INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

COMPUTATIONAL GEOMETRY WITH THE ROTATING CALIPERS

by

Hormoz Pirzadeh

School of Computer Science
McGill University
Montréal, Québec
Canada

May 1999

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
OF MCGILL UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

Abstract

The Rotating Calipers is a paradigm used to solve a number of problems in the field of Computational Geometry. The original algorithm was presented by Shamos in 1978 in order to find the diameter of a convex polygon; a few years later, Toussaint showed that the essence of the algorithm can be used to solve a number of other problems. In the past twenty years, research done by Toussaint and others has shown the Rotating Calipers to be a paradigm in Computational Geometry, applicable to a variety of problems. The resulting algorithms are often optimal in their time complexity, and always efficient and easy to implement.

This work is an extensive collection of these results, including all those obtained at McGill University. It is the first time such a collection has been done, and would therefore be viewed as an important asset by many researchers around the world. The results in question are reviewed, and proofs lacking in their original articles are provided. Furthermore, a new result is obtained. The problems studied in this work include the diameter and width of convex polygons, the maximum and minimum distance between convex polygons, the minimum area and minimum perimeter enclosing rectangles for convex polygons, onion and spiral triangulations, quadrangulations, merging convex hulls, intersecting convex polygons, common tangents and critical support lines, and vector sums of convex polygons.

To supplement the theoretical aspect of this work, a synthesis of this work is provided on the Internet, as well as an interactive applet demonstrating the use of the Rotating Calipers in solving many of the above mentioned problems.

Résumé

Le paradigme des "Rotating Calipers" permet de trouver la solution à plusieurs problèmes en Géométrie Algorithmique. Cet algorithme a été proposé initialement par Shamos en 1978 pour trouver le diamètre d'un polygone convexe. En 1983, Toussaint démontra que le même algorithme sert à résoudre d'autres problèmes géométriques. Depuis lors, Toussaint ainsi que d'autres chercheurs dans le domaine ont déterminé que l'algorithme est en fait un paradigme pouvant être appliqué à plusieurs autres problèmes. Toujours simple à implémenter, l'algorithme offre souvent une complexité optimale.

Cette thèse est un recueil de la plupart de ces résultats, et en particulier tous ceux obtenus à l'université McGill. Cet ouvrage sera utile aux chercheurs du domaine de la Géométrie Algorithmique, puisque c'est le premier recueil de son genre. Les résultats en question sont ré-étudiés, les preuves manquant dans les articles originaux sont fournis, et un nouveau résultat est présenté. Les problèmes étudiés sont: le diamètre et la largeur de polygones convexes, la distance minimale et la distance maximale entre deux polygones convexcs, les rectangles d'aire et de périmètre minimaux d'un polygone convexe, les triangulations "oignon" et "spirales", les quadrangulations, l'union et l'intersection de polygones convexes, les lignes tangentes communes et les lignes de support critiques, ainsi que la somme vectorielle de polygones convexes.

Pour complémenter le côté théorique, une synthèse de ces résultats est presentée sur l'Internet, ainsi qu'un "applet" interactif montrant la résolution de la plupart des problèmes ci-dessus avec le paradigme des "Rotating Calipers".

Acknowledgments

First and foremost, I would like to thank my thesis supervisor, Prof. Godfried Toussaint, for accepting me as his student, and providing everything I needed in order to complete this work. Like many of his colleagues at the School of Computer Science, Prof. Toussaint loves to teach and this aspect makes him wonderful to work with.

The School, is in my mind an ideal environment for studies in Computer Science. The atmosphere is wonderful, thanks to the faculty and staff. Among the professors, Luc Devroye and Prakash Panangaden stand out, as they are two of the greatest teachers I have ever had, and I am certain I share this opinion with many students.

I extend my gratitude to the administrative staff of SOCS, whose help, patience, and friendliness have always impressed me. A special thank you goes out to the graduate secretary, Ms. Franca Cianci, whom I probably bothered too many times about my thesis!

I thank the technical staff and systems administrators of the School and the Computational Geometry Laboratory for providing wonderful tools to work with, and for their quick and thorough help whenever needed.

Many people have contributed to this work on a more personal level. I thank my family for their support, encouragement, understanding, and basically for being wonderful people! Special thanks go out to my *new* family for everything they have done to help me finish this work. Although one furry little member of this family was sometimes disruptive on long nights of programming, I'm sure he couldn't help it. Still, I already miss little Sacha.

I would like to thank a wonderful group of friends I am lucky to have. Among the "best", I thank Lorenzo, Seth and Riccardo. Sadly, I did not get to see much of Seth, but he always had his *special* words of encouragement to keep me going. Lorenzo and Riccardo had a more direct contribution. Lorenzo helped me a great deal with the applet; he introduced me to Sun Java (there can only be one), and probably spent too much time testing the applet! To complement this, Riccardo helped with the theoretical work, and the actual thesis. His input is always helpful, and somehow he managed to make time for me in his busy new life. I hope I can repay my debt to each one.

Finally, there remains my treasured bride, whom I could not thank in mere words... To my dearest Elizabeth, I dedicate this thesis.

Contents

Abstract ii

Résumé iii

Acknowledgments iv

1 Introduction and Preliminaries 1

1.1 Introduction 1

1.2 Review of Literature 2

1.3 Geometric preliminaries 4

1.4 Analysis of algorithms 5

2 Computing Distances 8

2.1 The diameter of a convex polygon 8

2.1.1 Introduction 8

2.1.2 Theoretical results and algorithm analysis 9

2.2 The width of a convex polygon 15

2.2.1 Introduction 15

2.2.2 Theoretical results and algorithm analysis 16
2.3 The maximum distance between two convex polygons 19
2.3.1 Introduction 19
2.3.2 Theoretical results 20
2.3.3 Algorithm and analysis 22
2.4 The minimum distance between two convex polygons 24
2.4.1 Introduction 24
2.4.2 Theoretical results 25
2.4.3 Algorithm and analysis 30

3 Enclosing Rectangles **33**

3.1 The minimum-area enclosing rectangle 33
3.1.1 Introduction 33
3.1.2 Theoretical results 34
3.1.3 Algorithm and analysis 38
3.2 The minimum-perimeter enclosing rectangle 39
3.2.1 Introduction 39
3.2.2 Minimum-perimeter versus minimum-area enclosing rectangles 40
3.2.3 Theoretical results 42
3.2.4 Algorithm and analysis 45
3.3 Width-determined enclosing rectangle 46
3.3.1 Bounds on the error 49

4 Triangulations **52**

4.1 Introduction 52

4.2 Onion Triangulations 53

4.2.1 Introduction 53

4.2.2 Theoretical results 54

4.2.3 Algorithm and analysis 56

4.3 Spiral Triangulations 60

4.3.1 Introduction 60

4.3.2 Algorithm and analysis 62

4.4 Quadrangulations 68

5 Properties of Convex Hulls **72**

5.1 Merging convex hulls 72

5.1.1 Introduction 72

5.1.2 Theoretical results 74

5.1.3 Algorithm and analysis 76

5.2 Common tangents 78

5.3 Intersecting convex polygons 80

5.4 Critical support lines 81

5.4.1 Introduction 81

5.4.2 Theoretical results 81

5.5 Vector sums of convex polygons 85

5.5.1 Introduction 85

5.5.2 Theoretical results 86

6 Thinnest Transversals **100**

6.1 Thinnest Strip Transversal 100

7 Conclusion **106**

A Rotating Calipers homepage **108**

B Rotating Calipers applet **109**

Bibliography **118**

List of Figures

2.1 The diameter of P is determined by an anti-podal pair. 11

2.2 Decomposing P into convex chains. 13

2.3 Illustrating the proof of Lemma 2.1.3. 13

2.4 Illustrating the proof of Lemma 2.1.3. 14

2.5 Illustrating the width of a convex polygon. 16

2.6 Illustrating the proof of Lemma 2.2.1. 18

2.7 Illustrating the proof of Theorem 2.3.1. 21

2.8 Illustrating Algorithm MAXDIST2POLY 22

2.9 Illustrating the proof of Theorem 2.4.1. 26

2.10 The three cases for the minimum distance between P and Q. 27

2.11 Illustrating the proof of Theorem 2.4.3. 28

2.12 Illustrating the proof of Theorem 2.4.3. 28

2.13 Illustrating the proof of Theorem 2.4.3. 29

2.14 Illustrating the proof of Theorem 2.4.3. 30

3.1 An example of a rectangle enclosing P. 35

3.2 A polygon and its enclosing rectangles. 40

3.3 A polygon with corresponding $R_{\min A}$ and R_W. 48

4.1 The *onion-peeling* of a set of points. 54

4.2 The *onion triangulation* of a set of points. 55

4.3 Illustrating the annulus of P and Q. 56

4.4 Illustrating the proof of Theorem 4.2.1. 58

4.5 A set of points (a) and its convex spiral (b). 61

4.6 Illustrating the initialization of Algorithm TRI-CS. 63

4.7 The *inner polygonal* and *outer spiral* regions 64

4.8 The outer spiral region. 64

4.9 An example of a quadrangulation 70

5.1 Merging two convex hulls by adding "bridges". 73

5.2 An example of two vertices forming a co-podal pair. 75

5.3 An example of a common tangent. 78

5.4 An example of Critical Support lines. 82

5.5 Illustrating the proof of Theorem 5.5.2. 88

5.6 Illustrating the proof of Theorem 5.5.2. 89

5.7 Illustrating the proof of Theorem 5.5.2. 89

5.8 Illustrating the proof of Theorem 5.5.3. 91

5.9 (a) Polygon P and its associated star diagram (b). 91

5.10 Illustrating the proof of Theorem 5.5.4. 92

5.11 Illustrating the proof of Theorem 5.5.5. 93

5.12 Illustrating the first case of Theorem 5.5.5. 94

5.13 Illustrating the second case of Theorem 5.5.5. 94

5.14 Illustrating the third case of Theorem 5.5.5. 95

5.15 Illustrating the proof of the first two cases of Theorem 5.5.5. 96

5.16 Illustrating the proof of the third case of Theorem 5.5.5. 97

6.1 Illustrating Lemma 6.1.1. 103

6.2 The minimum-width strip with orientation θ. 104

Chapter 1

Introduction and Preliminaries

1.1 Introduction

Computational Geometry is the study of geometric problems and the algorithms to solve them. Problems range from the straightforward to the very complex in their concept. However, even the least complicated problems rarely have simple algorithms to solve them efficiently. Usually, the best algorithm to solve a given problem is "tailor-made" to suit that specific problem. It is therefore rare to find a *paradigm* in computational geometry, i.e., a general method or approach to solve a variety of problems. The *Rotating Calipers* is such a paradigm. As well as being polyvalent, this method's other advantages are its simplicity in concept and its efficiency.

Invented along with the field by Shamos twenty years ago for computing the diameter of a convex polygon [74], Toussaint, who also coined the term "Rotating Calipers", was the first to show its polyvalence five years later [81]. Since then, many problems have been solved with the Rotating Calipers paradigm, either directly, or as part of an algorithm.

Yet, to date, the only paper dedicated to the Rotating Calipers is the conference paper by Toussaint [81], which contained no proofs of the results, and the journal

article never appeared. Due to the many requests over the years by researchers from around the world, this project has been revived in this thesis.

The purpose of this work is to collect all the results involving the Rotating Calipers obtained at McGill University, analyze them, and supply missing details and proofs. Such a collection has never been done, and would be a welcome and useful addition to the Computational Geometry literature.

However, this only constitutes the theoretical aspect of the work. In addition to this, an on-line source for information on the Rotating Calipers is provided in the form of a homepage on the Internet, with the purpose of providing the basics on how the paradigm works, its applications, and references to papers on the subject. Furthermore, an interactive on-line applet was developed to demonstrate various applications of the Rotating Calipers. Because the paradigm is very intuitive, it has great pedagogical value in Computational Geometry. Such an interactive applet would be an invaluable tool for anyone interested in the Rotating Calipers or in Computational Geometry. Please refer to Appendices A and B for more details on the experimental component of this work.

1.2 Review of Literature

In 1978, M.I. Shamos started the field of Computational Geometry within theoretical Computer Science with his Ph.D. thesis bearing the same title [74]. In his work, he discusses the problem of finding the diameter of a convex polygon (see section 2.1) and gives a fast and efficient algorithm as a solution. This algorithm is the basis for the Rotating Calipers paradigm.

Toussaint, in 1983, showed that Shamos' idea can be generalized to solve a variety of problems [81], such as the minimum-area rectangle enclosing a convex polygon, the maximum distance between two convex polygons, the vector sum of two convex polygons, merging convex hulls, linear separability of convex polygons, and many visibility problems. Since then, the Rotating Calipers have been used to obtain

many results, mostly by Toussaint. The paradigm has also been generalized to three dimensions, although we limit the scope of this work to the study of two-dimensional problems.

For this work, the results have been grouped according to the nature of the problem in order to provide some continuity to the text. The following is the set of problems studied:

1. Computing distances
   (a) The diameter of a convex polygon.
   (b) The width of a convex polygon.
   (c) The maximum distance between two convex polygons.
   (d) The minimum distance between two convex polygons.
2. Enclosing rectangles
   (a) The minimum area enclosing rectangle.
   (b) The minimum perimeter enclosing rectangle.
3. Computing triangulations and quadrangulations
   (a) Onion triangulations.
   (b) Spiral triangulations.
   (c) Quadrangulations.
4. Properties of convex polygons
   (a) Merging two convex hulls.
   (b) Critical Support lines.
   (c) Common tangents.
   (d) Vector sums.

(e) Intersecting two convex polygons.

5. Thinnest transversals

As the purpose of this work is to study each of these problems, it would be redundant to discuss here the previous results leading to them and the more recent solutions to these. Thus, we prefer to leave these discussions in the introductions to each problem.

Let us now review some preliminary concepts in geometry and analysis of algorithms. For a more elaborate discussion of these topics, consult the text by Preparata and Shamos [65].

1.3 Geometric preliminaries

The geometric objects studied lie in $\mathbf{R}^2$, which we shall refer to as the *plane*. We use the Cartesian coordinate system, defining a point O on the plane as the *origin*, and the x and y axes which form the coordinate system. Thus a point A on the plane can be defined by its coordinates a_x and a_y. Given two points A and B, the *distance* between them (denoted $\mathrm{dist}(A, B)$) is determined using the usual Euclidean metric, i.e., $\mathrm{dist}(A, B) = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2}$. The distance between a point and a line refers to the perpendicular distance. In other words, given a point A and a line L, $\mathrm{dist}(A, L) = \mathrm{dist}(A, B)$ such that $B \in L$, $[A, B] \perp L$. Finally, the distance between two parallel lines L_1 and L_2 refers to the distance between a point $A \in L_1$ and L_2.

Given a set of n distinct points $p_1, \ldots, p_n$ on the plane, the set of line segments $[p_i, p_{i+1}]$ $i = 1, \ldots, n-1$ together with $[p_n, p_1]$ form a closed path B. Let I be the closed region bounded by B (but excluding B). Let $P = B \cup I$, the union of B and I. P is referred to as a *polygon*, B as its boundary and I its interior, and we say that $p_1, \ldots, p_n$ determines P. A point p_i is a *vertex* of P, and a line segment $e = [p_i, p_j]$, $e \subset B$ is an *edge*. A polygon with n vertices can also be referred to

as an *n-gon*. In this work, a polygon includes its interior. For ease of notation, the boundary of a polygon P is often denoted ∂P.

A *simple polygon* is such that its edges never intersect, except for adjacent edges sharing a common vertex.

We define a *convex* polygon to be any (simple) polygon P satisfying the following property:

$$\forall p, q \in P, \quad [p, q] \subseteq P.$$

Finally, to avoid some degeneracies, the following constraint is assumed to be respected in all geometric data studied: all points are distinct, and no three points can be collinear. These constraints often occur in the field, and geometric input respecting these is often referred to as being given in *standard form*.

This forms the geometric basis for this work. All other objects, terms, and properties will be defined as the need arises.

1.4 Analysis of algorithms

As this work deals essentially with the analysis of solutions to computational geometric problems, i.e., algorithms, it is necessary to define some basic concepts.

The *model of computation* assumed for this work is a random-access machine (RAM) using real numbers. This means that the algorithms are assumed to be implemented on a machine with no restrictions on memory, but constrained by the fact that it can only perform one operation at a time on finite data. The memory is essentially an array of storage locations, each capable of holding one real number.

Hence, points (the simplest geometric data dealt with) will be stored in an array of two real numbers, containing the x and y coordinates. Line segments and polygons will be stored as arrays of points. This enables us to directly access a polygon's vertex if the vertex index is known.

The time taken by an algorithm—its *running time*— is the number of primitive operations executed. In the RAM model, the following operations are considered to be primitive, and assumed to take a constant amount of time to execute (referred to as *unit* time cost):

1. Arithmetic operations $+$, $-$, $\times$, $/$.
2. Real number comparisons $<$, $>$, $=$, $\leq$, $\geq$, $\neq$.
3. Indirect memory addressing.

This set can be expanded to include k-th root computations, trigonometric, exponential and logarithmic functions as well as floor and ceiling operations.

Usually, the running time of an algorithm will depend on the size of the input given. In computational geometry, the input usually consists of points, or line segments. For an input size of n, the running time is denoted $T(n)$.

Furthermore, because the running time depends not only on the size of the input but the input itself, we are (usually) interested in the *worst-case* behaviour of the algorithm. Hence, when the running time is discussed, the worst-case scenario is implied, unless otherwise specified.

Since running times are quite complicated to compute exactly (even for uncomplicated algorithms), the asymptotic behaviour of $T(n)$ is determined instead of its exact value. Therefore, running times of algorithms are classified and compared in terms of their order of growth. For instance, if a given algorithm's running time can be expressed as $T(n) = c_0 + c_1 n + c_2 n^2$ (where c_0,c_1,c_2 are real numbers and $c_2 > 0$) then $T(n)$ is asymptotically quadratic and we write: $T(n) = \Theta(n^2)$. Formally,

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2, n_0,\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0\}.$$

If $f(n) = \Theta(g(n))$, $g(n)$ is said to be an *asymptotic tight bound* for $f(n)$.

Based on this, we define the sets of functions $O(g(n))$ and $\Omega(g(n))$ for the *asymptotic upper* and *lower* bounds, respectively:

$$O(g(n)) = \{f(n) \mid \exists\, c_1, n_0 \;\; 0 \leq f(n) \leq c_1 g(n) \;\; \forall n \geq n_0\},$$

$$\Omega(g(n)) = \{f(n) \mid \exists\, c_1, n_0 \;\; 0 \leq c_1 g(n) \leq f(n) \;\; \forall n \geq n_0\}.$$

In describing the worst-case behaviour of an algorithm, $O(n)$ is used more frequently than its counterparts.

Finally, an algorithm's efficiency is often referred to as its *time complexity*. For instance, an algorithm whose running time is $O(n)$ will be said to have linear time complexity.

Bibliographical note

For the RAM model of computation, we refer the reader to Cook and Reckhow's paper where the concept is formally introduced [23]. The text by Aho, Hopcroft and Ullman [1] also offers a good description of the model. However, any good book introducing algorithms [2, 24] discusses models of computation, as well as the Θ, O, and Ω function sets. We invite the reader to consult Knuth's papers where he formally introduces the Θ notation [50] and the history behind this abstraction [49]. Finally, Preparata and Shamos' excellent text [65] offers a beautiful introduction to the field of Computational Geometry and the necessary preliminaries.

Chapter 2

Computing Distances

The problem of finding the diameter of a convex polygon was at the origin of the Rotating Calipers paradigm. Hence it is only natural to start our work with a close look at this result before moving on to other problems. Since the essence of the problem is distance computation, we dedicate the rest of the chapter to results of a similar nature.

