

# Polygon Reconstruction from Visibility Information

LILLANNE ELAINE JACKSON  
B.Sc., University of Alberta, 1983

A Thesis  
Submitted to the Council on Graduate Studies  
of the University of Lethbridge  
in Partial Fulfillment of the  
Requirements for the Degree

MASTER OF SCIENCE

LETHBRIDGE, ALBERTA  
April 3, 1996

©LillAnne Elaine Jackson, 1996

To my husband,  
Gerry Patrick Jackson.

# Abstract

Reconstruction results attempt to rebuild polygons from visibility information. Reconstruction of a general polygon from its visibility graph is still open and only known to be in PSPACE; thus additional information, such as the ordering of the edges around nodes that corresponds to the order of the visibilities around vertices is frequently added.

The first section of this thesis extracts, in  $O(E)$  time, the Hamiltonian cycle that corresponds to the boundary of the polygon from the polygon's ordered visibility graph. Also, it converts an unordered visibility graph and Hamiltonian cycle to the ordered visibility graph for that polygon in  $O(E)$  time.

The second, and major result is an algorithm to reconstruct an orthogonal polygon that is consistent with the Hamiltonian cycle and visibility stabs of the sides of an unknown polygon. The algorithm uses  $O(n \log n)$  time, assuming there are no collinear sides, and  $O(n^2)$  time otherwise.

## Acknowledgements

There are many people who have contributed substantially to this thesis. My appreciation goes out to every person.

My supervisor, Professor Stephen K. Wismath, is an excellent teacher and researcher whose patience, encouragement and motivation were instrumental in completing this thesis. The friendship Steve, his family and colleagues have provided throughout this educational time in my life is greatly appreciated.

The Math and Computer Science department at the University of Lethbridge and the Electronics Engineering Technology department at the Lethbridge Community College are two groups of incredibly supportive people whose positive feedback has contributed substantially to the completion of this thesis. Also, I am grateful for the time and financial assistance provided to me by my employer, the Lethbridge Community College, which allowed me to consider working toward a Master of Science degree in the first place.

My husband and our circle of friends and family have always had confidence in me, even when I had none myself. Without them, neither I nor this thesis would exist. Thank you.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                                       | <b>1</b>  |
| 1.1      | Art Gallery Problems . . . . .                            | 1         |
| 1.2      | Implementations . . . . .                                 | 5         |
| 1.3      | Overview . . . . .  | 6         |
| <b>2</b> | <b>Definitions</b>  | <b>9</b>  |
| 2.1      | Complexity Theory . . . . .                               | 9         |
| 2.2      | Graph Theory . . . . .                                    | 11        |
| 2.3      | Geometry . . . . .  | 13        |
| 2.4      | Visibility and Visibility Graphs . . . . .                | 14        |
| <b>3</b> | <b>Ordered Visibility Graphs of Polygons</b>              | <b>19</b> |
| 3.1      | Ordered Visibility Graph Specified . . . . .              | 21        |
| 3.1.1    | Algorithm: Determine Hamiltonian Cycle . . . . .          | 28        |
| 3.2      | Hamiltonian Cycle Specified . . . . .                     | 32        |
| 3.3      | Summary . . . . .   | 33        |
| <b>4</b> | <b>Orthogonal Polygon Reconstruction</b>                  | <b>35</b> |
| 4.1      | Horizontal Rectangles . . . . .                           | 37        |
| 4.2      | Identification of Rectangles . . . . .                    | 43        |
| 4.3      | Algorithm - Determine <i>CONVEX/REFLEX</i> . . . . .      | 45        |
| 4.3.1    | Classify and Identify Rectangles: Types 0 to 11 . . . . . | 46        |
| 4.3.2    | Identify Rectangles: Type 12 . . . . .                    | 47        |
| 4.3.3    | Determine Convexity of Vertices . . . . .                 | 50        |
| 4.4      | Algorithm - Reconstruct Polygon . . . . .                 | 53        |
| 4.5      | Related Results . . . . .                                 | 59        |
| 4.5.1    | Edge Visibility Trees . . . . .                           | 59        |
| 4.5.2    | Bar Visibility Graphs . . . . .                           | 62        |
| 4.6      | The Collinear Sides Assumption . . . . .                  | 68        |
| 4.7      | Summary . . . . .   | 71        |
| <b>5</b> | <b>Conclusions and Open Problems</b>                      | <b>72</b> |

# List of Figures

|      |  |    |
|------|--|----|
| 1.1  | Illuminating the Interior of a Building . . . . .                                  | 2  |
| 2.1  | An Example of a Graph . . . . .  | 11 |
| 2.2  | A Simple and a Non-Simple Polygon . . . . .  | 13 |
| 2.3  | An Orthogonal Polygon . . . . .  | 14 |
| 2.4  | Object $x$ is Visible to Object $y$ but not $z$ . . . . .                          | 15 |
| 2.5  | The Data Structure that Stores a Visibility Graph . . . . .                        | 17 |
| 3.1  | Left Side and Right Side of a Visibility Only Edge . . . . .                       | 23 |
| 3.2  | One Witness on Left Side . . . . .   | 24 |
| 3.3  | More than One Vertex on Each Side of the Polygon . . . . .                         | 25 |
| 3.4  | $p$ 's Adjacency Row . . . . .   | 29 |
| 3.5  | Node $i$ 's List and Pointer to $p$ . . . . .                                      | 30 |
| 4.1  | Example Input for the OPR Problem . . . . .  | 36 |
| 4.2  | An Orthogonal Polygon with Horizontal Stabs . . . . .                              | 37 |
| 4.3  | The Twelve Possible Horizontal Rectangles . . . . .                                | 38 |
| 4.4  | The Two Type 0 Rectangles . . . . .  | 38 |
| 4.5  | Turning Toward and Away from Rectangle . . . . .                                   | 39 |
| 4.6  | Two Vertices Correspond to Rectangle Corners . . . . .                             | 40 |
| 4.7  | A Vertex is Part of 3 Rectangles . . . . .   | 42 |
| 4.8  | Two Orientations of Rectangles Around a Vertex . . . . .                           | 43 |
| 4.9  | <i>same</i> and <i>opposite</i> Sets Corresponding to a Type 6 Rectangle . . . . . | 46 |
| 4.10 | A Type 12 Rectangle . . . . .  | 47 |
| 4.11 | A Polygon with Vertices Isolated by Type 12 Rectangles . . . . .                   | 48 |
| 4.12 | A Polygon with $O(n)$ Type 12 stabs to Some Vertical sides . . . . .               | 49 |
| 4.13 | Predecessor Segments and Preceding Vertices . . . . .                              | 55 |
| 4.14 | $X$ and $Y$ Digraphs . . . . .   | 58 |
| 4.15 | Reconstructed Polygon, from Figure 4.14 . . . . .                                  | 59 |
| 4.16 | Join Sides $x$ and $y$ through the Polygon's Interior . . . . .                    | 61 |
| 4.17 | Join Sides $x$ and $y$ . . . . .   | 63 |
| 4.18 | $\infty$ Visible Rectangles . . . . .  | 65 |
| 4.19 | A Graph with Cutpoints on Exterior Face . . . . .                                  | 67 |
| 4.20 | Extra Rectangles Possible with Collinear Sides . . . . .                           | 69 |
| 4.21 | Collinear Sides Along the <i>stabs</i> of a Type 1 Rectangle . . . . .             | 70 |

# Chapter 1

## Introduction

Computational geometry is a study of the complexity of computer algorithms that solve geometric problems. In the preface of *Computational Geometry - Methods, Algorithms and Applications*, Biere and Würzburg [BN91] describe computational geometry as a:

“nearly mathematical discipline, dealing mainly with complexity questions concerning geometrical problems and algorithms. But ... increasingly, questions of a more practical relevance are central, such as applicability, numerical behavior and performance for all kinds of input size.”

Several excellent surveys of the field are available [LP84], [PS85], [O’R93b], [Ede87].

### 1.1 Art Gallery Problems

An emerging area of computational geometry is the study of art gallery problems. The classic art gallery problem is to determine the minimum number of security people needed to guard the valuables in an art gallery. This could also be stated as, determining how many lights are needed to light up every corner of the interior of a building. A related problem asks how many security check points are needed to ensure that a guard walks around every part of a patrolled building. Another similar

problem, the prison yard problem, determines the minimum number of guard towers that are needed on the perimeter of a prison yard to be sure that no part of the exterior fence is hidden from view. Inherent in each of these problems is the question of where to place guards, check points, or lights, as well as how many of each are needed and the algorithmic complexity of placing them. In each case, the patrolled or illuminated area is modeled by a polygon. To be illuminated or guarded, there must be at least one light or guard that is not obstructed from seeing a part of the polygon.

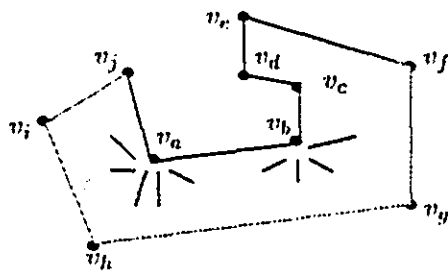


Figure 1.1: Illuminating the Interior of a Building

In the art gallery, lighting, or security guard problems the shaded areas of the polygon of figure 1.1 would be illuminated, or guarded by placing lights, guards, or check points on vertices  $v_a$  and  $v_b$ , but there is an obstruction (the side of the polygon) that hides the non-shaded area. Using vertex  $v_e$  instead of  $v_b$ , however, would guard or light the entire area. The polygon of figure 1.1 in the prison yard application would need at least three guard towers, located, for example, at vertices  $v_f$ ,  $v_h$ , and  $v_j$ . Applications of art gallery theorems arise in many diverse areas, such as graphics (eg, hidden line removal [PS85]), CAD [BM95], robotics [Kle92] [LOS95], VLSI design



[Len90].

All art gallery problems are concerned with the visibility relationship between a set of objects. That is, the problems study which other objects can be seen by each of the objects of the set. Obviously, if there is an obstruction, such as a wall of the gallery, separating two objects in the set, they cannot see each other. In the literature, the visibility between many kinds of objects has been studied. For example, the visibility among the vertices of a polygon, the sides of a polygon, line segments in the plane, rectangles in the plane, and various other objects in two and three dimensions, have all been recently considered.

A visibility graph is a model that indicates the visibility between each pair of objects in the set. For example, an internal vertex visibility graph has a node for each vertex of a polygon and an edge joining a pair of nodes if the corresponding vertices can see each other through the inside of the polygon. An endpoint visibility graph of a set of line segments has a node for each segment endpoint, and an edge joining a pair of nodes if the corresponding endpoints can see each other. These are only two of many examples of different visibility graphs that are considered in the visibility literature.

In the study of visibility, the following problem types have emerged:

- **Construction:** Given a set of objects in the plane, construct the corresponding visibility graph. Considerable literature exists on construction of these graphs. For the visibility graph of the endpoints of a set of line segments Sudarsan and Rangan [SR90] presented an  $O(|m| \log^2 n)$  algorithm, where  $m$  is the number

of edges in the visibility graph and  $n$  the number of nodes. For the visibility graphs of polygons  $O(n^2)$  algorithms have been presented by Asano, Asano, Guibas, Hershberger, and Imai [AAG<sup>+</sup>86] and by Welzl [Wel85]. For vertices of polygons, a more efficient  $O(m + n \log \log n)$  algorithm was presented in Hershberger [Her87], where  $n$  is the number of vertices in the polygon, and  $m$  is the number of internal visibilities between pairs of vertices. In fact, the  $n \log \log n$  factor reflects the triangulation<sup>1</sup> algorithm of Tarjan and Van Wyk [TV88]. Chazelle [Cha91] has since presented a triangulation algorithm that uses only  $O(n)$  time. Using Chazelle's algorithm the Hershberger result becomes  $O(m + n)$ , which is more properly expressed as  $\tilde{O}(m)$ , since  $2n - 3 \leq m \leq (n^2 - n)/2$  for the visibility graph of any polygon<sup>2</sup>.

- **Characterization and Recognition:** Characterization and recognition are related activities. The first involves characterizing the essential features of visibility graphs, while the second determines if a given graph is a visibility graph. Characterization and recognition have been examined for various objects, e.g., for simple polygons by Ghosh [Gho88] and for funnel-shaped polygons by Choi, Shin, and Chwa [CSC92], but in general the problems are not yet completely solved. Everett and Corneil [EC95] presented some negative characterization results for polygons, and Everett [Eve90] showed that the problem of recognizing a graph representing the vertices of a polygon is in PSPACE<sup>3</sup>. However,

---

<sup>1</sup>The triangulation of a polygon involves adding the maximum number of noncrossing internal diagonals to the polygon.

<sup>2</sup>See page 166 of [O'R87] for derivation of this inequality.

<sup>3</sup>This term is defined in section 2.1

Coullard and Lubiw [CL91] have presented an algorithm that solves the recognition problem for distance visibility graphs. The distance visibility problem is: given an edge-weighted graph  $G$ , is it the visibility graph of a simple polygon with the given weights as Euclidean distances?

- **Reconstruction:** Reconstruct the set of objects from its visibility graph. The Reconstruction problem is currently unsolved, except in very restricted cases. Often researchers augment the visibility graph with additional information, in order to complete the task. O'Rourke and Streinu [OS95] have shown that reconstruction of a pseudo-polygon from a vertex-edge visibility graph is in NP. This vertex-edge visibility graph is a bipartite graph that has nodes representing both vertices and sides of the polygon, and edges joining nodes when corresponding vertices can see corresponding sides.

Good coverage of existing visibility literature is presented in the book, *Art Gallery Theorems and Algorithms*, by O'Rourke [O'R87] and is updated by Shermer [She92] in the article *Recent Results in Art Galleries*. In the *Computational Geometry Column* of the SIGACT News [O'R93a], O'Rourke classifies and references results on visibility graphs prior to 1993.

## 1.2 Implementations

Software that implements computational geometry algorithms include: GraphEd [Bra], the Workbench for Computational Geometry [KMM<sup>+</sup>90] and GeoLab (an environment for development of Algorithms in computational geometry) [dRJ93]. LEDA

(Library of Efficient Data types and Algorithms) [NU95] is a library of C++ routines to assist in the development of implementations of computational geometry (and other) algorithms. A model, called Mocha [BCLT96], has been developed that uses the HotJava [GM95] browser to display animations of geometry algorithms for the World Wide Web.

A limited amount of work has been done writing implementations of visibility algorithms. At Smith College in Massachusetts, USA, a program that draws a polygon and then determines its vertex visibility graph was developed by Alef and Streinu [AS95]. VisPak [JPW95], a package of implementations of visibility algorithms uses LEDA to construct visibility graphs of various objects and was developed at the University of Lethbridge. VisPak contains a drawing editor to input various geometric objects, and has programs that determine the visibility graphs of: vertical line segments, rectangles, the vertices of a general simple polygon, the vertices of an orthogonal simple polygon, and the endpoints of a set of disjoint line segments. Also, VisPak contains an implementation of an algorithm written by Keil and Wismath [KW95] to determine the visibility polygons of the endpoints of a set of line segments.

### 1.3 Overview

This thesis presents two solutions related to reconstruction of objects from visibility information. As with all previous results, this work does not completely solve the reconstruction problem, only a restricted case.

The first result presented here could be a useful tool for future reconstruction so-

lutions. Notice that a visibility graph (also called an unordered visibility graph) does not contain any ordering information. The layout, or order, of the nodes of the graph bears no relation to the layout for the objects it represents. When reconstructing a *polygon* from its visibility graph, researchers often include the Hamiltonian cycle of the polygon to the input of the problem. The Hamiltonian cycle of a polygon is a list of the vertices in the order they appear on the boundary of the polygon. It identifies which edges of the graph represent sides of the polygon, as well as the order in which those sides occur. When reconstructing a set of *line segments* from their visibility graph, some researchers expect the edges about each node to be presented in cyclic order [Wis94]. That is, the cyclic order of edges around each node of the graph are in the same order as the visibilities to other segment endpoints as seen from the corresponding endpoint. The new work in chapter 3 presents an algorithm that converts the internal visibility graph (unordered) of the vertices of a polygon and its Hamiltonian cycle, to a cyclically ordered visibility graph, and vice versa. The purpose of this algorithm is to allow techniques developed for visibility graphs of the endpoints of disjoint line segments in the plane to be applied to visibility graphs of the vertices of a polygon.

