NOTICE

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

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

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

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

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

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

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

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

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

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, C.1970, c C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS RÉCU
PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

Full Name of Author — Nom complet de l’auteur

JANET ALEXANDER

Date of Birth — Date de naissance
1 February 1962

Country of Birth — Lieu de naissance
CANADA

Permanent Address — Résidence fixe
1134 PLANTÉ DR. E
OTTAWA, ONTARIO
K1V 9E8

Title of Thesis — Titre de la thèse

STRUCTURAL PROPERTIES OF IEC-701

University — Université
CARLETON UNIVERSITY

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF COMPUTER SCIENCE

Year this degree conferred — Année d'obtention de ce grade
1985

Name of Supervisor — Nom du directeur de thèse
DR. JAN KÜMPEL-BAAS

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission

Date
2 April 1985

Signature

L'autorisation est par la présente accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilm cette thèse et de prêter ou de vendre des exemplaires du film

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur

Signature
Structural Properties of Polygons

by

James Alexander Dean, B.Sc.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for
the degree of
Master of Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
April 12, 1985

© Copyright
1985, James Dean
The undersigned recommend to the Faculty of Graduate Studies and Research acceptance of the thesis "Structural Properties of Polygons" submitted by James Alexander Dean, B Sc. in partial fulfillment of the requirements for the degree of Master of Computer Science.

[Signature]
Thesis Supervisor

[Signature]
Chairman, School of Computer Science

Carleton University

April 19, 1985

Date
Abstract

Many algorithms in computational geometry can be made more efficient by having prior knowledge of the structure of the input polygons. The purpose of this thesis is twofold: First, to show one way in which the amount of winding in a polygon's boundary can be measured and later used. Secondly, to discuss several classes of polygons and show how their structures are used to develop efficient algorithms. In particular, the new class of pseudo-star-shaped polygons is defined and several results and algorithms using the properties unique to this class are provided.
Acknowledgements

I wish to express my gratitude to the many people who helped make this thesis a reality. First and foremost in this group is my supervisor, Dr. Jörg-Rüdiger Sack, to whom I am forever indebted for the inspiration, enthusiasm, and patience he provided. I would also like to thank my family and my lovely fiancée, Karen, for their support, and a special thanks goes to Gerhard Roth, Wilf Sullivan, and Eugene Basic for helping me so much in putting the final package together.

This work was supported by a Natural Sciences and Engineering Research Council (NSERC) postgraduate scholarship.
Dedication

I dedicate this thesis to the memory of my grandfather, James Edgar Dean, a teacher of mathematics, who passed away in December 1984, several months before the completion of this thesis. I will never forget the twinkle in his eye the day he found me, at age 8, carefully examining the properties of a Möbius strip I had made. Since that time he always shared with me his great love of mathematics and given me a tremendous amount of encouragement, not to mention a seemingly endless supply of textbooks. Thank you, Grandpa.
# Table of Contents

- **ACCEPTANCE SHEET**: ii
- **ABSTRACT**: iii
- **ACKNOWLEDGEMENTS**: iv
- **DEDICATION**: v
- **TABLE OF CONTENTS**: vi
- **LIST OF FIGURES**: viii
- **LIST OF TABLES**: x

**Chapter 1: Introduction** 1
  1.1 Definitions 2
  1.2 Winding Properties of Polygons 4
  1.3 Results of this Thesis 9

**Chapter 2: Measuring the Winding Properties of Polygons** 11
  2.1 Fourier Descriptors 11
  2.2 Simplicity 13
  2.3 The Labelling Scheme for Rectilinear Polygons 16
  2.4 Conclusions 22

**Chapter 3: The Labelling Scheme** 23
  3.1 Introduction 23
  3.2 The Definition of the Labelling Scheme 24
  3.3 Some Properties of the Labelling Scheme 27
  3.4 The Hidden Line Removal Problem 30
    3.4.1 The Parallel Model of Visibility 30
    3.4.2 The Perspective Model of Visibility 38
  3.5 Conclusions 42

**Chapter 4: Monotonicity and the Labelling Scheme** 43
  4.1 Introduction 43
  4.2 Necessary Conditions for Monotonicity 43
  4.3 Testing for Monotonicity 46
    4.3.1 Preparata and Supowit's Algorithm 46
    4.3.2 An Enhanced O(n) Algorithm 51
  4.4 Applying Monotonicity 59
  4.5 Conclusions 61

**Chapter 5: Star-Shaped Polygons** 62
  5.1 Introduction 62
  5.2 Using Star-Shaped Polygons 68
  5.3 Edge-Visibility 72
  5.4 Conclusions 74
Chapter 6: Pseudo-Star-Shapedness
  6.1 Introduction ........................................ 75
  6.2 Outer-Kernels ........................................ 75
  6.3 Testing for Pseudo-Star-Shapedness ............... 88
  6.4 Applying Pseudo-Star-Shapedness ................. 92
  6.4.1 Other Classes of Polygon ....................... 93
  6.4.2 The Spy Problem ................................ 98
  6.4.3 Point Inclusion ................................ 99
  6.4.4 Union and Intersection ......................... 100
  6.4.5 Triangulation .................................. 103
  6.5 Conclusions ........................................ 105

Chapter 7: Concluding Remarks and Open Problems .... 106

References .............................................. 109
List of Figures

1.1 The Hidden Line Removal Problem ............................................... 2
1.2 Right and Left Turns ........................................................................ 5
1.3 Two Polygons of Simplicity $s = 1$ .................................................. 7
1.4 A Labelled Rectilinear Polygon ....................................................... 9
2.1 Weakly Edge-Visible Polygons .......................................................... 15
2.2 A Labelled Rectilinear Polygon ....................................................... 17
3.1 Some Labelled Polygons .................................................................. 25
3.2 Non-Visible Edges ........................................................................... 31
3.3 Visibility Blocked by an Antagonist ................................................ 32
3.4 Visibility Blocked by Previous Edge ................................................. 33
3.5 The Hidden Line Removal Problem - Parallel Model ...................... 35
3.6 Pictures Produced by an Implementation of Algorithm 3.1 ................ 36
3.7 Pictures Produced by an Implementation of Algorithm 3.1 ............... 37
3.8 Transforming the Perspective Visibility Problem ............................ 40
3.9 Pictures Produced by a Perspective Model Visibility Algorithm ....... 41
4.1 A Non-monotone Chain .................................................................... 44
4.2 A Labelled Polygon .......................................................................... 48
4.3 The Final Polar Wedge Diagram ...................................................... 49
4.4 A Picture Produced by an Implementation of Algorithm 4.1 ............ 57
4.5 Pictures Produced by an Implementation of Algorithm 4.1 ............... 58
4.6 A Monotone Polygon with Slabs ...................................................... 60
5.1 Examples of Star-Shaped Polygons .................................................. 63
5.2 Pictures Produced by an Implementation of Algorithm [LeP79] .......... 65
5.3 A Picture Produced by an Implementation of Algorithm [LeP79] ........ 66
5.4 The Inner and Outer Cones of a Reflex Vertex ................................. 67
5.5 A Chain in a Non-Star-Shaped Polygon .......................................... 67
5.6 A Star-Shaped Polygon with Wedges .............................................. 69
5.7 Union and Intersection of 2 Star-Shaped Polygons ........................... 72
5.8 Edge-Visibility ................................................................................ 73
6.1 An Example of Keil's Pseudo-Star-Shaped Polygon ......................... 76
6.2 A Pseudo-Star-Shaped Polygon ..................................................... 77
6.3 Construction of $a_t$ ....................................................................... 78
6.4 Comparison of Monotonicity and Pseudo-Star-Shapedness ............... 78
6.5 Transforming the Problem ................................................................ 81
6.6 A Polygon with More Than One Outer-Kernel .................................. 82
6.7 Support Vertices .............................................................................. 84
6.8 Unbounded Outer-Kernels and Monotonicity 86
6.9 Unbounded Outer-Kernels are not Unique 87
6.10 Merged Outer-Kernels 88
6.11 Computing $A$ 91
6.12 Computing $B$ 91
6.13 Weak External Visibility 94
6.14 Outer-Kernels Inside the Convex Hull 95
6.15 Weak External Visibility and Pseudo-Star-Shapedness 96
6.16 Star-Shapedness and Pseudo-Star-Shapedness 97
6.17 The Point Inclusion Problem 99
6.18 Union and Intersection of Pseudo-Star-Shaped Polygons 101
6.19 Union and Intersection of Pseudo-Star-Shaped Polygons 102
6.20 Decomposition to Weakly Edge-Visible Pieces 104
List of Tables

3.1 Relationship Between Direction of Visibility and Labels ........................................... 32
4.1 The Wedges for the Polygon in Figure 4.2 ............................................................... 48
4.2 The Wedges at Termination of the Algorithm ............................................................ 49
4.3 The Interval List After Preprocessing ......................................................................... 56
Chapter 1  Introduction  Page 1

Introduction

Computational geometry, as presented by Shamos [Sha78], is the study of solving geometric problems efficiently by using computers. This young field is one of growing interest and practical importance. For example, problems in such areas as pattern recognition, graphics, and VLSI, often involve finding solutions to geometric problems. Using computational geometry, one can design efficient, and sometimes optimal, algorithms for solving many of these geometric problems.

In order to see how computational geometry techniques can lead to efficient algorithms, consider the hidden line removal problem in two dimensions. In this problem we wish to determine which parts of a two-dimensional object are visible to a person standing at a given point on the plane, called the viewpoint. This problem arises in computer graphics when projecting a three-dimensional object on a plane. Let this object be represented by a polygon. Then the problem becomes finding which edges, or parts of edges, are visible from the viewpoint. See Figure 1.1.

As a brute force approach we might test each edge of the polygon with each of the remaining edges to find which edges, or parts of edges, are visible from the viewpoint. If the polygon has \( n \) edges then the number of tests is \( O(n^2) \). Notice that as \( n \) increases, the number of tests grows dramatically.

One way to streamline an inefficient algorithm is to look for certain properties in the polygon being tested. For example, if a polygon has a spiral shape, then in the hidden line removal problem it is not necessary to test the part of the polygon which has spiralled inwards for visibility from outside. More formally, using computational geometry techniques we try to isolate structural properties of the polygon which would lead to a significantly smaller number of tests.

There are several kinds of structural properties of polygons besides being spiral shaped,
trigonometric, exponential, and logarithmic functions, and indirect addressing is also included.

A polygon $P$ will be represented by a list of its vertices, $p_1, \ldots, p_n$ listed in clockwise order along the boundary of $P$. Each vertex $p_i$ is specified by its Cartesian coordinates, denoted by $p_i, x$ and $p_i, y$. The point with maximum $y$-coordinate will be called the $y$-max point, or $p_y\text{-max}$ (in case of a tie, we use the point with larger $x$-coordinate); similarly the point with minimum $y$-coordinate will be the $y$-min point, $p_y\text{-min}$ (in case of a tie, we use the point with the smaller $x$-coordinate). A similar notation will hold for the points with maximum and minimum $x$-coordinate. Furthermore, we permit modulo operations on indices so that $p_0 = p_n, p_{n+1} = p_1$, etc.

All polygons discussed in this thesis are assumed to be simple, i.e. not self-intersecting. By the Jordan Curve Theorem [CoR41], a polygon partitions the plane into two regions, the interior and the exterior. A point $v$ is inside a polygon if it lies in the interior of the polygon; outside otherwise. We shall include the edges which define the polygon, the boundary, in the interior. Unless stated otherwise, a polygon will be referred to as having $n$ vertices. A polygon $P = (p_1, \ldots, p_n)$ is in standard form if:

1. $p_1 = p_y\text{-max}$;
2. the vertices $p_1, \ldots, p_n$ are in clockwise order around $P$; and
3. no three consecutive vertices are collinear.

Unless otherwise stated, all polygons discussed in this thesis will assumed to be in standard form.

A polygon $P$ which has $n$ vertices also has $n$ edges, $e_1, \ldots, e_n$. Each edge $e_i$ is specified by an ordered pair of its endpoints, which are $p_i$ and $p_{i+1}$. The edges
trigonometric, exponential, and logarithmic functions, and indirect addressing is also included.

A polygon $P$ will be represented by a list of its vertices, $p_1, \ldots, p_n$ listed in clockwise order along the boundary of $P$. Each vertex $p_i$ is specified by its Cartesian coordinates, denoted by $p_i.x$ and $p_i.y$. The point with maximum $y$-coordinate will be called the $y\text{-max}$ point, or $p_{y\text{-max}}$ (in case of a tie, we use the point with larger $x$-coordinate); similarly the point with minimum $y$-coordinate will be the $y\text{-min}$ point, $p_{y\text{-min}}$ (in case of a tie, we use the point with the smaller $x$-coordinate). A similar notation will hold for the points with maximum and minimum $x$-coordinate. Furthermore, we permit modulo operations on indices so that $p_0 = p_n$, $p_{n+1} = p_1$, etc.

All polygons discussed in this thesis are assumed to be simple, i.e. not self-intersecting. By the Jordan Curve Theorem [CoR41], a polygon partitions the plane into two regions, the interior and the exterior. A point $v$ is inside a polygon if it lies in the interior of the polygon; outside otherwise. We shall include the edges which define the polygon, the boundary, in the interior. Unless stated otherwise, a polygon will be referred to as having $n$ vertices. A polygon $P = (p_1, \ldots, p_n)$ is in standard form if:

1. $p_1 = p_{y\text{-max}}$;
2. the vertices $p_1, \ldots, p_n$ are in clockwise order around $P$; and
3. no three consecutive vertices are collinear.

Unless otherwise stated, all polygons discussed in this thesis will assumed to be in standard form.

A polygon $P$ which has $n$ vertices also has $n$ edges, $e_1, \ldots, e_n$. Each edge $e_i$ is specified by an ordered pair of its endpoints, which are $p_i$ and $p_{i+1}$. The edges
of a polygon will therefore be directed from \( p_i \) to \( p_{i+1} \), and thus the interior of a polygon lies to the right of each of its edges.

Each edge \( e_i \) is associated with a line collinear with \( e_i \), having the same direction as \( e_i \). This line divides the plane into two half-planes, which are called the right half-plane and the left half-plane with respect to \( e_i \) and its direction.

A **polygonal chain** (also called a **polygonal line**) \( C_{i,j} = (p_i, p_{i+1}, \ldots, p_{j-1}, p_j) \) is the part of the boundary of a polygon \( P = (p_1, \ldots, p_n) \) encountered on a traversal of the boundary of \( P \) starting at \( p_i \) and ending at vertex \( p_j \). The first edge of \( C_{i,j} \) will therefore be \( e_i \), and the last edge will be \( e_{j-1} \).

For reasons which will be made clear in Chapter 3, we depart from convention and use the **gradient**, or **grad**, as the unit of angular measure. Note that \( \pi/2 \text{ radians} = 90^\circ = 100 \text{ grads} \).

A vertex \( p_i \) is **reflex** if the interior angle formed by edges \( e_{i-1} \) and \( e_i \) is greater than \( 200 \text{ grad} \); **convex** otherwise.

The **center** of a polygon will be the point with \( x \)-coordinate halfway between the \( x \)-min and \( x \)-max points and \( y \)-coordinate halfway between the \( y \)-min and \( y \)-max points. If \( c \) is the center of the polygon, then

\[
\begin{align*}
\text{c}.x &= (p_{x\text{-min}}.x + (p_{x\text{-max}}.x - p_{x\text{-min}}.x))/2; \\
\text{c}.y &= (p_{y\text{-min}}.y + (p_{y\text{-max}}.y - p_{y\text{-min}}.y))/2.
\end{align*}
\]

Note that the center of the polygon need not be included in the interior of the polygon.
1.2 Winding Properties of Polygons

One structural property that turns out to be useful in the hidden line removal problem is that of the amount of winding encountered when traversing the boundary of the polygon. That is, if we know how much, and in which directions, the polygon's boundary turns at each vertex, then we may have an idea which edges need not be tested for visibility from a point outside the polygon. We will show in Chapter 3 that this idea is a good one for more than just visibility testing. With this in mind, we define turn as follows:

The turn \( \theta \), at vertex \( p \), is equal to 200 \textit{grads} minus the interior angle at \( p \).

For a reflex vertex \( p \), \( \theta < 0 \), and we call this a \textit{left}, or \textit{counter-clockwise}, turn.

At a convex vertex \( p \), \( \theta > 0 \) is a \textit{right}, or \textit{clockwise}, turn. See Figure 1.2.

Unless the obvious distinction is made, we express the size of a turn by its absolute value. That is, a left turn of 100 \textit{grads} implies \( \theta = -100 \) and a right turn of 100 \textit{grads} implies \( \theta = +100 \). Given a polygonal chain \( C_{1:j} = (p_1, \ldots, p_j) \), we can calculate the sum of the turns at each vertex \( p_{i+1}, \ldots, p_{j-1} \). This sum represents the amount of turn in the polygonal chain. Equivalently, we say the polygonal chain \( C_{1:j} \) turns some number of \textit{grads} right or left, and the amount of turn is given by the sum.
A theorem we will refer to in subsequent chapters which can be applied to turns is the
Theorem of Turning Tangents [Kh78], [Car76]. A complete clockwise traversal of a simple
polygon describes a $2\pi$ revolution. We use the Theorem of Turning Tangents in this way.

The sum of the turns made during a complete clockwise traversal of a polygon is a
right turn of 400 grads, that is

$$\sum_{i=1}^{n} \theta_i = 400 \text{ grads}$$

While the Theorem of Turning Tangents reveals something about polygons in general,
it does not by itself reveal anything special about the winding properties of a given poly-
gen. In order to extract this information from a given polygon, three concepts have been
proposed by Zahn and Roskies [ZaR72], Chazelle [Cha83] and Sack [Sac83]. Fourier des-
tcriptors, sinuosity and the labelling scheme, respectively. We will briefly introduce these
ideas here, with a more complete description of each following in Chapter 2.

Zahn and Roskies [ZaR72] use a set of numbers to describe a plane closed curve. Since
a simple polygon is a plane closed curve, the idea can be applied to polygons. A starting
point on the boundary of the polygon is chosen arbitrarily. Then the boundary of the
polygon is represented by a function of cumulative change in the direction of the boundary
from the starting point. This function is thus a function of boundary (or arc) length. By
expanding this function in a Fourier series, we may use the coefficients of the series to
describe the curve. This method is quite complicated in comparison to the other ideas, as
we shall show when we take an in-depth look at it in Chapter 2.

Chazelle [Cha83] developed the notion of sinuosity to count the number of times the
boundary of a polygon alternates between spirals of opposite orientation. Specifically, the
quantity $s$ measures the sinuosity of a polygon as follows: We let $C_{ij} = (p_i, \ldots, p_j)$ be a
simple polygonal chain and initialize a counter to 0. Consider the motion of the straight
line passing through edge $e_k$ as $k$ goes from $i$ to $j - 1$. Every time the line reaches the
vertical position clockwise (counter-clockwise) we increment (decrement) the counter. We
say that $C_{i,j}$ is *spiralng* (*anti-spiraling*) if the counter is never decremented (incremented) twice in a row. Any simple polygon $P$ can be decomposed into spiralng and anti-spiraling polygonal lines in a single linear-time pass. Note that a new polygonal line starts only when the previous line ceases to be spiralng or anti-spiraling. In general, a line will be of both types at the start and then fall into one type. The *sinuosity*, $s$, is the number of polygonal lines obtained by this procedure.

We can see that the larger $s$ is, the more winding the boundary of the polygon is. Chazelle states that in practice, $s$ is usually a small integer and at worst, $s$ is a slowly growing function of $n$. While sinuosity proves to be a fairly good measure of the winding properties of a polygon, it does not tell everything about the winding properties of a polygon's boundary. For example, we give the polygons in Figure 1.3. These are two polygons of equal sinuosity, but they have different shapes. Therefore, another measure of the winding properties of a polygon is desirable.