2.1 The diameter of a convex polygon

2.1.1 Introduction

The diameter of a convex polygon is the greatest distance between any two points of the polygon. We let $\text{diam}(P)$ denote the diameter of a polygon P and formally define:

$$\text{diam}(P) = \max_{\forall p,q \in P} \{\text{dist}(p,q)\}.$$

Note that it is possible for multiple pairs of points to determine the diameter of a convex polygon; consider for example an equilateral triangle, a square, or a rectangle. Intuitively, the pair (p,q) determining the diameter cannot belong to the interior of P

(otherwise we could extend the line through p and q, intersect it with the boundary of P and the intersection points would have greater distance between them). Thus, if one wanted to compute the diameter of a given convex n-gon P, one would already restrict the search for p and q to the boundary of the polygon. It can also be established that p and q must be vertices of P (assume q belongs to an edge $e = [a, b]$ but is not a vertex. Then either a or b (or both) are farther from p than q is). Hence, a quick look at the problem reveals already that we need only be concerned with the n vertices of P. A simple algorithm immediately comes to mind: compute all $n(n+1)/2$ pairwise distances and keep the maximum. This is the *brute-force* way and is the easiest method for determining the diameter. However its running time is $O(n^2)$.

In 1978, Shamos presented a much more efficient yet simple algorithm for determining the diameter [74]. His method was later described by Toussaint [81] as:

> [...] rotating a pair of dynamically adjustable calipers once around the polygon.

Thus did Toussaint label Shamos' idea the "Rotating Calipers". This algorithm runs in $O(n)$ time, which is in fact optimal: in some cases, $O(n)$ pairs of vertices determine the diameter. Consider for example a regular n-gon (n odd): n diameters are determined. One could slightly perturb one of the vertices such that one single new diameter is determined. In that case, an algorithm that does not check all antipodal pairs might not take in account the diameter determining pair of points.

Let us now analyze the result, as presented by Preparata and Shamos [65].

2.1.2 Theoretical results and algorithm analysis

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a convex polygon with n vertices in standard form, i.e., the vertices are specified according to Cartesian coordinates in a clockwise order and no three consecutive vertices are collinear.

Theorem 2.1.1 [90] further constrains our search for the diameter determining pair of vertices. Let us however define a few terms first.

Definition 2.1.1 *A line L is a* line of support *for a convex polygon P if it intersects P and the interior of P lies on one side of L.*

If L intersects P at a vertex v (or an edge e), we will say that v (or e) *admits* L.

Definition 2.1.2 *Given a convex polygon P, a pair of vertices $p, q \in P$ is called an* anti-podal pair *if p and q admit parallel lines of support.*

Theorem 2.1.1 *The diameter of a convex polygon P is the greatest distance between parallel lines of support of P.*

Proof: Given $P = \{q_1, \ldots, q_n\}$. Suppose that the pair of vertices $\{q_a, q_b\}$ determines the diameter of P. Let $d_{ab} = \text{dist}(q_a, q_b)$, let C_a be the circle of radius d_{ab} centered at q_a and C_b the circle of radius d_{ab} centered at q_b. Define the lines L_a as the tangent to C_b at q_a, and L_b as the tangent line to C_a at q_b, and finally let L be the line through q_a and q_b. See Figure 2.1. By definition of tangent lines, we have $L_a \perp L$, and $L_b \perp L$. Therefore, L_a is parallel to L_b. Now we claim L_a and L_b are lines of support (LS) of P.

Pick any point $p \in P$, $p \neq q_a$. p is on or inside C_b since $\text{dist}(q_b, p) \leq d_{ab}$, by the definition of diameter. Therefore $p \notin L_a$. Since this is true for all points of P except q_a, $L_a \cap P = q_a$, thus L_a is a line of support for P. Similarly, L_b is a line of support for P.

Since L_a, L_b are parallel and both lines of support for P, and

$$\text{diam}(P) = d_{ab} = \text{dist}(L_a, L_b)$$

we have that the diameter of P is determined by parallel lines of support.

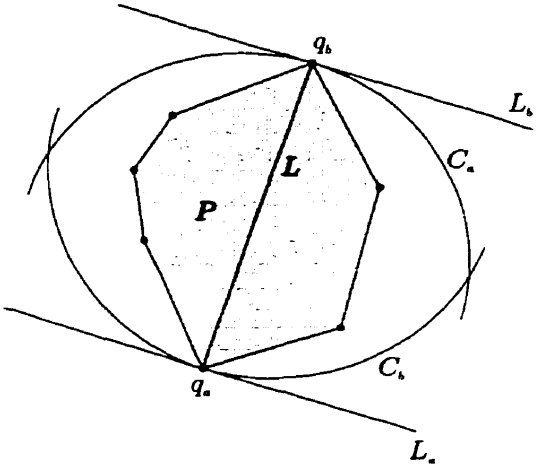


Figure 2.1: The diameter of P is determined by an anti-podal pair.

Finally, suppose there exist parallel lines of support L_1 and L_2 such that $\text{dist}(L_1, L_2) > d_{ab}$. Say L_1 and L_2 intersect P at vertices q_1 and q_2, respectively. Then $\text{dist}(q_1, q_2) \geq \text{dist}(L_1, L_2) > d_{ab}$, which contradicts our assumption that $d_{ab} = \text{diam}(P)$.

Hence, $\text{dist}(L_a, L_b)$ is the maximal distance between parallel lines of support for P, and we have that the diameter of P is the greatest distance between parallel lines of support. ∎

Corollary 2.1.2 *The diameter of a convex polygon P is the greatest distance between an anti-podal pair of P.*

Proof: It has already been established that any diameter pair is an anti-podal pair. Since no anti-podal pair can have a distance between its vertices greater than the diameter, the diameter is the greatest distance between an anti-podal pair. ∎

The following is an algorithm for generating all anti-podal pairs of a given convex polygon $P = \{p_1, \ldots, p_n\}$, as presented by Preparata and Shamos [65]. Note the addition of lines 10a and 12a. While 10a is a mere detail, 12a is absolutely necessary.

Algorithm ANTI-PODAL-PAIRS

```
1. begin p := p_n;
2.       q := NEXT[p];
3.       while (Area(p, NEXT[p], NEXT[q])
                > Area(p, NEXT[p], q)) do
                q := NEXT[q];
4.       q_0 := q;
5.       while (q ≠ p_0) do
6.             begin p := NEXT[p];
7.                   Print(p, q);
8.                   while (Area(p, NEXT[p], NEXT[q])
                           > Area(p, NEXT[p], q, )) do
9.                         begin q := NEXT[q];
10.                              if ((p, q) ≠ (q_0, p_0)) then Print(p, q)
10a.                             else return
                           end;
11.                  if (Area(p, NEXT[p], NEXT[q])
                       = Area(p, NEXT[p], q)) then
12.                    if ((p, q) ≠ (q_o, p_n)) then Print(p, NEXT[q])
12a.                   else Print(NEXT[p], q)
             end
  end.
```

Now, to the proof of correctness. We will show that Algorithm ANTI-PODAL-PAIRS does indeed generate all anti-podal pairs of P.

Suppose we want to generate all the points forming an anti-podal pair with a given vertex p_i. Let q^- be the furthest vertex from the edge $p_{i-1}p_i$ and q^+ the furthest vertex from p_ip_{i+1}. Consider Figure 2.2 where edges are shown in thick lines and the polygon's bounds in thin lines. Vertices in the convex chains $p_{i+1}q^-$ and q^-q^+ must lie in triangles A and B, respectively.

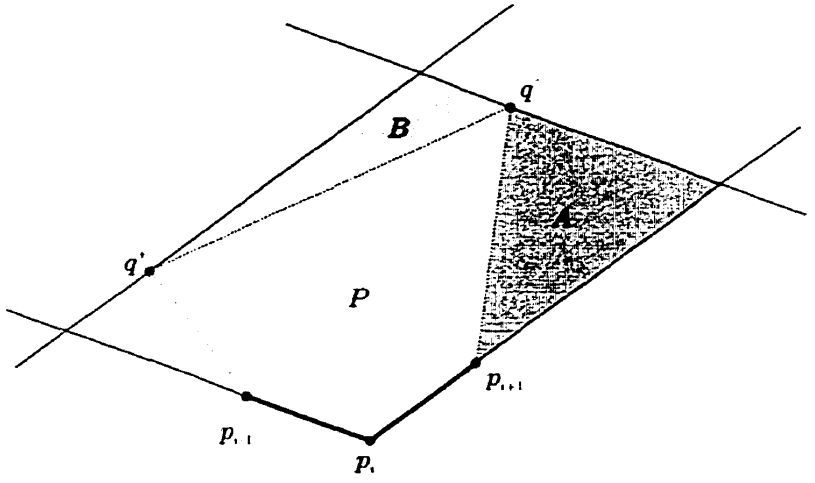


Figure 2.2: Decomposing P into convex chains.

Lemma 2.1.3 *All vertices forming an anti-podal pair with p_i are in the q^-q^+ convex chain.*

Proof: Suppose a vertex p_0 outside the q^-q^+ chain forms an anti-podal pair with p_i. (Without loss of generality, say it is between p_{i+1} and q^-). See Figure 2.3. Therefore p_0 must lie in A (see Figure 2.3). The dark grey

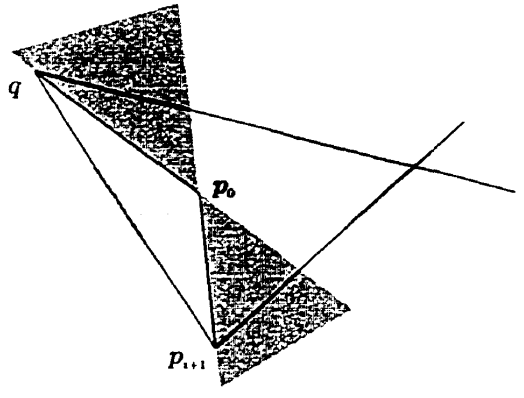


Figure 2.3: Illustrating the proof of Lemma 2.1.3.

area shows the range of possible lines of support at p_0. The light grey area shows the range of directions possible for p_i. (Note this range is always

delimited by lines parallel to the lines delimiting A). These areas do not intersect, showing the impossibility of parallel lines of support for p_0 and p_i. This is true for any point in A. Hence no point in A can have a line of support parallel to a line of support at p_i. Similarly, no point in the q^+p_{i-1} convex chain can form an anti-podal pair with p_i.

Now suppose we take a point q in the q^-q^+ convex chain. Then q must lie in B. See Figure 2.4. The grey area (S_q) shows the possible range of

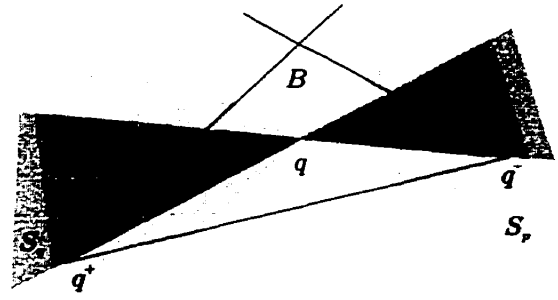


Figure 2.4: Illustrating the proof of Lemma 2.1.3.

directions for a line of support at q. The light grey area (S_p) shows the range of directions possible for p_i. (Note this range is always delimited by lines parallel to the lines delimiting B). The non-empty intersection of these areas ($S_p \cap S_q$) is shown in darker grey. Since this does not depend on the location of q, it is always possible to find parallel lines of support for any point $q \in B$ and p_i.

Therefore, all points forming an anti-podal pair with p_i are in the q^-q^+ convex chain. ∎

Algorithm ANTI-PODAL-PAIRS determines q^- and q^+ for a given vertex, and by the result above, it counts all vertices in this convex chain as forming anti-podal pairs with p_i. This procedure is repeated for the vertices following p_i, until the original pair is reached. Therefore, all anti-podal pairs of P are generated.

Theorem 2.1.4 *Algorithm ANTI-PODAL-PAIRS generates all anti-podal pairs of a convex polygon P in $O(n)$ time.*

> **Proof**: Note first that P is determined by $p_0, \ldots, p_n$. The algorithm starts with an initialization step consisting of finding q^- for p_0. So each vertex, starting with p_1 is checked until the farthest from $p_n p_0$ is found. This vertex is denoted q_0. This consists of a comparison, so the initialization time is $O(n)$.
>
> Once the initialization done, the algorithm keeps track of two points: p, the main vertex, and q, the vertices in the q^-q^+ convex chain of p. For a given p, and knowing that q is at q^- for p, the algorithm outputs (p, q) as an anti-podal pair, and "increments" q until q^+ is reached. It then "increments" p, at which point the old q^+ automatically becomes the new q^-, so no recalculation is needed. This process is repeated until the original pair (p_0, q_0) is reached. At the end of the main loop, p or q have been incremented. Since the algorithm never backtracks and stops without ever counting any point twice, it must run in $O(n)$ time. ∎

Using the results above, the diameter of a given convex polygon can be determined in $O(n)$ time using the "rotating calipers" as a basis: the anti-podal pairs are determined, each pair's distance computed, the maximum updated until all pairs have been checked.

Note that the algorithm outputs one anti-podal pair at each step, and two only if the polygon has parallel edges. The number of anti-podal pairs for a convex n-gon is then bound by $3n/2$ [65].

2.2 The width of a convex polygon

2.2.1 Introduction

Let us start by stating the problem formally.

Definition 2.2.1 *Given a convex polygon P, the width of P is the minimum distance between parallel lines of support of P.*

As its definition suggests, the concept of width is closely related to that of the diameter. Intuitively, one can think of the width of P as the smallest-size opening through which one could slide the "object" P(see Figure 2.5). In fact, the notion of

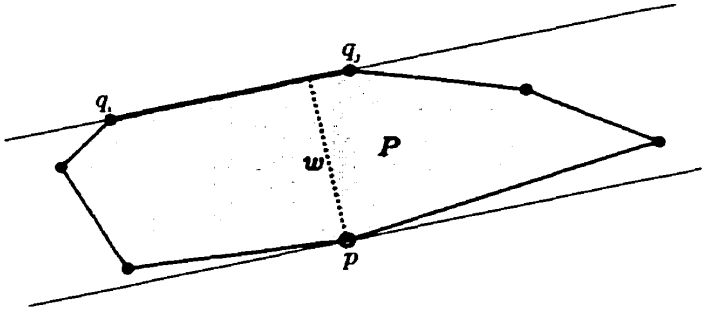


Figure 2.5: The width w of P is determined by lines of support through p and q_iq_j.

width is often used in collision-avoidance problems [82]. Other applications include approximating polygonal curves and line fitting [43, 44, 51]. Often, the general problem of computing the width of a set of points is discussed, but the convex hull of points must be determined in any case, so the two problems are essentially the same.

Results dating back to 1975 [43] help characterize the problem by constraining the possible 'candidates' for the width. In 1982, Kurozomi and Davis give two algorithms for computing the width of a convex n-gon [51]. The algorithms run in $O(n^2)$ and $O(n \log n)$ (using binary search). Houle and Toussaint [42] reduce the time complexity to $O(n)$, providing two algorithms: one is based on the Rotating Calipers, and the other on previous work by Brown for determining the diameter of a convex polygon [11]. Our interest lies in Houle and Toussaint's Rotating Calipers solution.

2.2.2 Theoretical results and algorithm analysis

Houle and Toussaint [42] present the following result, obtained by Ichida and Kiyono [43]:

Lemma 2.2.1 *The width of a polygon P is the minimum distance between parallel lines of support passing through a vertex-edge pair of P.*

Before proving this result, let us refine our notion of anti-podal pairs. As discussed in Section 2.1, an anti-podal pair is a pair of vertices admitting parallel lines of support. But what happens if one or both lines coincide with edges of the polygon?

We define three 'types' of anti-podal pairs:

1. The anti-podal *vertex-vertex* case.
2. The anti-podal *edge-edge* case, where both lines of support coincide with edges of the polygon.
3. The anti-podal *vertex-edge* case, where one line of support coincides with an edge of the polygon.

Lemma 2.2.1 states that we need only be concerned with the third case in order to compute the width. Here is the proof:

Proof: Let us analyze cases 1 and 2.

1. vertex-vertex: Given any two points p and q and distinct parallel lines l_1, l_2 passing through them, we can always decrease the distance d separating the two lines by rotating them about p and q in the *preferred direction of rotation.* The example in Figure 2.6 shows that by rotating l_1 and l_2 counterclockwise about p and q, d will decrease. Thus, for this case, the distance between the parallel lines of support can always be decreased until one of the lines hits an edge. Therefore, the original lines of support determining a vertex-vertex anti-podal pair could *not* determine the width.

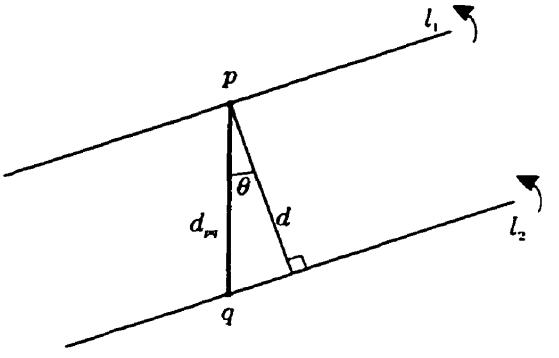


Figure 2.6: Illustrating the proof of Lemma 2.2.1.

2. edge-edge: In this case, the distance between the lines of support is the same as the distance between a vertex of one edge and the line of support through the other edge. Therefore, it can be considered as a special case of the vertex-edge case.

■

With this result, the width can now be easily determined using the idea of section 2.1. Houle and Toussaint [42] describe a linear-time procedure for obtaining the width.

Theorem 2.2.2 *Using the Rotating Calipers, it is possible to determine the width of a convex polygon P in $O(n)$ time.*

Proof: Given a convex polygon P, Algorithm ANTI-PODAL-PAIRS (see section 2.1) outputs all anti-podal pairs of P. We have to determine all vertex-edge cases. Vertex p and edge qq' admit lines of support *if and only if* (p, q) and (p, q') are both anti-podal pairs of P. Given our list of anti-podal pairs for P, we can go through the list in $O(n)$ time, and determine all vertex-edge pairs. Every time a vertex-edge pair is found, we calculate the vertex-edge distance ($O(1)$ time) and compare it to our "minimum-so-far" ($O(1)$ time). When all pairs have been found, we output the

"minimum-so-far" as the width of the polygon. The whole process cannot take more than $O(n)$ time since there are only $O(n)$ pairs to begin with.

■

2.3 The maximum distance between two convex polygons

2.3.1 Introduction

This problem arises from the computation of the maximum distance between two finite planar sets. Given two sets of points on the plane, the maximum distance between them is the greatest distance determined by a pair of points, one from each set. This problem occurs often in pattern recognition [26], more specifically in clustering problems. One procedure in particular, the *furthest neighbour clustering algorithm* [26] uses the maximum distance between sets. Hence an efficient algorithm for solving this problem is essential. Of course, the brute force method (given n points in total) is the easiest but yields an $O(n^2)$ algorithm. Bhattacharya and Toussaint [9] presented a somewhat complicated $O(n \log n)$ algorithm in a paper that appeared in 1983. Improving on this, Toussaint and McAlear [86] presented a much simpler $O(n \log n)$ algorithm as a successor [1].

Given the two point sets, the algorithm proposed in [86] determines the convex hulls, and computes the maximum distance between the polygons, using a procedure based on the Rotating Calipers but applied on two polygons. Despite the modifications on the paradigm, the main algorithm remains very simple.

As our interest lies in the use of the Rotating Calipers, the problem studied in this work is that of determining the distance between two convex polygons. It can be easily shown that the maximum distance between two sets of points on the plane is

[1] Ironically, this paper appeared before its predecessor

the maximum distance between their convex hulls. Let us then begin with a formal definition of the problem:

Definition 2.3.1 *Given two convex polygons P and Q, the maximum distance between P and Q (denoted $d_{\max}(P,Q)$) is given by:*

$$d_{\max}(P,Q) = \max\{\text{dist}(p,q),\ \forall p \in P, \forall q \in Q\}$$

2.3.2 Theoretical results

The main result, Theorem 2.3.1 by Toussaint and McAlear [86] introduces the concept of an anti-podal pair between two polygons to characterize the solution to the maximum distance problem.