The second and major result of this thesis is the reconstruction of an orthogonal polygon from its (extended) visibility information. The general visibility reconstruction problem is sufficiently difficult that it has not yet been completely solved. There are a few results that reconstruct specific objects from their visibility information, with various additional inputs. For example, ElGindy [ElG85] reconstructed monotone polygons from maximal outerplanar graphs. The reconstruction solution of this

thesis is a link between two previous results, the *Edge Visibility Trees* of Boothe and O'Rourke [O'R87] and Wismath's *Bar Visibility Graphs* [Wis85], which were previously viewed as unrelated special cases of the reconstruction problem. The Orthogonal Polygon Reconstruction (OPR) result of this thesis expects the visibility information to be input as the internal and external stabs of the sides of the polygon. For a side  $\overline{v_i v_j}$  of a polygon, the  $stab(v_i v_j)$  is defined to be the first side of the polygon intersected by a ray from  $v_i$  to  $v_j$ . If the ray does not intersect any side,  $stab(v_i v_j)$  is set to  $\infty$ . Note that for each polygon side, there are two stabs:  $stab(v_i v_j)$  and  $stab(v_j v_i)$ . This could also be expressed as: there are two stabs, one horizontal and one vertical, from each vertex. In addition this result uses the Hamiltonian cycle of the polygon. Note that the orthogonal restriction of the polygon reduces the possible angle measures at each vertex from an infinite number to exactly two,  $\Pi/2$  and  $3\Pi/2$ . This significant restriction still leaves the OPR problem challenging to solve, assuming the polygon has more than four vertices. An  $O(n \log n)$  algorithm that solves OPR is presented.

The remaining chapters are organized as follows: Definitions from the fields of complexity theory, graph theory, geometry and visibility are reviewed in Chapter 2; terms used in this work generally conform with existing literature. Any differences are slight and are described in the chapter. Chapter 3 presents the first result of this thesis: conversion (both directions) between an ordered visibility graph and the Hamiltonian cycle of the vertices of a simple polygon. The theory and algorithms of the orthogonal polygon reconstruction result are presented in Chapter 4. As well, a comparison is made between this result and closely related literature. Finally, the results are summarized and related open problems are discussed in Chapter 5.

# Chapter 2

## Definitions

This chapter contains terms and notation that will be used throughout the remainder of this thesis. Occasional differences between terminology used here and that found in the literature are indicated. Terms from complexity theory are presented because the field of computational geometry is concerned with presenting efficient algorithms. Since graph theory and geometry are fundamental to the study of visibilities, some definitions from these fields are covered. Finally, existing visibility concepts are defined.

### 2.1 Complexity Theory

The definitions of this section follow *Introduction to Algorithms* by Cormen, Leiserson and Rivest [CLR90]. Assume that  $x$  is the size of the input to a computer algorithm, and  $f(x)$  is a function of  $x$ .

Algorithms are usually analysed in terms of their asymptotic running times, that is, the limit as the input size increases, of the running time of the algorithm. The two asymptotic running times or bounds of an algorithm are expressed by  $O$  or  $\Omega$ .

The **upper bound** or  **$O$ -notation** is defined as:

$$O(g(x)) = \{f(x) : \exists \text{ positive constants } c \text{ and } x_0 \text{ s.t. } 0 \leq f(x) \leq cg(x), \forall x \geq x_0\}$$

The upper bound is often used to bound the worst case running time of an algorithm for any input. The **lower bound** or  **$\Omega$ -notation** is defined as:

$$\Omega(g(x)) = \{f(x) : \exists \text{ positive constants } c \text{ and } x_0 \text{ s.t. } 0 \leq cg(x) \leq f(x), \forall x \geq x_0\}$$

A **polynomial algorithm** or more properly, a **polynomially bounded algorithm** is one whose running time is  $O(x^k)$  for some constant  $k$ . A problem that can be solved by an algorithm whose running time is polynomially bounded is said to be *in the complexity class  $P$* . Some problems have not been shown to have polynomially bounded algorithms, but a (guessed) solution to a given problem of this type can be checked, *i.e.*, shown to be correct or incorrect, in polynomial time. These problems, including those that are in  $P$ , are said to be *in the complexity class  $NP$* , where  $NP$  stands for *non-deterministic polynomial*. We know that  $P \subseteq NP$ , but whether  $P = NP$  is not known.

In  $NP$  there is a group of problems that are called  **$NP$ -Complete**. None of these problems have been shown to have a lower bound that is more than polynomial, but as yet no polynomial time solutions have been found. To prove that a given problem is  $NP$ -Complete, it must be shown to be in  $NP$  and a transformation must be presented that converts a known  $NP$ -Complete problem to the given problem in polynomial time. In fact, if a polynomial algorithm that solves one  $NP$ -Complete problem is ever found, all  $NP$ -Complete problems will have polynomial time solutions. Since



much effort has been spent trying to find such a solution, most researchers view *NP*-Complete problems as intractable ones.

*NP*-**Hard** problems have polynomial time transformations from *NP*-Complete problems, but are not known to be in *NP*.

Discussions of algorithm complexity often involve an examination of the amount of time required to solve the problem. Another issue is the amount of space the algorithm uses. A problem that is in **PSPACE** is one that uses a polynomial amount of space. A **PSPACE-complete** problem is one that is in PSPACE and there is a polynomial transformation from another problem that is PSPACE-complete to the given problem.

The above terms will be used throughout the thesis to express the efficiency of algorithms, or the difficulty of finding solutions to problems.

## 2.2 Graph Theory

A **graph** is a mathematical abstraction of real world objects that consists of a set of *nodes* and a set of pairs of nodes called *edges*. An example is given in figure 2.1. Usually the nodes represent various objects and the edges indicate some type of

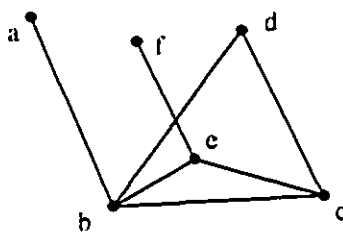


Figure 2.1: An Example of a Graph

relationship between them. The letter  $n$  will be used to indicate the number of nodes

in a graph and  $E$  to indicate the number of edges.

A **planar graph** is one that can be redrawn in the plane so that none of its edges intersect, except at nodes. The graph of figure 2.1 is planar since redrawing it with node  $f$  inside the triangle of nodes  $bcd$  would not change the nodes and edges of the graph, but would avoid any edge crossings. A graph is called **connected** if between every pair of nodes in the graph there is a path of edges, otherwise it is **disconnected**. A **1-connected graph** is one in which the removal of any one node and its incident edges would result in a disconnected graph. In a 1-connected graph, any node whose removal causes the graph to be disconnected is called a **cutpoint**. The graph in figure 2.1 is 1-connected, and nodes  $b$  and  $c$  are cutpoints. A **2-connected graph** is one that is not 1-connected and the removal of some two nodes and their incident edges would result in a disconnected graph. A sequence of edges that goes through a number of nodes of a graph and back to the original node is called a **cycle**, eg, edges  $(b, c)$ ,  $(c, d)$  and  $(d, b)$  in figure 2.1 form a cycle. A **Hamiltonian cycle** is a cycle that passes through every node of the graph exactly once. Some graphs do not have a Hamiltonian cycle, as illustrated by the graph of figure 2.1. A connected graph that has no cycles (Hamiltonian or not) is called a **tree**.

In this thesis, upper case letters,  $A, B, C, \dots$  are used for the names of entire graphs. Lower case letters,  $a, b, c, \dots$ , name the nodes of any single graph, and edges are expressed as unordered pairs of nodes such as  $(a, b)$ .

## 2.3 Geometry

All geometric objects considered in this work are two dimensional; they can be drawn in the plane. The individual points comprising the plane are identified by the Cartesian coordinate system.

A **line segment** is defined as two distinct points in the plane and all points between them on the unique line that contains the points. The two extreme points of the line segment are called its **endpoints**.

A **polygon** is a connected set of line segments with every segment endpoint shared by exactly two segments. The ordered line segments that define a polygon are called its **boundary**, and the endpoints of each of the segments are the **vertices** of the polygon. The boundary segments will also be called **sides** of the polygon. In the literature, the word *edge* is often used as a synonym for side (or boundary segment) of a polygon. We will not use this term here, in order to avoid confusion with its use in graph theory. A **simple polygon** is one in which the only common points between consecutive segments of the polygon boundary are vertices, and non-consecutive segments have no common points (figure 2.2). A simple polygon divides the plane into

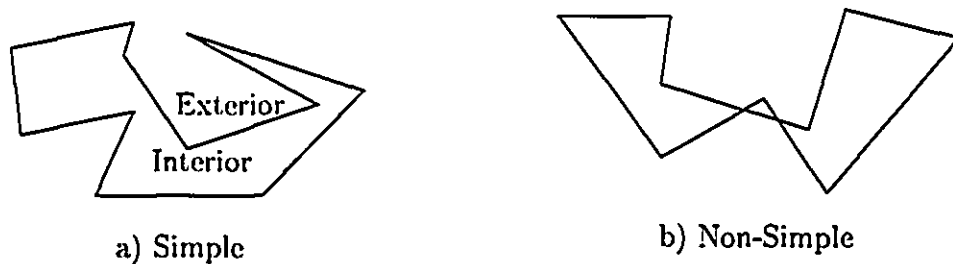


Figure 2.2: A Simple and a Non-Simple Polygon

an **interior** and **exterior** as shown in figure 2.2 a.

An **orthogonal polygon** is a polygon that has an internal angle of  $\Pi/2$  radians or  $3\Pi/2$  radians at every vertex, as in figure 2.3 for example. A vertex with interior angle of  $\Pi/2$  is called **convex** while one with an interior angle of  $3\Pi/2$  is **reflex**. All orthogonal polygons considered will also be simple polygons. It is assumed, without loss of generality, that the sides of an orthogonal polygon are oriented parallel to the  $X$  or  $Y$  axes of the cartesian coordinate system. **Collinear sides** of an orthogonal polygon are two or more non-adjacent boundary segments that share the same  $X$  or  $Y$  coordinate.

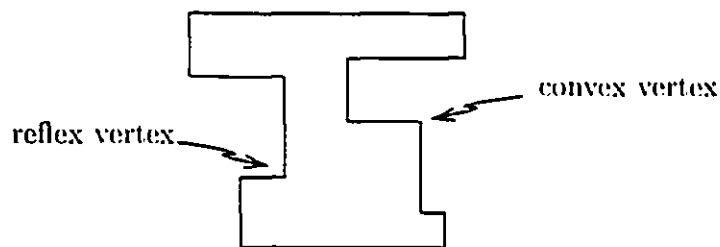


Figure 2.3: An Orthogonal Polygon

Throughout this thesis the letter  $n$  is used to indicate the number of vertices and sides of the polygon. The notation used to identify the vertices of a polygon is  $v_1, v_2, v_3, \dots, v_n$ , while its sides are described as  $\overline{v_1v_2}, \overline{v_2v_3}, \overline{v_3v_4}, \dots, \overline{v_nv_1}$ .

## 2.4 Visibility and Visibility Graphs

Visibility is described in terms of both geometric objects such as points, line segments, polygons, etc., and in terms of graphs.

An object  $x$  is **visible** to an object  $y$  if there exists a point  $x_i$  on  $x$  and a point  $y_j$

on  $y$  such that the line segment  $\overline{x_i y_j}$  connecting the points intersects no other objects of the set of objects. Figure 2.4 contains two sets of objects, and indicates a visibility and an invisibility in each set. The polygon of figure 2.4b uses the vertices of the



Figure 2.4: Object  $x$  is Visible to Object  $y$  but not  $z$

polygon as primary objects, and the sides of the polygon as objects that may obstruct visibility.

A **visibility graph** abstractly represents the visibility among the objects of interest. It has a node for each object in the set and an edge joining two nodes if the corresponding objects are visible to each other. For example, the visibility graph representing a simple polygon could have a node representing each vertex of the polygon, with edges indicating that the two corresponding vertices are visible to each other. In another context, each node of a visibility graph of a simple polygon represents a side of the polygon and the edges of the graph indicate that the corresponding polygon sides are visible to each other. As with general graphs, visibility graphs can be redrawn in the plane in any configuration that maintains the nodes and edges of the graph. The graph does not need to be laid out in a fashion that corresponds to the original objects. In fact, the ordering of the visibility edges around each node does

not need to be maintained in the ordinary, unordered visibility graph.

An **ordered visibility graph** of a set of objects has the same nodes and edges as an ordinary visibility graph, however, the order of visibilities about each node does correspond to the order of visibilities around each object in the geometric set of objects, and further, the ordering about each node is always in the same direction (*e.g.*, clockwise). Thus for each node in the graph there is a cyclic list that represents the clockwise (or counter clockwise) cycle of the objects seen from the corresponding object.

Below, the visibility graphs of some specific objects are defined so they can be easily referenced later.

The **internal vertex visibility graph**,  $V_v(P)$ , of a simple polygon,  $P$ , is a graph whose nodes correspond to vertices of the polygon and whose edges are incident with pairs of nodes that represent vertices that are visible *through the interior* of the polygon. The two endpoints of each boundary segment are defined as being visible to each other. The **ordered internal vertex visibility graph**,  $V_{vo}(P)$ , of  $P$  is an ordered version of  $V_v(P)$ . An **internal side visibility graph**,  $V_s(P)$ , of a simple polygon,  $P$ , is a graph whose nodes correspond to sides of the polygon, and edges join sides that are visible to each other through the interior of the polygon. (In the literature, this graph is usually called the internal edge visibility graph.) It is possible to order the edges of  $V_s(P)$ , thus giving an **ordered internal side visibility graph**,  $V_{so}(P)$ . Some references in the literature also consider an external vertex visibility graph of a polygon (chapter 6 [O'R87]). However, in this thesis, such a graph will not be used.

The **endpoint visibility graph**,  $V_e(S)$ , of a set of line segments,  $S$ , is a graph whose nodes correspond to the endpoints of a set of disjoint line segments, and edges join pairs of nodes that represent endpoints that are visible. The two endpoints of each segment are considered to be visible to each other. When the cyclic ordering of the edges around each node is specified, this is called the **ordered endpoint visibility graph**,  $V_{eo}(S)$ . A **bar visibility graph**,  $V_b(S)$ , of a set of parallel line segments,  $S$ , is a graph that has a node for each segment (also called bar) in the set and an edge joining pairs of nodes that correspond to bars that are visible to each other through a visibility path that is perpendicular to the direction of the bars. The ordering of this graph gives an **ordered bar visibility graph**,  $V_{bo}(S)$ .

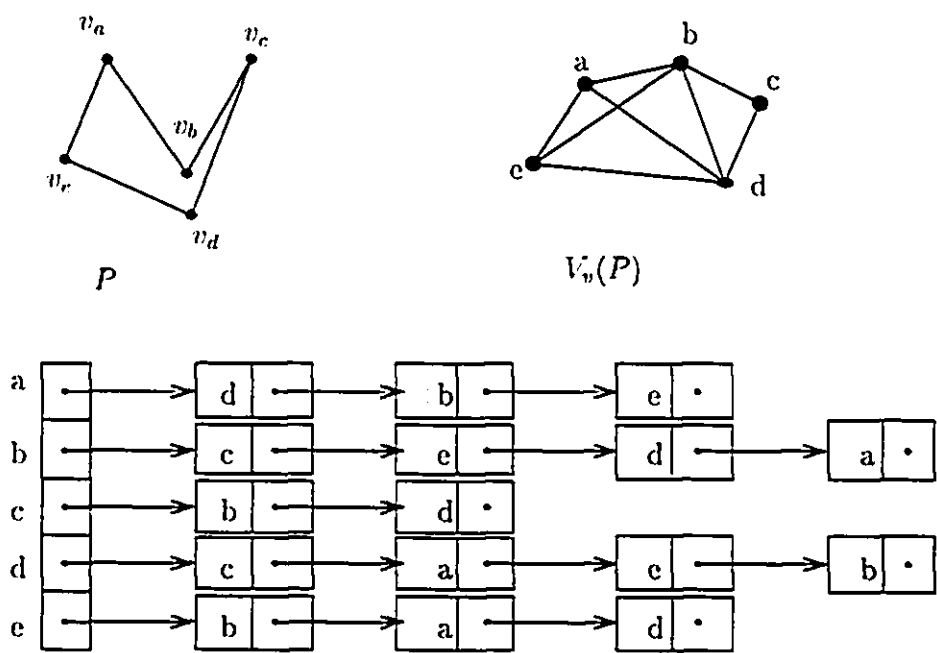


Figure 2.5: The Data Structure that Stores a Visibility Graph

The data structure used to store each of the visibility graphs described above is

an array of adjacency lists. That is, for every node  $i$  of the graph there is a list that contains each of the nodes that are incident to  $i$ . The example given in figure 2.5 shows a polygon,  $P$ , its corresponding internal vertex visibility graph,  $V_v(P)$ , and the adjacency lists that store  $V_v(P)$ . If the visibility graph is not ordered, the nodes are stored in random order in each list. If it is ordered, the order of each node in the list corresponds to the order of the vertices about the corresponding vertex. In the ordered graph, it is assumed that the node at the end of each list is linked to the node at the beginning of the same list, creating a cyclic list.