Another method of extracting winding property knowledge about a given polygon is
Tangents (see Chapter 1, §1.1). It can also be seen that when \( \gamma \) is a rectilinear polygon, \( \phi(l) \) is analogous to the labels calculated by the labelling scheme.

We wish to expand \( \phi(l) \) in a Fourier series, which is a periodic function. Therefore, we normalize to the interval \([0, 2\pi]\) as follows

\[
\phi^*(t) = \phi\left(\frac{Lt}{2\pi}\right) + t
\]

The essence of this is that \( \phi^*(t) \) measures the way that \( \gamma \) differs from a circular shape. Expanding as a Fourier series,

\[
\phi^*(t) = \mu_0 + \sum_{k=1}^{\infty} (a_k \cos kt + b_k \sin kt)
\]

which, in polar form is

\[
\phi^*(t) = \mu_0 + \sum_{k=1}^{\infty} A_k \cos (kt - \alpha_k)
\]

where \((A_k, \alpha_k)\) are the polar coordinates of \((a_k, b_k)\). \(A_k\) and \(\alpha_k\) are known as the Fourier Descriptors (FDs) of \(\gamma\).

Zahn and Roskies also applied this concept to polygons. Let \( P = (p_1, \ldots, p_n) \) be a polygon, such that each edge \( e_i \) has length \( \Delta l_i \), so that \( L = \sum_{i=1}^{n} \Delta l_i \). The change in angular direction at vertex \( p_i \) is \( \Delta \phi_i \) (this is the turn defined in Chapter 1) and \( \phi(l) = \sum_{i=1}^{n} \Delta \phi_i \).

* Manipulating in a manner similar to the above, we arrive at

\[
\phi^*(t) = \mu_0 + \sum_{n=1}^{\infty} A_n \cos(nt - \alpha_n)
\]

with Fourier Descriptors \( A_n \) and \( \alpha_n \). \( A_n \) is called the \( n^{th} \) harmonic amplitude and \( \alpha_n \) is called the \( n^{th} \) harmonic phase angle.

As Zahn and Roskies show, FDs are a useful tool in several ways. For example, given a set of Fourier Descriptors, it is possible, though not immediately obvious, to reconstruct the original curve/polygon. This cannot be done with the labels generated by the labelling scheme for simple polygons defined in Chapter 3 of this thesis.
1.3 Results of this Thesis

In this thesis, we expand the labelling scheme so that it is applicable to all simple, not necessarily rectilinear, polygons. We do this by defining the scheme so that the labels change relative to the turn at each vertex, i.e., $l_i = l_{i-1} + e	heta_i$, where $c$ is a constant based on the unit of angular measure. In the following chapter, we will expand on the descriptions in §1.2 of the methods of describing the winding properties of a given polygon. In particular, we show how the labelling scheme has been applied to problems in rectilinear computational geometry.

In Chapter 3 we will define the polygon labelling scheme for simple polygons, show that it is a generalization of the scheme for rectilinear polygons, and prove several of its
properties. In subsequent chapters we will show how the labelling scheme and its properties can be used. For example, in Chapter 4, we give an example of how to design an efficient, conceptually clear algorithm based on the winding information provided by the labelling scheme.

A polygonal chain \(C_{i,j}\) is **monotone** with respect to a line \(\ell\) if the projections of the vertices \(p_k, (k = 1, \ldots, j)\), on \(\ell\) are ordered in exactly the same way as they are on \(C_{i,j}\). Furthermore, a polygon \(P = (p_1, \ldots, p_n)\) is **monotone** if it can be partitioned into 2 polygonal chains \(C_{i,j}\) and \(C_{j,k}\) which are both monotone with respect to the same line \(\ell\). Chapter 4 is a study of how the labelling scheme for simple polygons can be used to test a polygon for monotonicity.

Chapters 5 and 6 concentrate on some of the aspects of visibility. Formally, we say that two points \(u\) and \(v\) are **visible** if the line segment joining them, denoted \(uv\), lies entirely in the interior, or entirely in the exterior, of a polygon. Alternatively, we say that \(u\) and \(v\) see each other. We distinguish between internal and external visibility when the need arises.

One aspect of internal visibility is **star-shapedness**. A polygon \(P\) is **star-shaped** if there exists at least one point \(u\) inside \(P\) such that \(u\) sees every point \(v\) of \(P\). The set of points which can see every point of \(P\) is called the **kernel** of \(P\). If \(P\) is star-shaped then the kernel is non-empty; otherwise it is empty. In Chapter 5 we discuss star-shapedness, some of its properties, and show how the structure of star-shaped polygons has been exploited in certain algorithms.

Finally, in Chapter 6 we present the main result of this thesis, a new class of polygons that we call **pseudo-star-shaped**. We will show that pseudo-star-shapedness is related to monotonicity and star-shapedness, and that in some cases the structure of pseudo-star-shaped polygons can be used to create simple, clear, algorithms. We leave a summary of this thesis and some open problems in Chapter 7.
Measuring the Winding Properties of a Polygon

In Chapter 1, §1 2, we introduced three concepts for measuring the amount of turn in the boundary of a simple polygon. In this chapter we present a summary of the literature concerning those methods. We first describe the method of Fourier Descriptors, and then Chazelle's sumosity. This is followed by Sack's integer labelling scheme for rectilinear geometry. During these discussions, we will show that the labelling scheme we will propose has certain advantages over the known methods for measuring the winding properties of polygons.

2.1 Fourier Descriptors

In order to use a finite set of numbers to describe a plane closed curve, Zahn and Roskies [ZaR72] expanded on an idea that had been introduced 12 years earlier. Given a plane, closed curve, a starting point on the boundary is chosen and the curve is represented by a function of cumulative change in direction of the curve from the starting point, which is in turn a function of arc length. This function is expanded in a Fourier series, and the coefficients of the series are used to describe the curve.

For a complete discussion of Fourier Descriptors, how to obtain them and how to use them, see [ZaR72]. We will present a brief summary of that work, using its terminology. Let \( \gamma \) be a closed curve in the plane with clockwise orientation. We represent \( \gamma \) parametrically as \( Z(l) = (x(l), y(l)) \), where \( l \) is arc length from some starting point, and \( L \) is the entire length of \( \gamma \). Let \( \theta(l) \) be the function which denotes the angular direction of \( \gamma \) at point \( l \) (i.e. the angle of the tangent at \( l \)), and \( \delta_0 = \theta(0) \) is the tangent angle at the starting point.

Define the cumulative angular function \( \phi(l) \) as the net amount of angular bend between the starting point and point \( l \). Note that \( \phi(l) + \delta_0 = \theta(l) \) plus some multiple of \( 2\pi \) radians. At this point, we note that \( \phi(L) = 2\pi \) radians as a consequence of the Theorem of Turning
Tangents (see Chapter 1, §1.1). It can also be seen that when \( \gamma \) is a rectilinear polygon, \( \phi(t) \) is analogous to the labels calculated by the labelling scheme.

We wish to expand \( \phi(t) \) in a Fourier series, which is a periodic function. Therefore, we normalize to the interval \([0, 2\pi]\) as follows:

\[
\phi^*(t) = \phi\left(\frac{Lt}{2\pi}\right) + t.
\]

The essence of this is that \( \phi^*(t) \) measures the way that \( \gamma \) differs from a circular shape. Expanding as a Fourier series,

\[
\phi^*(t) = \mu_0 + \sum_{k=1}^{\infty} (a_k \cos kt + b_k \sin kt)
\]

which, in polar form is

\[
\phi^*(t) = \mu_0 + \sum_{k=1}^{\infty} A_k \cos (kt - \alpha_k)
\]

where \((A_k, \alpha_k)\) are the polar coordinates of \((a_k, b_k)\). \(A_k\) and \(\alpha_k\) are known as the Fourier Descriptors (FDs) of \(\gamma\).

Zahn and Roskies also applied this concept to polygons. Let \( P = (p_1, \ldots, p_n) \) be a polygon, such that each edge \( e_i \) has length \( \Delta l_i \), so that \( L = \sum_{i=1}^{n} \Delta l_i \). The change in angular direction at vertex \( p_i \) is \( \Delta \phi_i \) (this is the turn defined in Chapter 1) and \( \phi(l) = \sum_{i=1}^{k} \Delta \phi_i \).

Manipulating in a manner similar to the above, we arrive at

\[
\phi^*(t) = \mu_0 + \sum_{n=1}^{\infty} A_n \cos (nt - \alpha_n)
\]

with Fourier Descriptors \( A_n \) and \( \alpha_n \). \( A_n \) is called the \( n^{th} \) harmonic amplitude and \( \alpha_n \) is called the \( n^{th} \) harmonic phase angle.

As Zahn and Roskies show, FDs are a useful tool in several ways. For example, given a set of Fourier Descriptors, it is possible, though not immediately obvious, to reconstruct the original curve/polygon. This cannot be done with the labels generated by the labelling scheme for simple polygons defined in Chapter 3 of this thesis.
Fourier descriptors are similar to O'Rourke's Signature of a curve [ORo82], in which every point $p$ of a plane closed curve $\Gamma$ is associated with the length of $\Gamma$ to the left of $p$ or on the line tangent to $\Gamma$ at $p$. Signatures are particularly useful when applied to rectilinear computational geometry. To quote O'Rourke:

"In general the signature of a curve does not uniquely identify the curve. For example, all smooth convex figures with the same perimeter have the same signature: a constant equal to the perimeter. But there is an important special class of curves for which the signature is a unique identifier: closed rectilinear curves..."

The main advantage of the labelling scheme over FDs is its simplicity and clarity. Recall that $\psi(l)$ very closely resembles the labels, and expanding into a Fourier series seems not only to complicate matters, but to take us away from our objective: describing algorithms and properties in conceptually clear form. For the problems studied in this thesis, Fourier Descriptors contain far more information than is needed. In the following section, we look at a concept that provides less information than the labelling scheme.

2.2 Sinuosity

Although it is known that some classes of polygons (e.g. monotone) can be triangulated in $O(n)$ time, it is not yet clear whether or not every polygon can be triangulated in $O(n)$ time. Shamos [Sha78] proves that $\Omega(n \log n)$ comparisons are necessary to triangulate $n$ points in the plane, and several algorithms which triangulate simple polygons in $O(n \log n)$ time have been written, including that of Chazelle [Ch83]. However, this lower bound does not hold for simple polygons because the boundary of a polygon is a set of $n$ points in the plane which have the additional property of being connected by a simple path. Thus it is open whether triangulating any simple polygon can be accomplished in linear time. The motivation for sinuosity was to help the triangulation algorithm approach $O(n)$ complexity for simple polygons.
Simply put, the idea behind sinuosity is to partition a given polygon into a number, \( s \), of spiraling or anti-spiraling chains. Chazelle shows that a polygon \( P \) with sinuosity \( s \) can be triangulated in \( O(n \log s) \) time. Since \( s \) is generally a small constant or a slowly growing function of \( n \), a triangulation algorithm in conjunction with sinuosity would generally be efficient, i.e. it would approach \( O(n) \). In the worst case, however, this is still \( O(n \log n) \).

Recall from §1.2 the details of the definition of sinuosity. Given a simple polygon \( P \), it can be decomposed into spiraling and anti-spiraling polygonal chains in a single, linear-time operation. A new polygonal chain starts only when the previous one ceases to be spiraling or anti-spiraling. The sinuosity, \( s \), is the number of polygonal lines obtained by this procedure.

The resulting triangulation algorithm is of the divide-and-conquer type. Given a polygon \( P \), it is partitioned into \( s \) spiraling or anti-spiraling chains. Each chain is then decomposed into vertical trapezoids in linear time. The vertical decompositions are merged in \( O(n \log s) \) time, and since a vertical trapezoidal decomposition can be triangulated in linear time, the complexity of the algorithm is \( O(n \log s) \).

Sinuosity provides a means to triangulate most simple polygons very efficiently, so why do we consider trying to find another way of measuring winding properties (Chazelle uses the phrase shape complexity) of polygon boundaries? One reason is this: it appears that this triangulation algorithm is more complicated than necessary for several classes of polygons. For example, it is fairly straight-forward to triangulate a monotone polygon [Tou83], but under sinuosity, the triangulation algorithm would have us use a decomposition into trapezoids first.

There is another class of polygons which can be triangulated in linear time, called weakly edge-visible polygons [AvT81]. Weakly edge-visible polygons, unlike monotone polygons, do not fall into one category as far as sinuosity is concerned. We can easily construct a weakly edge-visible polygon which has sinuosity equal to a linear function
Figure 2.1 Weakly Edge-Visible Polygons

Note that the sinuosity is proportional to \( n \), (i.e. \( s = \frac{(n-1)}{g} \)).

of \( n \), and therefore the sinuosity triangulation algorithm runs in \( O(n \log n) \) time instead of the \( O(n) \) algorithm presented in [ToA81]. See Figure 2.1.

The labelling scheme for simple polygons has the advantage that sinuosity, if desired, is available from it. Given a polygon \( P = (p_1, \ldots, p_n) \), we start at \( p_1 \) and traverse the polygon. We compare consecutive labels to indicate how the line going through the edges moves. For example, if \( l_i < 1 + 4k \) and \( l_{i+1} > 1 + 4k \), where \( k \) is some integer, then the line has passed the vertical clockwise, and we decrement the sinuosity counter. (This will be clear after the formal definition in Chapter 3). Thus, in a simple traversal of \( P \), we can
calculate the sinuosity of $P$ directly from the labels. In this way we claim that the labels are more general because they contain all the sinuosity information and more.

2.3 The Labelling Scheme For Rectilinear Polygons

In Chapter 1 we introduced the inspiration for this thesis, the integer labelling scheme for rectilinear polygons. From it, we claim that we can develop a similar scheme for simple non-rectilinear polygons using real numbers, and show that the integer scheme can be seen as a subset of the real number scheme. This is shown in Chapter 3. In this section we will summarize some of the uses of the labelling scheme for rectilinear polygons presented in [Sac84].

Recall the definition of the integer labelling scheme. Let $P = (p_1, \ldots, p_n)$ be a simple rectilinear polygon in standard form. Then attach labels to the edges of $P$ as follows. We first set $l_0 = 0$. Then for $i = 1, 2, \ldots, n - 1$ we have

$$l_i = \begin{cases} l_{i-1} + 1 & \text{if } p_i \text{ is convex;} \\ l_{i-1} - 1 & \text{otherwise.} \end{cases}$$

We now review some of the properties of this labelling scheme (the proofs are omitted). We have included Figure 2.2 as a representative example. The first property, which we will show in Chapter 3 (see Property 3.7), is that horizontal edges have even labels, and vertical edges have odd labels. Secondly, the net turn between parallel edges oriented in the same direction is zero. (That is, the cumulative turn is a multiple of 400 grades).

Because half of the edges are vertical and half of the edges are parallel in such a polygon, certain structural properties are easier to extract than they would be for a non-rectilinear polygon. For example, given a polygon $P$, a horizontal cut $h$ is defined as a horizontal line which intersects each vertical edge at the point with the same $y$-coordinate as $h$. Then $(e_1, \ldots, e_k)$ denotes the list of edges intersected by $h$, where $e_1$ is the leftmost edge and $e_k$ is the rightmost. Vertical cuts are similarly defined. The set of labels $(l_1, \ldots, l_k)$ is called a cut-sequence.
Now, given a polygon $P$ and a horizontal or vertical cut, let $e_i$ and $e_j$ be two edges of $P$ adjacent in the cut. Let $q_i q_j$ be the part of $h$ which joins $e_i$ and $e_j$. Define the function $\text{sign}(x)$ so that it returns $-1$ when $x < 0$, and $+1$ otherwise. The following properties result:

1. $q_i q_j$ is inside $P$ iff $l_j - l_i = +2 \times \text{sign}(j - i)$; and
2. $q_i q_j$ is outside $P$ iff $l_j - l_i = -2 \times \text{sign}(j - i)$.
From these properties we learn two things. First, that the labels of edges adjacent in the cut differ by 2. Secondly, \( l_1 = 1 \) and \( l_3 = 3 \) for a horizontal cut. That is, rightmost edges have label equal to 1 and leftmost edges have label equal to 3. This follows as a consequence of the Jordan Curve Theorem, since \( l_2 = 1 \) and the polygon is simple.

A generalized cut-sequence is analogous to the horizontal and vertical cuts except that the line which intersects \( P \) is any oriented line. In this case, we know that the labels of edges adjacent in the cut-sequence differ by some integer, which is at most 3.

These properties are used to examine the effect of inserting line segments in both the interior and the exterior of a polygon \( P \) such that the segment joins two boundary points of \( P \). Unfortunately, the corresponding properties for the non-rectilinear case are not as well-defined. Instead of vertical edges, we have the larger categories of up-pointing and down-pointing edges, which have real, not integer, labels. Thus, in a horizontal cut, we can only say that the labels of edges adjacent in the cut differ by some real number, which is less than 4. The point is that the labels are especially powerful when combined with the rigid structure of rectilinear polygons, and some of this power is lost when they are applied to simple polygons in general.

Define a label sequence as the set of labels \( l_0, \ldots, l_{n-1} \) for a given polygon. A label sequence is valid if

1. \( l_0 = 0 \);
2. \( l_s = l_{s-1} \pm 1 \) for \( s = 1, \ldots, n - 1 \); and
3. \( l_{n-1} = 3 \).

(1) and (2) follow immediately from the definition, while (3) is a consequence of the Theorem of Turning Tangents.

The integral nature of this labelling scheme lends itself very nicely to a set of production rules useful for creating valid label sequences for rectilinear polygons. The production rules are defined as follows:
1. \( S \to 0\ 1\ A \)
2. \( x\ A \to x\ x + 1\ A \)
3. \( x\ A \to x\ x - 1\ A \)
4. \( 3\ A \to 3 \)

Here \( S \) is the Start and \( A \) stands for a sequence of labels. By rules (1) and (4), we see that \( l_0 = 0 \), \( l_1 = 1 \), and \( l_{n-1} = 3 \) as required for any rectilinear polygon. Rules (2) and (3) combine to give the definition of the general step in the labelling algorithm that 
\[ l_x = l_{x-1} \pm 1 \]

We made the comment in §2.1 that we can not reconstruct a polygon from a set of labels. However, given a valid sequence of labels, Sack provides a method to construct "all" rectilinear polygons. Similarly, a method of constructing a polygon from a set of turns is given in [CuR84]; it does not guarantee that a polygon can be reconstructed from its turns.

It is shown that if the labels of a given rectilinear polygon cannot be generated by the above set of production rules, then the polygon is not simple. This provides a linear-time test of a necessary condition for simplicity, a property which can be proved only in \( \Omega(n \log n) \) time [Jar84]. An algorithm which tests for simplicity in \( O(n \log n) \) time can be found in [Sha78].

Another use of the labels is found in a test for star-shapedness. That is, does there exist at least one point of \( P \) that sees every other point of \( P \)? Using \((ab)^*\) to denote zero or more repetitions of labels \( ab \), and using \((ab)^+\) to denote one or more such repetitions (i.e., \((ab)^+ = (ab)(ab)^*\)) it is shown that star-shaped rectilinear polygons must have a label sequence of the following form:
\[(0\ 1)^+(2\ 1)^+(2\ 3)^+(4\ 3)^*\]

In Chapter 5, we will show how the labelling scheme provides a similar necessary condition for star-shapedness in simple polygons.
Similarly, the labels can be used in tests of monotonicity. Recall that a polygon is monotone with respect to a line $\ell$ if it can be partitioned into two chains such that the vertices of each chain can be projected on $\ell$ in the order that they appear on the chain. Sack shows that

1. The label sequence is $(0\ 1)^+(2\ 3)^+$ if and only if $P$ is monotone to all lines between 100 and 200 grads.
2. The label sequence is $(0\ 1)(2\ 1)^+(2\ 3)(4\ 3)^+$ if and only if $P$ is monotone to all lines between 0 and 100 grads.
3. The label sequence is $((0\ 1)(2\ 1)^+)((2\ 3)(4\ 3)^+)$ if and only if $P$ is monotone with respect to the $y$-axis; and
4. The label sequence is $((0\ 1)(0\ -1)^+)((2\ 3)(2\ 1)^+)((4\ 5)^+(4\ 3))$ if and only if $P$ is monotone with respect to the $z$-axis.