Definition 2.3.2 *Vertices $p \in P$ and $q \in Q$ determine an anti-podal pair between polygons P and Q if there exist directed lines L_p and L_q through p and q respectively such that:*

- *L_p is a line of support for P and P lies to the right of L_p,*
- *L_q is a line of support for Q and Q lies to the right of L_q,*
- *L_p and L_q are parallel and have opposite direction.*

Note that P and Q need not lie to the right of their respective lines of support, for they could both lie to the left of the lines.

Theorem 2.3.1 *Given two convex polygons P and Q, the maximum distance between P and Q is determined by anti-podal pair* between P *and* Q.

Proof: Suppose the maximum distance between P and Q is determined by points p and q, with $p \in P$ and $q \in Q$. We will show that p and q determine an anti-podal pair between P and Q.

Let $d_{pq} = \text{dist}(p, q)$. Construct the discs C_p and C_q, both of radius d_{pq} and centered at p and q respectively. Obviously $p \in C_q$ and $q \in C_p$.

Since p and q determine the maximum distance between P and Q, then all points of Q are inside C_p:

$$\forall u \in Q,\ \text{dist}(u, p) \leq d_{pq} \Rightarrow u \in C_p$$

Similarly, all points of P are inside C_q. Let us now construct L_p and L_q, the tangent lines to C_q and C_p through p and q, respectively, and L_{pq}, the line through p and q. See Figure 2.7. Since L_p and L_q are tangent lines,

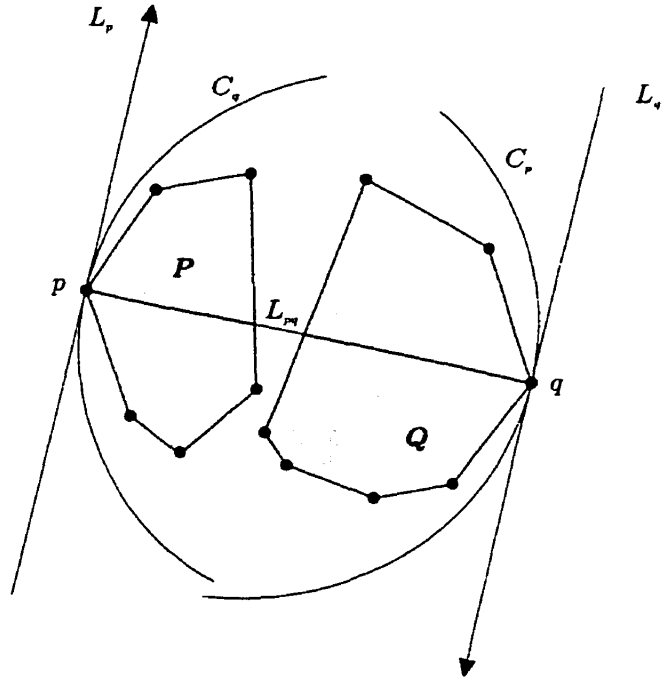


Figure 2.7: Illustrating the proof of Theorem 2.3.1.

we have:

$$L_p \perp L_{pq},\ L_q \perp L_{pq} \Rightarrow L_p \parallel L_q$$

Furthermore, L_p is a line of support for P at p since it is tangent to C_q and P is inside C_q. Similarly, L_q is a line of support for Q at q. Now assign directions to L_p and L_q such that P and Q lie to the right of their respective lines of support. Therefore, L_p and L_q are parallel lines of support in opposite directions, making p and q an anti-podal pair between P and Q. ∎

2.3.3 Algorithm and analysis

Using theorem 2.3.1, Toussaint and McAlear [86] present an algorithm that determines the maximum distance between two convex polygons P and Q. By checking all anti-podal pairs between P and Q, the algorithm's correctness is ensured. So, given polygons P and Q (in standard) form with m and n vertices (in clockwise order) respectively, let $N = m+n$ and consider the following algorithm along with Figure 2.8:

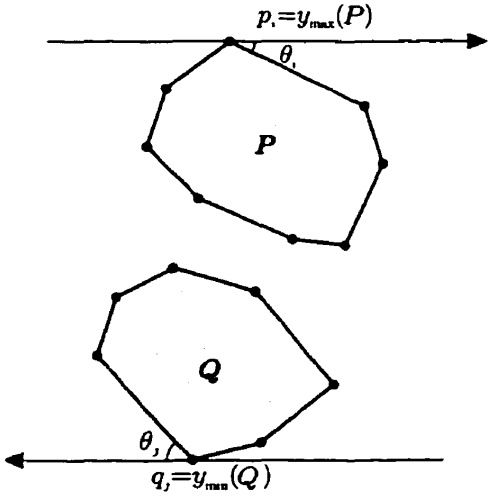


Figure 2.8: Illustrating Algorithm MAXDIST2POLY

Algorithm MAXDIST2POLY

1. Compute the vertex with minimum y-coordinate for P (call it $y_{\min}(P)$) and the one with maximum y-coordinate for Q (call it $y_{\max}(Q)$. This takes $O(N)$ time.

2. Start with two lines of support(i.e. calipers) parallel to the x-axis, touching P and Q at $y_{\max}(P)$ and $y_{\min}(Q)$. Choose a direction of rotation (say clockwise). The lines of support determine two angles θ_i, θ_j, which are computed. (All of this takes $O(1)$ time).

3. Since the lines of support already determine an anti-podal pair between P and Q, compute the distances between the vertices and keep it as the maximum. ($O(1)$ time cost).

4. Compute $\theta = \min(\theta_i, \theta_j)$. (unit time cost).

5. Rotate clockwise by θ, thus making one line of support flush with one edge. Since we have hit a new vertex, one new anti-podal pair between is considered (in the case of parallel edges between P and Q, a maximum of three new anti-podal pairs between the polygons can be considered). Compute all relevant distances, and compare them to our "maximum-so-far". This step is also done in $O(1)$ time.

6. Repeat steps 4–5 until calipers return to original position (or until our total rotation angle is greater than 2π). After each rotation, a new vertex *must* be hit. Since we only have N vertices, and since steps 4 and 5 are both $O(1)$, This step takes $O(N)$ time.

7. Output the "maximum-so-far" as $d_{\max}(P, Q)$.

Now to prove the correctness of the algorithm: two parallel lines of support for P and Q are used. By rotating them by the minimum angle, it is ensured that the next edge (or vertex) is not overlooked. Therefore, the next possible anti-podal pair is always taken in account. Since the lines are rotated until their original position is

reached, it is ensured that *all* anti-podal pairs are considered. Hence, since the maximum distance between anti-podal pairs between P and Q is output, by Theorem 2.3.1, $d_{\max}(P, Q)$ is output.

The running time of the algorithm is dominated by finding the extrema, and by rotating the lines. Each of these processes takes $O(N)$ time. Hence, the total running time is $O(N)$, and we have:

Theorem 2.3.2 *Given two convex polygons P and Q of m and n vertices, respectively, the maximum distance between P and Q can be computed in $O(m+n)$ time.*

2.4 The minimum distance between two convex polygons

2.4.1 Introduction

Analogous to the previous problem, the purpose is now to find the minimum distance between two convex polygons. Applications of this problem are found in pattern recognition and collision avoidance [71, 27].

This problem, unlike the maximum distance, does not stem from computing the minimum distance between two sets of points. In 1981, Avis proved an $\Omega(N \log N)$ lower bound for computing the minimum distance between point sets with a total of N points [5]. Optimal algorithms for this problem were obtained by Toussaint and Bhattacharya [85].

For polygons, the problem has two versions. First: given two convex polygons (which include their interiors), find the minimum distance between them. The second version restricts the minimum distance to occur between vertices of the polygons.

The unrestricted problem is the easier of the two. If the polygons intersect, then the distance is zero. A possible intersection can be detected in $O(\log(m+n))$

time [12, 16]. If the polygons are disjoint, then we are looking for a pair of points on the boundaries of the polygons. In 1981, Schwartz provided a $O(\log m \log n)$ time algorithm to solve the problem [71]. Edelsbrunner later proved a $\Omega(\log m + \log n)$ lower bound, and described an optimal algorithm [27].

The vertex-restricted version is somewhat more complicated. First, the intersecting case must be studied as well as the disjoint case, and even in the overlapping case if one polygon is contained within the other then that case must be studied separately. Toussaint and Bhattacharya applied their algorithm for point sets to solve the non-intersecting case, yielding a $O((m+n)\log(m+n))$ time algorithm. Optimal linear-time algorithms for this case were later independently obtained by McKenna and Toussaint [57] and by Chin and Wang [21]. Toussaint later solved the problem for intersecting convex polygons, obtaining an optimal linear-time algorithm [77, 78].

The minimum distance between convex polygons can be solved using a very elegant application of the Rotating Calipers, which we study here. Although the algorithm has linear time complexity (and is therefore not optimal), its simplicity and ease of implementation still make it a viable solution for this problem, if the input size is not too large. For a reasonable input size, this algorithm should run faster than Edelsbrunner's logarithmic-time solution.

2.4.2 Theoretical results

Let us start as usual with a formal definition of the problem:

Definition 2.4.1 *Given two convex polygons P and Q, the minimum distance between P and Q is given by:*

$$d_{\min}(P,Q) = \min\{\text{dist}(p,q),\ \forall p \in P, \forall q \in Q\}$$

We assume that the polygons have been checked for intersections, and are disjoint.

Theorem 2.4.1 *Given two convex polygons P and Q and a given pair of points (p, q) ($p \in P$, $q \in Q$) such that $d_{\min}(P, Q) = \text{dist}(p, q)$, p and q must either be vertices of P and Q (respectively) or belong to edges of P and Q (respectively).*

Proof: Let us assume without loss of generality that q is an interior point of Q, and that $d_{\min}(P, Q)$ is determined by the pair (p, q). Refer to Figure 2.9. Consider the line segment $[p, q]$. Since q is an interior

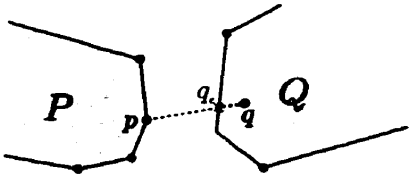


Figure 2.9: Illustrating the proof of Theorem 2.4.1.

point, then $[p, q]$ must intersect Q in a subsegment $[q_e, q] \subset [p, q]$. We have $\text{dist}(p, r) \leq \text{dist}(p, q)$ $\forall r \in [q_e, q]$. And in particular, $\text{dist}(p, q_e) < \text{dist}(p, q)$. Since $q_e \in Q$, this contradicts our assumption that $\text{dist}(p, q) = d_{\min}(P, Q)$. Hence, q cannot be an interior point of Q, so it must either be a vertex of q or belong to an edge of Q. By symmetry of the argument, the same can be established for p. ∎

The next result follows directly, as illustrated in Figure 2.10:

Corollary 2.4.2 *Given two convex polygons P and Q, the minimum distance between P and Q determined by a pair of points (p, q) occurs in three different cases:*

1. *The* vertex-vertex *case where p, q are both vertices of P and Q respectively. (See Figure 2.10 (a)).*

2. *The* vertex-edge *case where one of p, q is a vertex while the other is not a vertex. (See Figure 2.10 (b)).*

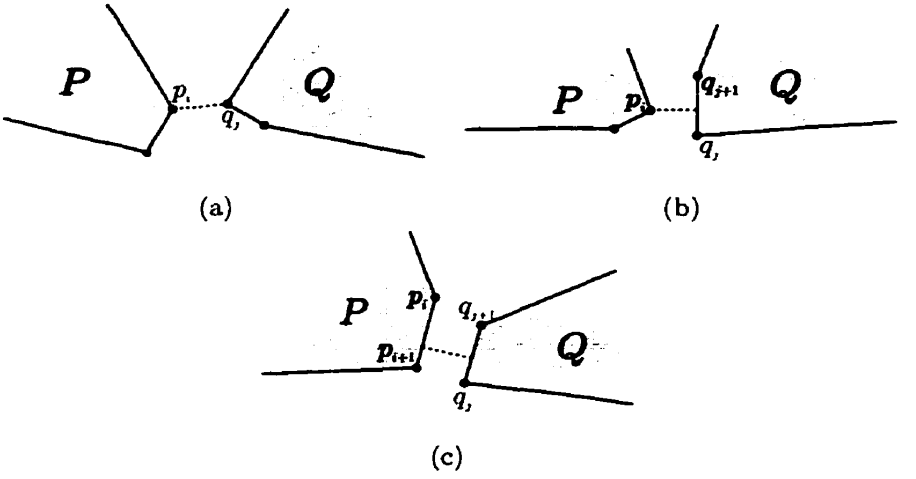


Figure 2.10: The three cases for the minimum distance between P and Q.

3. *The* edge-edge *case where neither p not q are vertices of P and Q. (See Figure 2.10 (c)).*

Theorem 2.4.3 *Given two convex polygons P and Q, with the minimum distance between P and Q determined by a pair of points (p, q) then p and q admit parallel lines of support in opposite directions.*

Proof: Using Corollary 2.4.2 we proceed case by case.

Vertex-vertex case: p and q are vertices of P and Q respectively. Construct the discs C_p centered at p and C_q centered at q, both of radius $\text{dist}(p, q) = d_{\min}(P, Q)$. Therefore, there exists no point of P inside the disc C_q, and there exists no point of Q inside C_p (the existence of such points would contradict (p, q) determining the minimum distance). Hence, all of P (except for p) lies outside of C_q and all of Q (except for q) lies outside of C_p.

Consider L_p, a directed line tangent to C_q at p such that q lies to its left. We claim that L_p is a line of support for P: L_p and P intersect at p. Suppose there exists a point $r \in P$ lying outside C_q but lying left of

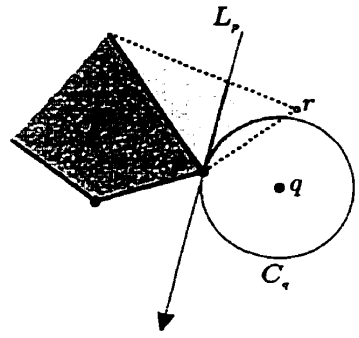


Figure 2.11: Illustrating the proof of Theorem 2.4.3, vertex-vertex case.

L_p. See Figure 2.11. Since P is convex, the segment $[p, r]$ must lie inside P. Now, L_p is also tangent to C_q at p. Therefore, part of $[p, r]$ must lie inside C_q. This implies that there exits a point of P other than p inside C_q, which contradicts our original assumptions. Since no point of P can lie to the left of L_p, all of P lies to the right of L_p. Hence, L_p is a line of support for P at p.

Similarly, consider L_q, a directed line tangent to C_p at q, and such that p lies to its left. As above, we can prove that L_q is a line of support for Q at q and that all of Q lies to the right of L_q. Now, since both L_p and L_q are perpendicular to the segment $[p, q]$, L_p and L_q are parallel.

Therefore, we can say that p and q admit parallel lines of support in opposite directions. See Figure 2.12.

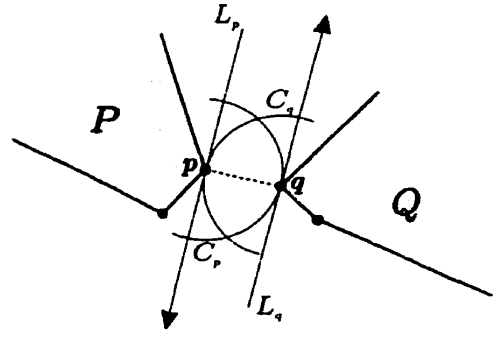


Figure 2.12: Illustrating the proof of Theorem 2.4.3, vertex-vertex case.

Vertex-edge case: assume without loss of generality that p is a vertex of P while q belongs to an edge. Again, by the fact that (p, q) determines $d_{\min}(P, Q)$, there exists no point in the interior of P inside the disc C_q and there exists no point in the interior of Q inside the disc C_p. Thus, all of P lies outside of C_q and all of Q lies outside C_p (again with the exceptions of p and q respectively). As above, let L_p be the directed line tangent to C_q at p and such that q lies to its left, and let L_q be the directed line tangent to C_p at q with p lying to its left. See Figure 2.13.

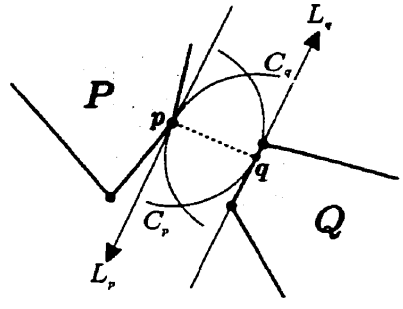


Figure 2.13: Illustrating the proof of Theorem 2.4.3, vertex-edge case.

The argument used above can be applied again in this case to show that L_p is a line of support for P. Now, since $d_{\min}(P, Q)$ represents the distance from the point p to the edge e_q that q lies on, the segment $[p, q]$ must be perpendicular to e_q. Since L_q is tangent to C_q at q, then L_q must also be perpendicular to $[p, q]$. Therefore, L_q and e_q must be parallel. And since they both include q, then e_q lies on L_q. This means L_q is flush with an edge of Q. Then L_q is a line of support for Q at q. And since L_p is perpendicular to $[p, q]$, L_p and L_q are parallel.

Thus, in this case, we can also say that the points determining the minimum distance between P and Q admit parallel lines of support.

Edge-edge case: in this case both p and q lie on edges e_p and e_q of P and Q respectively. We repeat the construction of C_p, C_q, L_p and L_q as in the previous cases. See Figure 2.14.

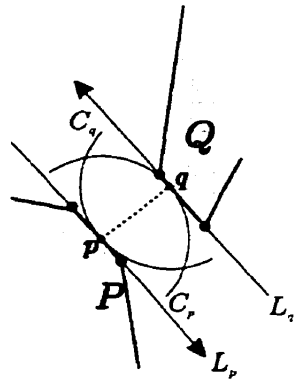


Figure 2.14: Illustrating the proof of Theorem 2.4.3, edge-edge case.

As above, it can easily be shown that L_p and L_q are flush with the edges where p and q lie respectively, thus making them lines of support for P and Q. Furthermore, as previously discussed, L_p and L_q are parallel and have opposite direction.

Hence, p and q admit parallel lines of support.

■

2.4.3 Algorithm and analysis

Theorem 2.4.3 restricts the pair determining the minimum distance to an anti-podal pair of points. Thus, the Rotating Calipers can be used very much the same way as in Algorithm MAXDIST2POLY. The only subtlety is the fact that anti-podal vertex-vertex, edge-vertex and edge-edge cases must all be considered. Otherwise we have a very similar algorithm. Again, let us assume we are given two disjoint convex polygons P and Q (in standard from), with m and n vertices each (given in clockwise order), and $N = m + n$.

Algorithm MINDIST2POLY

1. Compute the minimum and maximum y-coordinate extreme points $y_{\min}(P)$ and $y_{\max}(Q)$ for P and Q respectively. This takes $O(N)$ time.

2. Start with two lines of support (i.e. calipers) parallel to the x-axis, touching P and Q at $y_{\max}(P)$ and $y_{\min}(Q)$, and directed such that P and Q lie to the right of their respective lines of support. The lines of support lie on vertices $p_i \in P$ and $q_j \in Q$, and determine two angles θ_i, θ_j, which are computed. (All of this takes $O(1)$ time).

3. Since the lines of support already determine an anti-podal pair between P and Q, compute the distance between the vertices and keep it as the minimum. ($O(1)$ time).

4. Compute $\theta = \min(\theta_i, \theta_j)$. ($O(1)$ time).

5. Rotate the lines of support clockwise about p_i and q_j by an angle θ, thus making one line of support flush with one edge.