## Chapter 3

# Ordered Visibility Graphs of Polygons

Reconstruction problems in general are the reverse of construction problems. For example, given a set of points in the plane, constructing the cyclic ordering of all other points about each point is easily computed in  $O(n^2 \log n)$  time. However, the corresponding point reconstruction problem is known to be *NP*-Hard [Sho91]. The **Point Reconstruction**, PR, problem is defined as: given the cyclic ordering of all other points around each point (unembedded), determine an embedding of the points in the plane, consistent with the ordering information.

A related problem, the **Line Segment Reconstruction**, LSR, problem is defined as: reconstruct a set of line segments from the clockwise ordered visibilities around the endpoints of a set of (unembedded) line segments in the plane. An instance of PR can be transformed, in polynomial time, to LSR by stretching each of the points to a small line segment. Thus it is natural to allow additional input information to the visibility graph in order to reconstruct the line segments.

Reconstruction of a polygon from its visibility graph is a related unsolved problem. One result, by O'Rourke and Streinu [OS95], shows that reconstruction of a

pseudo-polygon from a vertex-edge visibility graph is in  $NP$ . To simplify the problem of reconstructing a polygon,  $P$ , from its internal vertex visibility graph,  $V_v(P)$ , the Hamiltonian cycle of the polygon is frequently added as part of the input. A visibility graph could contain many different Hamiltonian cycles. Determining if a graph contains a Hamiltonian cycle is  $NP$ -Complete [GJ79]. The Hamiltonian cycle used as input and referred to in the remainder of this chapter is the one that follows the boundary segments of the corresponding polygon.

Techniques for reconstructing a set of line segments are being developed independently from those for reconstructing a polygon. Typically, line segment reconstruction techniques use the *ordered* endpoint visibility graph instead of a Hamiltonian cycle. The two reconstruction problems, polygons and line segments, are similar, but due to the differing input, a solution to one does not imply a solution to the other. This chapter presents algorithms to convert  $V_v(P)$  of a polygon,  $P$ , and its Hamiltonian cycle to the *ordered* internal vertex visibility graph,  $V_{vo}(P)$ , of that polygon, and vice versa.

The chapter first presents an algorithm that extracts the Hamiltonian cycle of the boundary segments of a simple polygon from its *ordered* visibility graph,  $V_{vo}(P)$ . The algorithm performs this operation in  $O(E)$  time, where  $E$  is the number of edges in  $V_{vo}(P)$ . Recall that, for a simple polygon,  $2n - 3 \leq E \leq (n^2 - n)/2$ .

Also, an algorithm to create an ordered visibility graph,  $V_{vo}(P)$ , from the Hamiltonian cycle and unordered visibility graph,  $V_v(P)$ , of a simple polygon is presented. The routine, as developed here, also uses  $O(E)$  time.

Both results are thus worst case optimal and sensitive to the size of the output.

Using the results of this chapter, techniques presented for reconstructing a set of line segments can be applied to reconstructing a polygon.

### 3.1 Ordered Visibility Graph Specified

**INPUT:**

- $V_{vo}(P)$  : The Ordered Vertex Visibility graph of a simple polygon,  $P$ .

**OUTPUT:**

- The Hamiltonian cycle of  $V_{vo}(P)$  that corresponds to the boundary of the polygon,  $P$ .

In this section the Hamiltonian cycle of the polygon,  $P$ , is extracted from the polygon's ordered internal vertex visibility graph,  $V_{vo}(P)$ . Lemma 3, the main lemma of this section, is instrumental in the algorithm. It determines which edges of  $V_{vo}(P)$  represent sides of the polygon. First, some definitions and lemmas are presented.

Recall that an ordered vertex visibility graph of a polygon has a cyclic linked list (unidirectional) for each vertex of the polygon. The order of the nodes in each list corresponds to the clockwise order of each of the corresponding vertices around the given vertex.

An edge of the graph corresponds to either a **boundary segment** of the polygon, or a **visibility only segment** between two vertices through the interior of the polygon. However,  $V_{vo}(P)$  contains no indication of whether an edge represents a boundary or a visibility only segment.

A **witness** of a boundary segment or a visibility only segment of a simple polygon

is defined as a vertex that can see both endpoints and the *entire segment between the endpoints*. Thus, in the corresponding visibility graph (ordinary or ordered), the witness is adjacent to both nodes that represent the vertices at the endpoints of the segment.

Every node in the visibility graph of a polygon is adjacent to at least two other nodes. If a node,  $a$ , has exactly two neighbours,  $b$  and  $c$ , then node  $a$  (and corresponding vertex  $v_a$ ) is called a **blind ear**, and  $(a, b)$  and  $(a, c)$  are edges of the graph that represent boundary segments  $(\overline{v_a v_b})$  and  $(\overline{v_a v_c})$  of the polygon.

**Lemma 1** *Every edge of the visibility graph of a polygon,  $V_v(P)$  or  $V_{vu}(P)$ , that represents a boundary segment of the polygon has at least one witness.*

Proof: The triangulation theorem ([O'R87], page 12) shows that every polygon must admit a triangulation. Each triangle of a triangulation is on the interior of the polygon. Assume an arbitrary polygon  $P$  and a given triangulation of that polygon. Every boundary segment of  $P$  is part of one of the triangles of its triangulation. The vertex at the apex of that triangle is a witness of that boundary segment. This guarantees one witness for every boundary segment of the polygon. Since a polygon could have many different triangulations, a boundary segment may have more than one witness.  $\square$

**Lemma 2** *Every edge of the visibility graph that represents a visibility only segment of the polygon has at least two witnesses, one on each side of the segment.*

Proof: Any arbitrary visibility only segment can be used to cut a polygon into two separate polygons, with the segment (along the cut) as a part of the boundary of each of the new polygons. Applying the proof of Lemma 1 to each of the new polygons, ensures that the segment (along the cut) has at least one witness in each new polygon. Thus the visibility only segment has at least two witnesses, one on each side of the polygon.  $\square$

A visibility only segment,  $\bar{s}$ , with endpoints  $v_a$  and  $v_b$  is adjacent to two chains or sides of the polygon, the **left side** from  $v_b$  to  $v_a$ , and the **right side** from  $v_a$  to  $v_b$ , as shown in figure 3.1.

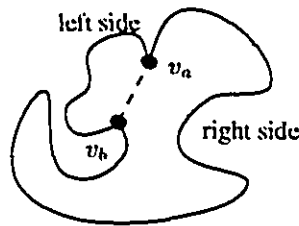


Figure 3.1: Left Side and Right Side of a Visibility Only Edge

Since our ultimate goal is to classify the edges of  $V_{vo}(P)$ , we now consider how the witnesses of the previous lemmas can be used in this endeavor.

**Lemma 3** *An edge,  $(a, b)$ , in an ordered visibility graph with more than 3 nodes represents a visibility only segment in a corresponding polygon if and only if at least one of the following three conditions exist:*

1. *One of the witnesses to  $(a, b)$  is a blind ear.*

2. *Some witness of  $(a, b)$  sees another node between  $a$  and  $b$  (with respect to the ordering around that witness).*
3. *There exists a pair of witnesses that see  $(a, b)$  consecutively and in opposite order, i.e., one sees first  $a$  then  $b$ , the other sees first  $b$  then  $a$  with no other nodes between them.*

Proof: ( $\Rightarrow$ ) Let nodes  $a$  and  $b$  correspond to vertices  $v_a$  and  $v_b$  on the polygon. Lemma 2 states that there is at least one witness vertex on each side of the visibility segment,  $\overline{v_a v_b}$ . Call these vertices  $v_c$  and  $v_d$ , and their corresponding nodes,  $c$  and  $d$ .

- Case 1: Assume that there is only one vertex,  $v_c$ , on the left side (similarly for the right side) of the segment, then  $v_c$  must be the witness for that side. (see figure 3.2) There are two possibilities:

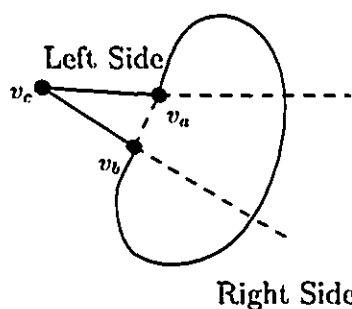


Figure 3.2: One Witness on Left Side

- None of the vertices of the polygon,  $P$ , that are visible to  $v_c$ , are in the sweep of the arc  $v_a v_c v_b$ . Then, in  $V_{v_c}(P)$ , node  $c$  is adjacent to just two nodes,  $a$  and  $b$ . Thus  $c$  is a blind ear, which is condition 1. On a blind ear, we have no indication of the ordering of the two nodes  $a$  and  $b$ , but

we do know that edges  $(c, a)$  and  $(c, b)$  represent boundary segments of the polygon, and that  $b$  is a witness to  $(c, a)$ , and  $a$  is a witness to  $(c, b)$ .

- There exists at least one vertex,  $v_x$  of the polygon that is visible to  $v_c$  and in the arc  $v_a v_c v_b$ . Thus,  $v_c$  and  $v_x$  are visible and, in  $V_{vo}(P)$  about node  $c$ ,  $x$  occurs between  $a$  and  $b$ . This is condition 2.

- Case 2: There is more than one vertex on each side of the polygon.(see figure 3.3) There are two possible situations:

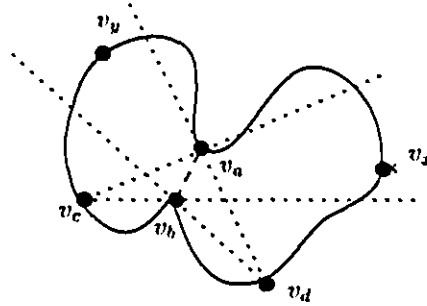


Figure 3.3: More than One Vertex on Each Side of the Polygon

- There exists at least one vertex  $v_x$  of the polygon that is visible to a witness  $v_c$  of segment  $\overline{v_a v_b}$  and in the arc  $v_a v_c v_b$ . Thus,  $v_c$  and  $v_x$  are visible and, in  $V_{vo}(P)$  about node  $c$ ,  $x$  occurs between  $a$  and  $b$ . This is condition 2.
- There are no visible vertices in either arc  $v_a v_c v_b$  or  $v_a v_d v_b$ , where  $v_c$  and  $v_d$  are witnesses of segment  $\overline{v_a v_b}$  on opposite sides of the polygon. Since  $V_{vo}(P)$  is constructed in a consistent (clockwise) order witness  $c$  will see  $a$  first then  $b$ , and witness  $d$  will see  $b$  first then  $a$ . This is condition 3.

Since no other possibilities exist, an edge in  $V_{\text{vis}}(P)$  that represents a visibility only segment implies that one of conditions 1, 2, or 3 are true.

( $\Leftarrow$ ) Now, assume at least one of the three conditions 1, 2, or 3 is met.

- Case 1: Assume condition 1 is met. Thus, one of the witnesses, say  $x$ , to edge,  $(a, b)$ , is a blind ear. On a blind ear, each of the edges attached to  $x$  represents a boundary segment of the polygon. If  $c$  also represents a boundary segment the polygon would be closed after only 3 vertices, violating one of the premises of the lemma. Thus, if a witness to an edge is a blind ear, that edge represents a visibility only segment of the polygon.
- Case 2: Assume condition 2 is met. That means, at least one witness of the edge  $(a, b)$  sees another node,  $x$ , between  $a$  and  $b$ . Assume that  $(a, b)$  represents a boundary segment, called  $\overline{v_a v_b}$ . Vertex  $v_x$  that corresponds to node  $x$ , must be between  $v_a$  and  $v_b$  as seen by the witness. Thus  $v_x$  must either be in front of or behind segment  $\overline{v_a v_b}$ . The triangle formed by  $\overline{v_a v_b}$  and the witness is empty, so  $v_x$  is not in front of  $\overline{v_a v_b}$ . Then the vertex  $v_x$  must be placed behind  $\overline{v_a v_b}$ . By the definition of visibility, the witness will not see  $v_x$  since it is behind a boundary segment. Thus  $\overline{v_a v_b}$  cannot be a boundary segment; it must be a visibility only segment, and  $(a, b)$  must represent a visibility only segment.
- Case 3: Assume condition 3 is met. There exists one witness,  $x$ , of  $(a, b)$  that sees first  $a$  then  $b$  with no other nodes between them, and one witness,  $y$  that sees first  $b$  then  $a$  with no other nodes between them. Because the lists for each



node are constructed in clockwise order, the corresponding witnesses vertices  $v_x$  and  $v_y$  must be on opposite sides of  $\overline{v_a v_b}$ . (Recall that the left side of a visibility only segment was previously defined to be the chain of vertices from  $v_a$  to  $v_b$  and the right side to be the chain of vertices from  $v_b$  to  $v_a$ .) Boundary segments have only one side, thus  $(a, b)$  must represent a visibility only segment.

So, if at least one of the conditions 1, 2, or 3 are met, then edge  $(a, b)$  of the visibility graph represents a visibility only segment.  $\square$

Equivalently, an edge represents a boundary segment of the polygon if and only if none of the conditions 1, 2, or 3 of lemma 3 are satisfied in the ordered visibility graph.

For a single node  $a$  in an ordered visibility graph, finding an adjacent edge that represents a boundary segment of the polygon would require searching through the lists of nodes adjacent to  $a$  for existence of any of the three conditions of lemma 3. Depending on the degree of  $a$  this could involve searching all of the edges,  $E$ , of the graph, or  $O(E)$  searches.

It is assumed that the ordered visibility graph,  $V_{vo}(P)$ , is stored as  $n$  adjacency lists, one for each vertex, and that the lists are circular, that is, the end of the list has a link to the beginning. The order of the nodes in each list corresponds to the clockwise order of the corresponding vertices around the given vertex. For every node, exactly two of the edges in its list represent boundary segments of the polygon. Due to the ordering of  $V_{vo}(P)$ , those two edges are adjacent in the list.

### 3.1.1 Algorithm: Determine Hamiltonian Cycle

The algorithm to determine the Hamiltonian cycle from the ordered visibility graph begins by finding one edge of  $V_{eo}(P)$  that definitely represents a boundary segment of the polygon, then goes through each adjacency list of  $V_{eo}(P)$ , finding the neighbouring node that also represents a boundary segment. The algorithm is described briefly below:

1. Choose any starting node, call it  $p$ . Traverse  $p$ 's list, counting the number of visible nodes.
2. If  $p$  has only two visibilities,  $a$  and  $b$ , it is a blind car. Adjacent edges  $(p, a)$  and  $(p, b)$  both represent boundary segments of the polygon,  $\overline{p_a p_a}$  and  $\overline{p_b p_b}$ . Choose one of them, say  $(p, a)$ , and determine which direction  $(p, a)$  or  $(a, p)$  is in the same direction as the ordered visibility graph was constructed. (Examine  $b$ 's ordered visibility list and choose the order of  $p$  and  $a$  there. This is possible since the degree of  $b$  is greater than or equal to three.)
3. If  $p$  has  $k$  visibilities ( $2 < k \leq n$ ), find an edge that represents a boundary segment of the polygon that is adjacent to node  $p$ . This is done by searching the visibility list of  $p$  for a node  $a$  such that  $(p, a)$  does not meet either conditions 2 or 3 of lemma 3. Edge  $(p, a)$  would represent a boundary segment of the polygon.
4. Starting at the boundary segment found above, walk through the ordered visibility lists, building the Hamiltonian cycle along the way, until returning to the

|               |   |   |   |   |   |   |   |     |  |
|---------------|---|---|---|---|---|---|---|-----|--|
|               |   | 1 | 2 | 3 |   |   |   | $n$ |  |
| $p\_adj\_row$ | 1 |   |   |   |   |   |   |     | 1 if node is adjacent to $p$ , 0 otherwise |
|               | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | directional indicator                      |

Figure 3.4:  $p$ 's Adjacency Row

starting node  $p$ .