Similarly, label sequences are developed to test rectilinear polygons for classes which are called pyramids and histograms. The most interesting use of label sequences is a test for another class of polygon called weakly edge-visible (we give the definition in §5.3). This test runs in linear time for rectilinear polygons: in general, a simple polygon can not be tested for this property in better than $O(n^2)$ time. In Chapter 4, we will use the labels to find all directions of monotonicity for a given simple polygon, although the method is not that of label sequences.

The labelling scheme can also provide visibility information. We first define two types of visibility. A boundary point $u$ of a polygon $P$ is externally visible from a point $v$ iff the open line segment $uv$ lies completely in the exterior of $P$. Point $u$ is internally visible from $v$ if the open line segment $uv$ lies completely in the interior of $P$.

For external visibility, we have two models: If the viewpoint $v$ is located at infinity, $\infty$, with respect to $P$, we have the parallel model, so called because the line segments $uv$ for each $u$ in $P$ are parallel. Otherwise, we have the perspective model. Note that internal visibility is always the perspective model. Sack presents algorithms to solve the hidden line
removal problem for both models of visibility. We do not discuss his algorithms here since a detailed description generalizing the result obtained for rectilinear geometry is presented in the next chapter.

Finally, the integer labelling scheme is applied to the movement of robots. Given a rectilinear polygon \( P \) and a robot on a point in the exterior of \( P \), we wish to free the robot from the convex hull of \( P \), \( CH(P) \). The result is a shortest rectilinear path, in which all movements of the robot are north, south, east, or west. To accomplish this, the robot "looks" in these four directions. We assume that what he sees is the label, rather than the facing edge. That is, to the north of him he sees an edge, which we call the north edge (with respect to the current robot position), to the west he sees the west edge, etc. If the robot can see the edge, we allow him to know the label. If there is no facing edge, we say that the label is equal to \( \infty \). The algorithm runs as follows:

Pick an arbitrary orientation and look at the facing edge.

\[
\text{WHILE (Robot is inside } CH(P)) \text{ DO}
\]
\[
\text{BEGIN}
\]
\[
\text{IF (label = } \infty \text{) THEN}
\]
\[
\text{MOVE towards that edge and exit;}
\]
\[
\text{ELSE}
\]
\[
\text{BEGIN}
\]
\[
\text{IF (facing edge is horizontal) THEN}
\]
\[
\text{IF (west edge label < east edge label) THEN}
\]
\[
\text{MOVE south;}
\]
\[
\text{ELSE}
\]
\[
\text{MOVE north;}
\]
\[
\text{ELSE}
\]
\[
\text{IF (north edge label < south edge label) THEN}
\]
\[
\text{MOVE west;}
\]
\[
\text{ELSE}
\]
\[
\text{MOVE east;}
\]
\[
\text{END}
\]
\[
\text{MOVE until side edges change labels or robot hits the facing edge THEN change orientation;}
\]
\[
\text{END}
\]
\[
\text{END}
\]

The result is a shortest rectilinear path out of the convex hull of \( P \). Tracing the algorithm, we notice that whenever the robot is in motion, the label on its left is always greater than the label on its right. After defining a labelling scheme for general simple
polys, we will be able to use a similar method to remove a robot from the convex hull of a polygon, but it is likely that the path will not be rectilinear.

2.4 Conclusions

In this chapter we have compared three known methods for determining the winding properties of a polygon. The first is called Fourier Descriptors. They involve lengthy formulae, and since they include information about the length of edges, FDs tell us more than we need to know for now. The second concept is known as sinuosity. It proves to be a good measure of shape complexity, but it didn't allow quick detection of certain classes of polygon.

The third, and most attractive idea for our purposes is that of the labelling scheme for rectilinear polygons. As we have shown, the information in the labelling scheme is also used by Fourier Descriptors, and the sinuosity of a polygon can be derived from the labels created by such a scheme. We also showed some of the uses of the integer labels in rectilinear polygons, and their power when combined with the inherent structure of such polygons.

In Chapter 3 we will formally define an extension of the integer labelling scheme which uses real numbers. In two subsequent chapters we will show that the real number labelling scheme can be used for many of the same purposes when applied to general simple polygons.
The Labelling Scheme

In this chapter we define a tool for computational geometry, the labelling scheme. This concept was first introduced by Sack [Sac83] and applied to rectilinear polygons (see also [Sac84]). We give an alternate definition and show its applications to arbitrary simple polygons. We state and prove several properties that can be derived from the labelling scheme, and then apply it to the hidden line removal problem.

3.1 Introduction

The idea of using a labelling scheme is to extract knowledge about the winding properties of a given polygon. Previously the labelling scheme was employed to exploit the structure of rectilinear polygons. Using this tool as a preprocessing step to provide structural information became a means of improving the efficiency of certain algorithms in rectilinear computational geometry; it also enabled the elimination of some cases and subcases tested in the body of some algorithms, which makes clearer algorithms. We shall see in Chapter 4 how, given an existing algorithm, the labelling scheme can lead to a clearer and simpler algorithm.

The labelling scheme for rectilinear polygons is defined as follows: The turn at a vertex of a rectilinear polygon is either a right turn or a left turn of \(\pi/2\) radians. We arbitrarily choose the horizontal edge with maximum y-coordinate, \(e_0\), and assign it label \(l_0 = 0\). Then \(l_i = l_{i-1} + 1\) if \(\theta_i\) is a right turn; \(l_i = l_{i-1} - 1\) otherwise. Now the reason for using \(\text{grads}\) becomes clear: \(l_i = l_{i-1} \pm 1\), and each turn is \(\pm 100\) \(\text{grads}\). We can derive the labels from the turns, and using \(\text{grads}\) as the unit of angular measure makes the conversion very simple.

An edge \(e_i\) of a rectilinear 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 $c_i$, respectively. Recall Figure 2.2. Observation, which we shall later prove to be correct, shows that top-edges $e_t$ have labels $l$, such that $l \mod 4 = 0$, and for bottom-edges, $l \mod 4 = 2$. Similarly, for left-edges $l \mod 4 = 3$, and for right-edges $l \mod 4 = 1$.

In simple, non-rectilinear, polygons the property $\theta_i = \pm 100 \text{ grads}$ is not necessarily valid. In fact, the restriction on the turns becomes $-200 < \theta_i < 200$ for all $i$. This implies that it may be difficult to create an integer labelling scheme for simple polygons but it does not exclude the possibility of a real number labelling scheme. In the next section we define such a labelling scheme.

3.2 The Definition of the Labelling Scheme

The labelling scheme was introduced in an algorithm to solve the hidden line removal problem in rectilinear geometry [Sac83]. Although applied only in the context of rectilinear geometry, the labelling scheme was also defined in the general form. We present an alternate definition using grads.

Let $P$ be a simple polygon in standard form. (Note $\theta_i \neq 0 \forall i$). Then calculate $\alpha_i$, the clockwise angle made by $e_i$ and the horizontal line through $p_i$. Then $l_i = \alpha_i / 100$. Subsequently,

$$l_i = l_{i-1} + \frac{\theta_i}{100}, \quad i = 2, \ldots, n.$$  

See Figure 3.1.
Figure 3.1 Some Labelled Polygons
Property 3.1: For rectilinear polygons, the labelling scheme will provide the same labels as that of Sack’s (with the exception of $l_n$).

Proof: Let $P$ be a rectilinear polygon in standard form. Then $e_n$ is the horizontal edge with maximum $y$-coordinate. By Sack’s scheme, $l_n = l_n = 0$. Since $p_1 = p_{y_{max}}$, $\theta_1 = 100$ grades and therefore $l_1 = 1$.

Consider $P$ and the labelling scheme defined above. Edge $e_1$ is clearly a right-edge with $p_1, y < p_1, y$. Thus $\alpha_1 = 100$ grades and therefore $l_1 = 1.0$.

In the integer scheme, each subsequent turn causes a difference in consecutive labels of $+1$ for right turns and $-1$ for left turns. In the real number scheme, each right turn also causes an increase of 1 (since $\theta/100 = 100/100 = 1$) and each left turn causes a decrease of 1. Thus, labels $l_i$, $i = 2, \ldots, n - 1$, from the two schemes will be equivalent. We know that the integer scheme has $l_0 = l_n = 0$. The real number scheme started the traversal at $e_1$. By the time it returns to $p_1$, a net turn of $+400$ grades will be made, the result of the Theorem of Turning Tangents. Therefore the final edge’s label will be $l_n = 4.0$. ■

Property 3.2: If $P$ is a polygon labelled according to the scheme with labels $l_1, \ldots, l_n$, then $0 < l_1 < 2$ and $2 < l_n < 4$.

Proof: By definition, $l_1 = \alpha_1/100$ where $\alpha_1$ is the clockwise angle made by $e_1$ and the horizontal line through $p_1$. Since $p_1 = p_{y_{max}}$, $e_1$ is directed downwards and $0 < \alpha_1 < 200$, which implies the result for $l_1$. A clockwise traversal of the polygon starting at $e_1$ and returning to $e_1$ is a $+400$ grade turn, by the Theorem of Turning Tangents. The turn from $e_n$ to $e_1$ is a right turn $0 < \theta_n < 200$ since the angle at $p_1$ is convex. Thus $l_n = l_1 - \theta_n/100 + 4$ and therefore $2 < l_n < 4$. ■

Since Property 3.1 holds, the labelling scheme for simple polygons can be used, with only the slight of modifications, in the algorithms and proofs for rectilinear polygons. In the next section we will present some more properties of the labelling scheme.
3.3 Some Properties of the Labelling Scheme

The key idea of the labelling scheme is to find winding properties of a given polygon. The labelling scheme can certainly provide the most basic of these properties, which vertices are reflex.

Property 3.3: If \( l_i > l_{i-1} \), then \( p_i \) is a reflex vertex, otherwise \( p_i \) is convex (except for \( p_1 \) which is always convex).

Proof: The proof follows from the definitions of the labelling scheme and turns. Let \( P \) be a polygon labelled according to the scheme. Since \( P \) is in standard form, \( p_1 = p_{n-1} \), and therefore \( p_1 \) is always convex. For a pair of labels \( l_i \) and \( l_{i-1} \), \( 2 \leq i \leq n \), \( l_i = l_{i-1} + \theta_n/100 \). Therefore \( l_i > l_{i-1} \) only when \( \theta_n > 0 \). This happens only at convex vertices. Similarly, \( l_i < l_{i-1} \) only implies \( \theta_n < 0 \), which occurs only at reflex vertices.

Notice that we always have \( l_n > l_1 \) as a consequence of the Theorem of Turning Tangents (see Property 3.2). This can lead to complications when testing arbitrary adjacent pairs of labels; if we happen to choose \( l_n \) and \( l_1 \) then the convexity of \( p_1 \) is not detected unless we know we are at \( p_1 \), or unless we know the indices of the labels. We attempt to counter this apparent problem as follows: Whenever we consider a polygonal chain \( C_{i,j} \), if \( i > j \) then \( p_i \) is on \( C_{i,j} \) and we add 4 to each label \( l_k \), \( k \leq i-1 \). This differs from the original labels, but it has the advantage that \( l_k > l_{k-1} \) always implies that \( p_k \) is convex for \( p_k \) on \( C_{i,j} \). Thus, no special case test is required at \( p_1 \). Therefore, for the purposes of discussion throughout this thesis, any time we consider a polygonal chain \( C_{i,j} \), \( i > j \), we assume the labels have been adjusted as above to eliminate any special case testing.

A convex polygon \( P \) is a polygon with no reflex vertices, i.e. the angle at \( p_i \) is convex for all \( i \). A simple test of the labels of a polygon can tell whether or not a polygon is convex.

Property 3.4: A simple polygon is convex if and only if \( l_1 < l_2 < \ldots < l_{n-1} < l_n \).
Proof: \(\Rightarrow\) Let \(P\) be a labelled polygon. Assume \(t_1 < t_2 < \ldots < t_{n-1} < t_n\). Then by Property 3.3, \(P\) has no reflex vertices, and therefore \(P\) is convex.

\(\Leftarrow\) Let \(P\) be a labelled convex polygon. Then no vertex of \(P\) is reflex. Then for \(s = 2, \ldots, n\), \(l_s > l_{s-1}\) by Property 3.3, and thus the result holds.  

As a consequence of Property 3.4 we have the following: Let \(C_{i,j}\) be a polygonal chain, part of a labelled polygon. Then we call \(C_{i,j}\) a convex chain if vertices \(p_{i+1}, \ldots, p_{j-1}\) are all convex. Equivalently, \(C_{i,j}\) is a convex chain if and only if \(l_i < l_{i+1} < \ldots < l_{j-1} < l_j\) (provided that the labels have been adjusted when \(p_1\) is on \(C_{i,j}\)).

Property 3.5: If \(p_i\) is the first reflex vertex in a labelled polygon \(P\) following a reflex vertex \(p_k\) (in a clockwise in-order traversal of \(P\)), then \(l_i < l_{k-1}\).

Proof: Let \(p_i\) be a reflex vertex in a labelled polygon \(P\). Begin a clockwise traversal of \(P\) at \(p_i\). Since the first reflex vertex encountered after \(p_i\) is \(p_k\), \(C_{i,j}\) is a convex chain. Therefore, by definition of convex chain, \(l_i < l_{k-1}\).

In rectilinear geometry, we have top-, bottom-, left-, and right-edges. In non-rectilinear geometry we define the following types of edges:

An edge \(e\) is down-pointing if \(p_{i+1}.y < p_i.y\), since edges are directed from \(p_i\) to \(p_{i+1}\). Similarly, \(e\) is up-pointing if \(p_{i+1}.y > p_i.y\). Horizontal edges will be neither up-pointing nor down-pointing. An edge \(e\) is left-pointing if \(p_{i+1}.x < p_i.x\) and \(e\) is right-pointing if \(p_{i+1}.x > p_i.x\). Vertical edges will be neither left-pointing nor right-pointing.

Notice that non-horizontal and non-vertical edges will belong to two categories. For instance, an edge directed from \((0,0)\) to \((1,1)\) will be both up-pointing and right-pointing.

For any edge \(e_i\), its label \(l_i\) can be used to determine the clockwise angle \(\alpha\), formed by \(e_i\) and the horizontal line through vertex \(p_i\). The label can also be used to determine in
which direction the edge points.

**Property 3.6:** Let \( e \), be an edge of a labelled polygon \( P \), with label \( l \). Then \( \alpha = 100 \times (l, \mod 4) \).

**Proof:** Recall that, by definition, \( l_i = \alpha_i / 100 \). Therefore this property holds for \( e_1 \). Let \( \theta_j \) be the turn from \( e_{j-1} \) to \( e_j \), for \( j = 2, \ldots, n \). Then \( \theta_j \) also represents the angle formed by \( e_{j-1} \) and a line starting at \( p_{j-1} \), parallel to \( e_j \). By construction, \( l_j = l_1 + \sum_{i=2}^{j} \theta_i / 100 \) and therefore represents the sum (when multiplied by 100) of consecutive angles starting at the positive x-axis and ending at the angle with the same orientation as \( e_j \). Therefore the angle from the horizontal to \( e_j \) is equal to \( l_j \times 100 + \sum_{i=2}^{j} \theta_i \), the same as \( l_j \times 100 \). Since we generally use angles in the range \([0, 400] \text{ grads}\), we use \((l, \mod 4) \times 100 \).

**Property 3.7:** Let \( e \), be an edge of a labelled polygon \( P \), such that the label of \( e \), is equal to \( l \). Then the following relationships hold:

<table>
<thead>
<tr>
<th>Type of Edge ( e )</th>
<th>Label ( l )</th>
</tr>
</thead>
<tbody>
<tr>
<td>down pointing</td>
<td>0 &lt; ((l, \mod 4) &lt; 2)</td>
</tr>
<tr>
<td>left pointing</td>
<td>1 &lt; ((l, \mod 4) &lt; 3)</td>
</tr>
<tr>
<td>up pointing</td>
<td>2 &lt; ((l, \mod 4) &lt; 4)</td>
</tr>
<tr>
<td>right pointing</td>
<td>3 &lt; ((l, \mod 4) &lt; 4) or 0 ≤ ((l, \mod 4) &lt; 1)</td>
</tr>
</tbody>
</table>

**Proof:** The proof follows directly from Property 3.6. For example, if \( 0 < (l, \mod 4) < 2 \), then \( \alpha \), the angle made by \( e \), and the horizontal through \( p \), lies in the range \((0, 200)\). Clearly, such an angle belongs to a down-pointing edge.

Property 3.7 can be used to prove what was stated earlier: That is, in rectilinear geometry, top-edges have labels such that \( l, \mod 4 = 0 \), bottom-edges have labels \( l, \mod 4 = 2 \), right-edges have labels \( l, \mod 4 = 1 \), and left-edges have labels \( l, \mod 4 = 3 \). For instance, a right-edge is down-pointing since the interior lies to the left of it. Therefore, by Property 3.7, the label of this edge (modulo 4) lies in the range \((0, 2)\). However, rectilinear
polygons have integer labels and the only integer in this range is 1. Thus, right-edges have labels \( l \mod 4 = 1 \).

3. The Hidden Line Removal Problem

In the introductory chapter of this thesis we mentioned the hidden line removal problem: Given a polygon \( P \) and a point \( v \), called the viewpoint, which edges, or parts of edges, of \( P \) are visible from \( v \)? Both Lee [Lee83] and Sack [Sac83] have presented linear-time algorithms which solve the hidden line removal problem. In this section we will modify an existing algorithm to solve this problem given that the polygon has been labelled according to the labelling scheme.

The hidden line problem has two types of models as described in §2.3; the parallel model and the perspective model. Full discussions of these models are available in [NeS79], [FVD82], and [CaP78]. In the parallel model of visibility, the viewpoint \( v \) is located at infinity, \( \infty \), with respect to \( P \). If \( u_1 \) and \( u_2 \) are any two points of \( P \), then the line segments \( u_1v \) and \( u_2v \) are parallel. In the perspective model, \( v \) need not be located at infinity, and may even be located in the interior of \( P \). Thus line segments \( u_1v \) and \( u_2v \) are not necessarily parallel.

3.4.1 The Parallel Model of Visibility

In this section we present an algorithm which solves the hidden line removal problem for the parallel model of visibility. Without loss of generality, we assume that the viewpoint is located on the \( z \)-axis. We can do this since a polygon can be both rotated and labelled in \( O(n) \) time. Furthermore, we allow visibility along horizontal edges.

**Lemma 3.1**: If edge \( e \) is visible from \( v \) at infinity on the positive \( z \)-axis then \( 0 \leq l \leq 2 \).
Figure 3.2 Non-visible Edges

Edge $e$, has label $l$, $> 4.0$.

Proof: Edges with labels 0 or 2 are potentially visible from infinity on the $z$-axis since we allow visibility along horizontal edges. Down-pointing edges are potentially visible since the interior of $P$ lies to the right of any edge and in this case the viewpoint lies to the left of the down-pointing edge. Similarly, up-pointing edges are not visible. Thus, by Property 3.7, it is known that the potentially visible edges are those having label $l$, such that $0 \leq (l, \mod 4) \leq 2$.