   Say we have only hit vertex $q' \in Q$. Compute $\text{dist}(p_i, q')$ and compare it to our "minimum-so-far" ($O(1)$ time). Now, consider a point $p^{\perp}$ such that line segment $[p, p^{\perp}]$ and edge $[q_j, q']$ are orthogonal. Compute the intersection of lines $L(p_i, p^{\perp})$ and $L(q_j, q')$. If it exists and is equal to q_{int}, $q_{\text{int}} \neq q_j, q'$, then compute $\text{dist}(p_i, q_{\text{int}})$ and also compare it to our "minimum-so-far". A similar procedure is used if a vertex $p' \in P$ is hit instead. If, on the other hand, we have parallel edges between P and Q, in which case both vertices p' *and* q' are hit, then the following procedure must be used: as above, points $p^{\perp}$ and $q^{\perp}$ are considered to compute the possible intersections $q_{\text{int}} = L(p_i, p^{\perp}) \cap L(q_j, q')$ and $p_{\text{int}} = L(q_j, q^{\perp}) \cap L(p_i, p')$. If one intersection is found, then the other need not be computed. If an intersection exists, then the relevant distance q_{int} or p_{int} (this distance is the $[p_i, p'] - [q_j, q']$ edge-edge orthogonal distance) is considered as a possible new minimum. Of course, the usual $\text{dist}(p_i, q')$ and $\text{dist}(q_j, p')$ must be computed as compared to our "minimum-so-far". Furthermore, (p', q') also forms an anti-podal vertex-vertex pair and thus $\text{dist}(p', q')$ is also computed and considered. Finally, when all distances are obtained and necessary comparisons done, p_i, q_j, or both might need to be updated to p' and q' respectively.

The time complexity of this step remains $O(1)$ no matter which case is considered (a maximum of four distance computations are necessary in the case of parallel edges, still a $O(1)$ operation). Updates are of course $O(1)$, so the total cost of this step is unit-time.

6. Repeat steps 4–5 until calipers return to original position (or until our total rotation angle is greater than 2π). After each rotation, a new vertex *must* be hit. Since we only have N vertices, and since steps 4 and 5 are both $O(1)$, This step takes $O(N)$ time.

7. Output the "minimum-so-far" as $d_{\min}(P, Q)$.

The correctness of the algorithm follows from Theorem 2.4.3. The running time is dominated by steps 1 and 6. Both are $O(N)$, as discussed in the description of the algorithm. Hence MINDIST2POLY has linear time complexity. We conclude this section with the following result:

Theorem 2.4.4 *Given convex polygons P and Q with m and n vertices respectively, Algorithm MINDIST2POLY computes the minimum distance between P and Q in $O(m + n)$ time.*

Chapter 3

Enclosing Rectangles

This chapter is devoted to the study of the "smallest-box" problem. Given a convex polygon, the goal is to compute an enclosing rectangle, minimizing the area or the perimeter. Theoretical results and algorithms are given for both these problems, as well as a specific case highlighting their difference. Finally, a new problem is studied in detail: the width-determined enclosing rectangle.

3.1 The minimum-area enclosing rectangle

3.1.1 Introduction

The first and most commonly known version of the "smallest-box" problem involves finding the minimum area rectangle enclosing a convex polygon. Many applications can be found in packing and optimal layout problems [32, 35]. Consider for instance trying to find the minimum volume box (with fixed horizontal orientation) for an object. A bird's eye view picture is taken of the object, and its height is determined. A two-dimensional convex-hull is computed, and using the minimum-area rectangle, one can compute a minimal-volume box encasing the object.

Freeman and Shapira [32] studied the more general problem of encasing an arbitrary closed curve. The process is divided into first determining a minimum-perimeter convex polygon enclosing the curve, and then computing the enclosing rectangle for that polygon. The algorithm for the latter part has quadratic time complexity. In 1980, Toussaint provided a first $O(n)$ algorithm [80] and presented a second solution [81] using a result from Freeman and Shapira's paper. The latter, which we will focus on, uses a variation on the original Rotating Calipers, in which two pairs of orthogonal lines of support are used.

3.1.2 Theoretical results.

Let us first give a definition of the problem at hand. As usual, we assume polygons include their interior.

Definition 3.1.1 *Given a convex polygon P, a rectangle R, such that $\forall p \in P$, $p \in R$ is called an* enclosing rectangle *for P. If* area$(R) \leq$ area(R') *for all enclosing rectangles R', then R is an* minimum-area enclosing rectangle *for P.*

Intuitively, in order to minimize the area, the rectangle's edges would have to touch the polygon. Freeman and Shapira [32] give a result further constraining the possible minimal area rectangles to a finite set:

Theorem 3.1.1 *The rectangle of minimum area enclosing a convex polygon has a side collinear with one of the edges of the polygon.*

> **Proof**: Let us say we are given a convex polygon P, and let us assume the smallest-box is given and that it does *not* have one side collinear with one of P's edges. Therefore it only touches P at four vertices p_i, p_j, p_k, and p_l. Refer to Figure 3.1. We claim that it is always possible to find a smaller enclosing rectangle.

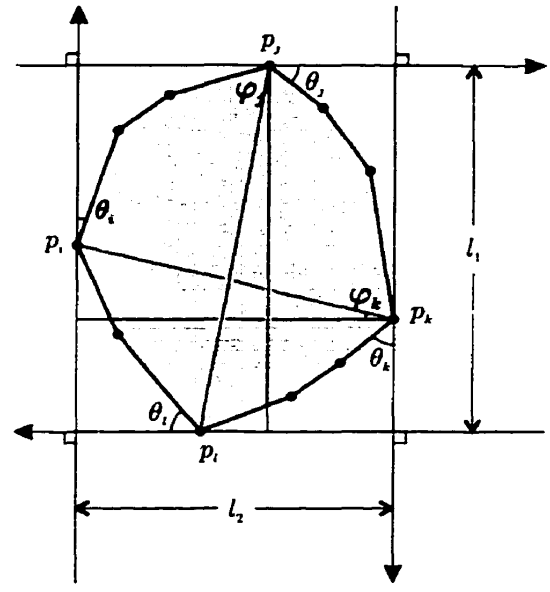


Figure 3.1: An example of a rectangle enclosing P.

Let us first define A as the area of the enclosing rectangle. We have $A = l_1 l_2$. Furthermore, let $d_{ik} = \text{dist}(p_i, p_k)$, and $d_{jl} = \text{dist}(p_j, p_l)$. We have

$$l_1 = d_{jl}\cos(\varphi_j)$$

$$l_2 = d_{ik}\cos(\varphi_k)$$

As we saw in section 2.2, both l_1 and l_2 can be decreased by rotating their corresponding lines of support in their respective preferred direction of rotation. Hence two cases emerge: case 1, where l_1 and l_2 can be decreased by rotating all lines of support in the same direction (they have the same preferred direction of rotation); and case 2, where rotating in a given direction decreases one length but increases the other (different preferred directions of rotation).

Case 1: an example of this case is shown in Figure 3.1. By rotating all lines of support counterclockwise by some (positive) angle η, both l_1 and l_2 are decreased. After rotating, we have a new box of area A' determined

by edges of length l_1' and l_2' where

$$l_1' = d_{jl}\cos(\varphi_j + \eta) \Rightarrow l_1' < l_1$$

$$l_2' = d_{ik}\cos(\varphi_k + \eta) \Rightarrow l_2' < l_2$$

$$\Rightarrow A' = l_1' l_2' < A$$

Hence in this case it is always possible to find a smaller enclosing box.

Case 2: here the preferred directions of rotation are different. Let us define δ_j as the maximum angle we can rotate the lines of support in l_1's preferred direction of rotation before we hit an edge, and similarly we define δ_k for l_2. Let $\delta = \min(|\delta_j|, |\delta_k|)$. Assume without loss of generality that the preferred direction of rotation for l_1 is clockwise and the preferred direction of rotation for l_2 is counterclockwise.

If we rotate clockwise, we obtain new lengths l_1', l_2' and a new area A_C, given by:

$$A_C = l_1' l_2' \quad \begin{cases} l_1' = d_{jl}\cos(\varphi_j + \delta) \\ l_2' = d_{ik}\cos(\varphi_k - \delta) \end{cases}$$

If we rotate counterclockwise, we have:

$$A_{CC} = l_1'' l_2'' \quad \begin{cases} l_1'' = d_{jl}\cos(\varphi_j - \delta) \\ l_2'' = d_{ik}\cos(\varphi_k + \delta) \end{cases}$$

Let us compare A with both A_C and A_{CC}, recalling that

$$A = d_{jl} d_{ik} \cos\varphi_j \cos\varphi_k > 0$$

$$\begin{aligned} \frac{A_C}{A} &= \frac{\cos(\varphi_j + \delta)\cos(\varphi_k - \delta)}{\cos\varphi_j \cos\varphi_k} \\ &= \frac{(\cos\varphi_j \cos\delta - \sin\varphi_j \sin\delta)(\cos\varphi_k \cos\delta + \sin\varphi_k \sin\delta)}{\cos\varphi_j \cos\varphi_k} \\ &= \frac{\cos\varphi_j \cos\varphi_k \cos^2\delta + \cos\varphi_j \sin\varphi_k \cos\delta \sin\delta}{\cos\varphi_j \cos\varphi_k} \end{aligned}$$

$$
\begin{aligned}
& - \frac{\sin\varphi_j \cos\varphi_k \cos\delta \sin\delta + \sin\varphi_j \sin\varphi_k \sin^2\delta}{\cos\varphi_j \cos\varphi_k} \\
= \; & \cos^2\delta + \tan\varphi_k \cos\delta \sin\delta - \tan\varphi_j \cos\delta \cos\delta \\
& - \tan\varphi_j \tan\varphi_k \sin^2\delta \\
= \; & \cos^2\delta + (\tan\varphi_k - \tan\varphi_j)\cos\delta \sin\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta
\end{aligned}
$$

$$
\begin{aligned}
\frac{A_{CC}}{A} &= \frac{\cos(\varphi_j - \delta)\cos(\varphi_k + \delta)}{\cos\varphi_j \cos\varphi_k} \\
&= \frac{(\cos\varphi_j \cos\delta + \sin\varphi_j \sin\delta)(\cos\varphi_k \cos\delta - \sin\varphi_k \sin\delta)}{\cos\varphi_j \cos\varphi_k} \\
&= \frac{\cos\varphi_j \cos\varphi_k \cos^2\delta - \cos\varphi_j \sin\varphi_k \cos\delta \sin\delta}{\cos\varphi_j \cos\varphi_k} \\
&\quad + \frac{\sin\varphi_j \cos\varphi_k \cos\delta \sin\delta - \sin\varphi_j \sin\varphi_k \sin^2\delta}{\cos\varphi_j \cos\varphi_k} \\
&= \cos^2\delta - \tan\varphi_k \cos\delta \sin\delta + \tan\varphi_j \cos\delta \sin\delta \\
&\quad - \tan\varphi_j \tan\varphi_k \sin^2\delta \\
&= \cos^2\delta + (\tan\varphi_j - \tan\varphi_k)\cos\delta \cos\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta
\end{aligned}
$$

If $A_C/A < 1$ then we rotate clockwise and we can obtain a smaller enclosing rectangle, and we are done. However, if $A_C/A \geq 1$, then we have:

$$
\begin{aligned}
& \frac{A_C}{A} \geq 1 \\
& \quad \Leftrightarrow \quad \cos^2\delta + (\tan\varphi_k - \tan\varphi_j)\cos\delta \sin\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta \geq 1 \\
& \quad \Leftrightarrow \quad (\tan\varphi_j - \tan\varphi_k)\cos\delta \sin\delta \leq \cos^2\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta - 1
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \frac{A_{CC}}{A} &\leq 2(\cos^2\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta) - 1 \\
&\leq 2(1 - \sin^2\delta - \tan\varphi_j \tan\varphi_k \sin^2\delta) - 1 \\
&\leq 1 - 2(1 + \tan\varphi_j \tan\varphi_k)\sin^2\delta \\
&< 1
\end{aligned}
$$

since $0 < \varphi_j, \varphi_k < \pi/2 \Rightarrow \tan\varphi_j \tan\varphi_k > 0$. And hence we have $A_{CC}/A < 1$, meaning that by rotating counterclockwise, we can obtain a smaller enclosing rectangle.

Therefore, in both cases, it is possible to find a smaller enclosing box, which contradicts our starting assumption that the rectangle had minimum area. ∎

3.1.3 Algorithm and analysis

Theorem 3.1.1 implies that for a given n-vertex polygon, the number of candidates for smallest enclosing rectangle is at most n. This greatly simplifies our task of trying to solve the smallest-box problem.

Freeman and Shapira [32] use this result and determine all n rectangles (assuming no pair of edges is parallel). Their procedure takes $O(n)$ time for each rectangle, yielding a $O(n^2)$ algorithm in the end. Toussaint [81] did away with lengthy calculations by using a modified algorithm based on the Rotating Calipers.

Indeed, theorem 3.1.1 implies that two (parallel) edges of the rectangle are determined by a vertex-edge anti-podal pair, while the two other edges are determined by anti-podal pairs. This is obvious since any enclosing rectangle with edges tangent to the polygon determines two sets of parallel lines of support.

Toussaint [81] presents the following algorithm applied to a convex polygon P (assume it is given in standard form, as usual):

1. Find the vertices of P with minimum and maximum x and y-coordinates. These determine two sets of "calipers" (in other words, two sets of parallel lines of support), parallel to the x and y axes, thus forming a rectangle enclosing P. These lines determine four angles, θ_i, θ_j, θ_k, and θ_l. See Figure 3.1.
2. Let $\theta = \min(\theta_i, \theta_j, \theta_k, \theta_l)$.
3. Rotate by θ, thus making the rectangle flush with one edge.
4. Compute the area of the rectangle. If the area is smaller than the previous rectangle's, keep the new rectangle as our new "minimum".

5. Recompute θ_i, θ_j, θ_k, and θ_l.

6. Repeat steps 2–5, until we have rotated calipers a total angle greater than $\pi/4$.

Theorem 3.1.2 *The smallest enclosing rectangle of a polygon can be found in $O(n)$ time.*

> **Proof**: After each rotation, the "box" is flush with one edge, and the area of that rectangle is considered. Every rectangle flush with P is considered: since we rotate an angle greater than $\pi/4$. Furthermore, no rectangle is considered twice, since any rotation past the $\pi/4$ limit only yields previously considered rectangles. This is due to the fact after rotating an angle equal to $\pi/4$, one set of calipers will be in the original position of the other set, and vice versa. Finally, since there are n edges, there can be at most n such rectangles, and thus the algorithm runs in $O(n)$ time. ∎

3.2 The minimum-perimeter enclosing rectangle

3.2.1 Introduction

Similar to the above, this problem consists of finding the rectangle of minimum perimeter enclosing a given convex polygon. In fact, the rectangles often coincide. This is not always true and a polygon exemplifying this is provided in section 3.2.2. Continuing with the sample application of the previous section, one could use the concept of smallest-perimeter enclosing rectangle (in conjunction with the smallest-area rectangle) to minimize the surface area of a box encasing an object.

This problem has been much less studied in the literature than its area counterpart. In 1987, DePano presents an algorithm for this problem [25], using the Rotating Calipers paradigm much in the same way as Toussaint had with the minimum-area version. Due to the similarity of the problems and the solutions, it is important at this point to differentiate these problems by giving a concrete example.

3.2.2 Minimum-perimeter versus minimum-area enclosing rectangles

Since decreasing the area of a rectangle usually also decreases its perimeter, for many polygons, the two versions of enclosing rectangles coincide. But this is not always true. Consider the polygon shown in Figure 3.2(a). Its minimum-area enclosing

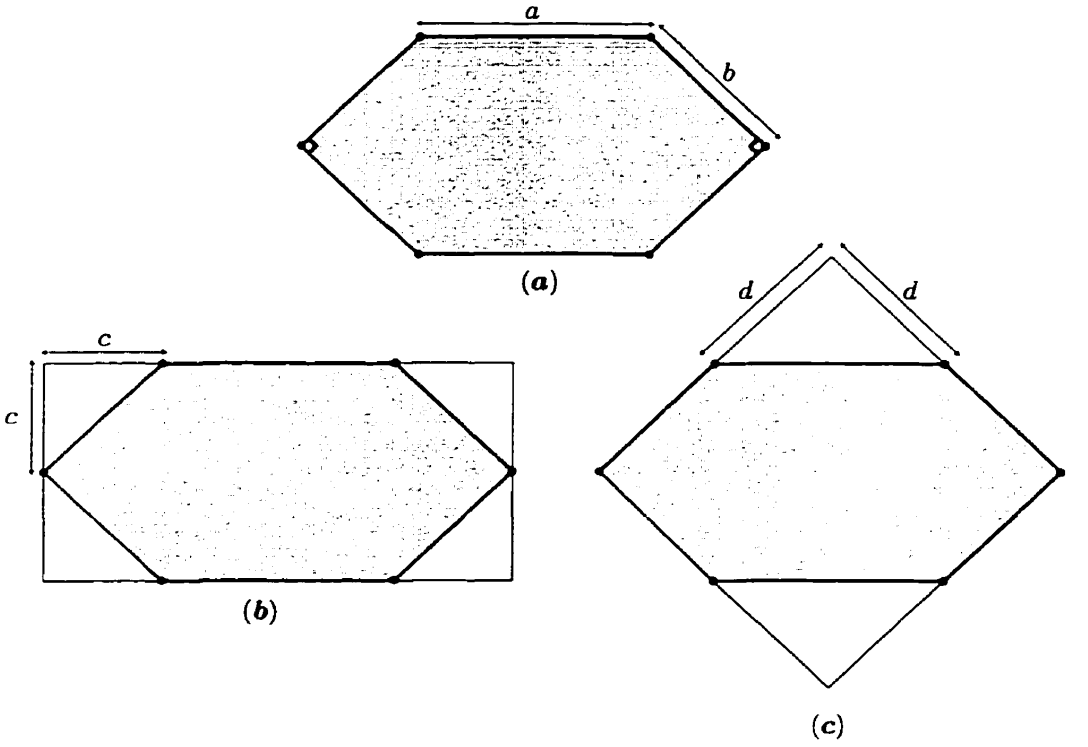


Figure 3.2: A polygon and its enclosing rectangles.

rectangle is shown in Figure 3.2(b) and its minimum-perimeter enclosing rectangle in (c). Finally, the dimensions are constrained such that

$$\frac{a}{2} < b < \frac{a}{\sqrt{2}}.$$

Let A_A and A_P denote the respective areas and P_A, P_P the respective perimeters of the minimum-area and minimum-perimeter enclosing rectangles. First,

$$c = \frac{b}{\sqrt{2}}, \quad d = \frac{a}{\sqrt{2}}.$$

Let us determine all areas and perimeters.

$$\begin{aligned}
A_A &= 2c(a+2c) \\
&= 2ac+4c^2 \\
&= \sqrt{2}ab+2b^2 \\
P_A &= 2a+8c \\
&= 2a+4\sqrt{2}b \\
A_P &= (b+d)^2 \\
&= (b+\frac{a}{\sqrt{2}})^2 \\
&= \frac{a^2}{2}+\sqrt{2}ab+b^2 \\
P_P &= 4(b+d) \\
&= 4(\frac{a}{\sqrt{2}}+b) \\
&= 2\sqrt{2}a+4b
\end{aligned}$$

Note that the polygon only has two distinct enclosing rectangles that are flush with at least one edge (Theorems 3.1.1 and 3.2.1 require this), so that only these two rectangles need to be considered. We show now that $A_A < A_P$ and $P_P < P_A$ if the bounds for b are respected.

$$\begin{aligned}
A_P - A_A &= \frac{a^2}{2}+\sqrt{2}ab+b^2-\sqrt{2}ab-2b^2 \\
&= \frac{a^2}{2}-b^2 \\
&> \frac{a^2}{2}-\frac{a^2}{2} \quad (b<\frac{a}{\sqrt{2}}) \\
\Rightarrow A_P - A_A &> 0 \\
\Rightarrow A_A &< A_P
\end{aligned}$$

For the perimeters, we have:

$$P_A - P_P = 2a+4\sqrt{2}b-2\sqrt{2}a-4b$$

$$
\begin{aligned}
&= 2a(1-\sqrt{2}) - 4b(1-\sqrt{2}) \\
&= 2(1-\sqrt{2})(a-2b) \\
&= 2(\sqrt{2}-1)(2b-a) \\
&= 4(\sqrt{2}-1)(b-\frac{a}{2}) \\
&> 0 \quad (b > \frac{a}{2}) \\
\Rightarrow P_A - P_P &> 0 \\
\Rightarrow P_A &> P_P
\end{aligned}
$$

Let us now analyze DePano's results and algorithm.