In order to analyse the above algorithm, more specific implementation details are required.

1. Traversing the chosen node  $p$ 's list will require  $O(n)$  time.
2. It only requires constant time to check if the number of visibilities in  $p$ 's list is two. If  $p$  is determined to be a blind ear, one of  $p$ 's adjacent edges, say  $(p, a)$  must be ordered consistently with  $V_{vo}(P)$ 's construction. (Assume  $a$  and  $b$  are the two nodes adjacent to  $p$ .) The process of ordering of an edge will require traversing  $b$ 's list until  $p$  and  $a$  are found, and using this order. Since  $b$ 's list could have up to  $n$  vertices in it, and  $p$  and  $a$  could appear at the end, this step of the algorithm requires  $O(n)$  time.

3. This step requires several substeps for efficient completion:

- Create a  $2 \times n$  matrix. Call this structure,  $p\_adj\_row$ , for  $p$ 's adjacency row.

See figure 3.4. In the first row, place a 1 in any location that is indexed by a node that is in  $p$ 's visibility list, and a 0 in any other location. This is analogous to one row of an adjacency matrix. Fill the second row with 0's. This

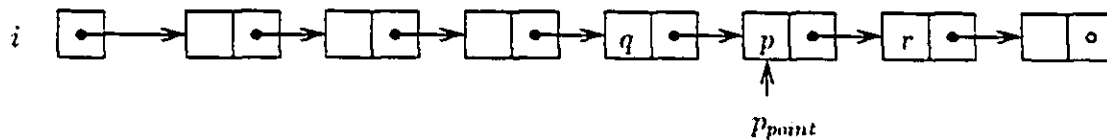


Figure 3.5: Node  $i$ 's List and Pointer to  $p$

second row is used as an ordering indicator. For node  $i$ ,  $p\_adj\_row[i, 2] = 1$  implies that the order  $(p, i)$  was indicated;  $p\_adj\_row[i, 2] = -1$  implies that the order  $(i, p)$  was indicated; and  $p\_adj\_row[i, 2] = 0$  implies that no information on the ordering of the edge has been indicated. The creation of this list requires  $O(n)$  time.

- Find the node  $p$ , in each of the, at most  $n$ , lists that are visible to  $p$ , and place a pointer,  $p\_point$  there in each list. See figure 3.5. This could require searching all the edges in all the lists, which needs  $O(E)$  time.
- Searching for condition 3 of lemma 3, go through the  $O(n)$  pointers:
  - Let  $q$  be the node before the pointer ( $p\_point$ ). Check if  $p\_adj\_row[q, 1] = 1$ . If so, check  $p\_adj\_row[q, 2]$ . If this entry is 0, set it to  $-1$  to indicate the ordering  $(q, p)$ . If this entry is  $-1$  do nothing. If this entry is 1, then condition 3 of lemma 3 has been met, and  $(q, p)$  must be removed as a potential boundary segment. That is, set  $p\_adj\_row[q, 1] = 0$ .
  - Let  $r$  be the node after the pointer ( $p\_point$ ). Check if  $p\_adj\_row[r, 1] = 1$ . If so, check  $p\_adj\_row[r, 2]$ . If this entry is 0, set it to 1 to indicate the ordering  $(p, r)$ . If this entry is 1 do nothing. If this entry is  $-1$ , then condition 3 of lemma 3 has been met, and  $(p, r)$  must be removed

as a potential boundary segment. That is, set  $p\_adj\_row[r, 1] = 0$ .

Thus  $O(n) \times O(1) = O(n)$  time is required.

- Go through the, at most  $n$ , lists of vertices that are visible to  $p$ , searching for condition 2 of lemma 3:
  - For node  $:=$  the node after  $r$  to the node before  $q$ ,
    - check if  $p\_adj\_row[node, 1] = 1$ . If it is, reset it to 0.

This could require checking  $O(E)$  edges, each needing  $O(1)$  time for a total of  $O(E)$  time.

- What remains in  $p\_adj\_row$ 's first row now is only two 1's. Both nodes, together with  $p$ , make edges of the visibility graph that represent boundary segments of the polygon. Go through  $p\_adj\_row$ 's first row until the first 1 is found, create an edge with this node and  $p$ , in the order indicated in the second row. Since the two 1's may be at the very end of  $p\_adj\_row$ ,  $O(n)$  time is needed.

The sum of the substeps, indicates that  $O(E)$  time is needed to check for conditions 2 and 3 of lemma 3.

4. The final walk through that creates the Hamiltonian cycle is accomplished as follows: Put the two nodes found above into the partial Hamiltonian cycle in the indicated order. While the cycle is less than  $n$  in length: search through the adjacency list of the last node in the partial cycle until the second last node of the partial cycle is found, and append its successor onto the partial cycle.

Since each vertex is ultimately in the Hamiltonian cycle, all  $n$  edge lists will be examined: in the worst case, all edges will be considered, resulting in  $O(E)$  time.

So, the overall complexity of the algorithm is  $O(n) + O(n) + O(E) + O(E)$ , which is  $O(E)$ .

### 3.2 Hamiltonian Cycle Specified

**INPUT:**

- $V_v(P)$  : The Unordered Vertex Visibility graph of a simple polygon,  $P$ .
- The Hamiltonian cycle of  $V_v(P)$  that corresponds to the boundary of the polygon,  $P$ .

**OUTPUT:**

- $V_{vo}(P)$  : The Ordered Vertex Visibility graph of the polygon,  $P$ .

The routine to convert from a Hamiltonian cycle and an unordered visibility graph,  $V_v(P)$ , to an ordered visibility graph,  $V_{vo}(P)$ , simply needs to sort each of the  $n$  lists into Hamiltonian cycle order. The lists could each be of length  $n - 1$ , so this operation might appear to require  $n \times O(n \log n) = O(n^2 \log n)$  time, to sort each of the  $n$  lists. However, in each list the entries are unique integers in the range  $1, 2, \dots, n$ . If the length of each list is  $l_i$ , a linear time sort like bucket and counting sort<sup>1</sup> can be used to sort each list in  $O(l_i)$  time, and all lists could be sorted in  $O(E)$  time since  $\sum_{i=1}^n l_i = O(E)$ .

The conversion algorithm is:

---

<sup>1</sup>These are described chapter 9 of [CLR90].

1. Assuming the Hamiltonian cycle is not in numerical (or alphabetic) order use a counting sort to create another array that is the inverse of the permutation of  $1, 2, \dots, n$  that represents the Hamiltonian cycle.
2. For each of the unordered lists of the visibility graph use a bucket sort to put the nodes into Hamiltonian cycle order. Here the inverse array will be used to convert the actual node (or vertex) number into the appropriate location of the Hamiltonian cycle.

All the elements of the Hamiltonian cycle will be unique integers in the range  $1, 2, \dots, n$ , so counting sort can be used here, and step 1 of the algorithm uses  $O(n)$  time. The bucket sort assumes that the input is well distributed over the range of buckets. If the length of each list is  $l_i$ , the bucket size for that list is set to be  $n/l_i$ . The fact that the entries in a list are unique numbers in the range  $1, 2, \dots, n$  ensures the entries in each list are distributed into various buckets. Thus all  $l_i$  bucket sorts will be completed in  $O(l_1 + l_2 + \dots + l_n) = O(E)$  time. Therefore the entire ordered visibility graph can be computed in  $O(E)$  time, and furthermore, the space used is also  $O(E)$ .

### 3.3 Summary

This chapter presented an algorithm that converts the ordered visibility graph of a simple polygon to its Hamiltonian cycle in  $O(E)$  time, where  $E$  is the number of edges of the given visibility graph. A second algorithm in the chapter converted the Hamiltonian cycle and unordered visibility graph of a simple polygon to the corresponding ordered visibility graph in  $O(E)$  time. Both algorithms are optimal.

These two results are interesting primarily because they link the results of two previously unrelated subareas in the visibility literature: line segments and polygons. For the reconstruction of a set of line segments,  $S$ , from the endpoint visibility graph,  $V_{eo}(S)$  is frequently supplied, whereas for the reconstruction of a polygon,  $P$ , from its vertex visibility graph, it is generally  $V_v(P)$  and the Hamiltonian cycle that is considered. Neither of these two reconstruction problems has been completely resolved in the general case.

In the next chapter, the reconstruction of an orthogonal polygon is considered, using line of sight visibility in the direction of each of the sides of the polygon.



## Chapter 4

# Orthogonal Polygon Reconstruction

An **orthogonal polygon** is a polygon that has an internal angle of  $\Pi/2$  (*CONVEX*) or  $3\Pi/2$  (*REFLEX*) radians at all corners. In this chapter, it is assumed that the orthogonal polygon is simple, has more than four vertices, and has no collinear sides. In section 4.6, the collinear sides assumption is discussed. A **stab** of a vertex of a simple polygon is an indication of the next side of the polygon seen by the vertex in the direction of the side. Both interior and exterior stabs of the polygon are specified. Every vertex of an orthogonal polygon has two stabs, one in the horizontal direction, and one in the vertical direction. If there is no side that is stabbed in the indicated direction, the stab is said to be a **stab to infinity** (denoted as  $\infty$ ). The **Hamiltonian cycle** of a polygon is a list of its vertices, in the order they appear around the polygon.

This chapter contains two algorithms; the second one relies on information provided by the first. The input to the algorithms is the set of stabs and the Hamiltonian cycle of an unknown orthogonal polygon. The first algorithm determines whether each vertex is forced by the stab information to be *CONVEX* or *REFLEX*. This is equiva-

lent to determining whether the stab is through the interior or exterior of the polygon, since stabs from convex vertices are on the exterior of the polygon and from reflex vertices are interior to the polygon. The second algorithm reconstructs an orthogonal polygon that is consistent with the input stabs and Hamiltonian cycle. (Actually, the reconstructed polygon is just one of a family of polygons that satisfies the input information.) Define the **Orthogonal Polygon Reconstruction** (OPR) problem to be the reconstruction of an orthogonal polygon given only its Hamiltonian cycle and stabs. Figure 4.1 is an example of the input information required by the OPR problem.

| Side | Orientation | Stab     |          |
|------|-------------|----------|----------|
| ab   | horizontal  | $\infty$ | $\infty$ |
| bc   | vertical    | $\infty$ | ef       |
| cd   | horizontal  | fg       | $\infty$ |
| de   | vertical    | $\infty$ | $\infty$ |
| ef   | horizontal  | $\infty$ | $\infty$ |
| fg   | vertical    | $\infty$ | ab       |
| gh   | horizontal  | bc       | ja       |
| hi   | vertical    | ab       | $\infty$ |
| ij   | horizontal  | fg       | $\infty$ |
| ja   | vertical    | $\infty$ | $\infty$ |

Figure 4.1: Example Input for the OPR Problem

The remainder of this chapter is organized in six sections. Section 4.1 defines and characterizes horizontal rectangles that are delimited by the horizontal stabs and the sides of the polygon. These rectangles are instrumental in determining the convexity of the vertices of the orthogonal polygon. Section 4.2 describes how to identify each of the horizontal rectangles. The first algorithm, determining whether each vertex is *CONVEX* or *REFLEX*, is presented in section 4.3. The algorithm to reconstruct the

orthogonal polygon is in section 4.4. Section 4.5 compares this OPR result to other similar results in the literature, and section 4.6 presents an OPR routine that allows the reconstructed polygon to have collinear sides.

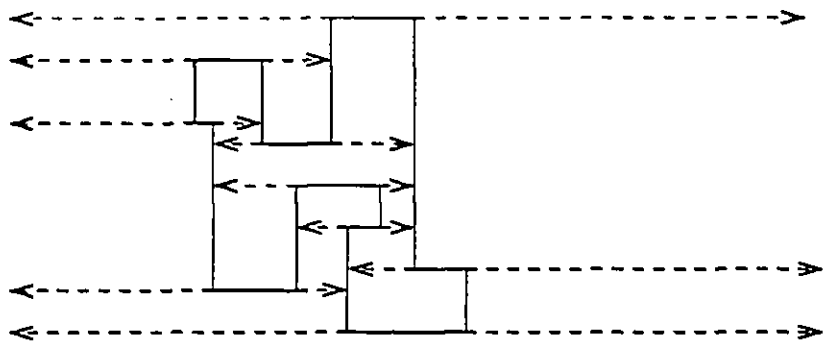


Figure 4.2: An Orthogonal Polygon with Horizontal Stabs

### 4.1 Horizontal Rectangles

The first algorithm of this chapter determines whether each vertex is *CONVEX* or *REFLEX*. In order to do this, the plane containing the polygon to be reconstructed is partitioned into rectangles, and those rectangles are classified. It is from this classification that the convexities of vertices are established.

Figure 4.2 is an example of an orthogonal polygon with all horizontal stabs drawn in. Notice that the plane is divided into a collection of different rectangles. It is assumed that those rectangles with stabs to infinity are completed by a pseudo side at infinity. Depending on how the stabs hit the sides of the polygon, twelve different types of rectangles, as enumerated in figure 4.3, are possible, ignoring horizontal and vertically symmetric situations.

Notice that every stab is part of exactly two rectangles, one above and one below

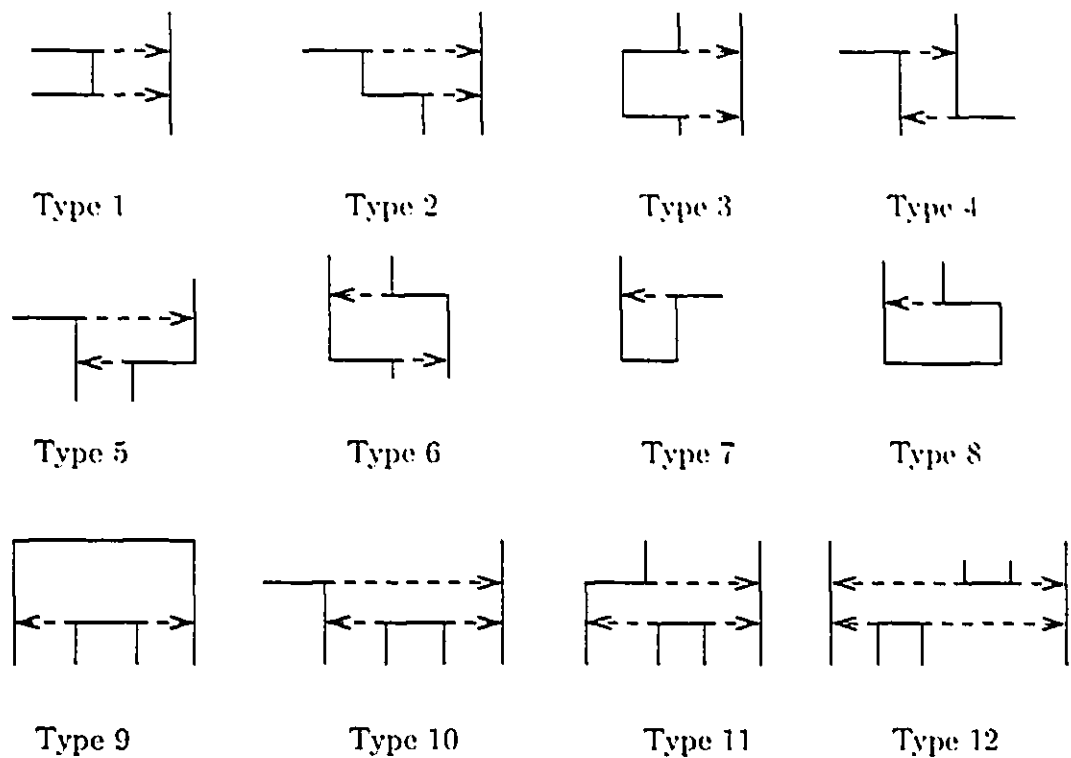


Figure 4.3: The Twelve Possible Horizontal Rectangles

it. For the previously stated assumption of no collinear sides, there exists a unique top most side and a unique bottom most side, each of which partially bound degenerate rectangles as shown in figure 4.4. These will be called type 0 rectangles and are easily identifiable.