Consider a down-pointing edge $e$, with $l > 4$. By definition of the labelling scheme, a traversal from $e_1$ to $e$, involves a right turn of more than 400 grada. Since $P$ is simple, this implies that there is at least one edge to the right of $e$, which blocks visibility from infinity. (See Figure 3.2.) Similarly, in the case of a down-pointing edge $e$, with label $l < 0$, the traversal from $e$, to $e_1$, involves a left turn of more than 400 grada, implying that there exists at least one edge which blocks visibility from the viewpoint. Thus the result holds.

By considering viewpoints at $x = -\infty$ and $y = +\infty$ and constructing proofs similar to that of Lemma 3.1, we construct Table 3.1.
<table>
<thead>
<tr>
<th>Location of Viewpoint</th>
<th>Range of Labels of Visible Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = +\infty, y = 0$</td>
<td>$[0, 2]$</td>
</tr>
<tr>
<td>$z = -\infty, y = 0$</td>
<td>$[2, 4] \cup [0]$</td>
</tr>
<tr>
<td>$z = 0, y = -\infty$</td>
<td>$[1, 3]$</td>
</tr>
<tr>
<td>$z = 0, y = +\infty$</td>
<td>$[3, 5] \cup [-1, 1]$</td>
</tr>
</tbody>
</table>

Table 3.1 Relationship Between Direction of Visibility and Label

Note that the second and the final entries have composite ranges; this is to account for the change in label sequence from $e_i$ to $e_j$.

Figure 3.3 Visibility Blocked by an Antagonist

Edge $e_j$ is the antagonist for edge $e_i$.

Not every example down-pointing edge $e_i$ with $l_i \in [0, 2]$ is visible. We present Figure 3.3 to illustrate this. No point $u$ on $e_i$ is visible from $v$ at $z = +\infty$ since the line segment $uv$ is intersected by several edges. We call the edge $e_j$, with the smallest index $j > i$ such that $e_j$ intersects $uv$ (and therefore blocks visibility) an antagonist for $e_i$.

**Lemma 3.2:** Let $e_i$ be an edge with label $l_i \in [0, 2]$. If point $u$ on $e_i$ is not visible from $v$ at $z = +\infty, y = 0$ then there exists either (a) an antagonist $e_j, j > i$, with $l_j \in (-2, 0)$
Figure 3.4 Visibility Blocked by a Previous Visible Edge

Edge $e_i$ is blocked by $e_k$.

and $u.y \leq p_{j+1}.y$ or (b) a visible edge $e_k$, $k < i$, with $u.y \geq p_{i+1}.y$, which blocks visibility.

Proof: Let $e_i$ be an edge with label $0 \leq l_i \leq 2$. By Property 3.7, $e_i$ is down-pointing or horizontal. Therefore it is potentially visible from $v = (\infty, 0)$, as indicated in Lemma 3.1. Assume no point of $e_i$ is visible from $v$. Then the horizontal half-line from any point $u$ of $e_i$, towards $v$ is intersected by at least one edge of $P$. Let $e_k$ be the edge with the rightmost intersection on the half-line. This intersection will be visible from $v$ since no edges lie between it and $v$. If $k < i$ then case (b) is true (Figure 3.4). Otherwise, $k > i$ (Figure 3.3). In this case, both $e_k$ and $e_i$ are down-pointing or horizontal, and thus, by the Jordan Curve Theorem and simplicity of the polygon, there exists an up-pointing edge $e_j$, $i < j < k$, which also intersects the half-line from $u$ to $v$. If there is more than one, let $e_j$ be the up-pointing edge with the intersection closest to $u$ on the half-line. Since $j > i$, a traversal can be made from $e_i$ to $e_j$ (without passing $p_i$), and this traversal is a left turn of no more than 400 grads. Since $0 \leq l_i \leq 2$, then $-4 \leq l_j \leq 2$. Furthermore, since $e_j$ is up-pointing, $-2 < l_j < 0$, and case (a) is true.

In Lemma 3.1 we describe the edges that we need to test for visibility, edges which have labels $l_i \in [0, 2]$. Lemma 3.2 describes the set of edges which block the visibility of potentially visible edges; previous edges and antagonists, edges with $l_j \in (-2, 0)$. Thus we
may now state the algorithm. The data structure used is a stack containing points which define the edges of the visibility chain, the polygonal chain which is visible from \( v \). The point at the top of the stack (the lowest point of the chain) is referenced by the command \( \text{TOP}(S) \).

**Algorithm 3.1: The Hidden Line Removal Problem – Parallel Model**

Input: A polygon \( P \) labelled according to the labelling scheme.

Output: The visibility chain, \( VC(P) \), as seen from the point at \( x = +\infty, y = 0 \).

BEGIN
\[
S := \emptyset; \quad (* \text{initialize an empty stack} *) \\
PUSH(p_1, S); \quad (* \text{extreme points are visible} *) \\
\text{FOR } i := 1 \text{ TO } n \text{ DO} \\
\quad \text{IF } (l, \in [0, 2]) \text{ AND} \\
\quad \quad (p_{i+1}.y \leq \text{TOP}(S).y) \text{ AND} \\
\quad \quad (p_i.y \geq \text{TOP}(S).y) \text{ THEN} \\
\quad \quad \text{BEGIN} \quad (* \text{push the visible part of } c, *) \\
\quad \quad \quad \text{IF } (\text{TOP}(S) \neq p_i) \text{ THEN PUSH}(p_i, S); \\
\quad \quad \quad \text{PUSH}(p_{i+1}, S); \\
\quad \quad \quad \text{END} \\
\quad \quad \text{ELSE IF } (l, \in [-2, 0]) \text{ AND} \\
\quad \quad \quad (p_i.y \leq \text{TOP}(S).y) \text{ THEN} \\
\quad \quad \quad \text{BEGIN} \quad (* c, may be an antagonist *) \\
\quad \quad \quad \quad \text{WHILE } (p_{i+1}.y > \text{TOP}(S).y) \text{ AND} \\
\quad \quad \quad \quad \quad (p_{i+1} \text{ lies to the left of the point in } S \text{ with the same } y\text{-coordinate}) \text{ DO} \\
\quad \quad \quad \quad \text{BEGIN} \quad (* \text{remove what the antagonist blocks} *) \\
\quad \quad \quad \quad \quad \text{LASTTOP} := \text{TOP}(S); \\
\quad \quad \quad \quad \text{POP}(S); \\
\quad \quad \quad \quad \text{END} \\
\quad \quad \quad \quad \text{IF } (p_{i+1}.y < \text{TOP}(S).y) \text{ THEN} \\
\quad \quad \quad \quad \text{BEGIN} \\
\quad \quad \quad \quad \quad q := \text{intersection of the horizontal through} \\
\quad \quad \quad \quad \quad p_{i+1} \text{ and edge } \text{TOP}(S) \text{ LASTTOP}; \\
\quad \quad \quad \quad \text{PUSH}(q, S); \\
\quad \quad \quad \quad \text{PUSH}(p_{i+1}, S); \\
\quad \quad \quad \quad \text{END} \\
\quad \quad \text{END} \\
\quad \text{END} \\
\text{END} \\
\text{END.}
\]

An example illustrating the parallel model is given in Figure 3.5.
Theorem 3.1: Algorithm 3.1 solves the hidden line removal problem for a viewpoint at \( \infty \) on the \( z \)-axis in \( O(n) \) time.

Proof: The complexity follows since each edge is tested exactly once. An edge which is removed from the stack is never returned to it. Since all we have changed from the original algorithm is the uses of ranges of labels instead of single labels, the proof of correctness follows from Lemmas 3.1 and 3.2, and from Sack’s proof.

Algorithm 3.1 has been implemented by the author in the C programming language on the Sun Graphics Workstation. Some pictures produced by sample runs of the program are given in Figures 3.6 and 3.7.
Figure 3.6 Pictures Produced by an Implementation of Algorithm 3.1.
Figure 3.7 Pictures Produced by an Implementation of Algorithm 3.1.
3.4.2 The Perspective Model of Visibility

In the perspective model of visibility problem, we wish to solve the hidden line removal problem from a viewpoint \( v \) somewhere on the plane. A point \( v \) in the exterior of a polygon is free exterior if there exists at least one half-line originating at \( v \) which does not intersect the polygon; blocked exterior otherwise. Thus, three cases of perspective visibility are distinguished [FrL67], depending on the location of the viewpoint:

1. Interior, if \( v \) is inside \( P \);
2. Blocked Exterior, and

It is shown in [Ne879] that parallel and perspective projections, and thus parallel and perspective visibility, are similar and therefore by mathematical manipulation one model can be transformed into the other. We will show how this is accomplished with an example of free exterior perspective visibility; interior and blocked exterior can be similarly shown by first partitioning the polygon with a line through \( v \) into two polygons such that \( v \) is free exterior with respect to both.

A polygonal chain is completely visible from a point \( v \) if \( v \) sees each point on the chain. The points of a completely visible chain are in angular order with respect to the viewpoint. In the parallel case it is known that the visibility chain is monotone, a linear ordering of the visible points. Similarly, the visibility chain in the perspective model is completely visible from the viewpoint and the visible points are angularly ordered with respect to the viewpoint. Thus, the transformation is a “bending” which works as follows: Given a polygon \( P = (p_1, \ldots, p_n) \) and a viewpoint \( v \), for each vertex \( p_i \), the Euclidean distance \( r_i \) from \( v \) to \( p_i \) is measured. The clockwise angle \( \theta \), made by the horizontal half-line from \( v \) towards \( z = +\infty \) and the line segment \( v p_i \), is also measured. In effect, the problem is positioned in a polar coordinate system with \( v \) at the origin. Subsequently, in a second coordinate system, this one rectangular, the points of the polygon are plotted so that \( -v \)
is on one axis, $\theta$ on the other. Note that a completely visible chain in the first system becomes a monotone chain in the second system.

The problem of parallel visibility is solved from $r = \infty$. This produces a monotone visibility chain. Finally, the reverse transformation is made, and the points of the visibility chain are seen to be angularly ordered, and visible, with respect to $v$ as required, completing the solution. We illustrate with the polygon of Figure 3.8(a).

We first measure the distance and angle, $r$, and $\theta$, respectively, for each point $p$, with respect to the viewpoint $v$. The results are given in the table below.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$, cm</td>
<td>5.1</td>
<td>0.8</td>
<td>4.4</td>
<td>1.4</td>
<td>2.6</td>
<td>2.0</td>
<td>5.8</td>
<td>1.5</td>
<td>2.3</td>
<td>0.9</td>
<td>2.6</td>
<td>5.5</td>
<td>3.4</td>
<td>6.3</td>
</tr>
<tr>
<td>$\theta$, grads</td>
<td>86.7</td>
<td>38.9</td>
<td>91.1</td>
<td>128.9</td>
<td>184.4</td>
<td>121.1</td>
<td>176.7</td>
<td>230.0</td>
<td>263.3</td>
<td>250.0</td>
<td>322.2</td>
<td>250.0</td>
<td>225.5</td>
<td>173.3</td>
</tr>
</tbody>
</table>

We transform the polygon in Figure 3.8(a) to that of 3.8(b) by plotting the points on an $(r, \theta)$ rectangular graph. In $O(n)$ time we label the polygon and solve the hidden line removal problem for the parallel model from $r = \infty$ using Algorithm 3.1. The result is illustrated in Figure 3.8(c). Finally, the transformed polygon is returned to its original state, and the solution to the perspective problem is obtained, as shown in Figure 3.8(d).

Lee has developed a linear-time algorithm which solves each of the three types of perspective visibility problem [Lee83]. His algorithm uses a single stack and does not involve the type of transformation illustrated in the above example. Independently, Sack has also created a linear time algorithm for perspective visibility in rectilinear geometry [Sac83]. It uses a single stack and is made clearer by using the labelling scheme. This algorithm has been modified and applied to the general case; pictures of an implementation of this modified algorithm are shown in Figure 3.9. The results may be found in [DeS85].
Figure 3.8 Transforming the Perspective Visibility Problem
Figure 3.9 Pictures Produced by a Perspective Model Visibility Algorithm
3.5 Conclusions

In this chapter we have defined the labelling scheme for simple polygons: \( l_i \) equals the clockwise angle formed by \( e_i \) and the positive horizontal half-line originating at \( p_i \). Then, for \( i = 2, \ldots, n \), \( l_i = l_{i-1} + \theta_i \), where \( \theta_i \) is the turn at \( p_i \), as defined in Chapter 1.

With Property 3.1 we showed that the integer labelling scheme for rectilinear polygons is an instance of this general scheme. These labels can be used to derive a number of properties of a polygon. For instance, Property 3.3 gives a simple test of reflexivity of a vertex, while Property 3.4 provides a very quick test for convexity of a polygon. We also showed how some simple properties of edges can be derived from labels.

We also presented a hidden line removal algorithm which illustrates the way in which labels can be used to create simple, conceptually clear algorithms. In the following chapter we will take advantage of the labelling scheme to provide a clear, efficient algorithm to test polygons for monotonicity. We will also show how to use the labels in preprocessing tests of necessary conditions before using the polygon in a given algorithm.
Monotonicity and the Labelling Scheme

In this chapter, we investigate the notion of monotonicity. We will then use the labelling scheme to show some necessary conditions for monotonicity. A linear-time algorithm to test a polygon for monotonicity [PrS81] will be discussed, and finally, given a labelled polygon, we will give a second algorithm which also tests a polygon for monotonicity in $O(n)$ time but with less work than the first algorithm.

4.1 Introduction

Recall from Chapter 1 the definitions relevant to monotonicity. We say a polygonal chain monotone if it is monotone with respect to any line $\ell$. Equivalently, a polygonal chain is monotone in a given direction if it is monotone with respect to a line $\ell$ has that direction.

Property 4.1: Each sub-chain of a monotone polygonal chain is itself monotone in the same directions.

Proof: Follows immediately from the definition. □

As with polygonal chains, we will use the phrase "monotone in a given direction" synonymously with "monotone with respect to a line which has a given direction". Note that a convex polygon is monotone in all directions. This suggests a relationship between reflex vertices and monotonicity, which will be explored further in §4.3.

4.2 Necessary Conditions for Monotonicity

In this section we see how the labelling scheme might provide monotonicity information with a very simple test. We begin by proving a simple lemma:
Lemma 4.1: A polygonal chain $C_{i,j}$ which turns more than 200 grades (i.e. $|l_i - l_{j-1}| > 2.0$) is not monotone in any direction.

Proof: A polygonal chain which turns more than 400 grades in any direction is clearly non-monotone. Therefore, let $C_{i,j}$ be a monotone polygonal chain which turns more than 200 grades but less than 400 grades, such that $|l_i - l_{j-1}| > 2.0$, $j > i$, and there is no $k$, $i < k < j$, with $|l_i - l_k| > 2.0$ or $|l_k - l_{j-1}| > 2.0$. Assume that $L$ is a line of monotonicity for $C_{i,j}$. In order for the turn of the chain to be more than 200 grades, $C_{i,j}$ must have at least 2 vertices between $p_i$ and $p_j$, since the turn at any vertex is less than 200 grades.

By assumption, the order of the projections of the vertices of $C_{i,j}$ on $L$ will be preserved. Now, a turn of more than 200 grades implies that $e_i$ has one direction with respect to $L$, (say right-pointing), while edge $e_{j-1}$ has the opposite direction, (left-pointing) because 200 grades is a semi-circle shape. Thus, if the projection of $p_{i+1}$ lies to the right of the projection of $p_i$, then the projection of $p_j$ lies to the left of the projection of $p_{j-1}$. See Figure 4.1. This contradicts the assumption that $C_{i,j}$ is monotone, as it does not preserve the order of projections required by monotonicity. Therefore, the assumption was not valid, and the lemma is proved. 

Convex polygons can certainly be partitioned into two chains such that one of them turns more than 200 grades right, since the whole polygon is a 400 grad right turn with no left turns. As we have seen, convex polygons are monotone in all directions, so having this
chain as part of the polygon did not disturb its monotonicity. The same can not be said of chains which turn more than 200 \textit{grads} left, as we prove with the following lemma:

\textbf{Lemma 4.2:} Let $C_{a,b}^1$ be a polygonal chain of a labelled polygon $P$ such that $l_a - l_{a-1} > 2.0$ (i.e. a left turn of more than 200 \textit{grads}). Then $P$ is not monotone in any direction.

\textbf{Proof:} To prove that a polygon is monotone, we must be able to split it into two monotone chains, $C_{a}^1$ and $C_{b}^1$. If either chain contains $C_{a,b}^1$ entirely, then it can't be monotone, as it would contain a non-monotone sub-chain, contradicting Property 4.1. Assume that we partition the polygon into two chains, with at least one partition made in $C^1$. We know by the Theorem of Turning Tangents (Chapter 1) that a traversal of the boundary of the polygon is a right turn of 400 \textit{grads}. Since $C_{a,b}^1$ is a left turn of at least 200 \textit{grads}, then the remaining part of the boundary of $P$, is a right turn of much more than 400 \textit{grads}. If we partition $P$ with at least one partition in $C_{a,b}^1$ then it is clear that at least one of the two chains will contain a sub-chain which turns right more than 200 \textit{grads} regardless of where the second partition is made. By Lemma 4.1 this sub-chain is not monotone, so by Property 4.1 the chain is not monotone, and thus by definition the polygon is not monotone. \[\square\]

The first test we consider is a test of the size of the labels. A label greater than 8 implies that the chain from $p_i$ to the edge with that label implies a "double spiral" shape (i.e. it turns more than 800 \textit{grads} clockwise). This could never be part of a monotone polygon. Similarly, a label smaller than $-4$ also implies a double spiral, which also could not be part of a monotone polygon. Can we improve on these bounds? As we show now, the answer is yes.

\textbf{Lemma 4.3:} For any labelled polygon $P$, if there exists an edge $e_i$ of $P$ with label $l_i \notin [-2.0, 6.0]$, then $P$ is not monotone in any direction.
Proof: Assume \( P \) is a monotone polygon having an edge \( e_m \) with label \( l_m > 6.0 \). By Property 3.2 it is known that \( l_s < 4.0 \). This implies that the polygonal chain from \( e_m \) to \( e_s \) inclusive, \( C_{m,s} \), is a left turn of more than 200 degrees. By Lemma 4.2, this contradicts the assumption that \( P \) is monotone. Similarly, \( l_m < -2.0 \) implies that \( C_{1,m} \) is a left turn of more than 200 degrees, thus proving the lemma.

In this section, we have shown that the labelling scheme can provide us with at least two necessary conditions for monotonicity. By Lemma 4.3, we have the following: If a polygon \( P \) has an edge labelled with a label outside the range \([-2.0, 6.0]\) then \( P \) is not monotone with respect to any line \( L \). Similarly, by Lemma 4.2 we have: If there exists a polygonal chain \( C_{i,j}, j > i, \) of \( P \) such that \( l_i - l_{i-1} > 2.0 \), then \( P \) is not monotone. In the next section we will present two algorithms for testing a polygon for monotonicity, providing us with necessary and sufficient conditions for monotonicity test.

4.3 Testing for Monotonicity

4.3.1 Preparata and Supowit's Algorithm

Preparata and Supowit [PrS81] have written an algorithm to test a polygon for monotonicity, proving also that it runs in linear time. First they define a wedge as follows: let \( \theta \), be the counter-clockwise polar angle at edge \( e \), with respect to a given direction (say \( e_0 \)). Then \( \alpha \), is the counter-clockwise wedge from \( \theta_{i-1} \) to \( \theta_i \) if the external angle at \( p_i \) is \( \geq 180^\circ \) the clockwise wedge from \( \theta_{i-1} \) to \( \theta_i \), otherwise. We will sometimes refer to clockwise wedges as reflex vertex wedges since they are formed when \( p_i \) is a reflex angle. Their algorithm is reproduced here.