3.2.3 Theoretical results

Definition 3.2.1 *Given a convex polygon P and a rectangle R enclosing P, if we have* perimeter(R) $\leq$ perimeter(R') *for all enclosing rectangles R', then R is a* minimum-perimeter enclosing rectangle *for P.*

DePano [25] shows that the smallest-perimeter enclosing rectangle holds the same property as its area counterpart:

Theorem 3.2.1 *The rectangle of minimum perimeter enclosing a convex polygon has a side collinear with one of the edges of the polygon.*

> **Proof**: Suppose we are given a convex polygon P and its minimum perimeter enclosing rectangle such that none of the polygon's sides is flush (or collinear) with the rectangle's. Again, we refer to Figure 3.1, where P touches the rectangle at vertices p_I, p_j, p_k and p_l. We claim that it is possible to find an enclosing rectangle of smaller perimeter.

Referring to Figure 3.1, let us define D as the perimeter of the enclosing rectangle. We have $D = 2(l_1 + l_2)$. Let $d_{ik} = \text{dist}(p_i, p_k)$ and $d_{jl} = \text{dist}(p_j, p_l)$. We have

$$\left.\begin{aligned} l_1 &= d_{jl}\cos(\varphi_j) \\ l_2 &= d_{ik}\cos(\varphi_k) \end{aligned}\right\} \quad 0 \leq \varphi_j, \varphi_k < \frac{\pi}{2}$$

Again, by rotating the lines of support determining l_1 and l_2 (in their preferred direction of rotation), it is possible to decrease their values, and we have two cases to consider:

Case 1: l_1 and l_2 have the same preferred direction of rotation (for example, in Figure 3.1, counterclockwise) and by rotating the lines of support in this direction by some positive angle η we can find a new enclosing rectangle, with sides l_1' and l_2', and perimeter D' where

$$l_1' = d_{jl}\cos(\varphi_j + \eta) \Rightarrow l_1' < l_1$$

$$l_2' = d_{ik}\cos(\varphi_k + \eta) \Rightarrow l_2' < l_2$$

$$\Rightarrow D' = 2(l_1' + l_2') < D$$

Thus in this case, we can find an enclosing rectangle with smaller perimeter.

Case 2: l_1 and l_2 have opposite preferred direction of rotation. Define δ_j as the maximum angle we can rotate the lines of support in l_1's preferred direction of rotation before we're flush with an edge, and similarly define δ_k for l_2. Let $\delta = \min(|\delta_j|, |\delta_k|)$. Without loss of generality, assume that the preferred direction of rotation for l_1 is clockwise and the preferred direction of rotation for l_2 is counterclockwise.

Rotating the lines of support clockwise, we obtain new lengths l_1', l_2' and a new perimeter D_C given by:

$$D_C = 2(l_1' + l_2') \quad \left\{\begin{aligned} l_1' &= d_{jl}\cos(\varphi_j + \delta) \\ l_2' &= d_{ik}\cos(\varphi_k - \delta) \end{aligned}\right.$$

And rotating counterclockwise, we obtain:

$$D_{CC} = 2(l_1'' + l_2'') \quad \begin{cases} l_1'' = d_{jl}\cos(\varphi_j - \delta) \\ l_2'' = d_{ik}\cos(\varphi_k + \delta) \end{cases}$$

Let us now rewrite the perimeters in function of δ and simplify, keeping in mind $\cos\varphi_j, \cos\varphi_k \neq 0$:

$$\begin{aligned} D_C &= 2(l_1' + l_2') \\ &= 2(d_{jl}\cos(\varphi_j + \delta) + d_{ik}\cos(\varphi_k - \delta)) \\ &= 2\left(\frac{l_1}{\cos(\varphi_j)}(\cos\varphi_j\cos\delta - \sin\varphi_j\sin\delta)\right. \\ &\quad \left. + \frac{l_2}{\cos\varphi_k}(\cos\varphi_k\cos\delta + \sin\varphi_k\sin\delta)\right) \\ &= 2(l_1(\cos\delta - \tan\varphi_j\sin\delta) + l_2(\cos\delta + \tan\varphi_k\sin\delta)) \\ &= 2(l_1 + l_2)\cos\delta + 2(l_2\tan\varphi_k - l_1\tan\varphi_j)\sin\delta \end{aligned}$$

Let $C = 2(l_1\tan\varphi_j - l_2\tan\varphi_k)$

We have

$$D_C = D\cos\delta - C\sin\delta$$

As for D_{CC}:

$$\begin{aligned} D_{CC} &= 2(l_1'' + l_2'') \\ &= 2(d_{jl}\cos(\varphi_j - \delta) + d_{ik}\cos(\varphi_k + \delta)) \\ &= 2\left(\frac{l_1}{\cos(\varphi_j)}(\cos\varphi_j\cos\delta + \sin\varphi_j\sin\delta)\right. \\ &\quad \left. + \frac{l_2}{\cos\varphi_k}(\cos\varphi_k\cos\delta - \sin\varphi_k\sin\delta)\right) \\ &= 2(l_1(\cos\delta + \tan\varphi_j\sin\delta) + l_2(\cos\delta - \tan\varphi_k\sin\delta)) \\ &= 2(l_1 + l_2)\cos\delta + 2(l_1\tan\varphi_j - l_2\tan\varphi_k)\sin\delta \\ &= D\cos\delta + C\sin\delta \end{aligned}$$

If $D_C/D < 1$ we are done. Now, suppose $D_C/D \geq 1$. We have:

$$\frac{D\cos\delta - C\sin\delta}{D} \geq 1$$

Let $K = C/D$. Note that $D \neq 0$. The above inequality simplifies to:

$$\cos\delta - K\sin\delta \geq 1$$

$$\Leftrightarrow K\sin\delta \leq \cos\delta - 1$$

Keeping in mind our assumption, let us now turn to D_{CC}:

$$\begin{aligned} \frac{D_{CC}}{D} &= \frac{D\cos\delta + C\sin\delta}{D} \\ &= \cos\delta + K\sin\delta \\ &\leq 2\cos\delta - 1 \\ &< 2 - 1 = 1 \qquad (\cos\delta < 1) \\ \Rightarrow \frac{D_{CC}}{D} &< 1 \end{aligned}$$

Hence, either D_C or D_{CC} ends up being smaller than D. In other words, we can always rotate the lines of support to find an enclosing rectangle with smaller perimeter.

In both cases, our assumption leads to a contradiction. Hence, the minimum perimeter enclosing rectangle must have an edge flush with the polygon. ∎

3.2.4 Algorithm and analysis

Paralleling Toussaint [81], DePano gives an algorithm using the Rotating Calipers for finding the minimum area enclosing rectangle, which we present here. Given a convex n-gon P in standard form:

1. Find vertices of P with minimum and maximum x and y-coordinates. These determine two sets of "calipers" (in other words, two sets of parallel lines of support), parallel to the x and y axes, thus forming a rectangle enclosing P. These lines determine four angles, θ_i, θ_j, θ_k, and θ_l. See Figure 3.1.

2. Let $\theta = \min(\theta_i, \theta_j, \theta_k, \theta_l)$.

3. Rotate by θ, thus making the rectangle flush with one edge.

4. Compute the perimeter of the rectangle. If the perimeter is smaller than the previous rectangle's, keep the new rectangle as our new "minimum".

5. Recompute θ_i, θ_j, θ_k, and θ_l.

6. Repeat steps 2–5, until we've rotated calipers a total angle greater than $\pi/4$.

7. Output minimum perimeter enclosing rectangle.

Hence the main result:

Theorem 3.2.2 *The minimum perimeter enclosing rectangle of a convex polygon can be found in $O(n)$ time.*

> **Proof**: After each rotation, the rectangle is flush with one edge of the polygon, and its perimeter is determined. Since the algorithm performs a total rotation with angle greater than $\pi/4$, every rectangle flush with P is considered. Furthermore, no rectangle is considered twice, since the algorithm stops right after the rectangle's original position is reached (rotation with an angle equal to $\pi/4$ yields one set of calipers in the original position of the other set). Finally, since there are n edges, there can be at most n distinct rectangles flush with edges of P. Thus the algorithm runs in $O(n)$ time. ∎

3.3 Width-determined enclosing rectangle

It has already been shown that the width of a convex polygon P is determined by a vertex-edge pair of that polygon (see section 2.2, Lemma 2.2.1). This pair is equivalent

to two parallel lines of support L_1 and L_2. Let us consider the lines of support of P in the direction perpendicular to L_1 and L_2. Call these $L_1^{\perp}$ and $L_2^{\perp}$. Together, these four lines determine an enclosing rectangle for P which is flush with one of the polygon's edges.

Thus, to every polygon Q is associated the width-determined enclosing rectangle, which shall be denoted R_W, the minimum-area enclosing rectangle, which shall be denoted $R_{\min A}$, and the minimum perimeter enclosing rectangle, which shall be denoted $R_{\min P}$.

For many polygons, in fact, these three rectangles coincide. D.T. Lee [52] raised the following interesting question:

Question 3.3.1 *Given a convex polygon Q, are $R_{\min A}$ and R_W the same?*

Naturally, this question raises another:

Question 3.3.2 *Are $R_{\min P}$ and R_W the same?*

If yes, then computing the minimal enclosing rectangles for a convex polygon might become a much simpler task.

The answer to both questions, however, is no, as the following case demonstrates.

Consider Q, a convex polygon, with the following vertices: $(0,0)$, $(0,1)$, $(a,1)$, $(1,a)$, and $(1,0)$. See Figure 3.3.

The polygon Q is a square with a corner cut off, whose size is determined by the parameter a, and that has width $w = b + c$. We need $w \leq 1$ for our example to make sense. Obviously, w depends on a, and this restriction on w is what determines the ranges of possible values for a. First, we have $b^2 + b^2 = 1$, or

$$b = \frac{1}{\sqrt{2}}.$$

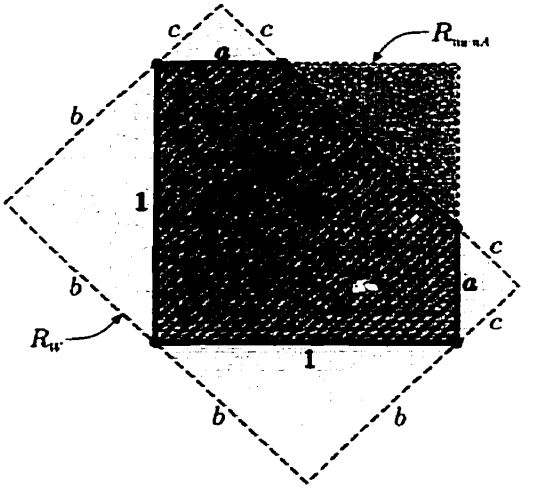


Figure 3.3: A polygon with corresponding $R_{\min A}$ and R_W.

We also have $c^2 + c^2 = a^2$, or

$$c = \frac{a}{\sqrt{2}}.$$

Finally, $w = b + c \leq 1$, in other words

$$\frac{a+1}{\sqrt{2}} \leq 1,$$

i.e., $0 \leq a \leq \sqrt{2} - 1$.

Note that only two rectangles flush with one of Q's edges exist: the unit square (corresponding to $R_{\min A}$), and the rectangle flush with the slanted edge (corresponding to R_W).

$R_{\min A}$ always has an area of one, while R_W has an area $A = 2b(b + c)$, which ranges from 1 (for $a = 0$) to

$$2\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}} + \frac{\sqrt{2}-1}{\sqrt{2}}\right) = \sqrt{2}.$$

Thus for $0 < a \leq \sqrt{2} - 1$, R_W and $R_{\min A}$ are distinct, thus answering Question 3.3.1.

In this case, $R_{\min P}$ coincides with $R_{\min A}$. The perimeter of $R_{\min P}$ is four, while perimeter$(R_W) = 3\sqrt{2} + \sqrt{2}a$, which ranges from $3\sqrt{2} \approx 4.243$ (for $a = 0$) to

$$3\sqrt{2} + 2 - \sqrt{2} = 2(\sqrt{2} + 1) \approx 4.828$$

Thus R_W and $R_{\min P}$ are distinct for $0 \leq a \leq \sqrt{2} - 1$.

3.3.1 Bounds on the error

Consider for a given convex polygon Q, the minimum-area rectangle $R_{\min A}$, the minimum-perimeter rectangle $R_{\min P}$ and the width-determined enclosing rectangle R_W. Consider the ratios

$$\varepsilon_A(Q) = \frac{\text{area}(R_W)}{\text{area}(R_{\min A})}$$

and

$$\varepsilon_P(Q) = \frac{\text{perim}(R_W)}{\text{perim}(R_{\min P})}.$$

Finally, let

$$E_A = \max_{\forall Q} \varepsilon_A(Q),$$

and

$$E_P = \max_{\forall Q} \varepsilon_P(Q).$$

In other words, E_A and E_P describe the errors in the worst-case scenario when using the width of a polygon to determine its enclosing rectangles.

These ratios are meaningful in that they give a preliminary idea of how significant the error can be if one decides to use the width-determined enclosing rectangle.

Finding the exact value of E_A and E_P would imply finding every possible polygon for which R_W has greater area than $R_{\min}$ and comparing their areas and perimeters. It is much more practical to find bounds on E_A and E_P to get at least a range for its value. We first determine the bounds on E_A.

1. Bounds on E_A: A lower bound for E_A has already been determined. Our initial example polygon described above had $\text{area}(R_{\min A}) = 1$ and $\text{area}(R_W) = \sqrt{2}$. Thus, in that case, $\varepsilon_A(Q) = \sqrt{2}$. We know therefore, that $E_A \geq \sqrt{2}$.

Now to find an upper bound. Consider for arbitrary polygon Q the associated $\varepsilon_A(Q)$. Let $R_{\min A}$ have sides l_1 and l_2 (with $l_2 \geq l_1$), and let R_W have sides $l_1' = \text{width}(Q)$ and l_2'. Then we have:

$$\begin{aligned} \varepsilon_A(Q) &= \frac{\text{area}(R_W)}{\text{area}(R_{\min A})} \\ &= \frac{\text{width}(Q)l_2'}{l_1 l_2} \end{aligned}$$

l_1 and l_2' are both determined by pairs of parallel lines of support for Q. Therefore $l_2' \leq \text{diam}(Q)$ (by Theorem 2.1.1). Furthermore, width(Q) is the minimum distance between parallel lines of support (Definition 2.2.1), so $l_1 \geq \text{width}(Q)$ and

$$\frac{\text{width}(Q)}{l_1} \leq 1.$$

Hence

$$\begin{aligned} \varepsilon_A(Q) &= \frac{\text{width}(Q)l_2'}{l_1 l_2} \\ &\leq \frac{\text{diam}(Q)}{l_2} \end{aligned}$$

Now, consider $R_{\min A}(Q)$. Since the rectangle encloses Q, the maximum distance between any pair of points of Q is no greater than the diagonal distance of $R_{\min A}(Q)$. Therefore, $\text{diam}(Q) \leq \sqrt{{l_1}^2 + {l_2}^2}$. We have:

$$\begin{aligned} \varepsilon_A(Q) &\leq \frac{\text{diam}(Q)}{l_2} \\ &\leq \frac{\sqrt{{l_1}^2 + {l_2}^2}}{l_2} \\ &\leq \frac{\sqrt{{l_1}^2 + {l_2}^2}}{\frac{l_1+l_2}{2}} \quad \left(l_2 \geq l_1 \Rightarrow l_2 \geq \frac{l_1+l_2}{2}\right) \\ &\leq 2\frac{\sqrt{(l_1+l_2)^2}}{l_1+l_2} \\ &\leq 2 \end{aligned}$$

In conclusion, we have:

$$\sqrt{2} \leq E_P \leq 2.$$

2. Bounds on E_P: The example in Figure 3.3 provides a good lower bound for E_P. No matter what the value of a is, the perimeter of R_W is always greater than that of $R_{\min P}$. The difference is greatest when $a = \sqrt{2} - 1$ and $\text{perim}(R_W) = 2(\sqrt{2} + 1)$. Thus in that case,

$$\varepsilon_P(Q) = \frac{2(\sqrt{2}+1)}{4} = \frac{\sqrt{2}+1}{2} \approx 1.207$$

Hence $E_P \geq (\sqrt{2}+1)/2$.

For the upper bound, consider as above $R_{\min P}$ having sides l_1 and l_2 (with $l_1 \leq l_2$), and R_W with sides $l'_1 = \text{width}(Q)$ and l'_2. We have:

$$\begin{aligned}
\varepsilon_P(Q) &= \frac{\text{perim}(R_W)}{\text{perim}(R_{\min P})} \\
&= \frac{2(\text{width}(Q) + l'_2)}{2(l_1 + l_2)} \\
&= \frac{\text{width}(Q) + l'_2}{l_1 + l_2} \\
&\leq \frac{\text{width}(Q) + \text{diam}(Q)}{l_1 + l_2} \qquad (l'_2 \leq \text{diam}(Q)) \\
&\leq \frac{\text{width}(Q) + \sqrt{{l_1}^2 + {l_2}^2}}{l_1 + l_2} \\
&\leq \frac{\text{width}(Q)}{l_1 + l_2} + \frac{\sqrt{{l_1}^2 + {l_2}^2}}{l_1 + l_2} \\
&\leq \frac{\text{width}(Q)}{2l_1} + \frac{\sqrt{(l_1 + l_2)^2}}{l_1 + l_2} \qquad (l_1 \leq l_2) \\
&\leq \frac{1}{2} + 1 = \frac{3}{2} \qquad \left(\frac{\text{width}(Q)}{l_1} \leq 1\right)
\end{aligned}$$

In conclusion, we have:

$$\frac{\sqrt{2}+1}{2} \leq E_P \leq \frac{3}{2} \text{ or } 1.207 \leq E_P \leq 1.5$$

Chapter 4

Triangulations

Our study of the Rotating Calipers leads us to examine the use of this paradigm in computing triangulations of a set of points. Two specific triangulations are examined: the "onion" triangulation, and the "spiral" triangulation, which can be used in order to obtain a quadrangulation.

4.1 Introduction

Definition 4.1.1 *Given S, a set of points in the plane, a triangulation $T(S)$ is a graph $G(S, E)$ such that:*

- *The points of S are its nodes.*
- *Nodes are connected by edges E.*
- *Edges intersect only at nodes.*
- *Every node is connected to at least two other nodes.*
- *As many triangles as possible are created.*

The triangulation graph occurs very often in computational geometry due to its many applications. It is used often in graphics, pattern recognition, statistics, scattered data interpolation, finite element methods, medical imaging, GIS (Geographic Information Systems), to give a few examples. And of course, the graph has its applications in computational geometry itself. We refer the reader to the following papers [3, 4, 7, 6, 59, 70, 83, 87, 88, 91].