Figure 4.4: The Two Type 0 Rectangles

**Lemma 4** *Aside from the two type 0 rectangles, rectangles of types 1 through 12 are the only possible rectangles created from the sides of an orthogonal polygon and its*

*horizontal slabs.*

Proof: Every rectangle has exactly four corners. It is possible that zero to four of those corners correspond to vertices of the polygon.

- Case 0: If *zero* corners of a rectangle correspond to vertices of the polygon, there is  $\binom{4}{0} = 1$  possible rectangle. The only rectangle that has no polygon vertices on its corners is type 12.
- Case 1: If *one* corner of a rectangle corresponds to a vertex of the polygon, there are  $\binom{4}{1} = 4$  possible locations for that correspondence. At that corner, the polygon could turn toward the rectangle, or away from it, as in figure 4.5 thus giving  $4 \times 2 = 8$  rectangles. Ignoring the four horizontal and vertically



Figure 4.5: Turning Toward and Away from Rectangle

symmetric situations leaves only two different rectangles. The two rectangles with one corner corresponding to vertices of the polygon are types 10 and 11.

- Case 2: If *two* corners of a rectangle correspond to vertices of the polygon, there are  $\binom{4}{2} = 6$  possible locations for those correspondences, as shown in figure 4.6.

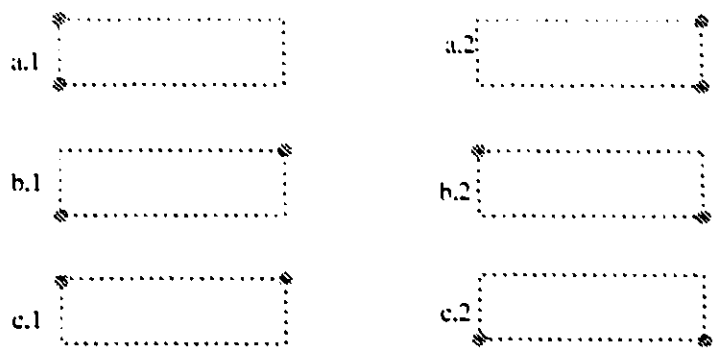


Figure 4.6: Two Vertices Correspond to Rectangle Corners

Notice that locations  $a.1$  and  $a.2$  differ by a vertical flip, and thus will be considered as equivalent. Locations  $b.1$  and  $b.2$  and locations  $c.1$  and  $c.2$  differ from their partners by a horizontal flip, and thus will each be considered equivalent.

- In case  $a$ , the vertices can both turn away from the rectangle, one turn toward the rectangle and one turn away from the rectangle, or both turn away from the rectangle. This gives three possible configurations, that correspond to rectangles of types 1, 2, and 3.
- In case  $b$ , the polygon vertices can turn in the same three directions as in case  $a$ . These configurations are rectangles of types 4, 5, and 6.
- In case  $c$ , the vertices can only turn toward the rectangle and be consecutive on the Hamiltonian cycle. If either or both turned away, a stab would be created that defines part of the rectangle, and the vertex would then not correspond to a corner of the rectangle. If both turned away from the rectangle and were not consecutive on the Hamiltonian cycle, collinear sides would be created. The one rectangle obtained here is a type 9 rectangle.

Thus, there are seven possible rectangles that have two corners that correspond to vertices of the polygon.

- Case 3: If *three* corners of a rectangle correspond to vertices of the polygon, there are  $\binom{4}{3} = 4$  possible locations for those correspondences, all of which are symmetrically related, leaving only one possibility.
  - The two of the three vertices cannot turn away from the rectangle, since this would result in an orthogonal polygon with collinear sides, violating one of the assumptions of this chapter.
  - The three vertices could have one turning away from the rectangle, and two turning toward it. However, in order to avoid collinear sides, the three vertices must be consecutive on the Hamiltonian cycle of the polygon. This is a type 7 rectangle.
  - The three vertices could all turn toward the rectangle. However, in order to avoid collinear sides, the three vertices must then be consecutive on the Hamiltonian cycle of the polygon. This is a type 8 rectangle.

Thus, there are two possible rectangles that have three corners that correspond to vertices of the polygon.

- Case 4: If the *four* corners of the rectangle correspond to the vertices of a polygon, there is  $\binom{4}{4} = 1$  possible location for this correspondence. However, this case is not possible under the given assumptions of: a simple polygon with more than four vertices and no collinear sides.

Therefore, there are exactly twelve possible configurations of rectangles created from the sides and horizontal stabs of an orthogonal polygon, and they are the types 1 through 12 rectangles as defined previously.  $\square$

**Lemma 5** *Every vertex of the polygon is part of exactly three horizontal rectangles.*

Proof: Refer to figure 4.7. There is one horizontal rectangle above and one below

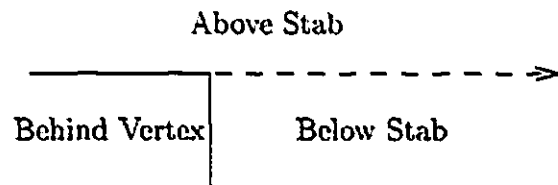


Figure 4.7: A Vertex is Part of 3 Rectangles

every horizontal stab, and one rectangle behind the vertex.  $\square$

**Corollary 6** *Around each vertex, there is either:*

- *one horizontal rectangle above and two below it, or*
- *two horizontal rectangles above and one below it.*

Proof: Since these are the horizontal rectangles created by the horizontal stabs, these are the only possible configurations. See figure 4.8.  $\square$





Figure 4.8: Two Orientations of Rectangles Around a Vertex

## 4.2 Identification of Rectangles

This section uses the characterizations of the previous section to identify the type of each horizontal rectangle. The identification of all rectangles of type 0 to type 11 will be shown to be an  $O(n)$  step, but identification of type 12 rectangles require  $O(n \log n)$  time. This typing will be used in the *CONVEX/REFLEX* algorithm in section 4.3.

Starting at one vertex of the polygon, and traversing through the Hamiltonian cycle, the algorithm determines the three types of rectangles around each vertex. Define the value returned by the function  $stab[v]$  to be the side that is stabbed by vertex  $v$  along a *horizontal* stab. Also, define  $stab[v].ver$  to be one of the vertices that are on the vertical side that is stabbed by vertex  $v$ . (Which of the two vertices depends on the orientation of the rectangle and the layout order, clockwise or counter clockwise, of the Hamiltonian cycle and is left as an implementation detail.) The boolean function  $IsHoriz(\overline{v_i v_{i+1}})$  determines whether the given side is horizontal or not. If  $i$  is the current vertex on the Hamiltonian cycle, then  $i + 1$  and  $i - 1$  (modulo  $n$  arithmetic) respectively refer to the next and previous vertices on the Hamiltonian

cycle. Refer to figure 4.3 while reading the descriptions.

Identifying the types 0 through 11 rectangles can be achieved by testing the following conditions for a vertex  $i$ :

- Type 0:  $stab[i] = \infty$  and  $stab[i + 1] = \infty$  AND  $IsHoriz(\overline{r_i r_{i+1}})$
- Type 1:  $stab[i] = stab[i + 1]$  AND NOT  $IsHoriz(\overline{r_i r_{i+1}})$
- Type 2:  $stab[i] = stab[i + 2]$  AND NOT (a type 1 rectangle)
- Type 3:  $stab[i] = stab[i + 3]$  AND NOT (a type 1 or type 2 rectangle)
- Type 4:  $stab[stab[i].ver].ver = i$
- Type 5:  $stab[stab[i].ver + 1].ver = i$
- Type 6:  $stab[stab[i].ver + 1].ver - 1 = i$
- Type 7:  $stab[i].ver = i + 2$
- Type 8:  $stab[i].ver = i + 3$
- Type 9:  $stab[i].ver + 1 = stab[i - 1].ver$  AND  $IsHoriz(\overline{v_i v_{i+1}})$
- Type 10:  $stab[stab[i].ver] = stab[i - 1]$  AND  $IsHoriz(\overline{v_i v_{i-1}})$  AND NOT (a type 4 rectangle)
- Type 11:  $stab[stab[i].ver + 1] = stab[i - 1]$  AND  $IsHoriz(\overline{v_i v_{i-1}})$  AND NOT (a type 3 or type 5 rectangle)

For a given vertex  $i$ , detecting the types 0 through 11 rectangles incident upon  $i$  requires  $O(1)$  time.

- Type 12: A type 12 rectangle is detectable when two pairs of vertices have common stabs. (That is,  $stab[i] = stab[j + 1]$  and  $stab[i + 1] = stab[j]$ , assuming  $IsHoriz(\overline{v_i v_{i+1}})$  and  $IsHoriz(\overline{v_j v_{j+1}})$ ). Detecting this type of rectangle will require examining all horizontal stabs to each vertical side. Since it is possible that  $O(n)$  stabs could hit one side (see figure 4.12, for example), it might appear that this operation could take  $O(n^2)$  time. However, in section 4.3, a data

structure is presented that reduces the overall time needed to identify all type 12 rectangles to  $O(n \log n)$  time in total.

### 4.3 Algorithm - Determine *CONVEX/REFLEX*

**INPUT:**

- The stabs of the horizontal and vertical sides of an unknown simple orthogonal polygon,  $P$ .
- The Hamiltonian cycle that corresponds to the boundary of the polygon,  $P$ .

**OUTPUT:**

- The convexity (*CONVEX/REFLEX*) of each vertex of the polygon,  $P$ .

The following algorithm determines the convexity of the vertices of an orthogonal polygon given the Hamiltonian cycle and the stab information. Traversing the Hamiltonian cycle of the polygon, all sides are assigned to be either horizontal or vertical:  $h_1, v_1, h_2, v_2, \dots, h_{n/2}, v_{n/2}$ . First, the algorithm will identify the rectangles adjacent to each horizontal stab. Each rectangle contains two, three, or four vertices of the polygon, and the convexity properties of the involved vertices are not independent. For each vertex,  $v$ , of the polygon, the algorithm maintains two sets, *same*[ $v$ ] and *opposite*[ $v$ ]. Ultimately, all the vertices on a rectangle containing  $v$  will be included in either *same*[ $v$ ] or *opposite*[ $v$ ]. These two sets indicate whether those vertices have the same or opposite convexity as  $v$ . For example, figure 4.9 shows a type 6 rectangle, and the corresponding *same* and *opposite* sets associated with each polygon vertex of the rectangle. Note that at this stage it has not yet been established whether the rectangle is in the interior or exterior of the polygon. In the final step, using all these

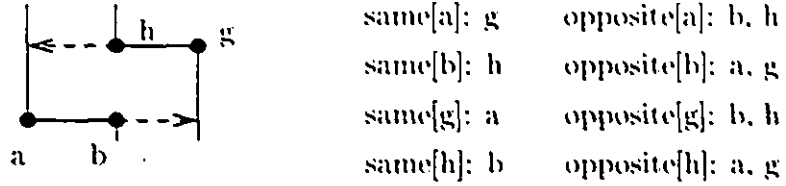


Figure 4.9: *same* and *opposite* Sets Corresponding to a Type 6 Rectangle

sets, the algorithm will assign the label *CONVEX* or *REFLEX* to each vertex.

The algorithm is described in three parts: classifying types 0 through 11 rectangles and identifying the vertices on each, identifying vertices on type 12 rectangles, and finally determining the convexity of each of the vertices.

#### 4.3.1 Classify and Identify Rectangles: Types 0 to 11

This part of the algorithm walks through the Hamiltonian cycle of the polygon, checking each horizontal stab for inclusion as part of any type 0 through 11 rectangles. For each vertex,  $v$ , append the other vertices on the same rectangle to either its *same*[ $v$ ] or *opposite*[ $v$ ] set, and count the number of rectangles to which it has been assigned.

- *Initialize*: For each vertex  $v$ , in Hamiltonian cycle order do:
  1.  $number\_of\_rectangles[v] := 0$ .
  2. initialize *same*[ $v$ ] to the empty set.
  3. initialize *opposite*[ $v$ ] to the empty set.
- *Classify/Identify*: For each vertex,  $v$ , in Hamiltonian cycle order do:
  - if conditions 0 to 11 of section 4.2 are satisfied with vertex  $v$ :
    - \* For every pair of vertices,  $j$  and  $k$ , on the rectangle
      1. increment  $number\_of\_rectangles[j]$
      2. either  $INSERT(j, same[k])$  or  $INSERT(j, opposite[k])$  appropriately<sup>1</sup>

---

<sup>1</sup>This is easily determined from figure 4.3

Analysis: The *initialize* loop uses  $O(n)$  time since each of the operations inside the loop use constant time, and the loop is executed  $n$  times. In the *classify/identify* loop, checking each of conditions 0 through 11 requires constant time (as shown in section 4.2). Since every vertex is part of exactly three horizontal rectangles, each of which contains from two to four vertices of the polygon, the *same* and *opposite* sets for each vertex will together contain no more than twelve vertices, a constant number. Inserting a constant number of vertices into constant length sets is an  $O(1)$  time operation, as is incrementing a variable. So the *classify/identify* loop also uses  $O(n)$  time, and the overall analysis of this part of the algorithm is  $O(n)$ . Also, the space used by the above routines is bounded by  $O(n)$ .

Now, label any vertex that has been assigned to three rectangles as *classified* and the rest as *unclassified*. The next section will use this *classified/unclassified* labelling to identify the type 12 rectangles.

### 4.3.2 Identify Rectangles: Type 12

A type 12 rectangle could be on either the inside or the outside of the polygon. Any vertex that is now unclassified must be part of some type 12 rectangle. The difficulty

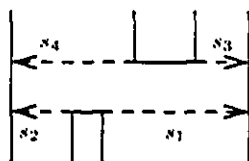


Figure 4.10: A Type 12 Rectangle

is identifying which other stabs are also part of this same rectangle; refer to figure

4.10. The stab  $s_2$  that is on the other end of the horizontal side from  $s_1$  can be identified in constant time, simply by looking at the stab of the next vertex on the Hamiltonian cycle. The two stabs  $s_3$  and  $s_4$  of the same rectangle are more difficult to find.

The necessity of identifying type 12 rectangles is shown by the polygon of figure 4.11. If type 12 rectangles are not considered, the marked vertices would not appear in the *same* or *opposite* sets of any of the other vertices. The marked vertices and more, would be isolated if we examined the vertical, instead of the horizontal rectangles.

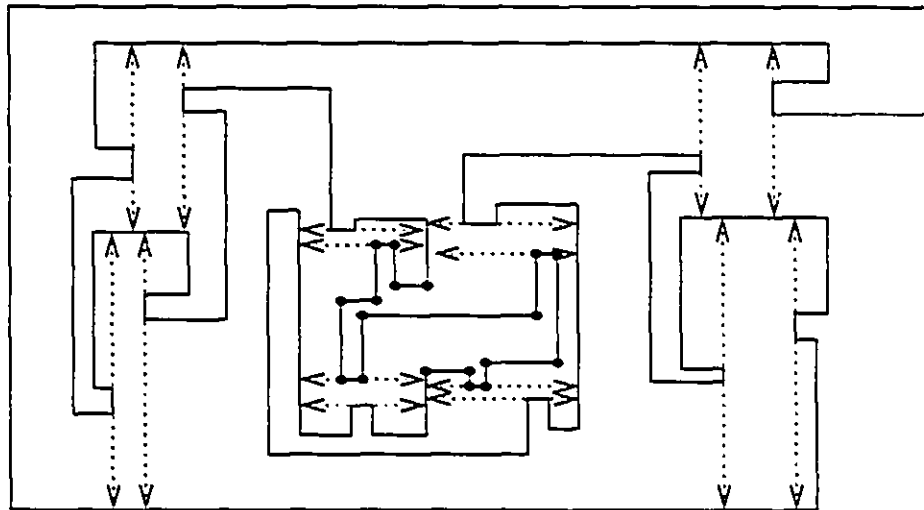


Figure 4.11: A Polygon with Vertices Isolated by Type 12 Rectangles

Even though the total number of all types of rectangles created by a polygon's horizontal stabs is  $O(n)$ , there could be  $O(n)$  type 12 rectangles. A natural conjecture would be that each of the  $O(n)$  vertical sides stabbed by type 12 rectangles, only have a constant number of such stabs. Vertical side,  $s$ , on the polygon of figure 4.12 shows that this conjecture is incorrect and a more elaborate procedure is required.