**Initial Step:** Set up a doubly linked list with pointer \( \theta_0 \). 0 is chosen conventionally as \( \theta_0 \). There is a single wedge, with counter 0. Insert into the list angle \( \theta_1 \) and set the counter on the wedge determined by \( \alpha_1 \) to 1, and \( \theta \) is set to \( \theta_1 \).
General Step: Let $\theta$ be the current position and assume that the list is such that the angles are in increasing order and the wedge counters form a string of alternating 1's and 2's, with the possible exception of one 0. (These counters are also known as the multiplicity of the wedge). Process $\alpha$. If $\theta$, is larger than $\theta$, scan the list forward, otherwise, scan it backwards. The scan terminates when $\theta$, can be inserted. In this process, each wedge counter different from 2 is incremented and any two consecutive wedges with identical counters are merged (by deleting the list node separating the two wedges). With regard to the updating of $\theta$, suppose that $\theta$, is to be inserted into wedge $[\beta, \beta']$: if the pointer from $\beta$ to $\beta'$ has counter 0 or 1, then a new list node is created and $\theta$ is set to $\theta$; else no new node is created and $\theta$ is set to $\beta'$.

Result: At termination, a partition of the polar range into at most $O(n)$ wedges with alternating counters 1 and 2 is obtained. Scanning the sequence of angles, one can determine pairs of antipodal wedges (wedges which are opposite each other in the polar wedge diagram—a line can be drawn which passes through both wedges and the origin) of multiplicity 1. If any pairs of antipodal wedges of multiplicity 1 exist, a line drawn through them and the origin will be perpendicular to a line of monotonicity.

We give a brief example to illustrate how the algorithm works. Although this algorithm does not require the labelling scheme, we may still use it to derive the wedges. We will take the x-axis as the reference line for the polar angles made by each edge, and produce Table 4.1 from the polygon in Figure 4.2. Note the two clockwise wedges in Table 4.1 which correspond to the two reflex angles in the polygon. After the initial step, we have two wedges: the counter-clockwise wedge from 230 to 106 with counter 1, and the remaining wedge from 106 to 230 with counter 0. The pointer $\theta$ points at 106. After inserting the next wedge, we have: a counter-clockwise wedge from 230 to 175 with counter 1, a clockwise wedge from 106 to 175 with counter 2, and the remaining wedge from 106 to 230.
with counter 0. If we continue to insert wedges until completion, merging adjacent wedges which have identical counters, we have at termination the values in Table 4.2, which are also given as a polar wedge diagram in Figure 4.3.

Examination of either the table or the polar wedge diagram reveals that we can find,
Table 4.2 The Wedges at Termination of the Algorithm

<table>
<thead>
<tr>
<th>Wedge</th>
<th>Orientation</th>
<th>Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>224 to 175</td>
<td>Counter-Clockwise</td>
<td>1</td>
</tr>
<tr>
<td>106 to 175</td>
<td>Clockwise</td>
<td>2</td>
</tr>
<tr>
<td>106 to 284</td>
<td>Counter-Clockwise</td>
<td>1</td>
</tr>
<tr>
<td>224 to 284</td>
<td>Clockwise</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 4.3 The Final Polar Wedge Diagram

In this case by splitting the largest wedge, two pairs of antipodal wedges with multiplicity 1: [(24 to 375), (224 to 175)] and [(106 to 84), (306 to 284)]. Since these pairs exist, we conclude that the polygon is in fact monotone. To find a direction of monotonicity, we draw a line through one of the pairs (and through the origin), and draw a perpendicular to it. Therefore, to find all directions of monotonicity, we add 100 grads to the wedges' limits. In this case, the directions of monotonicity will lie in the ranges [75, 124], [275, 324], [184, 204], [384, 400], and [0, 4]. (These ranges of counter-clockwise angles correspond to the ranges of labels [3.25, 2.76], [1,25, 0.76], [0.16, 0.0] and [4.0, 3.96].)

Comparing the final polar wedge diagram (Figure 4.2 and Table 4.3) with the list of wedges at each edge (Table 4.1) shows, at least in this case, that there is a definite
relationship between wedges of multiplicity 2 and reflex vertex wedges. Before proving the exact relationship we note the following: Recall that Preparata and Supowit direct the edges of a polygon in counter-clockwise order, and that the labelling scheme directs the edges in clockwise order. Adjusting for this difference, we see by comparing the definition of wedge and the definition of turn (Chapter 1) that wedges correspond to turns in the following manner: A clockwise wedge (reflex angle wedge) at \( e \), corresponds to a left turn at \( p_i \) (since reflex angles are left turns) and the size of the wedge is equal to the amount of the turn. Similarly, a counter-clockwise wedge at \( e \), corresponds to a right turn at \( p_i \), with the size of the wedge begin equal to the amount of the turn. We now state the relationship between reflex vertex wedges and multiplicity.

**Lemma 4.4:** In Preparata and Supowit's algorithm, a wedge is of multiplicity 2 if and only if it contains a wedge determined by a reflex vertex (i.e. a clockwise wedge).

**Proof:** \( \Rightarrow \) We show that a wedge is of multiplicity 2 if it contains a reflex vertex (i.e. clockwise) wedge. Consider the polar wedge diagram in Figure 4.3 produced at termination of Preparata and Supowit's algorithm. Assume that the diagram contains a wedge with multiplicity 2 such that this wedge contains no portion of a clockwise wedge. Then it must be the intersection of at least two counter-clockwise wedges. Let us examine the simplest case, that it is the intersection of only two wedges. After insertion of the first of these counter-clockwise wedges (say, w.l.o.g. from \( \theta_{i-1} \) to \( \theta_i \)) into the polar wedge diagram, wedges are inserted until pointer \( \theta \) passes \( \theta_i \) again. This corresponds to a right turn of more than 400 grads (since a left turn of more than 400 grads would have produced a clockwise wedge which would have intersected with the first wedge). Therefore, the second wedge in the intersection will cover, say w.l.o.g., \( \theta_{j-1} \) to \( \theta_j \), where \( \theta_{j-1} \approx \theta_{i-1} + 400 \). However, by the Theorem of Turning Tangents, a right turn of more than 400 grads must be followed in the in-order traversal by at least one left turn, so that the sum of all turns is a right turn of 400 grads. Since left turns correspond to clockwise wedges, the counter-clockwise wedge \( \theta_{j-1} \) to \( \theta_j \) will be entirely contained in an intersection with one or more clockwise
wedges. This violates the assumption that the wedge under consideration did not intersect any clockwise wedges, thus proving this part of the theorem by contradiction.

"\( \leq \)" We show that each reflex vertex (i.e. clockwise) wedge is contained in a wedge of multiplicity 2. By definition, wedges are oriented either clockwise or counter-clockwise. As we have shown above, we can associate a turn with a wedge: Clockwise wedges are left turns (and the amount of turn is negative), and counter-clockwise wedges are right turns (where the turn is positive). By the Theorem of Turning Tangents the sum of the (now) positive and negative turns must be 400 grada. Therefore, any clockwise wedge must be either preceded or followed (or both) by counter-clockwise wedges so that the sum of all wedges is always 400. Therefore, a clockwise wedge must intersect with one or more counter-clockwise wedges (to "cancel it out"). Thus, clockwise wedges will always be contained in wedges of multiplicity 2. \( \square \)

With the proof of this theorem, we have shown what was suggested in §4.1, that there is a direct relationship between reflex vertices and monotonicity. We know from this that in Preparata and Supowit's algorithm we need to consider only the reflex vertex wedges to find all wedges of multiplicity 2 and thus to determine all directions of monotonicity. In the next section, we will use this result to build a slightly more efficient algorithm for testing a polygon for monotonicity.

4.3.2 An Enhanced O(n) Algorithm

In the previous section we showed that it is possible to determine whether or not a polygon is monotone in any direction, and to find those directions, by considering only the reflex vertices. What follows is a discussion leading to the development of an enhanced linear-time algorithm to do just that, by using Lemma 4.4 to avoid processing non-reflex vertices.

Our method will be somewhat similar to Preparata and Supowit's with the difference that, for the wedge construction, only reflex vertices need to be processed. Instead of a
wedge from angle $\theta_{i-1}$ to angle $\theta_i$, we will talk of the interval from label $l_{i-1}$ to label $l_i$. Therefore, instead of the polar wedge diagram stored in a circular doubly linked list that they use, we will use a doubly linked list of intervals. Using Lemma 4.3 we can limit this list of intervals to the range $[-2, 0, 6, 0]$, and thus it need not be circular, unlike a wedge diagram.

We first want to find the reflex vertices, so the preprocessing step will be to label the polygon as per the labelling scheme. The first step of the algorithm is to examine the vertices in clockwise order. Create a list of reflex vertices where each reflex vertex $p_i$ is represented by the interval $[l_{i-1}, l_i]$, the labels of the incoming and outgoing edges, respectively. (i.e. $[s_i, o_i]$) Note that, since the vertices are reflex, $l_{i-1} > l_i$ (Property 3.3). During creation of the list, intervals from consecutive reflex vertices $p_i$ and $p_{i+1}$ can be joined since $s_i = o_{i+1}$, thus creating one set, namely $[i, o_{i+1}]$.

We now describe the general step of the algorithm. An interval created at a reflex vertex in the preprocessing step can be seen to correspond exactly to a wedge in Preparata and Supowit’s algorithm. We will thus invoke the result previously proven, that is: only reflex vertex wedges, (in this case, reflex vertex intervals) need be considered to find all regions (if any) of monotonicity. We start with an empty doubly linked list which will cover the region of labels $[-2, 0, 6, 0]$. (Recall Lemma 4.3.) We then insert each of the intervals $[i, o_i]$ created in the preprocessing step into the doubly linked list, noting that this interval is full. The insertion step is done by scanning the list forwards from the last inserted interval to a position where node $\{i\}$ can be inserted. We know from Property 3.5 that this is always a forwards scan. A backwards scan from node $\{i\}$ is used to determine where node $\{o\}$ will be inserted. Overlapping intervals are merged to become one interval by deleting intervals when crossed during the backwards scan. The result will be a set of alternating empty (0) and full (1) intervals.

At a similar point in the Preparata and Supowit algorithm, we wish to locate antipodal wedges of multiplicity 1. Since we don’t have a wedge diagram, we must somehow align
the intervals created by the general step of the algorithm so that antipodal wedges can be
detected. We do this by breaking the list into the four pieces which cover the intervals
\([-2, 0]\), \([0, 2]\), \([2, 4]\), and \([4, 6]\) and perform a MOD 2 operation on each so that they now
all cover the region \([0, 2]\). These 4 pieces are then merged into 1 piece by merging any full
intervals which intersect. This is a linear time process, since each piece is a sorted list.
This step yields either one full interval from 0 to 2 or a series of alternating empty and
full intervals over the same range. A line having a label in an "empty" interval will clearly
not intersect a reflex vertex interval, and therefore it will run perpendicular to a line of
monotonicity in the same way that lines drawn through antipodal wedges of multiplicity 1
do in the other algorithm. Therefore, the existence of empty intervals in this step proves
that a polygon is monotone. Finally, we add 1 to each label in the interval list, to avoid
having to find perpendiculars. This step provides a set of intervals in the range \([1, 3]\), and
labels in the empty intervals of this range will correspond to directions of monotonicity (as
will such a label \(\pm 2\)). We will show later that this algorithm is \(O(n)\), but is made more
efficient by only having to process reflex vertices.

For a data structure, we will maintain a doubly linked list of intervals over the range
\([-2, 0, 6, 0]\). Pointers will separate adjacent intervals, and will have a counter so that a
pointer with counter 0 points at an empty interval, and a pointer with counter 1 points
at a full interval, which results from the insertion of, or union with, an interval from the
original list. We will also maintain an insertion pointer, say \(I\), to keep track of where
insertions are/will be made in the list.

We now present the algorithm:

Algorithm 4.1: Testing for Monotonicity

Input: A list of reflex vertices represented by \(p_i = [l_{i-1}, l_i]\)

Output: A list of intervals of labels in which directions of lines of monotonicity can be
found.
BEGIN
Initialize List = the empty interval \([-2, 6]\],
\(I = \{-2\}\).

FOR \(j = 1\) TO \(r\) DO (* \(r\) is number of reflex vertex intervals *)
BEGIN
(* process interval \(j = [i, o]\) *)
IF \((i, o) \notin [-2, 6]\) OR
\((o, \notin [-2, 6])\) THEN
(* STOP the polygon is not monotone *)
RETURN (List = the full interval \([\{1\}, \{3\}]\)).
ELSE
BEGIN
(* Insert the interval into the list *)
(* SCAN forward from \(I\) to see where to insert \([i, o]\) *)
WHILE \(I.\text{Forward} < i\) DO \(I = \{I.\text{Forward}\}\);
IF \(i, o\) falls in an empty interval THEN
BEGIN
INSERT \(\{i, o\}\) into doubly linked list after \(I\);
\(\text{Forward Counter} := 0; \text{Backward Counter} := 1\).
END
BEGIN
\(I := \{i\}\);
(* SCAN backwards from \(I\) to see where to insert \([i, o]\) *)
(* Delete nodes we cross during the scan. *)
WHILE \(I.\text{Backward} > o\) DO DELETE \(\{I.\text{Backward}\}\);
IF \(o, i\) falls in an empty interval THEN
BEGIN
INSERT \(\{i, o\}\) into doubly linked list before \(I\);
\(\text{Forward Counter} := 1; \text{Backward Counter} := 0\).
END
BEGIN
\(I := \{I.\text{Backward}\}\);
END
END (* of process step loop *)
END (* of FOR loop *)

(* Merge Step *)

SPLIT List into 4 pieces such that
\(\text{List}[1] = \{[-2], \ldots, \{0\}\}\),
\(\text{List}[2] = \{0\}, \ldots, \{2\}\),
\(\text{List}[3] = \{2\}, \ldots, \{4\}\), and
\(\text{List}[4] = \{4\}, \ldots, \{6\}\);

(* Move them all to \([0, 2]\) *)
FOR \(j := 1, 3, 4\) DO
FOR EACH Node In List\(j\) DO Node := Node mod 2;
(* Merge the 4 pieces *)
List = Union of full intervals \(\text{List}[1], \text{List}[2], \text{List}[3], \text{List}[4]\);

(* Shift the intervals 100 grade *)
FOR EACH Node in List DO
Node := Node + 1; (* Range is now \([1, 3]\) *)
RETURN (List);
END (* of the algorithm *)
The following will get the directions of monotonicity out of the list of intervals

\[\text{FOR EACH Node in List DO}\]
\[\text{IF Node Forward Counter} = 0 \text{ THEN}\]
\[\text{PRINT "Directions of monotonicity: ", Node, " to ", Node Forwards.}\]

**Theorem 4.1:** Algorithm 4.1 detects monotonicity and reports all directions of monotonicity in \(O(r)\) time.

**Proof:** We first show correctness. Given a labelled polygon, reflex vertices are detected by Property 3.3. The proof of the general step follows from Lemma 4.4, which shows that only reflex vertex wedges must be considered, and from the proof of correctness in [PrS81].

The merge step is proven as follows. Let \(l_k\) and \(l_{k+1}\) be the labels which define a reflex vertex wedge. Then the antipodal wedge is defined by labels \(l_k \pm 2\) and \(l_{k+1} \pm 2\), since an increment or decrement of 2 implies a turn of 200 grads. The correctness of the "break and merge" step follows.

We now show complexity.

1. **Preprocessing:** Labelling the polygon is \(O(n)\) by definition.

2. **Initial Step:** Creating the list of reflex vertices is clearly accomplished in linear time, since each vertex is visited exactly once. This yields at most \(r\) intervals, where \(r\) is the number of reflex vertices, and \(r \leq n - 3\).

3. **General Step:** We process at most \(r\) reflex vertex intervals. We set up a single list node in constant time. Once a node has been inserted into the list, it can be scanned. It can be encountered on a backwards scan only once since it is deleted when scanned backwards. We stop whenever we encounter a label less than -2 or greater than 6, by Lemma 4.3. Otherwise, we stay inside the range \([-2, 6]\) without scanning outside this range, so we know that we can only scan a node forwards once. Therefore, each node is scanned at most twice, and therefore this step is \(O(n)\).
Reflex  |  Interval
------|--------------------
Vertex |                    
2     | [0.94, 0.25]       
7     | [3.76, 3.16]       

Table 4.3 The Interval List After Preprocessing

4. Merge Step: Break list into 4 pieces in $O(n)$ time by traversing the list. The 4 pieces contain a total of at most $r + 3$ intervals ($r$ intervals in the list, and the breaking might split as many as 3 intervals). Each piece is sorted. Therefore, merging the 4 lists can be done in $O(n)$ time.

5. Finding ranges of monotonicity: $O(n)$, since it is done by scanning the merged list.

Therefore, we have shown that Algorithm 4.1 runs in $O(n)$ time. 

We illustrate with the polygon of Figure 4.2. The preprocessing step is to find the reflex vertices and create the list of intervals. See Table 4.3.

After inserting each of the intervals, we have the following list:

\[
\{\{-2.0\}, \{(0.25), (0.94)\}, \{(3.16), (3.76)\}, \{6.0\}\}
\]

where \{ \} represents a node in the list, and two nodes inside parenthesis, ({}), represent a full interval.

Splitting into four lists, we have:

1. \[\{\{-2.0\}, \{0.0\}\}\] (empty)
2. \[\{(0.0), \{(0.25), (0.94)\}\}, \{2.0\}\]
3. \[\{(2.0), \{(3.16), (3.76)\}\}, \{4.0\}\]
4. \[\{(4.0), (6.0)\}\] (empty)
Taking each node in each list MOD 2, merging the four lists, and adding 1, we have

\[
\{1.0\}, \quad \{1.25\} \cup \{1.94\}, \quad \{2.16\}, \{2.76\}, \quad \{3.0\}\]

Therefore, the polygon is monotone with respect to any line which has a label in any of the ranges \([1.0, 1.25] ,\ [1.94, 2.16] ,\ \text{and} \ [2.76, 3.0]\). These ranges, and those in the opposite directions, can be seen to encompass exactly the same directions as the ranges found in the first example.

Figures 4.4 and 4.5 illustrate some sample runs of an implementation of Algorithm 4.1 on a graphics workstation.
Figure 4.5 Pictures Produced by an Implementation of Algorithm 4.1.
4.4 Applying Monotonicity

We now examine monotonicity to see how it is used in problems of computational geometry. It is clear from the definitions that a polygon monotone in some direction can be rotated until it is monotone with respect to either the $x$- or $y$-axis. In other words, after the rotation, the vertices are in order by one of the coordinates. In a sense, they are pre-sorted (in $O(n)$ time, since rotation is $O(n)$). As we have shown in §4.3, one can test a polygon for monotonicity in $O(n)$ time. Since sorting is in the worst case an $O(n \log n)$ process, we can use a monotonicity test to create an algorithm which will ensure that the vertices of a polygon are in order by coordinate in $O(n)$ time whenever the polygon is monotone.

Some algorithms (e.g. regularization) require that the order of the vertices with respect to some coordinate axis be known. Generally, this means that a sorting step will be used, which means that the complexity of the algorithm is at least $O(n \log n)$. Therefore, when sorting is required, we can make use of the knowledge that a given polygon is monotone. Testing a polygon for monotonicity and rotating so that its vertices are in order with respect to either coordinate axis can be done in $O(n)$ time, reducing the complexity of the sorting step. Therefore, any algorithm which has a sorting step as the only contributor to an $O(n \log n)$ total complexity (i.e. it would otherwise run in $O(n)$ time) can be made to run in $O(n)$ time for a monotone polygon.