The problem of triangulating a simple n-gon has existed for almost a century. Lennes [53] gives a first algorithm achieving the computation in $O(n^2)$ time. In the last two decades, the complexity has been gradually reduced to linear-time (thanks to Chazelle [15]) due to a tremendous amount of research on the problem. We invite the reader to consult the following papers for many of the intermediate results [33, 41, 19, 84, 75, 22, 48].

Given a set of points, often certain constraints are added to yield special triangulations. Perhaps the best known is the Delaunay Triangulation [65]. Another interesting class of triangulations are *Hamiltonian* triangulations. A Hamiltonian triangulation is such that its dual graph is a chain. This is a great advantage, since the triangulation graph can be easily maintained as a list of adjacent triangles, and every triangle can be easily "reached". Two specific Hamiltonian triangulations are the "onion" and the "spiral" triangulation [83].

4.2 Onion Triangulations

4.2.1 Introduction

Consider a set of n points on the plane. By computing the convex hull, a smaller set of points remains in the interior of the hull. If one successively computes the hulls of the remaining points, the *onion-peeling* of S is obtained. An example of an *onion-peeling* of a set S is given in Figure 4.1.

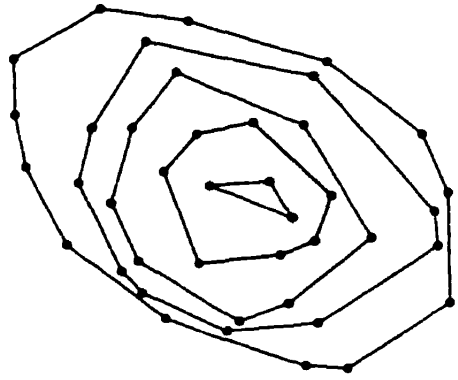


Figure 4.1: The *onion-peeling* of a set of points.

This structure has, among others, applications in statistics. Consider the last hull generated. The average of the points of that hull is referred to as the *Tukey median.* Intuitively, one can see that the onion-peeling of a set gives some information about the "layering". We refer the reader to [83, 89, 14].

The *onion triangulation* of a set S is a triangulation of S including the onion-peeling of S as a subset. Apart from the fact that it maintains the useful layered structure, this triangulation is also hamiltonian. Furthermore, once the onion-peeling of a set has been obtained, the triangulation can be easily obtained using an algorithm by Toussaint, based on the Rotating Calipers [83]. Let us now carefully examine this procedure.

4.2.2 Theoretical results

Before proceeding to the results, let us give formal definitions of the graph structures:

Given a set of points S, the *onion-peeling* of S, denoted $OP(S)$, is the graph resulting from the following procedure:

1. Let $S' = S$.
2. Compute $P_S = CH(S')$. Add $CH(S')$ to $OP(S)$.

3. Let $S' = S' \setminus V(P_S)$, that is, remove the vertices of P_S from S'.

4. Repeat steps 2–3 until $S' = \emptyset$.

Definition 4.2.1 *The* onion triangulation *of a set of points S is a triangulation graph G of S such that $OP(S) \subset G$.*

Figure 4.2 illustrates the onion triangulation of the same set of points as in Figure 4.1.

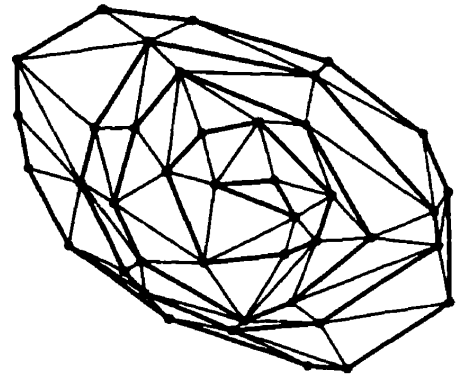


Figure 4.2: The *onion triangulation* of a set of points.

Definition 4.2.2 *The* annulus *of two convex polygons P and Q (such that Q lies in the interior of P), denoted $ANN(P,Q)$ is defined as the union of ∂P, ∂Q and the region of the plane interior to P and exterior to Q.*

An example of an *annulus* of two convex polygons is given in Figure 4.3 (the annulus is shown in grey).

The *onion-peeling* of a set S is then a series of nested *annuli.* This is obvious from the procedure defining $OP(S)$.

Thus, the *onion triangulation* of a set S can be obtained by computing the triangulations of its *annuli.* This is the main idea behind Toussaint's triangulation algorithm [83], which we now present.

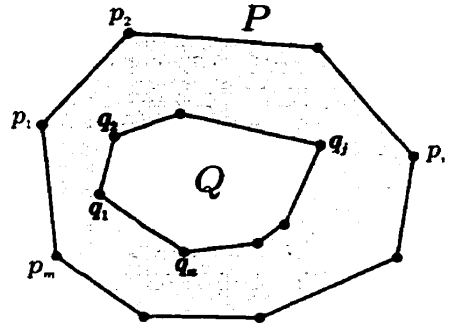


Figure 4.3: Illustrating the annulus of P and Q.

4.2.3 Algorithm and analysis

Let $P = \{p_1, p_2, \ldots, p_m\}$ and $Q = \{q_1, q_2, \ldots, q_n\}$ be two convex such that Q lies in the interior of P and let T be the desired triangulation of $ANN(P, Q)$. We assume that P and Q are given in clockwise order and in standard form.

Algorithm TRI-ANNULUS

1. Insert vertices of P and Q as nodes into T.
2. Insert edges of P and Q as edges into T.
3. Find vertices $x_{\min}(P)$ and $x_{\min}(Q)$. Re-index vertices of P and Q such that $p_1 = x_{\min}(P)$ and $q_1 = x_{\min}(Q)$.
4. Insert edge (p_1, q_1) into T.
5. Let $p_i = p_1$, $q_i = q_1$.
6. Let L_P and L_Q be two directed vertical lines passing through p_1 and q_1, respectively and such that P and Q lie to the right of L_P and L_Q, respectively. These two lines of support will be the *calipers*.
7. Rotate the calipers until one of the lines "hits" a vertex.
   - if the vertex belongs to P add a new edge $p_{i+1}q_i$ to T, set $p_i := p_{i+1}$.

- if the vertex belongs to Q add a new edge p_iq_{i+1} to T, set $q_i := q_{i+1}$.
- if both lines have hit vertices, add new edge $p_{i+1}q_{i+1}$ and either of edges p_iq_{i+1} or $p_{i+1}q_i$ to T, set $p_i := p_{i+1}$ and $q_i := q_{i+1}$.

8. Repeat previous step until both p_1 and q_1 have been reached.

Theorem 4.2.1 *Algorithm TRI-ANNULUS computes the triangulation of* $ANN(P,Q)$ *in* $O(m+n)$ *time.*

Proof: We begin by first showing that the output of the algorithm (the graph T) is indeed a valid triangulation of $ANN(P,Q)$. The first two steps add P and Q into T. This is valid since no edges of P or Q can intersect, and since Q lies within P, no edge of Q can intersect an edge of P. We proceed by first showing the validity of the first triangle, and applying the same argument to others. First, we claim that the creation of the first triangle (the first two edges added) is valid.

The first edge added is $e_1 = (p_1, q_1)$. No edge of Q can intersect e_1, since q_1 is the "left-most" point of Q by definition. Furthermore, any edge $e = (p_l, p_{l+1})$ of P cannot intersect e_1 either, since e' must have all points of P and Q (thus p_1 and q_1) lie to its right.

The second edge added, e_2, is either (p_1, q_2) (see Figure 4.4(a)) or (p_2, q_1) (Figure 4.4(b)) (the parallel case can be separated into two steps, first adding one of the above, then adding (p_2, q_2)). Suppose e_2 is not a valid edge. It cannot intersect (p_1, q_1). Therefore, it must intersect an edge of P or Q. But no edge $e = (p_l, p_{l+1})$ of P could intersect e_2 since p_1, p_2, q_1 and q_2 must all lie to its right. That leaves an edge $e = (q_l, q_{l+1})$ of Q, which is also impossible since one of e's vertices would have to lie to the left of (q_1, q_2). Hence, e_2 is always valid.

This argument applies for subsequent triangles: each time an edge is added, it consists of a pair of points admitting parallel lines of support.

We have thus shown that the first triangle created is valid. Suppose now that k triangles have been created (all valid) and the algorithm is in the process of adding the next edge. Thus, current points are p_i and q_i (edge (p_i, q_i) is already in T), and again, we have two cases for the next edge e: (p_{i+1}, q_i) or (p_i, q_{i+1}). Suppose e is not valid. By the argument above, no

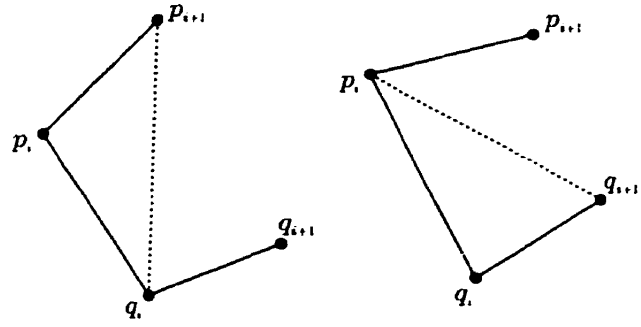


Figure 4.4: Illustrating the proof of Theorem 4.2.1.

edge of P or Q could intersect e. So it must be an edge $e_T = (p_k, q_l) \in T$. However, there are restrictions on e_T. Because all the previous triangles are valid, e_T (belonging to two previous triangles) cannot intersect (p_i, q_i), nor any edge of P or Q. Furthermore, p_k must lie to the right of (p_i, p_{i+1}), and q_l must lie to the right of (q_i, q_{i+1}). Now, if p_k lies to the right of (q_i, q_{i+1}) then we have no intersection. Thus p_k lies to the left of (q_i, q_{i+1}) and q_l to its right. But since Q lies to the right of (q_i, q_{i+1}), then e_T must intersect ∂Q, which contradicts our assumption that e_T was a valid edge. Thus, by induction, all edges (or triangles) added are valid.

We now show that the algorithm has linear time complexity. At each step of the loop, a vertex of P or Q is hit by the calipers. Each time a vertex is hit, an edge (thus a triangle) is added. In the parallel case, two vertices are hit and two triangles added. No vertex can be skipped and no vertex is checked twice except for p_1 and q_1. Since we have $m + n$ vertices in total, the algorithm's loop runs in $O(m+n)$ time. Initialization also takes

$O(m+n)$ time due to finding the minima. Thus, the whole algorithm runs in linear time. ∎

Now, the complete algorithm for triangulating a set S of n points has only main steps:

Algorithm TRI-ONION

1. Compute $OP(S)$.
2. Use algorithm TRI-ANNULUS to triangulate each annulus in $OP(S)$.

Theorem 4.2.2 *Given a set S of n points, algorithm TRI-ONION computes a hamiltonian triangulation of S in $O(n \log n)$ time.*

Proof: Let us first prove the correctness and establish the fact that the resulting graph is hamiltonian. Since two consecutive annuli only share edges, triangles can only intersect at edges and vertices. Hence the union of two annulus triangulations remains a valid triangulation, and algorithm TRI-ONION yields a valid triangulation.

Now, consider a triangulated annulus. Since we started with two nested polygons (with the vertices and edges already in the triangulation), the only edges added to the triangulation graph were edges connecting a vertex of the outer polygon to a vertex of the inner (a "cross-polygon" edge, so to speak). Therefore, each triangle can be viewed as determined by an edge belonging to one of the polygons and a vertex belonging to the other. But then two triangles can only intersect at those cross-polygon edges (and vertices). Since triangles were added one by one, each sharing a cross-polygon edge with the previous, the triangulation's dual must be a chain: i.e., the triangulation is hamiltonian. Now, since this is true for all annuli, all chains (meaning duals) can be trivially concatenated, yielding

a new chain for the triangulation of the whole set. Hence, algorithm TRI-ONION yields a hamiltonian triangulation.

Let us now analyze the time complexity of the algorithm. Step 1 can be accomplished in $O(n \log n)$ time using the procedure by Chazelle [13, 14]. Step 2 must be looked at a bit more carefully. Suppose our set of points yields h layers H_1 through H_h ($H_1 = CH(S)$), the union of which is $OP(S)$. Suppose each layer (polygon) has n_i points, with $n = n_1 + \ldots + n_h$. So far, we know that TRI-ANNULUS must be used $h - 1$ times, first on the $ANN(H_1, H_2)$, then on $ANN(H_2, H_3)$, and so on. The running times are therefore $O(n_1 + n_2)$, $O(n_2 + n_3)$, etc. "Adding" these running times together yields a total running time of $O(n)$ for step 2. Therefore, the running time for the algorithm is dominated by step 1, and algorithm TRI-ONION runs in $O(n \log n)$ time. ∎

Corollary 4.2.3 *Computing the onion triangulation of a set of n points has a $\Omega(n \log n)$ lower bound.*

Proof: If one computes the onion triangulation of a set S, then $CH(S)$ is obtained trivially. Since computing the convex hull of n points has a $\Omega(n \log n)$ lower bound [61] then so does the onion triangulation problem. ∎

Hence we conclude that algorithm TRI-ONION is optimal.

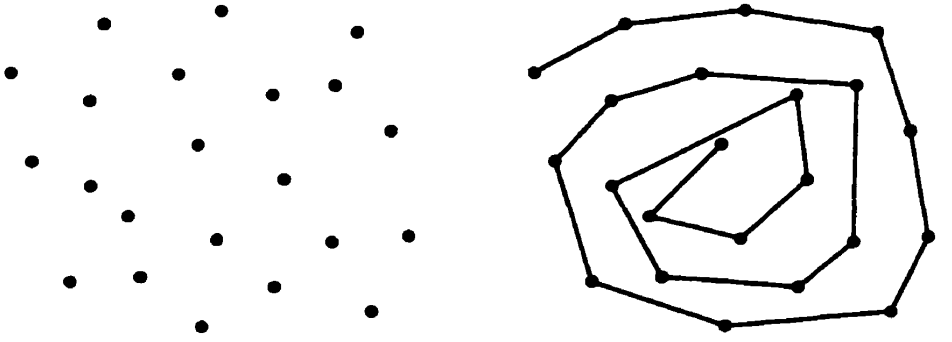
4.3 Spiral Triangulations

4.3.1 Introduction

Given a set S of n points on the plane, consider the procedure of attaching a string to one of the points[1](call it h_i) and wrapping the string (say clockwise) around the set of

[1]Some restrictions apply here, and are discussed below.

points, ignoring those the string already touches. This procedure resembles that for obtaining the convex hull, the loop is never closed and keeps spiraling inwards. The resulting graph is called the *convex spiral* of S, and is denoted $CS(S)$. An example is illustrated in Figure 4.5.



(a) (b)

Figure 4.5: A set of points (a) and its convex spiral (b).

The convex spiral also has applications to statistics, and it is not surprising — due to the similarities between CS and OP — that the structure is also used to define the median for two-dimensional data [10]: in this case, the last point on the spiral.

The convex spiral structure is, in fact, closely related to the onion-peeling graph. They both give some information as to the layering structure of the set of points, and the convex spiral maintains the annulus structure (locally). Finally, given one graph, the other can be obtained in $O(n)$ time [74, 65].

Some important differences exist. Given S, $OP(S)$ is unique since $CH(S)$ is unique. $CS(S)$, however isn't: it depends on the starting point (referred to above as h_i), and the direction of the graph (clockwise or counterclockwise). Restrictions apply to h_i, however. Only a vertex of the convex hull of S may be chosen (otherwise the procedure for obtaining $CS(S)$ fails). Thus, if $CH(S)$ has h vertices, $2h$ distinct convex spirals exist.

Finally, similar to the onion triangulation, the *spiral triangulation* of a set S is a triangulation graph containing $CS(S)$ as a subgraph [83, 10]. One interesting

application of this graph is its use in obtaining quadrangulations [10]. This will be discussed in the next section.

Bose and Toussaint [10] present an algorithm yielding a spiral triangulation of a set of points, and show that the triangulation is hamiltonian. We study their procedure, which they refer to as the Spiraling Rotating Caliper Algorithm.

4.3.2 Algorithm and analysis

First, let us give formal definitions for the structures so far introduced.

Given a set of n points S in the plane, the *convex spiral* of S is a polygonal chain resulting from the following procedure when applied to S:

1. Find $x_{\min}$, the point in S with minimum x-coordinate. If two such points exists, then let $x_{\min}$ be the one with smaller y-coordinate. Let $p_1 := x_{\min}$, and initialize a chain index i at 2.
2. Construct L, a vertical directed line through $x_{\min}$ pointing in the positive y direction.
3. Rotate L clockwise until the line "hits" a point x of S. Let $p_i := x$, and $i := i+1$. Remove x from S.
4. Repeat previous step until $S = \emptyset$.
5. Output $CS(S) = \{p_1, \ldots, p_n\}$.

Definition 4.3.1 *The* spiral triangulation *of a set of points S is a triangulation graph G of S such that $CS(S) \subset G$.*

The algorithm itself is somewhat more complex than those previously studied in its initialization steps. Nevertheless, the core of the procedure is quite straightforward, as it is based on the Rotating Calipers paradigm.

Given a set of n points S in the plane given in standard form, and its *convex spiral* $P = \{p_1, p_2, \ldots, p_n\}$ given in clockwise order, the goal is to compute the corresponding triangulation T.

Algorithm TRI-CS

1. *Initialization*

   (a) Insert P into T.

   (b) Find point p_h of P (in order, starting with p_2) such that $p_{h-1}p_hp_1$ is a right-turn but $p_hp_{h+1}p_1$ is a left-turn. The first few points of P define its convex hull (this is guaranteed by the *Spiraling Procedure* used to create the convex spiral); p_h is the last such point. In other words, $CH(P) = \{p_1, p_2, \ldots, p_h\}$.

   (c) Extend edge (p_{n-1}, p_n) until it intersects P and label that point as q'.

   (d) Construct a line L locally tangent to P at q', and rotate it counterclockwise until it meet the first point q of P such that $L \parallel (p_{n-1}, p_n)$. Refer to Figure 4.6.

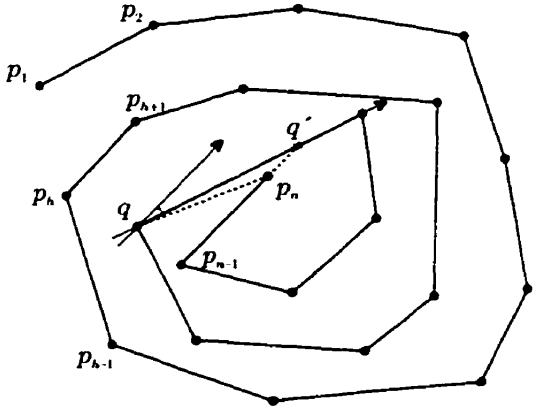


Figure 4.6: Illustrating the initialization of Algorithm TRI-CS.

   (e) Insert (p_n, q) into T.

   The above construction splits the region of the convex spiral (in other words the interior of $CH(P)$) in two, as is illustrated in Figure 4.7: the

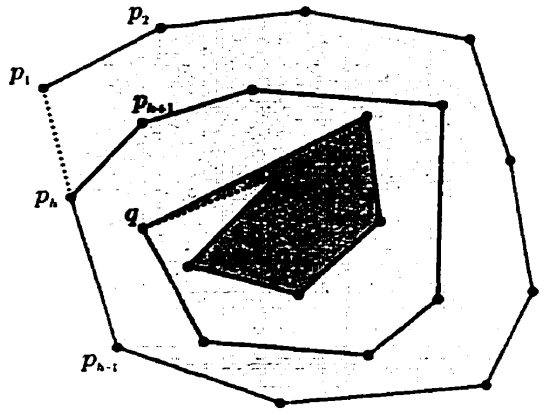


Figure 4.7: The *inner polygonal* and *outer spiral* regions

outer spiral region P_o (shown in light grey) and the inner polygonal region P_i (shown in dark grey). Furthermore, the spiral region P_o can itself be viewed as the outer polygonal chain $C_o = \{p_1, p_2, \ldots, q\}$ and the inner polygonal chain $C_i = \{p_h, p_{h+1}, \ldots, q, \ldots, p_{n-1}, p_n\}$. See Figure 4.8.