- initialize *binary\_tree*[*s*] to empty.
- *Count stabs*: For each vertex, *v*, in Hamiltonian cycle order do:
  - If (*number\_of\_rectangles*[*v*] = 2 ) increment *unclassified\_count*[*stab*[*v*]].
- *Create Trees*: For each vertex, *v*, in Hamiltonian cycle order do:
  - if (*number\_of\_rectangles*[*v*] = 2) AND (*number\_of\_rectangles*[*v* + 1] > 3)
    - \* if (*unclassified\_count*[*stab*[*v*]] < *unclassified\_count*[*stab*[*v* + 1]])
    - /\* vertical side *stab*[*v*] is the least stabbed of the two sides \*/
    - if (*MEMBER*(*stab*[*v* + 1], *binary\_tree*[*stab*[*v*]]))
    - /\* a type 12 rectangle has been found. \*/
    - For every pair of vertices, *j* and *k*, on the rectangle:
    - INSERT*(*j*, *same*[*k*])
    - *DELETE*(*stab*[*v* + 1], *binary\_tree*[*v*])
    - else *INSERT*(*stab*[*v* + 1], *binary\_tree*[*stab*[*v*]]).
    - \* else /\* vertical side *stab*[*v*] is NOT the least stabbed of the two sides \*/
    - if (*MEMBER*(*stab*[*v*], *binary\_tree*[*stab*[*v* + 1]]))
    - /\* a type 12 rectangle has been found. \*/
    - For every pair of vertices, *j* and *k*, on the rectangle:
    - INSERT*(*j*, *same*[*k*])
    - *DELETE*(*stab*[*v*], *binary\_tree*[*v* + 1])
    - else *INSERT*(*stab*[*v*], *binary\_tree*[*stab*[*v* + 1]]).

Analysis: The *initialize* and *count stabs* loops are each  $O(n)$  loops. The *create trees* loop is an  $O(n \log n)$  loop, since it is executed  $n$  times, and each binary tree could have  $O(n)$  entries in it. (Searching, and inserting into a balanced binary tree of size  $O(n)$  requires  $O(\log n)$  time.) So, the overall time needed by this part of the algorithm is  $O(n \log n)$ . However the space required here is only  $O(n)$ , since the number of entries in all the binary trees never exceeds  $n$ .

### 4.3.3 Determine Convexity of Vertices

This stage starts with any vertex, *v*, that has a stab to infinity, marks it as *CONVEX*, and initializes a queue (called *to\_be\_done*) with this vertex. Then a loop is created



that dequeues a vertex,  $v$ , from the front of the queue, marks the vertices in  $same[v]$  as the same convexity as  $v$ , and those in  $opposite[v]$  as opposite to  $v$ . For each of these vertices, if they were not previously marked, enqueue them to the back of the queue. The loop continues until the queue is empty.

- initialize  $to\_be\_done$  to be an *EMPTY* queue.
- *Initialize*: For each vertex,  $v$ , in Hamiltonian cycle order do:
  - $has\_been\_queued[v] := false$
  - if  $(stab[v] = \infty)$  and  $(to\_be\_done = EMPTY)$ 
    - \*  $ENQUEUE(v, to\_be\_done)$
    - \*  $has\_been\_queued[v] := true$
    - \*  $vertex[v] := CONVEX$
- *Determine Convexity*: While  $(to\_be\_done \neq EMPTY)$  do:
  - $i := DEQUEUE(to\_be\_done)$
  - for every vertex,  $j$ , in  $same[i]$  do:
    - \* if  $NOT(has\_been\_queued[j])$ 
      - $ENQUEUE(j, to\_be\_done)$
      - $has\_been\_queued[j] := true$
    - \* if  $(vertex[i] = CONVEX)$  then  $vertex[j] := CONVEX$
    - \* else  $vertex[j] := REFLEX$
  - for every vertex,  $j$ , in  $opposite[i]$  do:
    - \* if  $NOT(has\_been\_queued[j])$ 
      - $ENQUEUE(j, to\_be\_done)$
      - $has\_been\_queued[j] := true$
    - \* if  $(vertex[i] = CONVEX)$  then  $vertex[j] := REFLEX$
    - \* else  $vertex[j] := CONVEX$

Analysis and Correctness: The *initialize* loop clearly uses  $O(n)$  time. The correctness of the *determine convexity* loop requires the following definition and lemma.

Define an **isolated group of vertices** to be a *proper* subset,  $K$ , of all the vertices of an orthogonal polygon such that these vertices appear in each other's *same* and

*opposite* sets, but not in the *same* and *opposite* sets of any other vertices. Furthermore, the *same* and *opposite* sets of the vertices of  $K$  do not contain any vertices that are not in  $K$ .

**Lemma 7** *After the algorithms to classify and identify types 0 through 12 rectangles in subsections 4.3.1 and 4.3.2, an isolated group of vertices does not exist.*

Proof: Assume an isolated group of vertices does exist and that the original orthogonal polygon is  $P$ . The vertices must all be on rectangles that are defined by the vertices of the group. From lemma 5 we know that every vertex is part of exactly 3 horizontal rectangles, and from corollary 6 we know that at least one of those three must be above the vertex and one below it. So, every vertex will be part of the *same* or *opposite* set of the vertices on one rectangle above it and one below it. Every rectangle has at least one vertex on its top edge and at least one vertex on its bottom edge. Thus there is a strip extending from the top of the plane to the bottom, consisting of rectangles that were created from the isolated group of vertices. However, there must be some vertices, and thus some rectangles that are not part of the group since an isolated group is a *proper* subset of the total set of vertices. So there is another strip going from the top of the plane to the bottom consisting of rectangles that were created from vertices that are not part of the isolated group. These two strips must be mutually exclusive. Note that the horizontal segment with the highest y-coordinate delineates the upper type 0 rectangle, and that this rectangle extends completely across the plane, from right to left - similarly for the lower type 0 rectangle. So, the two strips of rectangles would have the upper and lower rectangles in

common. Thus the two strips cannot be mutually exclusive and therefore the isolated group of vertices cannot exist.  $\square$

Lemma 7 indicates that the *determine convexity* loop correctly identifies all the vertices as *CONVEX* or *REFLEX*, since there are no isolated vertices. The loop does not terminate until the queue is empty, which will only happen when all are marked. Each vertex is put onto the queue once, and pulled off once. Thus the *determine convexity* loop is executed exactly  $n$  times, with each iteration requiring constant time. Therefore, this entire stage of the algorithm uses  $O(n)$  time.

Thus, the time used to determine whether each vertex of an orthogonal polygon of  $n$  vertices is *CONVEX* or *REFLEX* is dominated by the  $O(n \log n)$  needed to identify the type 12 rectangles. The space requirement is only  $O(n)$ .

## 4.4 Algorithm - Reconstruct Polygon

**INPUT:**

- The stabs of the horizontal and vertical sides of an unspecified simple orthogonal polygon,  $P$ .
- The Hamiltonian cycle that corresponds to the boundary of the polygon,  $P$ .

**OUTPUT:**

- An orthogonal polygon,  $P$ , that abides by the input information.

In this section, an efficient algorithm is presented to reconstruct an orthogonal polygon from its stabs and Hamiltonian cycle, after the convexity of the vertices is established. This algorithm creates two lists, representing the relationships between the  $x$

and  $y$  coordinates of all vertices. One list  $\{x_{min}, \dots, x_{max}\}$  represents the  $x$  coordinates of each of the vertical sides, the other list  $\{y_{min}, \dots, y_{max}\}$  represents the  $y$  coordinates of the horizontal sides. These two lists will be created in such a way that when the sides are laid out on the  $x$  and  $y$  coordinates, the result will be an orthogonal polygon. The lists are not unique since it is not possible to determine the relationships between the stabs on opposite sides of any boundary segment. Placing all vertices on these coordinates, however, does reconstruct an orthogonal polygon that respects the given stabs and Hamiltonian cycle.

1. Run the convex/reflex algorithm of section 4.3.  $O(n \log n)$  time and  $O(n)$  space is needed to complete this task.
2. Find the four segments with both horizontal and vertical stabs to infinity. The two vertical ones must be located at  $x_{min}$  and  $x_{max}$ , while the two horizontal ones must be at  $y_{min}$  and  $y_{max}$ . Start with a horizontal extreme segment, assign it to  $y_{min}$ , then follow through the Hamiltonian cycle. Assign the other three segments to  $x_{min}$ ,  $y_{max}$  and  $x_{max}$ . These assignments will lay out the polygon so that its Hamiltonian cycle is in clockwise order. Completion of this step requires one pass through the cycle, doing a constant amount of work on each vertex. Therefore  $O(n)$  time is needed.
3. The segment that runs horizontally along  $y_{min}$  is laid out from right to left, since the Hamiltonian cycle is in clockwise order. Call this a *left* segment. The next segment on the Hamiltonian cycle, a vertical segment, must be an *up* segment, otherwise the first segment would not be at  $y_{min}$  and furthermore the corner

between the two must be a convex corner.

Recall that the sides of the polygon have been named  $h_1, v_1, h_2, v_2, \dots, h_{n/2}, v_{n/2}$  along the Hamiltonian cycle. For a horizontal (respectively vertical) segment, define its **predecessor segment** to be the horizontal (respectively vertical) segment immediately before it on the Hamiltonian cycle. ( $h_i$ 's predecessor is  $h_{i-1}$ , and  $v_j$ 's predecessor is  $v_{j-1}$ .) On any segment, vertical or horizontal, define its **two preceding vertices** to be the two vertices between  $h_i$  and  $h_{i-1}$  or between  $v_i$  and  $v_{i-1}$ . Figure 4.13 shows a horizontal segment and a vertical



Figure 4.13: Predecessor Segments and Preceding Vertices

segment and their respective predecessor segments. The arrows indicate the direction of the Hamiltonian cycle. In each case, vertices  $x$  and  $y$  are the preceding vertices to the segment.

For each of the remaining segments, in the cycle, if the preceding vertices have the same convexity, the segment must be opposite its predecessor segment, in the same dimension. If the preceding vertices have opposite convexity, the segment is the same as its predecessor segment, in the same dimension. In this way, assign *up/down*, *left/right* to each segment of the polygon.

Again, this step traverses the Hamiltonian cycle, examining a constant number of sides and vertices on each step. Thus a total of  $O(n)$  time and space is used.

4. Create two digraphs,  $X$  and  $Y$ , with a node in the  $X$  graph for each vertical side, and a node in the  $Y$  graph for each horizontal side. Add arcs as follows:
  - On the  $X$  graph, direct arcs from the node corresponding to the  $x_{min}$  side to every other node, and from all nodes to the node corresponding to the  $x_{max}$  side. This step adds  $(n - 2)$  arcs to the  $X$  digraph.
  - For every *right* segment in the polygon, put an arc in the  $X$  digraph from the node corresponding to the side containing the first endpoint to the node corresponding to the side containing the second endpoint.
  - For every *left* segment in the polygon, put an arc in the  $X$  digraph from the node corresponding to the side containing the second endpoint to the node corresponding to the side containing the first endpoint. This step and the immediately preceding one add a total of  $n/2$  arcs to the  $X$  digraph.
  - For every stab to a *right* segment put an arc from the node corresponding to the side containing the first endpoint of the stabbed segment, to the node corresponding to the side containing the endpoints of the stabbing segment, and another from the node corresponding to the side containing the endpoints of the stabbing segment to the node corresponding to the side containing the second endpoint of the stabbed segment.
  - For every stab to a *left* segment put an arc from the node corresponding to the side containing the second endpoint of the stabbed segment, to the

node corresponding to the side containing the endpoints of the stabbing segment, and another from the node corresponding to the side containing the endpoints of the stabbing segment to the node corresponding to the side containing the first endpoint of the stabbed segment. This step and the immediately preceding one add at most  $n$  arcs to the  $X$  digraph (stabs to  $\infty$  do not add arcs).

Thus, the  $X$  digraph contains less than  $5n/2 - 2 = O(n)$  arcs. The arcs for the  $Y$  digraph are created in a similar fashion, substituting *up* for *right* and *down* for *left*. The two digraphs represent partial orders for the  $x$  and  $y$  coordinates of the sides of the polygon. Figure 4.14 gives an example of the two digraphs.

Creating the two digraphs requires, two traversals of the Hamiltonian cycle doing constant work on each stop, and will use a total of  $O(n)$  time. Since the two digraphs together have  $n$  nodes and  $O(n)$  arcs,  $O(n)$  space is used.

5. Use a topological sort on each digraph to order the nodes from minimum to maximum. As described in *Graph Algorithms and NP Completeness* [Meh84], a topological sort of  $n$  vertices and  $e$  edges uses  $O(n + e)$  time. Each of the two digraphs have  $n/2$  vertices and  $O(n)$  edges, thus the two topological sorts will require only  $O(n)$  time.
6. Assign  $x$  and  $y$  integer track numbers to the nodes in the sorted order defined by the topological sorts. Each vertex will have an  $x$  track number and a  $y$  track number. In each case the track numbers are uniquely chosen from the range  $[1..n/2]$ , where  $n$  is the number of vertices in the polygon. Draw out the tracks

| Side | Orientation | Stab     |          | Convexity | Direction | Extreme   |
|------|-------------|----------|----------|-----------|-----------|-----------|
| ab   | horizontal  | $\infty$ | $\infty$ | Convex    | right     | $y_{max}$ |
| bc   | vertical    | $\infty$ | ef       | Convex    | down      |           |
| cd   | horizontal  | fg       | $\infty$ | Reflex    | right     |           |
| de   | vertical    | $\infty$ | $\infty$ | Convex    | down      | $x_{max}$ |
| ef   | horizontal  | $\infty$ | $\infty$ | Convex    | left      | $y_{min}$ |
| fg   | vertical    | $\infty$ | ab       | Convex    | up        |           |
| gh   | horizontal  | bc       | ja       | Reflex    | left      |           |
| hi   | vertical    | ab       | $\infty$ | Reflex    | down      |           |
| ij   | horizontal  | fg       | $\infty$ | Convex    | left      |           |
| ja   | vertical    | $\infty$ | $\infty$ | Convex    | up        | $x_{min}$ |

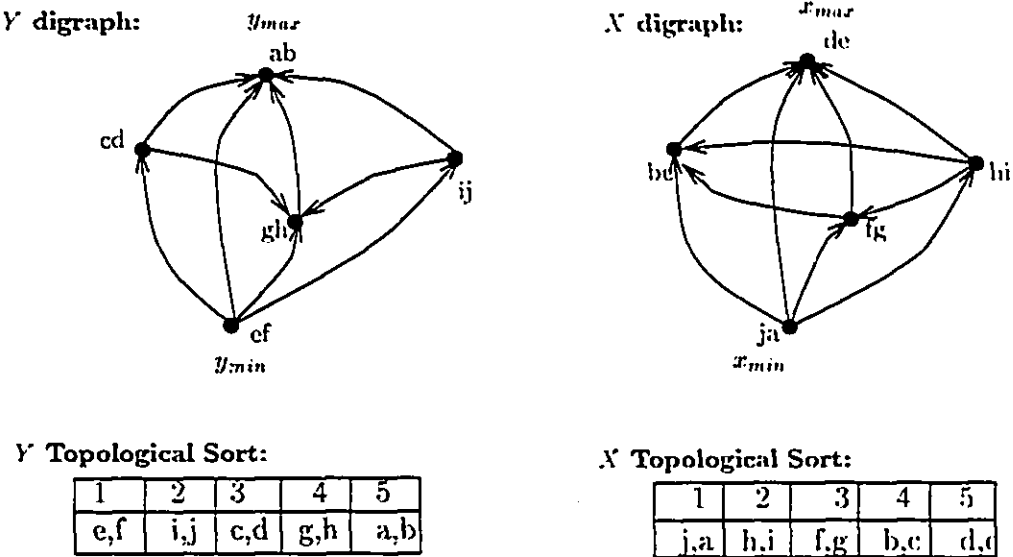


Figure 4.14: X and Y Digraphs

in each direction and follow through the Hamiltonian cycle laying each vertex on its respective track, putting a segment between each pair of consecutive vertices. The resulting orthogonal polygon respects the given Hamiltonian cycle and stabs and has no collinear sides. This step, also uses  $O(n)$  time. Figure 4.15 assigns integer track numbers and draws the polygon using the data of the example in figure 4.14.