A second way of making use of monotonicity is the so-called slab method. That is, a monotone polygon is rotated until it is monotone with respect to either axis, say the $z$-axis. Then a vertical line is drawn through each vertex. This partitions the plane into $O(n)$ "slabs" of infinite length. See Figure 4.6. Note that each slab intersects each of $C_{1,x}$ and $C_{1,y}$ over exactly one edge, except the first slab and the last slab, which are empty.

One simple use of these slabs is the point location problem: given a monotone polygon $P$ and a point $u$, is $u$ in the interior or the exterior of $P$? Having set up the slabs (in $O(n)$
Figure 4.6 A Monotone Polygon with Slabs

time since $P$ is monotone) a binary search is performed to determine in which slab the point is located. When that is known, the point is compared with the two edges that intersect the slab, and the problem is solved. Thus, given $O(n)$ preprocessing and $O(n)$ storage space, the point location problem can be answered in $O(\log n)$ query time for each point. In [LeF76] we are given a similar algorithm which runs with $O(n \log n)$ preprocessing, $O(n)$ storage, and $O((\log n)^2)$ query time, while [Pre78] describes an algorithm with $O(n \log n)$ preprocessing, $O(n \log n)$ storage and $O(\log n)$ query time.

Slabs can also be used to find the intersection and/or union of two polygons monotone
in the same direction. The slabs are set up and it is noted that each slab intersects each polygon in at most two edges [Sha78]. Since convex polygons are monotone in all directions, slabs can be set up without rotating first, and the union and intersection of two convex polygons can always be found in \( O(n) \) time by using slabs.

4.5 Conclusions

In this chapter we have presented a study of monotonicity, and we have shown how to use the labelling scheme defined in Chapter 3. The main result of this chapter is an enhanced \( O(n) \) algorithm to detect monotonicity which runs more efficiently than its predecessor by virtue of the fact that it tests only reflex vertices, based on properties derived from the labelling scheme.

In the following chapter, we examine the class of star-shaped polygons, which, as we shall show, is related to the class of monotone polygons.
Star-Shapedness

In the examination of monotone polygons presented in Chapter 4 we showed that one property of a monotone polygon is that its boundary can be partitioned at two vertices, separating the boundary into two polygonal chains. Each of these chains has the property that its vertices are in order with respect to the lines having a given slope. In this brief chapter we discuss the related class of star-shaped polygons, in which the vertices of a polygon are in angular order with respect to a set of designated points inside the polygon. We show that the labelling scheme can be used to give a necessary condition for star-shapedness, we show how to use star-shapedness to develop efficient algorithms, and we describe the related class of edge-visible polygons.

5.1 Introduction

In Chapter 3 we described the problem of visibility from a point located in the exterior of a given polygon. This chapter and the next will deal with a different aspect of visibility. In this section we describe the class of star-shaped polygons. We begin by recalling several definitions from Chapter 1.

Let $u$ and $v$ be two points in the interior of a simple polygon $P$. We say $u$ is visible from $v$ if the open line segment $uv$ lies entirely in the interior of $P$. Alternatively, we say $u$ and $v$ see each other.

A polygon $P = (p_1, \ldots, p_n)$ is star-shaped if there exists at least one point $u$ in the interior of $P$ such that $u$ sees every point $v$ in $P$. If $P$ is star-shaped, the set of points which can see every other point in $P$ is called the kernel of $P$, denoted $K(P)$.

Following the convention that the edges of the polygon are directed clockwise, the kernel of a star-shaped polygon can be constructed by intersecting the half-planes lying
Figure 5.1 Examples of Star-Shaped Polygons

The shaded areas designate the kernels.

to the right of the directed lines which contain each edge. This is the intersection of $n$ half-planes determined by the edges of polygon, and can be computed by the $O(n \log n)$
process described in [Shi76].

Lee and Preparata [LeP79] have developed an algorithm to compute the kernel of a polygon in $O(n)$ time. It takes advantage of the knowledge that the $n$ half-planes are ordered by virtue of the fact that they are determined by consecutive edges. The algorithm starts at any vertex $p$, and uses the half-planes defined by the directed lines containing $e_{i-1}$ and $e_i$ to define an unbounded cone-shaped initial region which must contain the kernel of the polygon. Subsequently, a traversal of the polygon is performed, in which each edge is associated with a half-plane. If the line defining the half-plane intersects the current kernel region, then part of the kernel region is cut away. Otherwise, either the current kernel region lies entirely in the half-plane and is not changed, or it does not, in which case the algorithm stops and the polygon is not star-shaped. To ensure that the algorithm runs in $O(n)$ time, the polygon records which vertices of the kernel region are support vertices with respect to the vertex at the end of the current edge of the polygon. In this way, it is easy to tell whether the current kernel region will intersect the next half-plane. A complete description of this algorithm can be found in [LeP79]. Like Algorithms 3.1 and 4.1, Lee and Preparata's algorithm has been implemented in the C programming language, and sample runs are pictured in Figures 5.2 and 5.3.
Figure 5.2 Pictures Produced by an Implementation of Algorithm [LeP79].
Figure 5.3 A Picture Produced by an Implementation of Algorithm [LeP79].

A third way of determining the kernel of a star-shaped polygon involves inner cones. Given a reflex vertex \( p_r \), the inner cone of \( p_r \), denoted \( IC(p_r) \), is the intersection of the half-planes lying to the right of the directed lines containing \( e_{r-1} \) and \( e_r \). The outer cone, \( OC(p_r) \), is the intersection of the half-planes lying to the left of the same lines. We use the term inner (outer) cone because it always intersects the interior (exterior) of the polygon. See Figure 5.4. It can be shown that the kernel of a polygon is equal to the intersection of the inner cones of each of the polygon's reflex vertices and the polygon itself.

The kernel of a star-shaped polygon has some properties that should be mentioned here. First, since \( K(P) \) is the intersection of \( n \) convex regions, \( K(P) \) is convex and has at most \( O(n) \) edges. Trivially, the kernel of a convex polygon is equal to the convex polygon itself. Secondly, if \( u \) is in \( K(P) \) and is not a vertex of \( P \), then vertices \( p_1, \ldots, p_n \) are in sorted angular order with respect to \( u \). If \( u \) is a vertex of \( P \), then vertices \( p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n \)
are in sorted angular order with respect to $p_i$. Thus we have an alternate definition of star-shapedness: A polygon $P$ is star-shaped if there exists a point $u$ in the interior of $P$ such that vertices $p_1, \ldots, p_n$ are in sorted angular order with respect to $u$.

As was the case for monotone polygons, the labelling scheme can be used to derive necessary conditions for star-shapedness. Interestingly enough, the same conditions necessary for monotonicity are necessary for star-shapedness.

**Lemma 5.1:** Let $P$ be a polygon with labels $l_1, \ldots, l_n$. If there exist two labels $l_i$ and $l_j$ such that $i < j$ and $l_j \geq l_i + 2$, then $P$ is not star-shaped.

**Proof:** Consider Figure 5.5. W.l.o.g. we have rotated so that $e_i$ is down-pointing. The dotted line represents an edge with label equal to $l_i - 2$. Since $e_i$ is down-pointing and $e_j$ is up-pointing, there exists at least one edge $e_k$ such that $e_k$ is right-pointing. By definition,
the kernel must lie in the half-plane to the right of $e_k$. However, $l$, and $l_i$, imply that the intersection of the half-planes associated with $e$ and $e_j$ must lie to the left of $e_k$. Thus the kernel, which is formed partly by intersecting these three half-planes, is empty and $P$ is not star-shaped.

**Property 5.1**: Let $P$ be a polygon with labels $l_1, \ldots, l_n$. Then $P$ is star-shaped only if $-2 < l_i < 6$ for $i = 1, \ldots, n$.

**Proof**: By Property 2.7, it is known that $0 < l_i < 2$. Therefore, $l_i \leq -2$, $i > 1$, implies that $P$ is not star-shaped by Lemma 5.1. Similarly, it is known that $2 < l_i \leq 4$ and therefore any $l_j > 6$, $j < n$, implies that $P$ is not star-shaped, also by Lemma 5.1.

Recall from Chapter 4 (Theorem 4.1) that the labels of a monotone polygon lie in the range $[-2.0, 6.0]$. With Property 5.1 we have shown that the same restriction on labels is necessary for star-shaped polygons.

### 5.2 Using Star-Shaped Polygons

In Chapter 4 we described the so-called "slab" method which can be applied to monotone polygons for such problems as point inclusion and intersection of monotone polygons. Star-shaped polygons can be similarly exploited using a "wedge" method, as we now describe.

We let $P = (p_1, \ldots, p_n)$ be a star-shaped polygon, with kernel $K(P)$, and we choose a point $u \in K(P)$. Then a half-line $L_u$ originating at $u$ is drawn through each vertex $p_i$ of $P$. This partitions the plane into $n$ wedge-shaped regions, each with the property that it intersects one edge of $P$, since the vertices of $P$ lie in angular order with respect to $u$. See Figure 5.6.

Recall the point inclusion problem: given a polygon $P$ and a point $v$, is $v$ in the exterior or the interior of $P$? The wedges set up by the above procedure can be used
to solve the point inclusion problem given that $P$ is star-shaped [Sha78]. As in the slab method for monotone polygons, the preprocessing is $O(n)$; testing for star-shapedness and locating $K(P)$ is $O(n)$ using Lee and Preparata's algorithm; creating wedges is also $O(n)$ since the vertices are ordered about any point in the kernel and each vertex is visited once. Considering the linear relationship between vertices and wedges, it is clear that the data structure used requires $O(n)$ storage space. Finally, by using a binary search method ($O(\log n)$) to find the wedge containing a given test point, and testing the point with the edge crossing the wedge in constant time, it is clear that this results in the same $O(\log n)$ query time per test point attained by the slab method for monotone polygons.

We may also apply the wedge method to the problems of finding the union and finding
the intersection of two star-shaped polygons. Shamos has shown that two star-shaped polygons may have $O(n^2)$ intersections, and in any case, just detecting intersection of two polygons may take $O(n \log n)$ time. However, extra information is gained when it is known that the kernels of two star-shaped polygons intersect, and we show this below.

Let $P = (p_1, \ldots, p_n)$, with edges $(e_1, \ldots, e_n)$, and $Q = (q_1, \ldots, q_m)$, with edges $(f_1, \ldots, f_m)$, be two star-shaped polygons. Kernels $K(P)$ and $K(Q)$ can be computed in $O(n)$ and $O(m)$ time, respectively. Since each kernel is convex and has at most $O(n)$ and $O(m)$ edges, respectively, a slab method can be used to compute their intersection, if any, in $O(n + m)$ time. Assuming that $K(P) \cap K(Q) \neq \emptyset$, we choose a point $w$ located in the intersection of the two kernels.

Since each polygon is star-shaped, the lists of vertices $(p_1, \ldots, p_n)$ and $(q_1, \ldots, q_m)$ are each in sorted angular order with respect to $w$. We therefore merge these lists in linear time, and insert the half-lines which define the wedges with respect to $w$. In a single star-shaped polygon, each wedge crosses the boundary of the polygon once. Therefore, for these two star-shaped polygons, each wedge will cross some edge $e_i$ of $P$ and some edge $f_j$ of $Q$.

Let $pq$ and $rs$ be two non-intersecting line-segments in a wedge subtended by $w$ such that any half-line originating at $w$ and lying in the wedge intersects both line segments $pq$ and $rs$, or intersects neither line segment. Then $pq$ is closer to $w$ than $rs$ if, for each point $u$ of $pq$, the line segment $uw$ does not intersect $rs$; further otherwise.

Within each wedge, we have one of three cases:

1. $e_i$ and $f_j$ do not intersect and $e_i$ is closer to $w$;
2. $e_i$ and $f_j$ do not intersect and $f_j$ is closer to $w$; and
3. $e_i$ and $f_j$ intersect.
Lemma 5.2: The union and intersection of two star-shaped polygons with intersecting kernels can be computed in \( O(n) \) time.

Proof: We prove for intersection; union is similar. Given that \( P \) and \( Q \) are star-shaped polygons with intersecting kernels, we choose a point \( w \) in the intersection of the kernels and determine the wedges in linear-time, as above. Refer to Figure 5.7. To find the intersection of \( P \) and \( Q \), we perform a traversal of the wedges. At each wedge we test the edges crossing the wedge and we add the edge closest to \( w \) to the intersection. In the case where \( e_i \) and \( f_i \) intersect, we partition the edges at the intersection and add the parts of each closest to \( w \) before and after the intersection. In each wedge, the region defined by the closer edge(s), the half-lines, and \( w \) belongs to the intersection since it is part of both \( P \) and \( Q \). Let \( v \) be a point in the wedge such that the line segment joining \( v \) and \( w \) intersects the closest edge in the wedge to \( w \). Then either \( v \) is in \( P \)'s exterior, or in \( Q \)'s exterior, or both. Thus the set of "closer" edges in each wedge properly defines the intersection of the two polygons. Since each wedge is examined once, and there are exactly two edges in each wedge, this is a linear-time \( O(n + m) \) process. 

Similarly, we can find the union of two star-shaped polygons whose kernels intersect. Using the same steps as outlined above, the union is found by taking the edges farthest from \( w \) in each wedge. Note that, by the ordering of the vertices of both polygons with respect to \( w \), both the intersection and union of star-shaped polygons with intersecting kernels are also star-shaped.

The structure of star-shaped polygons has also been used in the triangulation problem. In [ScL80], two algorithms are presented which triangulate a star-shaped polygon in \( O(n) \) time. Furthermore, the hidden line problem is easier to solve given that a polygon is star-shaped; this is shown in [RaT83].
5.3 Edge-Visibility

In this section we discuss the class of edge-visible polygons, which are closely related to star-shaped polygons. The term edge-visible describes a polygon which can be seen from at least one of its edges. This visibility has three types:
(a) is completely edge-visible, (b) is strongly edge-visible, and (c) is weakly edge-visible.

1. A polygon $P$ is **completely edge-visible** if there exists an edge $e_i$ of $P$ such that each point $u$ of $e_i$ sees every point $v$ of $P$.

2. A polygon $P$ is **strongly edge-visible** if there exists an edge $e_i$ of $P$ such that there exists at least one point $u$ of $e_i$ which sees every point $v$ of $P$.

3. A polygon $P$ is **weakly edge-visible** if there exists an edge $e_i$ of $P$ such that each point $v$ of $P$ sees some point $u$ of $e_i$.

See Figure 5.8.

We note that, by definition, all completely edge-visible polygons are strongly edge-visible, and all strongly edge-visible polygons are weakly edge-visible. Furthermore, all
complete and strong edge-visible polygons are star-shaped. We can use this fact to test a polygon for complete and strong edge-visibility in linear time by modifying Lee and Preparata’s kernel algorithm to record which edges of the polygon are also part of the kernel. However, there is no known algorithm which tests a polygon for weak edge-visibility in linear, or even $O(n \log n)$ time.

Avis and Toussaint [AvT81] present an $O(n)$ algorithm to test the type of visibility from a given edge of a given polygon. Applying their algorithm to each edge of the polygon results in an $O(n^2)$ test for weak edge-visibility. Finding a better algorithm is desirable as it is known that triangulating a weakly edge-visible problem can be accomplished in $O(n)$ time [ToA81].

5.4 Conclusions

In this chapter we described the class of star-shaped polygons, and the related class of edge-visible polygons. We also gave examples of how to exploit star-shapedness by using the so-called “wedge” method. With the wedge method, we showed how to determine, in linear time, the intersection and union of two star-shaped polygons whose kernels are known to intersect. We note that one reason star-shapedness is useful is that star-shaped polygons can be triangulated in linear time; and the point inclusion and hidden line removal problems are easier for star-shaped polygons than for the general case. Furthermore, edge-visible polygons can also be triangulated in $O(n)$ time.

In the next chapter we describe the class of pseudo-star-shaped polygons, which have many similarities with both monotone and star-shaped polygons.
Pseudo-Star-Shapedness

In this chapter we introduce the class of pseudo-star-shaped polygons. Monotonicity and pseudo-star-shapedness are related in that the boundary of each type of polygon can be partitioned into two chains such that monotonicity implies a linear ordering of the vertices on each chain, and pseudo-star-shapedness implies an angular ordering of the vertices on each chain with respect to a point outside the polygon. It is the purpose of this chapter to explore this class of polygons, to list some of their properties, to present an algorithm which tests a given polygon for pseudo-star-shapedness, and to describe some applications of this class of polygon.

6.1 Introduction

In previous chapters of this thesis we have looked at several aspects of the visibility problem. We define another type of visibility here. Given a polygon $P$ and two points, $u$ in the exterior of $P$ and $v$ in the interior, we say $v$ sees $u$ through an edge $e_i$ of $P$ if the line segment $uv$ intersects $e_i$ and intersects no other edge of $P$. Similarly, we say $u$ sees $v$ through polygonal chain $C_{i,j}$ if $uv$ intersects exactly one edge of $C_{i,j}$.

Keil [Kei83] introduced pseudo-star-shaped polygons in an algorithm to decompose a polygon into star-shaped pieces. We give his definition here and follow with a generalized version. Let $Q = (q_1, \ldots, q_n)$ be a simple polygon. Partition $Q$ with line segment $q_iq_j$ so that $P = (q_1, \ldots, q_j)$ is one of the two resulting polygons, and each vertex $q_{j+1}, \ldots, q_{i-1}$ lies to the right of the directed line containing $q_iq_j$. The edge $q_iq_j$ is called the base of polygon $P$, and $Q \supset P$. Then $P$ is pseudo-star-shaped if there exists a point $u$ in $Q$ but not in $P$ such that $u$ can see every point of $P$ through $q_iq_j$. See Figure 6.1.

Note that there are 3 restrictions on this definition:
1. \( P \) must be contained in some polygon \( Q \);
2. \( u \) must be in \( Q \); and
3. \( P \) must lie entirely to the right of some edge \( q,q_j \).

We are interested in finding a more general definition of pseudo-star-shapedness that would include that of Keil. Recall that a polygonal chain \( C_{i,j} \) is completely visible from a point \( u \) on the plane if \( u \) can see each point \( v \) on \( C_{i,j} \). Then we have the following lemma:

**Lemma 6.1:** Given a simple polygon \( P \), a polygonal chain \( C_{i,j} \) of \( P \) is completely visible from a point \( u \) in the exterior of \( P \) if and only if \( u \) is located in the half-plane lying to the left of each edge of \( C_{i,j} \).

**Proof:** Let \( u \) be a point in the exterior of \( P \) such that \( u \) sees \( p_i \) and \( p_j \). Assume \( u \) lies to the right of some edge \( e_k \) of \( C_{i,j} \). Then for \( v \) on \( e_k \), the line segment \( uv \) intersects the interior of \( P \) and \( v \) is not visible from \( u \), since \( u \) is in the exterior of \( P \). Now let \( u \) lie to the left of each edge of \( C_{i,j} \), and assume that there exists a point \( v \) on an edge \( e_k \) of \( C_{i,j} \) such
that \( u \) does not see \( v \). Then \( uv \) intersects \( P \) at some other point of \( P \) than \( v \). This implies that \( v \) lies to the right of some edge of \( C_{i,j} \) (since \( v \) sees \( p_i \) and \( p_j \)). This contradicts the assumption and proves the lemma.

We now let \( P \) be a simple polygon and let \( u \) be a point in the exterior of \( P \). \( P \) is *pseudo-star-shaped* with respect to \( u \) if the boundary of \( P \) can be partitioned into two polygonal chains, \( C_{i,j} \) and \( C_{j,i} \), such that \( C_{i,j} \) is completely visible from \( u \) and \( C_{j,i} \) is completely visible from \( u \) through \( C_{i,j} \). See Figure 6.2. Vertices \( p_i \) and \( p_j \) are called *support vertices* with respect to \( u \).