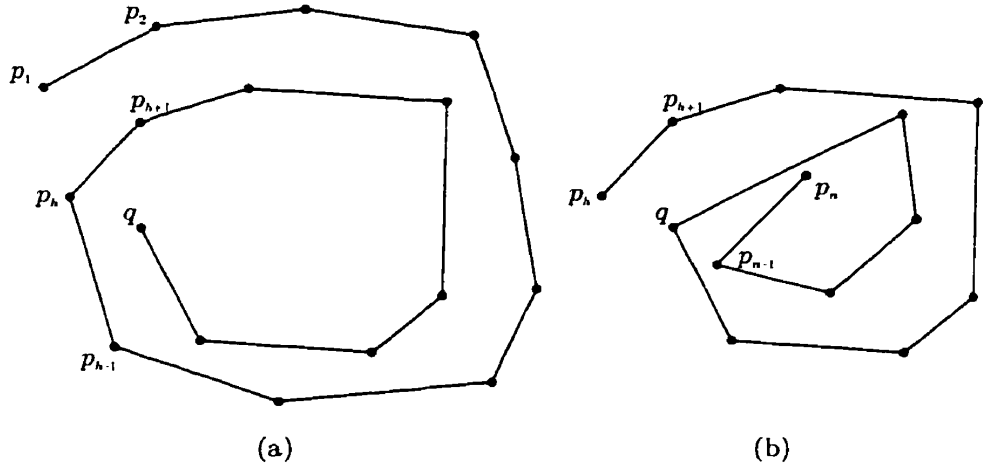


Figure 4.8: The outer spiral region, split in two: (a) the outer chain and (b) the inner chain.

2. *Triangulation*

The triangulation of P is done in two steps, by first triangulating P_o, then P_i.

3. P_o is triangulated similarly to an *annulus*. The triangulation is as follows:

   (a) Construct directed lines (calipers) L_o and L_i passing through p_1 and p_h, respectively and both flush with (p_h, p_1). L_o and L_i are associated with C_o and C_i, respectively. Let $q_o := p_1$, $q_i := p_h$. Add (p_h, p_1) to T.

   (b) Rotate calipers L_o and L_i clockwise, until a vertex v is hit: if it is by L_o, add edge (q_i, v) and let $q_o := v$. If it is by L_i, add edge (q_o, v) and let $q_i := v$. In the case of parallel edges in the two chains, L_o and L_i will hit v_o and v_i respectively. In that case, we add either of edges (q_o, v_i) or (v_o, q_i), along with edge (v_o, v_i), and we update: $q_o := v_o$, $q_i := v_i$.

   (c) Repeat previous step until $q_i = p_{n-1}$ and $v = p_n$.

4. P_i, by the construction above is a star-shaped polygon. A triangulation of the inner polygonal region is thus obtained by adding all adding all diagonals (p_n, v) $\forall v \in P_i,\ v \neq p_{n-1}, q$.

   The union of the two triangulations yields T, the triangulation of P.

Theorem 4.3.1 *Given a set S of n points, Algorithm TRI-CS yields the triangulation of $CS(S)$ in $O(n)$ time.*

Proof: First, we prove the correctness of the algorithm. Triangulating P_o, as previously mentioned is quite similar to triangulating an *annulus* (see section 4.2. This is true because viewed locally, it is an *annulus*, consisting of two non-intersecting convex chains, like the *convex spiral*. Since the algorithm used to triangulate P_o is essentially the same as the *annulus* triangulation algorithm, we claim that P_o is correctly triangulated.

The triangulation of P_i requires only for us to prove that the inner polygonal region is indeed star-shaped. By our construction, P_i is split by the segment (p_n, q'). This segment extend the edge (p_{n-1}, p_n) until we intersected the convex spiral. Consider RP_i, that region of P_i such that

every point lies to the right of (p_{n-1}, p_n), and LP_i, the region such that every point lies to the left of that edge. Now, $P_i = RP_i \cup LP_i$, and $RP_i = \{q', \ldots, p_{n-1}, p_n\}$. By our construction, it is obvious that RP_i is a convex polygon. Thus every point is visible from every other, in particular from p_n. Consider LP_i. By our construction, following P's vertices counterclockwise from q', q is the first vertex admitting a line of support parallel to edge $e_n = (p_{n-1}, p_n)$. Let $d = \text{dist}(e_n, q)$, the distance from q to e_n. We have $\forall x \in LP_i, \text{dist}(e_n, x) < d$. Hence all points of LP_i must lie within the sector qp_nq'. Hence all points of LP_i are visible from p_n. This implies all points of P_i are visible from p_n, therefore P_i is star-shaped from p_n.

We now show the algorithm runs in linear time. The initialization steps take $O(n)$ time: all that is involved is finding p_h (this is a walk through the vertices, in order), and the construction of q' and q. The linearity triangulation of P_o follows from the linearity of the *annulus* triangulation algorithm (Theorem 4.2.1). Since at most $2n$ points are involved in both chains C_i and C_o, this part takes $O(n)$ time as well. Finally, triangulating P_i is trivial since P_i is star-shaped. Hence, this part can also easily be done in $O(n)$ time. The algorithm therefore runs in linear time. ∎

Corollary 4.3.2 *The triangulation obtained using algorithm TRI-CS is hamiltonian.*

Proof: Consider first the triangulation of P_i. We have shown already that the inner polygon is star-shaped. And by adding all diagonals, its triangulation must be hamiltonian.

P_o, as mentioned above is triangulated in the same manner as an annulus. In the triangulation of P_o, all edges added to the graph connected one chain to the other. Since no edge connecting two vertices of the same chain was ever added, the triangles succeed each other in order, each

sharing one cross-chain edge with the next. Hence, the dual of this graph must be a chain, and the triangulation must be hamiltonian.

Finally, since q and p_n are the last vertices of chains C_i and C_o, $[q, p_n]$ must be an edge in the triangulation of P_o. Since it also belongs to P_i, and both triangulations are hamiltonian, the union (i.e., the whole triangulation) is hamiltonian as well. ∎

Using the above procedure, given $CS(S)$, a triangulation is easily obtained. But how can $CS(S)$ be obtained? Although the Rotating Calipers can be used (achieving the result in $O(n^2)$ time), this is not the most efficient way. The most efficient ways are by obtaining $OP(S)$ first (using Chazelle's algorithm [13, 14], or the procedure given by Hershberger and Suri [39, 40]) and then using $OP(S)$ to obtain $CS(S)$ as has been previously discussed [74, 65]. Obtaining $OP(S)$ takes $O(n \log n)$ time and $CS(S)$ takes $O(n)$ time subsequently. Hence this first step dominates the running time of the algorithm; the total running time is $O(n \log n)$.

Let us summarize these results:

Algorithm TRI-SPIRAL

1. Given S, compute $OP(S)$.
2. Obtain a $CS(S)$ using $OP(S)$.
3. Use algorithm TRI-CS to compute a spiral triangulation of S.

And we have:

Theorem 4.3.3 *Given a set S of n points on the plane, the spiral triangulation of S can be obtained in $O(n \log n)$ time.*

As in the onion triangulation, the convex hull of the point set S can easily be obtained (in $O(n)$ time) from the triangulation (or the convex spiral). Thus algorithm TRI-SPIRAL is optimal.

Corollary 4.3.4 *Computing the onion triangulation of a set of n points has a $\Omega(n \log n)$ lower bound.*

4.4 Quadrangulations

While triangulations may be a widely used structure, some cases have been recently pointed out where quadrangulations (Definition 4.4.1) might be a better structure. Bose and Toussaint [10] mention finite element methods and scattered data interpolation in particular.

Definition 4.4.1 *A* quadrangulation *of a set of points S is a subdivision such that:*

- *Its vertices corresponding to points of S,*
- *Its outer face is the convex hull of S,*
- *Each face (except the outer face) is a quadrilateral.*

Because of this, efficient, straightforward algorithms solving this problem are needed. Results and algorithms (for varieties of inputs) have been obtained in the past twenty years or so. The earlier results often dealt with restricted class of polygons, such as orthogonal polygons (whose edges are parallel to the coordinate axes) [68, 46, 56, 69], and an $O(n \log n)$ optimal algorithm was obtained [56, 69]. Variations of the problem included minimizing the sum of the length of diagonals [47, 56], and decomposing orthogonal polygons into rectangles, a procedure of great use in VLSI [54, 58]. Trapezoidalizations are another variation often used in order to obtain polygon triangulations [30].

General algorithms, however, are more difficult to obtain: the concept of quadrangulations is somewhat more complicated than for triangulations. For instance, some polygons do not admit quadrangulations. In these cases, additional points (called

Steiner points) must be added. The existence of quadrangulations and the necessity for Steiner points are discussed in several papers; we invite the reader to consult [56] and [29]. The following result discussed in [10], however, is noteworthy:

Theorem 4.4.1 *A set of points S admits a quadrangulation if and only if $CH(S)$ has an even number of points.*

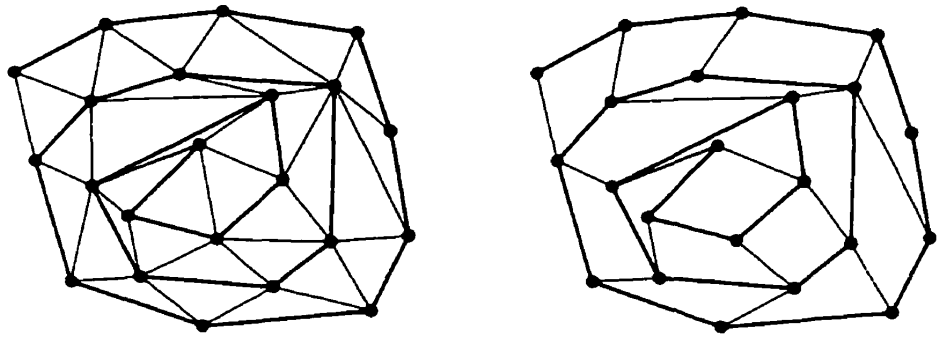
Several quadrangulation algorithms exist. One relatively easy approach is to convert a triangulation into a quadrangulation by adding up to $O(n)$ Steiner points (for a set of n points) [38, 45, 29]. Many algorithms construct a quadrangulation for a few points (such as the convex hull) and insert points into the structure, maintaining and updating the quadrangulation [8, 64, 20, 3, 4]. The most efficient (and optimal) of these Sequential Insertion (SI) algorithms was presented by O'Rourke [61], with a time complexity of $O(n \log n)$.

The algorithm studied in this work was presented by Bose and Toussaint [10]. The authors point out the drawbacks of the SI algorithms: the quadrilaterals produced are often long and non-convex; Bose and Toussaint then present an optimal algorithm yielding "nicer" quadrangulations. Interestingly, the essence of the algorithm consists of finding the spiral triangulation (see section 4.3). Once this is done, obtaining the quadrangulation is trivial.

Consider then a set S of n points, and T_S the spiral triangulation of S, obtained by using the algorithm described in section 4.3 (see Figure 4.9(a)).

Let D_S denote the sequence of k diagonals that were added in the process, $D_S = \{d_1, \ldots, d_k\}$, with $d_1 = [p_1, p_h]$. The procedure is quite straightforward:

- if k is even: remove diagonals $d_2, d_4, \ldots, d_m$.
- if k is odd:
    1. remove diagonals $d_k, d_{k-2}, \ldots, d_3$.



(a) (b)

Figure 4.9: (a): The spiral triangulation of a set of points and (b): the corresponding quadrangulation

2. add one Steiner point s outside $CH(S)$, near $[p_1, p_h]$.
3. Connect s to p_1 and p_h.
4. remove d_1.

We claim this procedure yields a quadrangulation (as in Figure 4.9(b)). If k is even, since the triangulation is hamiltonian, the triangles form a chain, attached to each other, so to speak, by the diagonals. Thus each two consecutive triangles form a quadrilateral. Removing the shared diagonal therefore is sufficient to convert it into a quadrangle. Since this is done for every second diagonal, each consecutive triangle pair is "converted" and the resulting graph is a proper quadrangulation.

If k is odd, we start from the last diagonal, and also remove each second diagonal. Again, each pair of consecutive triangles is converted into a quadrilateral. However, the very first triangle (it is either $\triangle(p_1, p_2, p_h)$ or $\triangle(p_h, p_{h+1}, p_1)$) is left out. This is why a Steiner point is needed, and by adding it, the last triangle is trivially converted.

The time complexity of this procedure is $O(n)$, since the removal of all diagonals takes at most $O(n)$ time, and if needed, adding the Steiner point and making the necessary update is $O(1)$. Therefore, in either the even or odd diagonals case, the

above procedure has linear time complexity. Recalling that the TRI-SPIRAL algorithm had $O(n \log n)$ running time (Theorem 4.3.3), the complete quadrangulation algorithm keeps the same time complexity. We thus conclude:

Theorem 4.4.2 *Given a set S of n points, the quadrangulation of S is obtained in $O(n \log n)$ time, with the addition of at most one Steiner point.*

Chapter 5

Properties of Convex Hulls

This chapter focuses on problems involving convex hulls. Many well-known problems are studied: merging and intersecting convex hulls, finding the common tangents and critical support lines of two convex polygons, computing the vector sum of convex polygons. We study the use of the Rotating Calipers paradigm in solving each of these problems in an efficient way.

5.1 Merging convex hulls

5.1.1 Introduction

Suppose two convex polygons (or their hulls) are given, and it is required to find the convex hull of the union of these. Of course, the problem could easily be solved by computing the convex hull for the set of points consisting of all involved vertices. But, by simple experimentation, one notes that sections of the previous hulls are maintained in the end of the merge operation (see Figure 5.1). Therefore, it would be better to use the information already given in the hulls.

The main application of this operation is in computing convex hulls: all "divide-and-conquer" convex hull algorithms need a merging procedure. In such algorithms,

the original point set is divided in two halves and the convex hull for each half is recursively obtained. When the subsets considered are small enough (three points), the convex hull is easily obtained in $O(1)$ time. Then, it remains to merge all the smaller hulls, two by two, until only one remains. The first divide-and-conquer convex hull algorithm is due to Preparata and Hong [63] and dates back to 1977. Thanks to a linear time complexity merge step, their algorithm (for an input size n) has optimal $O(n \log n)$ complexity. Other optimal divide-and-conquer algorithms exist, notably by Shamos [74] and Toussaint [80, 76, 81, 83].

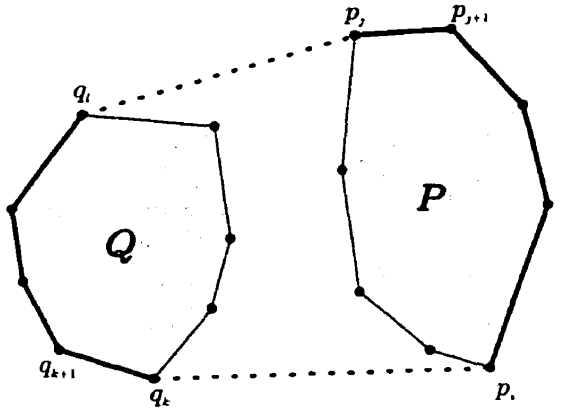


Figure 5.1: Merging two convex hulls by adding "bridges".

Merging two hulls consists, in essence, of finding the *bridge points* between the two (refer to Definition 5.1.1 and Figure 5.1). This consists of finding lines of support common to both hulls. The brute force way consists of trying each pair of points between the hulls and testing their validity as bridge points. But a quadratic number of such pairs exists. Preparata and Hong's method computes the desired pairs of points for disjoint hulls; in order to make sure all pairs of hulls considered during their algorithm are disjoint, the points are initially sorted by their x-coordinates.

In 1983, Toussaint [81, 83] presents another algorithm performing the merge step. The idea is to use the Rotating Calipers to compute the bridge points. The algorithm is just as straightforward and efficient, and has a main advantage: the hulls need not be disjoint; even in the case of intersecting hulls, all bridge points are computed.

This in turn enables one to dismiss the initial sort operation needed in Preparata and Hong's procedure. Furthermore, no backtracking is involved, another advantage over some optimal algorithms mentioned above [74, 80, 76]. Toussaint's algorithm [81, 83] is therefore more efficient while at the same time providing a general merge procedure. The details and analysis of this beautiful algorithm are provided in the rest of this section.

5.1.2 Theoretical results

We start with the definition of some key concepts discussed. First, a formal definition of bridge points.

Definition 5.1.1 *Given two convex polygons* $P = \{p_1, \ldots, p_m\}$ *and* $Q = \{q_1, \ldots, q_n\}$. *Suppose* $R = CH(P \cup Q)$. *Then* R *is formed by convex chains belonging to* P *and* Q, *joined by new edges called* bridges, *made up by vertices called* bridge points.

Definition 5.1.2 *Given two convex polygons* $P = \{p_1, \ldots, p_m\}$ *and* $Q = \{q_1, \ldots, q_n\}$, *two vertices* p_i *and* q_j *are said to form a* co-podal pair *if* P *and* Q *admit parallel lines of support in the same direction at* p_i *and* q_j.

The above definition complements the concept of anti-podal pairs. An example of a co-podal pair is illustrated in Figure 5.2.

The following result is the heart of the merge procedure. It characterizes the necessary and sufficient conditions for a pair of points to determine a bridge between two polygons.

Theorem 5.1.1 *Given two convex polygons* $P = \{p_1, \ldots, p_m\}$ *and* $Q = \{q_1, \ldots, q_n\}$, *two vertices* p_i *and* q_k *are bridge points if, and only if they form a co-podal pair and the vertices* $p_{i-1}, p_{i+1}, q_{k-1}, q_{k+1}$ *all lie on the same side of* $L(p_i, q_k)$.

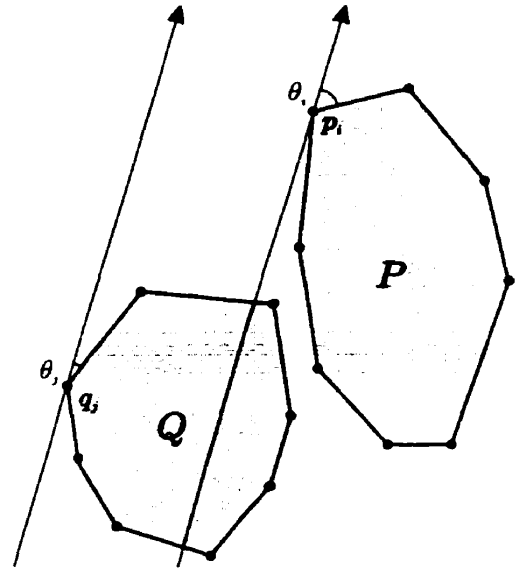


Figure 5.2: An example of two vertices forming a co-podal pair.

Proof:

$\Rightarrow$: Assume p_i and q_k are bridge points. Therefore they form an edge of the merged convex hull. Hence, all points of P and Q must lie on one side of this edge. Let $\ell = L(p_i, q_k)$. From the above, we have that ℓ is a line of support for P at p_i, and ℓ is also a line of support for Q at q_j. Now, because all points of P and Q lie on the same side of ℓ, p_i and q_k form a co-podal pair.

$\Leftarrow$: Let $\ell = L(p_i, q_k)$ once again. ℓ is such that p_{i-1}, p_{i+1}, q_{k-1}, q_{k+1} lie on the same side of ℓ. Therefore, ℓ is a line of support for P and for Q. Since p_i and q_k are co-podal, P and Q must lie on the same side of ℓ. Therefore, p_iq_k is a line segment with all points of P and Q lying to one of its sides. This can only be if p_iq_k is an edge of $CH(P \cup Q)$ (by the definition of a convex hull). Hence, we have that p_i, q_k are bridge points. ∎

5.1.3 Algorithm and analysis

Using Theorem 5.1.1, Toussaint [81, 83] suggests the following algorithm based on the Rotating Calipers. It is assumed, again, that P and Q are both convex polygons with m and n vertices each, and that $N = m + n$.