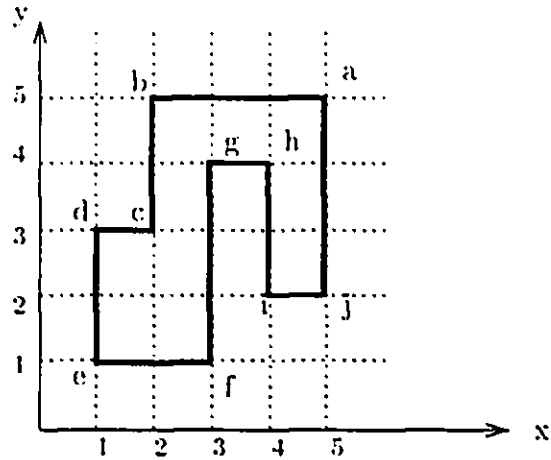


Figure 4.15: Reconstructed Polygon, from Figure 4.14

The first step of this algorithm uses  $O(n \log n)$  time and  $O(n)$  space, the rest use only  $O(n)$  time and space. Therefore, the overall time used to solve the Orthogonal Polygon Reconstruction problem, OPR, is  $O(n \log n)$  and the overall space is  $O(n)$ .

## 4.5 Related Results

The reconstruction result of this chapter is closely related to two others in the visibility literature: *Realization of Visibility Trees* by Booth and O'Rourke [O'R87], and *Bar Visibility Graphs* by Wismath [Wis85] (independently by Tamassia and Tollis [TT86]). Both of these results are discussed more fully in Chapter 7 of O'Rourke's classic text [O'R87]. A summary of the two results and a comparison of each to OPR follows.

### 4.5.1 Edge Visibility Trees

In the Booth and O'Rourke work, sides of an orthogonal polygon are represented by nodes in the visibility graph, and pairs of nodes are connected by edges if there is a horizontal or vertical line of sight between them that is *inside* the polygon. The

resulting visibility graph is disconnected, and is actually two trees, one representing horizontal and one representing vertical visibility. Booth and O'Rourke reconstruct an orthogonal polygon from two given labelled trees, *i.e.*, the nodes of the two trees are numbered (labelled)  $1, 2, \dots, n$  in the order that the corresponding sides appear on the polygon.

For every node in the horizontal tree with degree greater than one, an absolute ordering of the vertical sides in the visibility polygon about that node is made. The absolute orderings are then combined into one partial ordering of the  $x$  coordinates of the vertical sides. The same approach is applied to get a partial ordering of the  $y$  coordinates. Next, the two partial orderings are each assigned integer coordinates, from 0 to  $n/2$ , with two sides whose order is indistinguishable being assigned the same coordinate. Then each vertex is assigned the  $x$  and  $y$  coordinates from the two sides (horizontal and vertical) adjacent to it, and a polygon is drawn using these coordinates. Finally, the construction may require slight adjustment of some side lengths to avoid collision of same coordinate sides, and collinearities.

The two labelled trees of Booth and O'Rourke can be extracted from the stabs and Hamiltonian cycle that are input to the OPR problem, as follows:

1. Run the *CONVEX/REFLEX* algorithm of section 4.3.
2. Create a graph with  $n$  nodes, one for each side of the polygon. Label the nodes so they correspond to the Hamiltonian cycle, *i.e.*, the node representing the side between vertices 1 and 2 on the Hamiltonian cycle, is labelled 1, the node representing the side between vertices 2 and 3 on the Hamiltonian cycle, is

labelled 2, etc.

3. Create the vertical tree part of the visibility graph from the vertical stabs of *reflex* vertices. Refer to figure 4.16. First, join the nodes representing the hori-

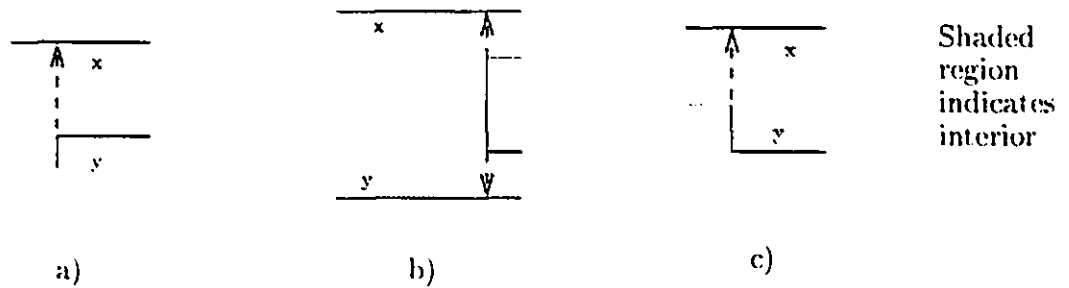


Figure 4.16: Join Sides  $x$  and  $y$  through the Polygon's Interior

zontal side attached to each reflex vertex to the node representing the horizontal side stabbed by that vertex's stab (diagram a). Next, whenever a vertical side has reflex corners on both ends, join the two nodes representing the horizontal sides stabbed by both vertical stabs (diagram b). Finally, whenever a vertical side has a reflex corner on one end and a convex corner on the other, join the node representing the horizontal side attached to the convex corner to the node representing the horizontal side stabbed by the vertical stab of the reflex corner (diagram c).

4. Create the horizontal tree from the horizontal stabs in a similar fashion.

Now the reconstruction can be completed using the  $O(n)$  algorithm designed by Booth and O'Rourke. The resulting orthogonal polygon agrees with the internal stabs, but may not be consistent with external stabs. The conversion from stabs and

Hamiltonian cycle to labelled trees uses  $O(n \log n)$  time and the Booth and O'Rourke algorithm uses  $O(n)$  time to reconstruct. So, the overall analysis of  $O(n \log n)$  is not improved, and the resulting polygon is less constrained than the OPR result.

It should be noted that the Booth and O'Rourke algorithm cannot be directly extended to polygons with both internal and external visibilities, since the horizontal and vertical graphs would each contain cycles, and thus the initial absolute orderings could not be made. Moreover, there is no obvious way of extending their result.

### 4.5.2 Bar Visibility Graphs

In [Wis85], a visibility graph represents a set of vertical line segments. (The segments are not connected into a polygon.) Every node in the graph represents a segment in the set, and two nodes are joined if the corresponding segments can see each other horizontally through a rectangle of non-zero height. Tamassia and Tollis[TT86] have labelled this as  $\varepsilon$  visibility. The restraint that disallows collinear sides of the orthogonal polygon problem is similar, but stronger than  $\varepsilon$  visibility. The *no collinear sides* model creates a more restricted class of graphs than  $\varepsilon$  visibility. This is due to the fact that three or more endpoints of consecutive line segments may line up on the same  $X$  (or  $Y$ ) coordinate, a condition that could never occur with the vertices of a non-collinear orthogonal polygon. From the stab information of the OPR problem, two  $\varepsilon$  visibility graphs can be extracted, one in each of the  $x$  and  $y$  dimensions.

1. Run the *CONVEX/REFLEX* algorithm of section 4.3.
2. Create the vertical visibility graph from the vertical stabs. Refer to figure 4.17.

First, join the node representing the horizontal side attached to each vertex to

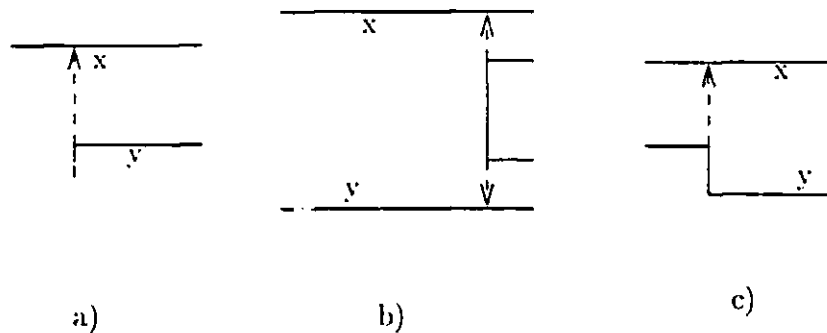


Figure 4.17: Join Sides  $x$  and  $y$

the node representing the horizontal side stabbed by that vertex's vertical stab, as in figure 4.17 a. Next, whenever a vertical side has similar convexity corners on both ends, join the two nodes representing the horizontal sides stabbed by both vertical stabs, as in figure 4.17 b. Finally, whenever a vertical side has opposite convexity corners, join the node representing the horizontal side attached to one end of the vertical side to the node representing the horizontal side stabbed by the vertex at the other end of the vertical side (diagram c).

3. Create the horizontal visibility graph from the horizontal stabs similarly.

In fact, these two graphs are similar to the visibility trees of the Booth and O'Rourke work, except that both *external* and internal visibility is considered. For the OPR result, it does not matter whether the stab is on the inside or the outside of the polygon.

Wismath reconstructs the set of line segments using an *st\** numbering of the nodes of the visibility graph. Before defining an *st\** numbering, a related concept, an *st* numbering, developed by Hopcroft and Tarjan [HT74] as an equivalent definition of

biconnectivity, must be considered.

An *st numbering* of a graph of  $n$  nodes is a one-to-one function  $\lambda$  that maps each node of the graph to a unique integer from  $\{1, 2, \dots, n\}$  in such a way that every node, except two, have an adjacent node with a number lower and an adjacent node with a number higher than its own. The two exceptions are special adjacent nodes, labelled as  $s$  and  $t$ , with  $\lambda(s) = 1$  and  $\lambda(t) = n$ . Vertex  $s$  has no adjacent node with a lower number and  $t$  has no adjacent node with a higher number.

Define  $\lambda_{min}$  to be a node that has no lower numbered adjacent node (only  $s$  in an *st* numbered graph), and  $\lambda_{max}$  to be a node that has no higher numbered adjacent node (only  $t$  in an *st* numbered graph). An *st\** numbering of a graph is a relaxation of the *st* numberings to allow more than one  $\lambda_{min}$  and more than one  $\lambda_{max}$ . A graph is said to be *st\** **numberable** if there is a one-to-one function,  $\lambda$ , that maps each node of the graph to a unique integer  $\{1 \dots n\}$ , and a planar embedding of the graph such that all  $\lambda_{max}$  and  $\lambda_{min}$  nodes are on the exterior face, and they are separable in such a way that all  $\lambda_{max}$  nodes can be connected to one new node and all  $\lambda_{min}$  nodes can be connected to another, and the resulting graph remains planar.

Wismath showed that a graph is representable by a set of vertical line segments if and only if it is *st\** numberable and furthermore that an *st\** numbering can be determined in  $O(n)$  time. In an alternate characterization he showed that a graph is representable by a set of vertical line segments if and only if there is a planar embedding of the graph with all cutpoints on the exterior face. The two graphs created from the stab information of the orthogonal polygon can be shown to have just such a planar embedding.

**Lemma 8** *The horizontal and vertical visibility graphs extracted from an orthogonal polygon (in section 4.4) have a planar embedding with all cutpoints on the exterior face.*

Proof: The proof is similar for the horizontal and vertical graphs, so only the vertical is considered. Locate the node that represents each horizontal segment on the midpoint of the segment. Join interior visible nodes through the interior of the polygon and exterior visible nodes through the exterior of the polygon, each along the path of visibility. This creates a planar embedding of the visibility graph, and is the embedding assumed in the remainder of this proof.

For every boundary segment or part of a boundary segment,  $s$ , of the polygon that has no segment directly below it (see figure 4.18), imagine a rectangle with  $s$  as one side,  $-\infty$  as the opposite side, and two parallel lines connecting the two. Do the same for segments or partial segments of the polygon with no segment directly above it. Call these  $\infty$  **visibility rectangles**. The nodes on the exterior face of the

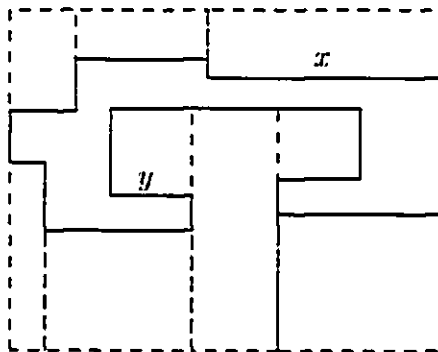


Figure 4.18:  $\infty$  Visible Rectangles

planar graph described above, correspond to a boundary segment of the polygon that

is either part of an  $\infty$  visible rectangle (segment  $x$  in figure 4.18), or a segment that has a stab through one of the  $\infty$  visible rectangles (segment  $y$  in figure 4.18).

Assume there is a cutpoint that is not on the exterior face of this embedding. Every node that is not on the exterior face has at least one node that is visible above it and one that is visible below it. So, if the cutpoint node is removed, the segment below and the segment above the cutpoint will be disconnected. But since the cutpoint is not on the exterior face of the graph, there must be a path to the left and a path to the right of the cutpoint. These paths will connect the two nodes that were supposedly disconnected by the removal of the cutpoint. Therefore, the assumption is incorrect, and all cutpoints must be on the exterior face of the graph, as laid out in this planar embedding.  $\square$

The converse of lemma 8 is not true.

**Lemma 9** *A planar embedding of a graph with all cutpoints on the exterior face is not necessarily a horizontal or vertical visibility graph of an orthogonal polygon.*

Proof: Figure 4.19 is an example of a planar embedding of a graph with all cutpoints on the exterior face. The figure also shows an orthogonal polygon that attempts to realize the given horizontal visibility graph. The polygon does not realize the graph since the graph edge between nodes  $g$  and  $f$  does not have a corresponding visibility path. This graph could not be a horizontal or vertical visibility graph of an orthogonal polygon, since the face bounded by  $dfg$  has only 3 edges. The horizontal and vertical visibility graphs of an *orthogonal* polygon would need all interior faces



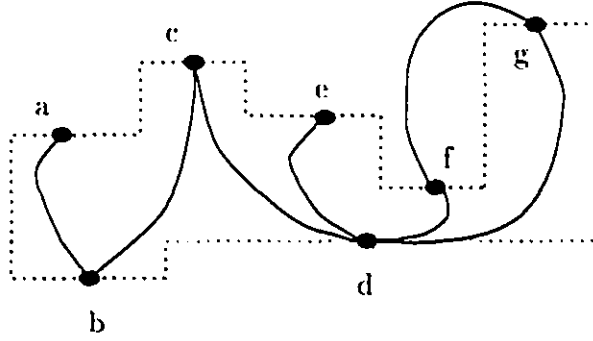


Figure 4.19: A Graph with Cutpoints on Exterior Face

to have at least four edges.  $\square$

An  $st^*$  numbering of the horizontal (or vertical) visibility graph extracted from an orthogonal polygon can be made, by including two super nodes, one at  $+\infty$  and one at  $-\infty$ . Assuming the embedding of the graph, as described in the proof of lemma 8, extend the graph by adding connections between all nodes located on segments that are part of an  $\infty$  visible rectangle to the appropriate new node:  $+\infty$  or  $-\infty$ .

Furthermore, if the correct choice of the  $st^*$  numbering is made, the reconstructed line segments represent one dimension (horizontal or vertical) of the edges of the orthogonal polygon. Notice that the topological sort numbering of section 4.3 is an  $st^*$  numbering. If that numbering is used, the bar reconstruction algorithm creates bars that can be connected into an orthogonal polygon. However, choosing this particular numbering would require running the entire OPR algorithm. Unless another method of choosing the  $st^*$  numbering is found, it is more reasonable to just use the algorithm of section 4.3.

## 4.6 The Collinear Sides Assumption

The algorithms of this chapter assume that the orthogonal polygon has no collinear sides. Allowing collinear sides is a natural extension of this problem, and needs to be considered.

Lemma 4 of section 4.1 showed that rectangles of types 0 through 12 (figure 4.3) were the only possible rectangles created from the sides of an orthogonal polygon and its horizontal stabs. If collinear sides were possible, the number of rectangle types would increase. Cases 0, 1 and parts a and b of case 2 of that proof, did not rely on this assumption, so those cases contain the complete set of rectangles. However, part c of case 2 and cases 3 and 4 both used this assumption, so the additional rectangles of figure 4.20 would be introduced. Algorithmically, checking each of these additional types of rectangles is no more difficult than checking each of types 0 through 11.