Alternatively, we compute angle \( \alpha_k \) as the angle which is formed at \( u \) by the segments from \( u \) to \( p_i \) and from \( u \) to \( p_k \). Then \( P \) is pseudo-star-shaped if, for \( C_{i,j} \), \( \alpha_k \) increases as a traversal is made from \( p_i \) to \( p_j \), and for \( C_{j,i} \), \( \alpha_k \) increases as a traversal is made from \( p_j \) to \( p_i \). See Figure 6.3. Note that the support vertices \( p_i \) and \( p_j \) are the extreme vertices with respect to the angles made with \( u \).

In this way we can see how monotonicity and pseudo-star-shapedness are related; monotonicity implies a linear order of vertices on the two chains, whereas pseudo-star-
shapedness implies an angular order. See Figure 6.4. We can liken this to the difference between the parallel and perspective models of visibility that we described in Chapter 3, because the visibility chain in the parallel model is monotone, and the visibility chain in the perspective model is star-shaped.
Finally, we note that we can always create a pseudo-star-shaped polygon from a set of $n$ points in $O(n \log n)$ time as follows.

Algorithm 6.1: Creating a Pseudo-Star-Shaped Polygon

Input: A set of $n$ points on the plane.

Output: A pseudo-star-shaped polygon.

BEGIN

1. Find the convex hull of the points;

2. Choose a point $u$ in the exterior of the convex hull created in step 1;

3. Sort the original $n$ points with respect to the angles made by the line segments joining each point to $u$. Store the resulting list as $v_1, \ldots, v_n$ in order by angle;

4. Create two empty chains, Top Chain, called $TC_{1,n}$ and Bottom Chain, called $BC_{n,1}$;

5. Add $v_1$ to end of $TC_{1,n}$;
   Add $v_1$ to beginning of $BC_{n,1}$;

6. FOR $i = 2$ TO $n - 1$ DO
   IF $uv_i$ intersects $v_1v_n$ THEN
     add $v_i$ to end of $TC_{1,n}$;
   ELSE
     add $v_i$ to beginning of $BC_{n,1}$;

7. Add $v_n$ to end of $TC_{1,n}$;
   Add $v_n$ to beginning $BC_{n,1}$;

8. Connect $TC$ and $BC$ at common vertices $v_1$ and $v_n$;

END

Lemma 6.2: Algorithm 6.1 correctly creates a pseudo-star-shaped polygon from a set of $n$ points in $O(n \log n)$ time.

Proof: We first show correctness. Since $u$ is outside the convex hull of the points, it is clear that $u$ is in the exterior of the resulting polygon. The two chains $TC$ and $BC$ intersect only at $v_1$ and $v_n$ since the other vertices of $TC$ lie on the left of $v_1v_n$ and the other vertices of $BC$ lie on the right of $v_1v_n$. Furthermore, both chains are non-intersecting
since the vertices are in angular order with respect to $u$. Since the vertices of $BC_{a_1}$ are in
sorted angular order with respect to $u$, $BC$ is completely visible from $u$. Likewise, $TC_{1,n}$ is
completely visible from $u$ through $BC$. Thus, the resulting polygon is pseudo-star-shaped
with respect to $u$.

The complexity of algorithm is derived as follows: Finding the convex hull of $n$ points
is $O(n \log n)$ [Sha78]. Choosing a point outside the convex hull is done in $O(n)$ time.
Calculating the angles for each point is a linear-time step, since each point is visited once.
The sorting step is $O(n \log n)$. Creating the two chains and merging is also $O(n)$ since
each vertex is visited exactly once and constant work is performed at each visit. Thus, the
complexity of the entire algorithm is $O(n \log n)$.

6.2 Outer-Kernels

Recall that for star-shaped polygons, the set of points in the interior of the polygon
which can see the entire polygon is called the kernel. Similarly, we introduce the term outer-
kernel, denoted $OK(P)$ to describe a set of points in the exterior of a polygon from which
the polygon is pseudo-star-shaped. In this section we discuss the form of outer-kernels and
some of their properties.

Let $P$ be a simple polygon and let $u$ be a point located in the exterior of $P$. Assume
$p_1$ and $p_i$ are the support vertices of $P$ with respect to $u$. In §6.1 we defined the term
completely visible, and showed in Lemma 6.1 that $C_{i,j}$ is completely visible if and only if
$u$ lies in the half-plane to the left of each edge of $C_{i,j}$. From this we derive the following:

**Lemma 6.3:** Given a simple polygon $P$ with $C_{i,j}$ completely visible from $u$, $C_{j,i}$ is com-
pletely visible from $u$ through $C_{i,j}$ if and only if $u$ is located in the half-plane lying to the
right of each edge of $C_{j,i}$. 
Proof: Since $C_{i,j}$ is completely visible from $u$, we delete $C_{i,j}$ from the polygon. This leaves $C_{j,i}$ (see Figure 6.5(a) and Figure 6.5(b)) and the proof follows along the lines of that for Lemma 6.1.

Assume $P$ is pseudo-star-shaped from a point $u$, with support vertices $p$, and $p_j$. Then we combine Lemmas 6.1 and 6.3 to prove the following:

Property 6.1: The outer-kernel of $P$ which contains $u$ is the intersection of the right half-plane for each edge $e_h$ of $C_{j,i}$ and the left half-plane for each edge $e_m$ of $C_{i,j}$. Furthermore, the outer-kernel is convex.

Proof: The result follows directly from Lemma 6.1 and Lemma 6.3. That $OK(P)$ is convex follows since it is the intersection of convex regions, i.e. half-planes.
Figure 6.6 A Polygon with more than one Outer-Kernel

Given a polygon and a point $u$ from which it is pseudo-star-shaped, one can use Property 6.1 to find the outer-kernel in which $u$ is contained. Using $u$, the support vertices $p_1$ and $p_4$ can be found. Then the outer-kernel is the intersection of the appropriate half-planes, determined by the edges of $C_{1,4}$ and $C_{3,1}$. Of course, for any arbitrary polygon there could be many such $C_{i,j}/C_{j,i}$ partitionings, depending on the location of $u$ with respect to $P$. This suggests that a pseudo-star-shaped polygon may have more than one outer-kernel. This is, in fact, true and we give a simple example in Figure 6.6.

Figure 6.6 introduces an interesting fact about outer-kernels. In star-shaped polygons, the kernel is bounded, obviously, since the kernel must be contained in interior of the polygon. This is not true for outer-kernels, and we have the following:

Property 6.3: Outer-kernels may be either bounded or unbounded.

Proof: We give the example of Figure 6.6, in which $\mathcal{OK}_u(P)$, with support vertices $p_2$ and $p_1$ and chains $C_{4,1}$ and $C_{5,1}$, is bounded, and $\mathcal{OK}_u(P)$, with support vertices $p_1$ and
p, and chains C₁,₁ and C₂,₁, is unbounded.

At this point we make note of the special case of convex polygons. Since a line from any point u in the exterior of a convex polygon intersects the polygon in at most two places, a convex polygon can always be shown to be pseudo-star-shaped from C. Thus, a convex polygon P is pseudo-star-shaped from every point of its exterior, and therefore the union of its outer-kernels is equal to the exterior.

Having seen that a pseudo-star-shaped polygon can have more than one outer-kernel, it is desirable to find out how many outer-kernels it can have. Then we will have an idea of the complexity of an algorithm which finds and reports all outer-kernels for a given polygon. We now show an upper bound on the number of outer-kernels for an arbitrary polygon.

**Lemma 6.4:** For any arbitrary simple n-vertex polygon P, the number of outer-kernels of P is bounded above by n².

**Proof:** We first show that if P is pseudo-star-shaped from points u and v, such that p, and pᵢ are the support vertices of P for both u and v, then u and v are contained in the same outer-kernel. From Lemma 6.1 and Lemma 6.3, it is clear that u and v lie in the right half-planes of the edges of Cᵢ, and the left half-planes of the edges of Cᵢ,j. Therefore, u and v lie in the intersection of these half-planes. Thus they are in the same outer-kernel.

This implies a direct relationship between support vertices and outer-kernels. Denoting pᵢ and pᵢ as the **right-support vertex** and **left-support vertex**, respectively, it is clear that for each pair of right- and left-support vertices there is at most one outer-kernel.

Finally, we set a bound on the number of pairs of support vertices. Let P be a simple polygon within vertices, r of which are reflex. Clearly, support vertices must be non-reflex, and P has (n−r) non-reflex vertices. Thus there will be (n−r)(n−r−1) pairs of non-reflex vertices, and therefore there is at most the same number of possible right- and left-support
pairs. From the above argument, there is at most one outer-kernel for each pair, and thus the number of outer-kernels is bounded by \((n-r)(n-r-1) \leq r^3\), which is of course \(O(n^3)\).

Since not every pair of convex vertices is actually a pair of support vertices for any point in the exterior of \(P\), this upper bound may possibly be refined. Furthermore, we can construct a polygon where \(p_a\) and \(p_b\) are right- and left-support vertices respectively for some point \(u\) in the exterior, and also show that there are no points \(v\) in the exterior for which \(p_a\) and \(p_b\) are left- and right-extremes, as in Figure 6.7. Such a situation eliminates one from the number of possible outer-kernels. Thus, the problem of giving an exact bound on the number of outer-kernels is an open problem resulting from this thesis (see Chapter 7).

At this point it is convenient to examine the implications of an unbounded outer-kernel. We first recall the alternate definition of pseudo-star-shapedness: Given a polygon \(P\), a point \(u\) in the exterior of \(P\), and the support vertices \(p_i\) and \(p_j\) of \(P\) with respect to \(u\), \(P\) is pseudo-star-shaped if the angles \(\alpha_{ij}\) formed by the segment joining \(p_i\) and \(u\), and
the segment joining $u$ and $p_s$, increase as the polygon is traversed from $p_i$ to $p_j$, clockwise along $C_{i,j}$ and counter-clockwise along $C_{j,i}$.

Given a pseudo-star-shaped polygon $P$ with an unbounded outer-kernel, we can choose a point $u$ in the interior of the unbounded outer-kernel such that the distance from $u$ to $P$ approaches infinity, and $P$ is pseudo-star-shaped from $u$. Thus, we say that a polygon $P$ is pseudo-star-shaped from infinity if $P$ has an unbounded outer-kernel. Then we have:

**Theorem 6.1:** $P$ is monotone in some direction if and only if $P$ is pseudo-star-shaped with at least one unbounded outer-kernel.

**Proof:** "$\Rightarrow$" We show that unbounded outer-kernels imply monotonicity. Let $P$ be a pseudo-star-shaped polygon with an unbounded outer-kernel and let $u$ be a point in this outer-kernel. As $u$ approaches infinity, the line segments joining the vertices of $P$ to $u$ become parallel. That is, the angular ordering becomes a linear ordering. Therefore, for any line perpendicular to these segments, the projections of the vertices on this line are in the order of their appearance on the chains $C_{i,j}$ and $C_{j,i}$, and $P$ is monotone by the definition of monotonicity in Chapter 4.

"$\Rightarrow$" We show that monotone polygons are pseudo-star-shaped with an unbounded outer-kernel. Let $P$ be a simple polygon, monotone with respect to a line $L$. Then there exist two polygonal chains $C_{i,j}$ and $C_{j,i}$ such that, for each chain, the order of the projections of the vertices onto $L$ is the same as the order in the chain. We extend the projection lines from the vertices, through $L$, towards infinity. Let $v$ be a point on one such projection line. As $v$ approaches infinity, the line segments from $v$ to the vertices of $P$ approach the projection lines, and the linear ordering of the vertices becomes an angular ordering with respect to $v$. Thus $P$ is pseudo-star-shaped from some point at infinity, and therefore the outer-kernel is unbounded. $\blacksquare$

**Property 6.3:** Unbounded outer-kernels define directions of monotonicity.
Figure 6.8 Unbounded Outer-Kernels and Monotonicity

Proof: Let $OK_1(P)$ be an unbounded outer-kernel of a polygon $P$. Two edges of $OK_1(P)$, say $e_a$ and $e_b$, extend to $\infty$, defining the unbounded edge of the outer-kernel. They also define the directions that one can draw a ray from any point of $P$ towards $\infty$ so that $P$ is pseudo-star-shaped from that direction. Lines perpendicular to these rays are directions of monotonicity. Therefore the directions of $e_a$ and $e_b$ can be altered by 100 grads to give the range of directions of monotonicity for $P$. See Figure 6.8.

This relationship between unbounded outer-kernels and monotonicity helps prove another property of unbounded outer-kernels; that is, that unbounded outer-kernels come in pairs.

Property 6.4: Let $P$ be a non-convex pseudo-star-shaped polygon with an unbounded outer-kernel. Then there is more than one such unbounded outer-kernel.

Proof: By assumption, $P$ has an unbounded outer-kernel, and thus $P$ is monotone by Theorem 6.1. As described in the proof of Property 6.3, we choose a line $L_1$ of monotonicity such that $L_1$ does not intersect $P$. W.l.o.g. we rotate the plane so that $L_1$ is vertical and
on the right of \( P \). By the construction in Theorem 6.1, there exists a point \( u \) with \( z = +\infty \) such that \( P \) is pseudo-star-shaped with respect to \( u \). We choose a line \( L_2 \), which does not intersect \( P \), parallel to \( L_1 \) on the left of \( P \). Since \( P \) is monotone with respect to \( L_1 \), \( P \) is monotone with respect to \( L_2 \) and there exists some point \( v \) with \( z = -\infty \) such that \( P \) is pseudo-star-shaped from \( v \). Since \( v \) is on the left, and \( u \) is on the right, of \( P \), the left- and right-support vertices of \( P \) with respect to \( u \) and \( v \) are different, and thus \( v \) lies in a different unbounded outer-kernel than \( u \). See Figure 6.9.

Recall Figure 6.6, in which we show a polygon with eight outer-kernels. We note that neighbouring outer-kernels sometimes share an edge. In this case it is tempting to merge adjacent outer-kernels to make one large outer-kernel. For instance, merging the outer-kernels of a convex polygon yields the exterior of the polygon. If we do this, the result may be an outer-kernel which is not convex, as shown in Figure 6.10. The disadvantage to this is that structure of convex polygons allows certain tests to run to be made more efficiently than they might on a non-convex polygon. The advantage, if any, would be in a new definition of outer-kernels with respect to inner and outer cones, which were described in §6.1.
6.3 Testing for Pseudo-Star-Shapedness

Using the alternate, angular, definition of pseudo-star-shapedness, we can easily write an algorithm which tests a given polygon $P$ for pseudo-star-shapedness from a given point $u$ in the exterior of $P$. (Recall Figure 6.2)

Algorithm 6.2: Testing for Pseudo-Star-Shapedness from a Given Point

Input: A simple polygon $P$ and a point $u$ in $P$'s exterior.

Output: "Yes" or "No" to the question, "Is $P$ pseudo-star-shaped with respect to $u"?

BEGIN
  calculate support vertices $p_i$ and $p_j$ of $P$ from $u$;
  FOR $k = i + 1$ TO $j$ IN STEPS OF $+1$ DO
     BEGIN
       $a_k =$ the angle made by $p_i$, $p_k$, and $u$, subtended by $u$;
       IF $a_k < a_{k-1}$ THEN
          BEGIN
            OUTPUT('NO'); STOP;
          END
     END
END
FOR \( k = 1 \) TO \( j \) IN STEPS OF \(-1\) DO
BEGIN
\( \alpha_k \) = the angle made by \( p_i \), \( p_k \), and \( u \), subtended by \( u \).
IF \( \alpha_k < \alpha_{k+1} \) THEN
BEGIN
OUTPUT('NO'), STOP.
END
END

OUTPUT('YES').
END

Lemma 6.5: Algorithm 6.2 correctly tests a polygon for pseudo-star-shapedness with respect to a given point in \( O(n) \) time.

Proof: The proof of correctness follows from the alternate definition of pseudo-star-shapedness. Each chain \( C_{i,j} \) and \( C_{j,i} \) is examined and if at any time \( \alpha_k \) decreases instead of increases, the algorithm halts. Otherwise, under the definition, \( P \) is pseudo-star-shaped with respect to \( u \). Complexity follows since each vertex is visited and compared with the previous vertex exactly once. 

In §6.2 we showed that \( n^2 \) is an upper bound on the number of outer-kernels of a pseudo-star-shaped polygon, and thus, while we can test a polygon to detect pseudo-star-shapedness from a given point in \( O(n) \) time, we cannot find and report all the outer-kernels of a pseudo-star-shaped polygon in less than \( O(n^2) \) time. By employing Algorithm 4.1, which detects monotonicity in \( O(n) \) time, we can detect pseudo-star-shapedness with unbounded outer-kernels also in \( O(n) \) time, under Theorem 6.1. However, this does not calculate the outer-kernels, nor can it detect pseudo-star-shapedness in polygons which have only bounded pseudo-kernels. Therefore, we resort to a brute force method to calculate all the outer-kernels of a pseudo-star-shaped polygon.

Algorithm 6.3: Testing a Polygon for Pseudo-Star-Shapedness

Input: A simple polygon \( P \).
Output: No, if the polygon is not pseudo-star-shaped, Yes otherwise, and the set of $P$'s outer-kernels

BEGIN
  IF $P$ is convex THEN
    BEGIN
      OUTPUT(YES); OUTPUT(exterior of $P$); STOP.
    END
  c := 0;
  FOREACH pair of left- and right-support vertices $(p_i, p_j)$ DO
    BEGIN
      FOR chain $C_{i,j}$, calculate $A$, the intersection of the left
      half-planes of each edge using Lee and Preparata's algorithm;
      FOR chain $C_{j,i}$, calculate $B$, the intersection of the
      right half-planes of each edge using Lee and Preparata's algorithm;
      IF $A \neq \emptyset$ AND $B \neq \emptyset$ THEN
        BEGIN
          $OK_{c+1}(P) := A \cap B$;
          IF $OK_{c+1}(P) \neq \emptyset$ THEN $c := c + 1$;
        END
    END
  IF $c = 0$ THEN
    OUTPUT(NO);
  ELSE
    OUTPUT(YES);
    FOR $i = 1$ TO $c$ DO
      OUTPUT($OK_i(P)$);
  END
END.

Theorem 6.2: Algorithm 6.3 detects pseudo-star-shapedness and reports all outer-kernels in $O(n^2)$ time.

Proof: We first prove correctness. We have shown in Lemma 6.4 that an outer-kernel can be associated with a distinct pair of support vertices. Therefore, we test at each pair of support vertices, $p_i$ and $p_j$. For chain $C_{i,j}$, Lemma 6.1 guarantees that the chain is completely visible from each point of $A$, the intersection of the left half-planes of each edge of the chain. Likewise, Lemma 6.3 ensures that $C_{j,i}$ is completely visible through $C_{i,j}$ from every point of $B$. By intersecting $A$ and $B$, we have a set of points which satisfy Property 6.1, and therefore $P$ is pseudo-star-shaped from this intersection.
The complexity analysis of this algorithm depends on two things: Lee and Preparata's algorithm to calculate the kernel of a star-shaped polygon, and Lemma 6.4. Firstly, testing $P$ for convexity is $O(n)$ since all we need do is test each vertex exactly once for reflexivity.
Then, by Lemma 6.4, it is known that the main loop of the algorithm will be repeated up to \((n - r)(n - r - 1)\) times, i.e. \(O(n^2)\). Within this loop, we use Lee and Preparata's algorithm to find \(A\) and \(B\), the intersections of left and right half-planes, respectively. This step is illustrated in Figures 6.11 and 6.12, in which Lee and Preparata's algorithm is applied to the bold edges from \(p_1\) and \(p_r\). As shown in [LeP79], each step will be \(O(n)\) and results in either a convex, or empty, region. Furthermore, the intersection of the two convex regions can be found in \(O(n)\) using the slab method of [Sha78]. Therefore the main loop of the algorithm is \(O(n^2)\). Reporting the set of outer-kernels is clearly accomplished in \(O(n^2)\) in the worst case, by Lemma 6.4, and therefore the complexity of the entire algorithm is \(O(n^3)\).