It is possible that along each *stab* of each rectangle, any number of collinear sides could exist, as shown in figure 4.21. Rechecking each rectangle type, allowing for the possibilities of such sides complicates the algorithm. It is necessary to traverse the list of sides that are horizontally collinear and are guaranteed to have the same convexity on each side, to find the side that would be stabbed if the collinearity did not exist. Define a **traversable horizontal side** as one that has same convexity vertices on each end, as indicated by a previously identified rectangle. For example, the bottom side of types 7, 8, 9, 10, 11, 12 and 13 and the top side of types 9, 26, 27, and 28 as drawn in figures 4.3 and 4.20 are traversable. Also, define a **non-traversable horizontal side** as one that has opposite convexity vertices, or has not yet been identified as

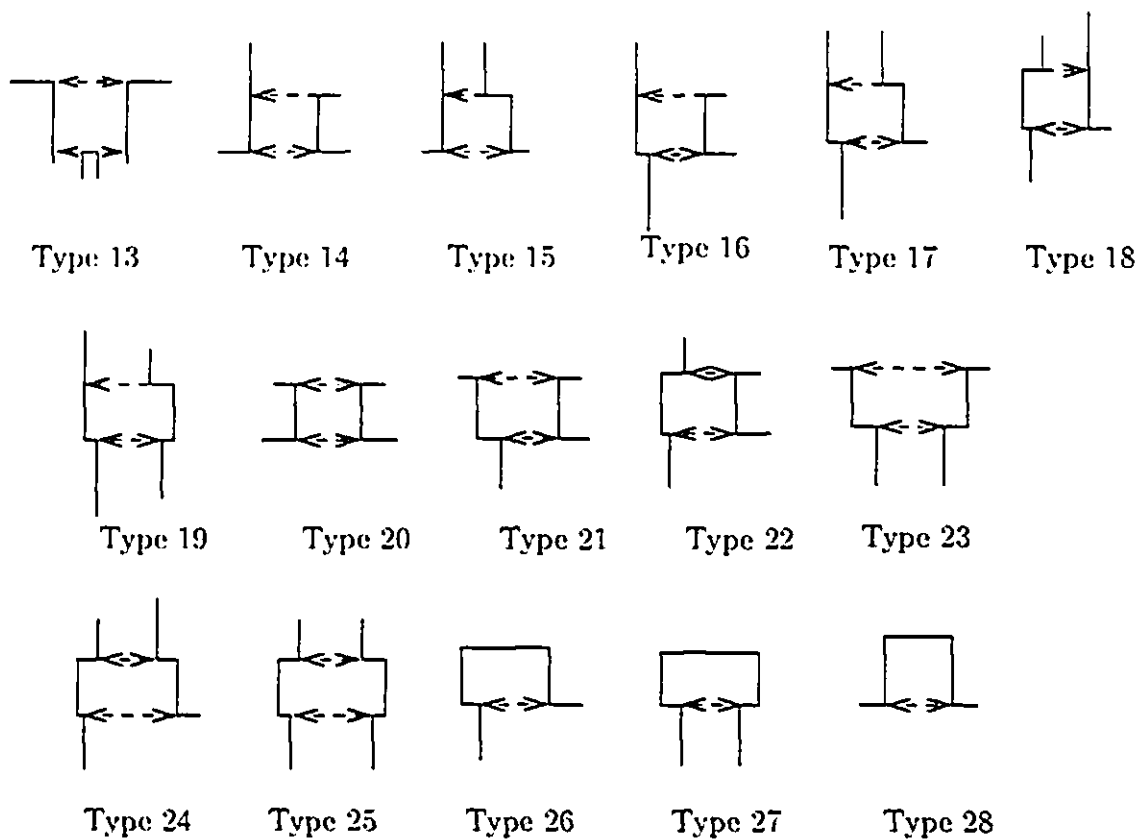


Figure 4.20: Extra Rectangles Possible with Collinear Sides

traversable. Recall that for non-collinear sides,  $stab[v]$  is the *side* stabbed by the horizontal stab emanating from vertex,  $v$ . For collinear sides,  $stab[v]$  is the collinear *vertex* stabbed by the horizontal stab from vertex,  $v$ . Assuming collinear sides are identifiable in constant time from the *stabs*, a straightforward algorithm to identify and classify rectangles is:

- for each horizontal side,  $s$ , in Hamiltonian cycle order do:
  - $side[s] := non-traversable$ .
- $stab\_can\_Change := FALSE$
- for each vertex,  $v$ , in Hamiltonian cycle order do:

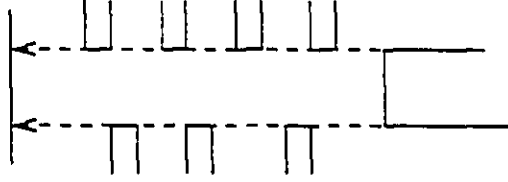


Figure 4.21: Collinear Sides Along the *stabs* of a Type 1 Rectangle

- $number\_of\_rectangles[v] := 0$
- Repeat /\*matching until below\*/ :
  - $stab\_can\_Change := FALSE$
  - for each vertex,  $v$ , in Hamiltonian cycle order do:
    - \* Check for types 0 through 11 rectangles and types 13 through 28 collinear rectangles. For every pair of vertices,  $j$  and  $k$ , on the identified rectangle
      - increment  $number\_of\_rectangles[j]$
      - either  $INSERT(j, same[k])$  or  $INSERT(j, opposite[k])$  appropriately
    - \* if the rectangle is type 7, 8, 9, 10, 11, 13, 26, 27, or 28:
      - $side[j] := traversable$ , where  $j$  indicates the vertices of the rectangle that are part of traversable sides, as indicated above.
      - $stab\_can\_Change := TRUE$ .
  - for each vertex,  $v$ , in Hamiltonian cycle order do:
    - \* if ( $stab[v]$  is collinear to  $v$ )
      - $a := stab[v], b := stab[v] + 1$
      - if ( $(side[\overline{ab}] = traversable)$  and ( $\overline{ab}$  is horizontal)) then  $stab[v] := stab[b]$
      - else
        - $a := stab[v], b := stab[v] - 1$
        - if ( $(side[\overline{ab}] = traversable)$  and ( $\overline{ab}$  is horizontal)) then  $stab[v] := stab[b]$
- Until ( $stab\_can\_Change = FALSE$ ) /\* Matches the repeat above \*/
- Check for type 12 rectangles.

The part of the algorithm that traverses the collinear sides dominates the analysis.

Since it is possible that a side may not become traversable until  $O(n)$  other associated

rectangles have been identified, the do/while loop could be executed  $O(n)$  times. Within the do/while loop are two  $O(n)$  for loops. Thus the analysis of the algorithm, allowing collinear sides is a more expensive  $O(n^2)$ .

## 4.7 Summary

In summary, this chapter presented an algorithm that reconstructed an orthogonal polygon when its Hamiltonian cycle and the stabs of its vertices are known. This algorithm runs in  $O(n \log n)$  time if the polygon is known to be without collinear sides and  $O(n^2)$  time otherwise. The result is related to but extends the Edge Visibility Trees of O'Rourke and Booth[O'R87] as described in section 4.5.1 and Wismath's Bar Visibility Graphs[Wis85] discussed in section 4.5.2.

## Chapter 5

# Conclusions and Open Problems

The purpose of this thesis was to examine and solve visibility reconstruction problems, that is, given visibility information of a set of objects, reconstruct the original objects. The thesis presented two results related to visibility graph reconstruction.

In chapter 3, a conversion technique was presented. The existing work on reconstructing polygons from their (unordered) vertex visibility graphs normally assumes the boundary Hamiltonian cycle of the polygon is known. However, when reconstructing a set of line segments from their endpoint visibility graphs, it is often assumed that the order of the edges around each node of the graph is in the same order as visibilities to other endpoints as seen by the corresponding endpoint. This thesis presented algorithms for converting between the Hamiltonian cycle (with the unordered vertex visibility graph) and the ordered vertex visibility graph for a simple polygon. These two results link the results of two different subareas in the study of visibility: line segments and polygons.

In chapter 4, an efficient algorithm for reconstructing orthogonal polygons was presented. The algorithm expects the Hamiltonian cycle and the stabs of the sides of the polygon as input. It determines whether each vertex is convex or reflex and

creates a partial ordering of the horizontal sides of the polygon in the  $Y$ -dimension and an ordering of the vertical sides of the polygon in the  $X$ -dimension of the cartesian coordinate system. These  $X$  and  $Y$  dimension orderings of the sides can be used to draw an orthogonal polygon consistent with the given stab information.

These results represent a small fraction of the work to be done in the reconstruction area of the study of visibility graphs. Some direct refinements and extensions of the presented results are discussed below.

The analyses of all non-collinear routines in chapter 4 were dominated by the  $O(n \log n)$  time needed to find the groups of four vertices that are on common type 12 rectangles. If this could be reduced to  $O(n)$  the entire analysis would be a more pleasing  $O(n)$ . Therefore, an open problem is to reduce the time needed to run the type 12 algorithm presented or alternatively, to prove that  $\Omega(n \log n)$  is a lower bound.

The orthogonal polygon reconstructed by the OPR algorithm is consistent with the given stab information. An extension of the OPR problem would be to reconstruct an orthogonal polygon that is consistent with the internal vertex visibility graph as well as the stabs and Hamiltonian cycle. Line segment reconstruction results such as [Wis94] together with the results of chapter 3 may be one approach to accomplish this.

A planar straight line graph, or PSLG, as described in Preparata and Shamos [PS85] is a general subdivision of the plane into attached polygons. A natural extension of the orthogonal polygon reconstruction of chapter 4 is to apply the algorithm to PSLG's that are orthogonal.

Other open problems include applying the techniques of the OPR algorithm to

general simple polygons, and extending the reconstruction result to three dimensional orthogonal polyhedra, however these are significantly more difficult problems.

The VisPak project [JPW95], located at <http://www.cs.uleth.ca/dept/wismath/vis.html>, is committed to implementing visibility algorithms. Currently, six visibility algorithms have been implemented, details are found in chapter 1. Most of the programs in the package are implementations of visibility graph construction algorithms. The visual nature of the output of these programs has proved invaluable in testing the correctness of research ideas relating to the design of visibility reconstruction results. Future releases of the package may include implementation of the OPR result of Chapter 4.



## Bibliography

- [AAG<sup>+</sup>86] Ta. Asano, Te. Asano, L. J. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1:49-63, 1986.
- [AS95] H. Alef and I. Streinu. A program that computes a visibility graph: xv-graph.c. Available by ftp at Smith College, Massachusetts, 1995.
- [BCLT96] J. E. Baker, I. F. Cruz, G. Liotta, and R. Tamassia. A new model for algorithm animation over the www. Technical report, Brown University, Providence, RI, USA, 1996.
- [BM95] P. Biscondi and J-M. Moreau. Computer aided road network design. In *Proceedings of the Seventh Canadian Conference on Computational Geometry*, pages 229-234, 1995.
- [BN91] H. Bieri and H. Nottmeier, editors. *Computational Geometry - Methods, Algorithms and Applications*, chapter Preface. Proceedings of the International Workshop on Computational Geometry CG '91, Bern, Switzerland. Springer-Verlag, Berlin Heidelberg, 1991.
- [Bra] F. J. Brandenburg. *GraphEd*. Universitaet Passau, Passau, Germany, version-4.0.2 edition.
- [Cha91] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485-524, 1991.
- [CL91] C. Coullard and A. Lubiw. Distance visibility graphs. In *Proceedings of the seventh Annual ACM Symposium on Computational Geometry*, pages 289-296, 1991.
- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Mass., 1990.
- [CSC92] S. H. Choi, S. Y. Shin, and K. Y. Chwa. Characterizing and recognizing visibility graphs of funnel-shaped polygons. In *Proceedings of the Third Annual International Symposium on Algorithms in Computing (ISAAC '92)*, 1992.

- [dRJ93] P. J. de Rezende and W. R. Jacometti. Animation of geometric algorithms using geolab. In *Proceedings of the ninth Annual ACM Symposium on Computational Geometry*, 1993.
- [EC95] H. Everett and D.G. Corneil. Negative results on characterizing visibility graphs. *Computational Geometry Theory and Applications*, 5:51–63, 1995.
- [Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany, 1987.
- [ElG85] H. A. ElGindy. *Hierarchical Decomposition of Polygons with Applications*. PhD thesis, School of Computer Science, McGill University, 1985.
- [Eve90] H. Everett. *Visibility Graph Recognition*. PhD thesis, University of Toronto, Department of Computer Science, January 1990.
- [Gho88] S. K. Ghosh. On recognizing and characterizing visibility graphs of simple polygons. In *Proceedings of the first Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 96–104. Springer-Verlag, 1988.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.
- [GM95] J. Gosling and H. McGilton. The java language environment: A white paper. Technical report, [http://java/sun/com/whitePaper/javawhitepaper.1.html](http://java.sun.com/whitePaper/javawhitepaper.1.html), 1995.
- [Her87] J. Hershberger. Finding the visibility graph of a simple polygon in time proportional to its size. In *Proceedings of the third Annual ACM Symposium on Computational Geometry*, pages 11–20, 1987.
- [HT74] J.E. Hopcroft and R.E. Tarjan. Efficient planarity testing. *Journal of the Association of Computing Machinery*, 21:549–568, 1974.
- [JPW95] L. Jackson, H. Pinto, and S.K. Wismath. VisPak: A Package of Visibility Algorithms written in LEDA. Technical Report Report TR-CS-01-95, University of Lethbridge, Lethbridge, Alberta, 1995.
- [Kle92] R. Klein. Walking an unknown street with bounded detour. *Computational Geometry Theory and Applications*, 1:325–351, 1992.
- [KMM<sup>+</sup>90] A. Knight, J. May, M. McAffer, T. Nguyen, and J.-R. Sack. A computational geometry workbench. In *Proceedings of the sixth Annual ACM Symposium on Computational Geometry*, page 370, 1990.

- [KW95] M. Keil and S.K. Wismath. Depth first spiralling on the endpoints of line segments in output sensitive time. Technical Report Report TR-CS-02-95, University of Lethbridge, Lethbridge, Alberta, 1995.
- [Len90] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Applicable Theory in Computer Science. John Wiley & Sons, WestSussex, England, 1990.
- [LOS95] A. López-Ortiz and S. Schuierer. Simple efficient and robust strategies to traverse streets. In *Proceedings of the seventh Canadian Conference on Computational Geometry*, pages 217–222, 1995.
- [LP84] D. T. Lee and F. P. Preparata. Computational geometry: a survey. *IEEE Transactions on Computers*, C-33:1072–1101, 1984.
- [Meh84] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, volume 2 of *Data Structures and Algorithms*. Springer-Verlag, Heidelberg, West Germany, 1984.
- [NU95] S. Naher and C. Uhrig. *The LEDA User Manual*. Max-Planck-Institut für Informatik, Saarbrücken, Germany, release r 3.2 edition, 1995.
- [O'R87] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987.
- [O'R93a] J. O'Rourke. Computational geometry column 18. *Sigact News*, 24(1):20–25, 1993.
- [O'R93b] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1993.
- [OS95] J. O'Rourke and I. Streinu. Visibility in pseudo-polygons and vertex-edge pseudo-visibility graphs. Technical report, Smith College, Northampton, MA, 1995.
- [PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, New York, NY, 1985.
- [She92] T. C. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.
- [Sho91] P. W. Shor. Stretchability of pseudolines is NP-hard. In *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, volume 4 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 531–554. AMS Press, 1991.
- [SR90] S. Sudarshan and C. Pandey Rangan. A fast algorithm for computing sparse visibility graphs. *Algorithmica*, 5:201–214, 1990.

- [TT86] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete and Computational Geometry*, 1(4):321–341, 1986.
- [TV88] R. E. Tarjan and C. J. Van Wyk. An  $O(n \log \log n)$ -time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17:143–178, 1988. Erratum in 17(1988), 106.
- [Wel85] E. Welzl. Constructing the visibility graph for  $n$  line segments in  $O(n^2)$  time. *Inform. Process. Lett.*, 20:167–171, 1985.
- [Wis85] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Proceedings of the first Annual ACM Symposium on Computational Geometry*, pages 147–152, 1985.
- [Wis94] S. K. Wismath. Reconstruction of parallel line segments from endpoint visibility information. In *Proceedings sixth Canadian Conference on Computational Geometry*, pages 369–373, 1994.