If we recall Keil's definition of pseudo-star-shapedness, we can derive several other problems that we can use Algorithm 6.3 to solve. For instance, given a polygon, we can test for "Keil pseudo-star-shapedness" with "base" equal to a given edge in \(O(n)\) time. To detect "Keil pseudo-star-shapedness" from any edge, we only need to compute outer-kernels for support vertices which subtend one edge of the polygon. The number of such pairs is \(O(n)\), where, in general, the total number of support vertex pairs is \(O(n^2)\). This reduces the complexity of the Algorithm 6.3 by one degree, to \(O(n^3)\). Finally, given a polygon \(Q = (q_1, \ldots, q_n)\), we may insert, one at a time, diagonals \(q_iq_j\), and test the resulting polygons (first for simplicity, in linear time) for "Keil pseudo-star-shapedness." Since there are \(O(n^3)\) possible diagonals, and an \(O(n)\) test for pseudo-star-shapedness with the diagonal as the base, the result is an \(O(n^4)\) algorithm to find all the "Keil pseudo-star-shaped" polygons defined by a given polygon.

### 6.4 Applications of Pseudo-Star-Shapedness

In the previous three sections we have introduced a new general class of polygons described by the term pseudo-star-shaped. We have illustrated some of its properties, and
shown how to detect them. In this section we will explain the motivation for this work, relate it to other known classes of polygons, and give some uses of it.

6.4.1 Other Classes of Polygons

Given a simple polygon $P$, the convex hull of $P$, denoted $CH(P)$, is defined as the smallest convex polygon which contains $P$. Having stated this we now give the definition of another class of polygons. A simple polygon $P$ is said to be weakly externally visible if, for each point $u$ on the boundary of $P$ there exists at least one point $v$ on the boundary of $CH(P)$ such that $u$ sees $v$. Alternatively, $P$ is weakly externally visible if for each point $p$ on the boundary of $P$ there exists a halfline originating at $p$ which does not intersect $P$ at any other point [EAT81]. This is analogous to "walking around" the polygon outside the convex hull and seeing every point of the boundary of the polygon at least once. See Figure 6.13.

**Theorem 6.3:** Let $P$ be a pseudo-star-shaped polygon. Then $P$ is weakly externally visible if there exists at least one outer-kernel of $P$ which intersects the boundary or the exterior of $CH(P)$.

**Proof:** Let $P$ be a pseudo-star-shaped polygon and let $u$ be a point that lies in any outer-kernel of $P$ and also in the exterior (or on the boundary) of the convex hull of $P$. Let $p_j$ and $p_i$ be the support vertices of $P$ with respect to $u$. By definition, $C_{i,j}$ is completely visible from $u$, and $C_{j,i}$ is completely visible from $u$ through $C_{i,j}$. Thus a half-line starting at $u$ which intersects $P$ in at most two places, once on $C_{i,j}$ and once on $C_{j,i}$. Thus every point on $C_{i,j}$ can be seen from a point outside $CH(P)$, and by following the lines from $u$ through $C_{j,i}$, we see that a half-line can be drawn from each point on $C_{i,j}$ such that they do not intersect $P$, (except at that point).
Figure 6.15 Weak External Visibility

The polygon in (a) is weakly externally visible and in (b) is not.

When the outer-kernel is inside the convex hull, weak external visibility is not guaranteed, as illustrated in Figure 6.14. Furthermore, not all pseudo-star-shaped polygons are weakly externally visible, and not all weakly externally visible polygons are pseudo-star-shaped, as illustrated in Figure 6.15.

As one might expect, there is a relationship between star-shaped polygons and pseudo-star-shaped polygons. We prove this in Property 6.5.

Property 6.5: A star-shaped polygon $P$ is pseudo-star-shaped if there exists an edge $f$ of $K(P)$ that is contained by an edge $e$, of $P$. 
Both polygons are pseudo-star-shaped, but the one in (b) is not weakly externally visible.

**Proof:** Since $P$ is star-shaped and an edge $f_i$ of $K(P)$ is contained by an edge $e_j$ of $P$, each point of $P$ lies in the right half-plane defined by $e_j$. Construct a region $Q$ by intersecting the left half-plane defined by $e_j$ with the right half-planes defined by $f_{i-1}$ and $f_{i+1}$. This region can be seen to be difference between the kernel formed by Lee and Preparata's algorithm applied to $C_{j+1,j}$ and $K(P)$. Thus $C_{j+1,j}$ is completely visible from any point in $Q$ when $e_j$ is deleted. We choose a point $u$ in $Q$ and move it close to $e_j$, so that $p_j$ and $p_{j+1}$ are support vertices of $P$ with respect to $u$. Then $e_j$ is completely visible from $u$ and $C_{j+1,j}$ is completely visible from $u$ through $e_j$. Therefore $P$ is pseudo-star-shaped with respect to $u$. See Figure 6.16. ■
Figure 6.15 Weak External Visibility and Pseudo-Star-Shapedness

(a) is weakly externally visible and pseudo-star-shaped, (b) is pseudo-star-shaped but not weakly externally visible, and (c) is weakly externally visible but not pseudo-star-shaped.
Figure 6.16 Star-Shapedness and Pseudo-Star-Shapedness

This polygon is both star-shaped and pseudo-star-shaped.

By Property 6.5 it is clear that both completely and strongly edge-visible polygons are pseudo-star-shaped. We can also relate Keil's original definition of pseudo-star-shapedness to the third type of edge-visibility, weakly edge-visible polygons.

Property 6.6: If $P$ is "Keil pseudo-star-shaped" then $P$ is weakly edge-visible.

Proof: Since $P$ is pseudo-star-shaped under Keil's definition, there exists an edge $e_i$ of $P$ and a point $u$ in the exterior of $P$ such that $P$ is visible from $u$ through $e_i$. This implies that each point of $P$ can be seen from some point on $e_i$, which proves weak edge-visibility.

The converse is not necessarily true. Keil's pseudo-star-shaped polygons have a "base": the entire polygon lies to one side of a certain edge. The same is not always true for weakly edge-visible polygons.
We can now see the relationship between pseudo-star-shaped polygons and other classes of polygons. By definition, all convex polygons are monotone, star-shaped, and pseudo-star-shaped. By Theorem 6.1, all monotone polygons are pseudo-star-shaped. As well, all completely edge-visible and strongly edge-visible polygons are pseudo-star-shaped. In Figure 6.6 we showed that some polygons are both pseudo-star-shaped and star-shaped, and in Property 6.6 we showed which polygons are both, but it is clear that neither class contains the other. The same holds for pseudo-star-shapedness and weak external visibility, and for pseudo-star-shapedness and weak edge-visible polygons.

6.4.2 The Spy Problem

One application of pseudo-star-shaped polygons is the "spy problem" which we introduce here. Let us assume that we wish to do some undercover surveillance on a closed room. The equipment available to us is an infrared camera which can see through exactly one wall, and a microphone which can hear through exactly one wall. The spy problem asks the question, "Where, outside the room, can we place the surveillance equipment in order that we may see and hear the whole room?" The answer is obvious given the class of pseudo-star-shaped polygons.

We first transform the walls of the room into the boundary of a polygon. Using Algorithm 6.3 we can test the polygon for pseudo-star-shapedness, and if the test is positive, we can place the surveillance gear at any point of any outer-kernel to see and hear the entire room. Of course, given that these are mechanical devices with some range limitations, it would realistically be desirable to find the "best" outer-kernel point. This can be done by testing each outer-kernel to minimize the maximum distance from the point in the outer-kernel to the far side of the polygon (see [BhT81] and [TMe82]) and choosing the outer-kernel with the minimum maximum distance.
6.4.3 Point Inclusion

In Chapter 4 we showed how the point inclusion problem can be solved with respect to a monotone polygon by using a "slab" method. For star-shapedness in Chapter 5, we described the similar "wedge" method for solving point inclusion. In this section we combine the two methods to produce a solution to the point inclusion problem given a pseudo-star-shaped polygon.

Let \( P \) be a pseudo-star-shaped polygon and let \( w \) be a point in the exterior of \( P \) from which \( P \) is pseudo-star-shaped. From \( w \) we draw a half-line through each vertex \( p_k \) of \( P \), as in the wedge method for star-shaped polygons. Since \( P \) is pseudo-star-shaped from \( w \), the set of half-lines divide the plane into \( n \) wedges. One of these wedges is empty, the other wedges cross two edges of \( P \), one from \( C_{1,j} \) and the other from \( C_{2,n} \), as in the slab method for monotone polygons. See Figure 6.17.
By the similarity with this problem on the aforementioned classes of polygon, it is clear that once again we have $O(n)$ storage space and an $O(\log n)$ query time per test point.

6.4.4 Union and Intersection

The union and intersection of two pseudo-star-shaped polygons can be found in linear time provided that there exists a region in the plane that is the intersection of at least two outer-kernels, one from each polygon. As in the point inclusion problem, we borrow elements of both the slab and wedge methods described in previous chapters.

Let $P = (p_1, \ldots, p_n)$, with edges $e_1, \ldots, e_n$, and $Q = (q_1, \ldots, q_m)$, with edges $f_1, \ldots, f_m$, be two pseudo-star-shaped polygons, and let $w$ be a point from which both $P$ and $Q$ are pseudo-star-shaped. The vertices of each are sorted angularly with respect to $w$; thus we merge the lists of vertices in linear time into list $v_1, \ldots, v_{n+m}$ and partition the plane into wedges by drawing half-lines from $w$ through each vertex $v_i$. Each wedge will then intersect:

1. No edges of $P$ and no edges of $Q$;
2. No edges of $P$ and edges $f_h$ and $f_i$ of $Q$;
3. Edges $e_j$ and $e_k$ of $P$ no edges of $Q$; or
4. Edges $e_j$ and $e_k$ of $P$ and edges $f_h$ and $f_i$ of $Q$.

In case (4) either edge $e_j$ or $e_k$ may intersect either edge $f_h$ or $f_i$, by testing the 4 edges in each wedge in constant time, we determine all intersection points in linear time. If there are no intersections, then the polygons do not intersect or one polygon contains the other. The method used to compute union or intersection is similar to that for two polygons monotone in the same direction [Sha78]; we illustrate for the union case. For vertex $p_n$, we call $e_n$ the outgoing edge, and $e_{n-1}$ the incoming edge. We assume that the polygons intersect, otherwise the union consists of the two separate polygons. Given the list of intersection points of the polygon's boundaries, we select one of these points and begin a traversal by selecting the leftmost outgoing edge at that point; that is, the edge
which does not go inside one of the polygons. If at any time this edge intersects another edge, we turn left onto the new edge. The traversal continues in this manner until we return to the original point. If any intersection points remain untraversed, then we repeat the process starting at one of these points. When all the intersection points have been visited, the set of traversed edges defines the union of the two polygons.

Recall that the union and intersection of star-shaped polygons having intersecting
kernels is also star-shaped. For two pseudo-star-shaped polygons having intersecting outer-kernels, the polygons need not intersect, and if they do, the intersection may be disjoint pieces. Each piece of the intersection is pseudo-star-shaped, once again by the angular ordering of the points with respect to a point from which both are pseudo-star-shaped. However, the union is not necessarily pseudo-star-shaped. In fact, it may be a polygon with one or more "holes", and thus is not a simple polygon as defined in this thesis. These particular cases of union and intersection are illustrated in Figure 6.19.
6.4.5 Triangulation

Recall from Chapter 2 that Chazelle derived the concept of smoothis to reduce the complexity of the triangulation problem from $O(n \log n)$ as close as possible to $O(n)$. We know that it is possible to triangulate certain classes of polygon (e.g., monotone, star-shaped, weakly edge-visible) in $O(n)$ time. We now show that we can also add pseudo-star-shaped polygons to this list.

Toussaint [Tou83] triangulates a monotone polygon in linear time by first decomposing it into weakly edge-visible pieces. The weakly edge-visible pieces are then triangulated using a method that was derived from Sklansky’s convex hull algorithm [ToA82]. We claim that we can do a similar decomposition by using the pseudo-star-shaped property.

Algorithm 6.4: Triangulating a Pseudo-Star-Shaped Polygon

Input: A pseudo-star-shaped polygon $P$ and a point $w$ from which it is pseudo-star-shaped.

Output: A partitioning of $P$ into triangles.

BEGIN
1. Calculate support vertices $p_i$ and $p_j$ with respect to $w$.
   Store the resulting chains as lists, sorted angularly with respect to $w$,
   $C_{p_j} = p_1, \ldots, p_i$, called 'p-vertices' and
   $C_{p_i} = q_1, \ldots, q_i$, called 'q-vertices'.
2. Merge the two sorted lists to obtain a new sorted list;
3. Scan the new list and whenever a $p$ (or $q$) vertex
   follows a $q$ (or $p$) vertex add a diagonal between them.
   This gives a decomposition of $P$ to weakly edge-visible polygons;
4. Apply the algorithm of [ToA82] to triangulate the weakly edge-visible polygons.
END

Theorem 6.4: Algorithm 6.4 correctly partitions a pseudo-star-shaped polygon into weakly edge-visible pieces in $O(n)$ time.

Proof: Since Algorithm 6.4 is taken directly from [Tou83], the proof is merely an adaptation of Toussaint’s proof, as we have changed from a linear ordering of the vertices in
the monotone case to an angular ordering in the pseudo-star-shaped case. Throughout the proof, refer to Figure 6.19.

We first show that the diagonals added lie completely in the polygon and that no two diagonals intersect. Let \( p, q \) be an added diagonal. The intersection of a wedge and a pseudo-star-shaped polygon is either empty, a triangle, or a convex quadrilateral, since each wedge intersects only one edge of \( C_{i,j} \) and only one edge of \( C_{i,j} \). We ignore the empty case and in the remaining two cases, \( p \) sees \( q \). Thus the diagonal \( p, q \) is inside at the intersection and therefore inside \( P \). By the fact that each diagonal is defined by a \( q \)-vertex.
and a p-vertex adjacent in the sorted list, it is clear that no two diagonals intersect.

The proof that each piece is weakly edge-visible, and that weakly edge-visible pieces can be triangulated, and the proof of $O(n)$ complexity follows directly from [Tou83] and [ToA82].

6.5 Conclusions

In this chapter we have introduced and examined a new class of polygons, called pseudo-star-shaped. In the hierarchy of classes of polygon, we note that the class pseudo-star-shapedness can be seen as a generalization of monotonicity. It is also related to star-shapedness, edge-visibility, and weak external visibility.

With this new class of polygons we were able to add a new wrinkle to the guard placement problem. We called this new problem the spy problem. We were also able to show that some known methods can be adapted to solve the point inclusion, intersection, and union, problems for pseudo-star-shaped polygons. Most importantly, we show that pseudo-star-shapedness is another class of polygons which can be triangulated in $O(n)$ time.

In the next chapter, we will summarize the work contained in this thesis and present some open problems.
Concluding Remarks and Open Problems

In the previous six chapters of this thesis we have studied some of the various structural properties of polygons. The first topic we examined was a way of measuring the amount of winding in a polygon's boundary. The method of choice for this thesis is the labelling scheme, a way of labelling each edge of the polygon with a real number representing the sum of the turns at each vertex from the vertex with maximum y-coordinate until the vertex at the end of the current edge.

In Chapter 3 we presented several properties obtained through the use of this labelling scheme. We then used the labels to adapt an algorithm, originally written for rectilinear polygons, to solve the hidden line elimination problem for simple polygons in the parallel model of visibility.

Several of the algorithms presented in this paper have been implemented on a Sun Graphics Workstation in the C programming language, using the Suncore graphics package. These algorithms include the parallel model of visibility (Algorithm 3.1), detection of monotonicity (Algorithm 4.1), and detection of star-shapedness [LeP79]. We have taken some sample pictures of these implementations and have reproduced them in Chapters 3-5.

In Chapter 4 we switched from the study of winding properties to the study of classes of polygons. The first class of polygons studied was the class of monotone polygons. Using labels, it was shown that the labels of a monotone polygon lie in the range \([-2,0,6,0]\). Using this fact, we showed that the labelling scheme can be used to enhance an earlier algorithm which detects monotonicity and computes directions of monotonicity. The new version runs in linear time but is more efficient in that the amount of work depends on the number of reflex vertices in the polygon.

More work can be done in the area of monotonicity. We leave as an open problem the following question: Given that \(P\) is a polygon not monotone in any direction, does
there exist a linear time algorithm which partitions the boundary of $P$ into a (minimum?) number of polygonal chains monotone in the same direction? It may be possible to use the result in the regularization problem [GJP78], in which a polygon is decomposed into a number of monotone polygons in $O(n \log n)$ time.

We next described the class of star-shaped polygons and the related class of edge-visible polygons. It was also shown that the labels of a star-shaped polygon must lie in the range $[-20, 60]$, just as in the monotone case. Labels provide another necessary condition for structural properties. It was shown that if there exists two labels $l_i$ and $l_j$, $j > i$, and $l_i > l_j + 2$, then the polygon is neither monotone nor star-shaped.

The final topic examined was a new class of polygons which we call pseudo-star-shaped. It has elements of star-shapedness and is seen as a generalization of monotonicity. Defining this class led to the study of outer-kernels, the sets of points from which a given polygon is pseudo-star-shaped. One open problem which results from this thesis is to determine the best upper bound for the number of outer-kernels for a pseudo-star-shaped polygon.

In the hierarchy of classes of polygons, it is shown that all monotone polygons are pseudo-star-shaped, and so are completely and strongly edge-visible polygons. This came as a result of showing which types of star-shaped polygons are pseudo-star-shaped. We also showed that certain pseudo-star-shaped polygons are weakly edge-visible and that there exists a type of pseudo-star-shaped polygon which is also weakly externally visible.

We presented an algorithm which detects pseudo-star-shapedness and calculates the set of outer-kernels, all in $O(n^3)$ time. As an open problem we leave the following: Does there exist a more efficient algorithm which detects pseudo-star-shapedness, (can it give at least one point from which the polygon is pseudo-star-shaped)? Furthermore, if the algorithm reports all outer-kernels, find an algorithm which is linear in the number of outer-kernels.

Pseudo-star-shapedness has some valuable applications apart from its stature as a "fa-
ther" of monotonicity. Knowing that a polygon is pseudo-star-shaped, the point inclusion problem can be solved with $O(n)$ storage space and $O(\log n)$ query time. Furthermore, if we know that two polygons have intersecting outer-kernels, we can compute union and intersection in linear time. The solutions to these problems are based on a "wedge" method.

The "spy problem" was introduced. In this problem we are equipped with a camera and/or a microphone which can see and/or hear through exactly one wall, and we wish to spy on a room. Given that the room is pseudo-star-shaped, the solution is to place the equipment in an outer-kernel near the polygon.

It is known that monotone polygons can be triangulated in linear time. The class of pseudo-star-shaped polygons is more general, but we showed a linear-time algorithm suffices. It remains an open problem whether all weakly externally visible polygons can be triangulated in $O(n)$ time.
References


[CuR84] Culberson, J., Rawlins, G., Turtle Polygons, CS-84-41, Department of Computer Science, University of Waterloo, Waterloo, Ontario, November 1984


[McT83] McKenna, M., Toussaint, G.T., *Finding the Minimum Distance between Two Distinct Convex Polygons in Linear Time*, Technical Report SOCS-83-6, School of Computer Science, McGill University, Montreal, Quebec, April 1983


END
210286
FIN