Algorithm: MERGE-HULLS

1. Compute the maximum y coordinates for both P and Q.
2. Start with the two lines of support (i.e., calipers) parallel to the x-axis, touching P and Q at $y_{\max}(P)$, and $y_{\max}(Q)$. Choose a direction of rotation, say clockwise. The two lines of support determine the angles θ_i, θ_j.
3. Compute $\theta = \min(\theta_i, \theta_j)$.
4. Rotate clockwise by θ, thus making one line of support flush with one edge. Since we have hit a new vertex, one new co-podal pair between P and Q is considered. In the case of parallel edges between P and Q, a maximum of three new co-podal pairs between the polygons can be considered.
5. Let p_i and q_j be the new vertices where the lines of support are touching P and Q, and let $\ell = L(p_i, q_j)$. Since the lines of support determine at least one co-podal pair between P and Q, check if p_{i-1}, p_{i+1}, q_{j-1}, q_{j+1} all lie on the same side of ℓ (each "check" can be done in $O(1)$ time using a signed area calculation). If all four points lie on one side of ℓ, label p_i, q_j as bridge points.

   In the case of parallel edges, it is not sufficient to check the new vertices hit. If (p_i, q_j) is the old vertex pair and (p'_i, q'_j) the new one, the pairs (p_i, q'_j) and (p'_i, q_j) are also co-podal, and thus the same set of computations applied to (p_i, q_j) must be applied to (p_i, q'_j) and (p'_i, q_j).
6. Repeat steps 3–5 until the total rotation angle is greater than 2π.

7. Given our list of all bridges

$$B = \{(p_{b1}, q_{b1}), \ldots, (p_{bk}, q_{bk})\},$$

the merged hull is obtained as follows: we start with the highest of the maximum y-coordinates (obtained in step 1), say it is equal to p_i. Then the merged hull is $\{p_i, \ldots, p_{b1}, q_{b1}, \ldots, q_{b2}, p_{b2}, \ldots\}$. In other words, either of the two polygons vertices are just added in order until the vertex coincides with a bridge point, at which point we switch between polygons.

Theorem 5.1.2 *Given two convex polygons P and Q (with m and n vertices each), algorithm MERGE-HULLS determines the merged convex hull $CH(P \cup Q)$ in $O(m+n)$ time.*

Proof: Let us first prove the correctness of the algorithm. The pair of lines of support are rotated an angle of at least 2π. Hence all possible co-podal pairs between P and Q are analyzed. From Theorem 5.1.1, all bridge points are correctly determined. It then remains to output the correct list of vertices for the merged hull. Since the algorithm starts with the maximum y-coordinates for P and Q, the maximum between these must be a vertex. Since the bridge pairs connect convex chains between the two polygons, each vertex along the current chain is added until a bridge is encountered. Then the same procedure is continued with the other polygon, and so on, until the first vertex (with maximum y-coordinate) is encountered again. The merged hull is therefore correctly output.

The running time analysis is also straightforward. Step 1 takes $O(N)$ time ($N = m + n$). Step 2 is initialization, and thus takes constant time $O(1)$. The combination of steps 3 to 5 is essentially a finite (fixed) number of calculations, and thus takes $O(1)$ time. This same combination is repeated (step 6) until every vertex has been "hit" by the lines of

support. Repeating N times an $O(1)$ set of operations has a $O(N)$ time cost. Finally, step 7 depends on the number of bridge pairs we have. If the polygons are disjoint and this is known, then we have two bridge pairs and a special procedure (for that case) can output the vertex list of the merged hull in $O(1)$ time. Otherwise, we may have up to N bridge pairs, and the worst-case time is therefore $O(N)$. We therefore conclude that algorithm MERGE-HULLS has linear time complexity. ∎

5.2 Common tangents

Common tangents for two polygons are exactly what their name indicates: lines which are lines of support for both polygons. We restrict the notion however: both polygons must lie on the same side of a common tangent[1]. For an example, refer to Figure 5.3 where line ℓ is a common tangent for P and Q. Note that p_i, q_j form a bridge. The

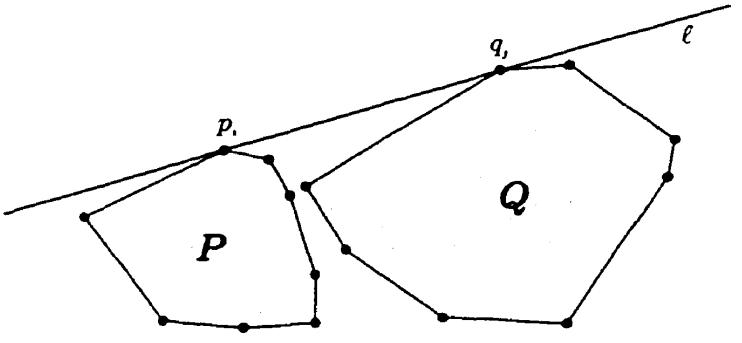


Figure 5.3: An example of a common tangent.

concept of common tangents is in fact very similar to that of bridge points. Indeed, they are equivalent:

Theorem 5.2.1 *Given two convex polygons P, Q, a line $\ell = L(p_i, q_j)$ ($p_i \in P$, $q_j \in Q$) is a common tangent for P and Q if, and only if p_i and q_j are bridge points.*

[1] When they lie on opposite sides, the line is called a Critical Support line. We study this in Section 5.4.

Proof: $\Rightarrow$: Let $\ell = L(p_i, q_j)$ be a common tangent of P and Q. Therefore, P and Q lie on the same side of ℓ, and ℓ is a line of support for both P and Q at p_i and q_j respectively. Hence, p_{i-1}, p_{i+1}, q_{i-1}, q_{i+1} must all lie on the same of ℓ, and (p_i, q_j) must form a co-podal pair. By theorem 5.1.1, we have that p_i and q_j are bridge points.

$\Leftarrow$: Suppose p_{i-1}, p_{i+1}, q_{i-1}, q_{i+1} all lie on the same side of $\ell = L(p_i, q_j)$, and assume that (p_i, q_j) form a co-podal pair.

Since p_{i-1}, p_{i+1} lie on the same side of ℓ, and ℓ is tangent to P at p_i, all of P must lie to one side of ℓ. This is due to the fact that P is convex (therefore P lies in the $p_{i-1}p_ip_{i+1}$ sector). Hence we have that ℓ is a line of support for P at p_i. By symmetry, ℓ is a line of support for Q at q_j.

Therefore, P and Q must both lie on the same of ℓ (since p_{i-1} and q_{i-1} lie on the same side).

Since ℓ is a line of support for both P and Q and these lie on one side of it, ℓ is a common tangent for P and Q. ∎

Using theorem 5.2.1, finding common tangents between two polygons P and Q boils down to finding bridge points. Therefore, we refer to algorithm MERGE-HULLS, described in section 5.1 which does exactly that. The only change is that the last step of the algorithm is removed.

Hence, we have the equivalent of Theorem 5.1.2 in the following:

Theorem 5.2.2 *Given two convex polygons P and Q, with m and n vertices respectively, the common tangents for P and Q can be determined in $O(m + n)$ time.*

5.3 Intersecting convex polygons

Suppose we are given two convex polygons P and Q (with m and n vertices respectively), with the purpose to compute their intersection[2].

Before proceeding to the computation of the intersecion, *whether* the polygons intersect or not should be known. Chazelle (with Dobkin) [12, 16] provides us with efficient $O(\log(m+n))$ time algorithms. If P and Q do indeed intersect, we can proceed to the computation.

Many algorithms exist: Shamos [72, 74], and earlier with Hoey [73] give linear-time ($O(m+n)$) algorithms. Although efficient, these methods are quite complex and elaborate. Later O'Rourke et al' [62] presents a much less complicated algorithm, without sacrificing performance.

Our interest lies in an algorithm by Toussaint [79, 83] which uses the Rotating Calipers. Indeed, a result by Guibas et al. [37] establishes a one-to-one correspondence between the intersections of the polygons' boundaries ∂P and ∂Q and the bridges between P and Q. Toussaint provides a proof of this result [79].

Based on this, given the bridge points, intersection points are computed (this is the key to the whole procedure). When these are joined together by convex chains of P and Q, $P \cap Q$ is determined. Thus we have another application of algorithm MERGE-HULLS (Section 5.1). Since it is beyond the scope of this work to present details of that algorithm we invite the reader to consult Toussaint's papers [79, 83] for all the details.

[2]We assume that the data is given in standard form, thus implying that P and Q intersect if, and only if their interiors intersect.

5.4 Critical support lines

5.4.1 Introduction

A critical support (CS) line of two polygons is a common line of support with the polygons on opposite sides (see Definition 5.4.1 for a formal definition). This problem has a multitude of applications, such as visibility, collision avoidance, range fitting and linear separability to name a few [81]. Such problems occur frequently in path planning in robotics. Critical support lines are also often referred to as *separating tangents*.

For instance, the CS lines for two disjoint polygons determine the proper visibility regions. They determine separability lines between polygons. They can be used to determine how much one can translate one polygon without colliding with the other. Finally, Grenander defines a method for measuring the distance between disjoint convex polygons [34]. We refer the reader to the following papers for more information on these problems [28, 36, 60].

It is therefore desirable to have an algorithm to compute the CS lines given two convex polygons. The brute-force way is easy to implement, but the procedure has quadratic time complexity, since all vertex pairs between polygons need to be checked.

Toussaint provides yet another application of the Rotating Calipers [81], to compute the CS lines. Given two disjoint polygons with a total of N vertices, his algorithm efficiently computes the CS lines in $O(N)$ time.

We present his results in detail in the remainder of this section.

5.4.2 Theoretical results

Let us formalize our previous definition of CS lines.

Definition 5.4.1 *Given two disjoint convex polygons P and Q, a Critical Support Line (CS line) is a line $\ell = L(p_i, q_j)$ ($p_i \in P$, $q_j \in Q$) such that: it is a line of support for P at p_i, it is a line of support for Q at q_j, and such that P and Q lie on opposite sides of ℓ.*

An example is shown in Figure 5.4: vertices p_i and q_j determine the Critical Support line ℓ. p_i and q_j form an anti-podal pair between P and Q; in addition, p_{i-1}, p_{i+1} and q_{j-1}, q_{j+1} lie on opposite sides of ℓ. We will see these are necessary and sufficient conditions for ℓ to be a CS line.

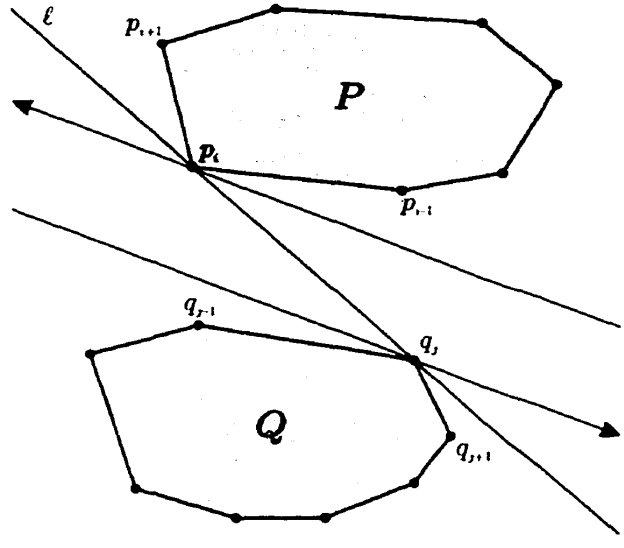


Figure 5.4: An example of Critical Support lines.

Theorem 5.4.1 *Two vertices $p_i \in P$ and $q_j \in Q$ determine a CS line $\ell = L(p_i, q_j)$ if, and only if they form an anti-podal pair and p_{i-1}, p_{i+1} lie on one side of ℓ while q_{j-1}, q_{j+1} lie on the other side of ℓ.*

Proof: $\Rightarrow$: Assume p_i, q_j determine a CS line $\ell = L(p_i, q_j)$. Therefore, by definition, we have:

- ℓ is a line of support for P at p_i.

- ℓ is a line of support for Q at q_j.
- P and Q lie on opposite sides of ℓ.

Since ℓ is a line of support for both P and Q, and since P and Q lie on opposite sides of ℓ, then p_i and q_j form an anti-podal pair. The case is peculiar since the two parallel lines of support usually considered happen to coincide in this case.

Since P and Q lie on opposite sides of ℓ, then p_{i-1}, p_{i+1} must lie on one side of ℓ, while q_{j-1}, q_{j+1} lie on the other side.

$\Leftarrow$: We know that (p_i, q_j) form an anti-podal pair, p_{i-1} and p_{i+1} lie on one side of ℓ while q_{j-1}, q_{j+1} lie on the other side of ℓ.

p_{i-1}, p_{i+1} lie on one side of ℓ. Hence all of P must lie on the same side, since ℓ touches P at p_i and P must lie in the $p_{i-1}p_ip_{i+1}$ sector (since it is convex). Similarly, all of Q must lie on one side of ℓ.

Now, since ℓ intersects P at p_i and Q at q_j, we have that ℓ is a line of support for both P and Q, at p_i and q_j respectively.

Furthermore, since p_{i-1} and q_{j-1} lie on opposite sides of ℓ, then P and Q must lie on opposite sides as well. ∎

Using theorem 5.4.1, the determination of a CS line now becomes very easy. The Rotating Calipers can be used in a similar way as in the MERGE-HULLS procedure 5.1. Toussaint [81] suggests an algorithm similar to CS-LINES below. The input is two convex, disjoint polygons P and Q (in standard form) with m and n vertices respectively given in clockwise order.

Algorithm CS-LINES

1. Determine the vertex with maximum y coordinate for P and the vertex with minimum y coordinate for Q.

2. Start with the two lines of support (i.e., the caliper) parallel to the x-axis, touching P and Q at $y_{\max}(P)$, and $y_{\min}(Q)$, and directed such that the polygons lie to their right. The two lines of support determine angles θ_i, θ_j.

3. Compute $\theta = \min(\theta_i, \theta_j)$.

4. Rotate clockwise by θ, thus making one line of support flush with one edge. Since we have hit a new vertex, one new anti-podal pair between P and Q is considered. In the case of parallel edges between P and Q, three new anti-podal pairs between the polygons are considered.

5. Let p_i and q_j be the new vertices where the lines of support are touching P and Q, and let $\ell = L(p_i, q_j)$. Since the lines of support determine at least one anti-podal pair between P and Q, check if:

   - p_{i-1} and p_{i+1} lie on the same side of ℓ
   - q_{j-1} and q_{j+1} lie on the same side of ℓ
   - p_{i-1} and q_{j-1} lie on opposite sides of ℓ.

   Each "check" can be done in $O(1)$ time using a signed area calculation. If all checks are true, p_i, q_j determine a CS line.

   In the case of parallel edges, it is not sufficient to check the new vertices hit. If p_i, q_j is the old vertex pair and p'_i,q'_j the new one, the pairs (p_i, q'_j) and (p'_i, q_j) are also anti-podal, and thus the same set of computations applied to (p_i, q_j) must be applied to (p_i, q'_j) and (p'_i, q_j).

6. Repeat steps 3–5 until the total rotation angle is greater than 2π, i.e., until the lines return to their original position.

7. Output the pairs of points determining the CS lines.

As a consequence, we have the following theorem:

Theorem 5.4.2 *Given two convex polygons P and Q (with m and n vertices respectively), algorithm CS-LINES computes the CS lines for P and Q in $O(m+n)$ time.*

Proof: The correctness of the algorithm follows from Theorem 5.4.1: all possible anti-podal pairs between the polygons are checked. Therefore, the output is correct.

The running time analysis is straight-forward: Step 1 consists of two computations of $O(m)$ and $O(n)$ for a total of $O(m+n)$. Step 2 is just initialization and takes constant time. Steps 3 to 5 make up the main "loop" of the procedure. Essentially, they consist of a fixed and finite number of computations, each taking up $O(1)$ time. The total time cost through one interation of the loop is thus $O(1)$. These three steps are repeated while the calipers are rotated around the polygons until their original positions is reached. Since at each iteration at least one new vertex is "hit" or checked, and subsequently never checked again, there are at most as many iterations as there are vertices. Therefore, step 6 takes $O(m+n)$ time. Finally, the result is output in step 7, which takes unit time. Therefore, the running time of the algorithm is dominated by steps 1 and 6, both of which take $O(m+n)$ time. Hence the total running time of CS-LINES is $O(m+n)$. ∎

5.5 Vector sums of convex polygons

5.5.1 Introduction

Computing the vector sum of two convex polygons, also referred to as the Minkowski sum, is a fundamental problem occurring in motion planning and collision-avoidance [81]. The problem considered is the following: suppose one has a convex object

that can be translated anywhere on the plane, but avoiding a given obstacle (i.e., they cannot collide). What region of the plane determines where one can translate the object to? The vector sum is used to answer this question. Lozano-Perez and Wesley [55] present an algorithm based on this, while O'Rourke's text offers a good introduction to the subject [61].

The formal definition of the vector sum is as follows:

Definition 5.5.1 *Given two convex polygons P and Q, the vector sum of P and Q, denoted by $P \oplus Q$ is given by:*

$$P \oplus Q = \{t = (x_r + x_s, y_r + y_s) \mid \forall r = (x_r, y_r) \in P, \quad \forall s = (x_s, y_s) \in Q\}$$

5.5.2 Theoretical results

Toussaint proposes using the Rotating Calipers to compute the vector sum of two convex polygons [81]. His suggested algorithm is efficient, having linear time complexity, and is easy to implement.

The following basic results (Theorems 5.5.1 to 5.5.5) characterize many aspects of the problem, necessary for the algorithm [81].

Theorem 5.5.1 *Given two convex polygons P and Q, $P \oplus Q$ is a convex polygon.*

Proof: Definition 5.5.1 can be rewritten as:

$$P \oplus Q = \bigcup_{r \in P} \{t = (x_t, y_t) \,|x_t = x_r + x_s, y_t = y_r + y_s, \forall s \in Q\}, r = (x_r, y_r)$$

Note that in the above equation, x_r and y_r are fixed inside the braces. Let us call the set inside the braces $Q \oplus r$. Therefore, $P \oplus Q$ can be rewritten as:

$$P \oplus Q = \bigcup_{r \in P} (Q \oplus r) = \bigcup_{s \in Q} (P \oplus s)$$

Now, suppose $P \oplus Q$ is not convex.

Therefore, $\exists\ t_1 = (x_1, y_1), t_2 = (x_2, y_2), t_1, t_2 \in P \oplus Q$ and a third point $t_0 = (x_0, y_0)$, with

$$\left.\begin{array}{rcl} x_0 & = & x_1 + (x_2 - x_1)\tau_0 \\ y_0 & = & y_1 + (y_2 - y_1)\tau_0 \end{array}\right| 0 < \tau_0 < 1, t_0 \notin P \oplus Q$$

In other words, there exists a point along the segment (t_1, t_2) that does not belong to $P \oplus Q$ though t_1 and t_2 do.

Now, $t_1, t_2 \in P \oplus Q$

$$\Rightarrow \left\{\begin{array}{rcl} t_1 & = & (p_{x_1} + q_{x_1}, p_{y_1} + q_{y_1}) \\ t_2 & = & (p_{x_2} + q_{x_2}, p_{y_2} + q_{y_2}) \end{array}\right.$$

with

$$\begin{array}{rcll} p_1 & = & (p_{x_1}, p_{y_1}) & \in\ P \\ p_2 & = & (p_{x_2}, p_{y_2}) & \in\ P \\ q_1 & = & (q_{x_1}, q_{y_1}) & \in\ Q \\ q_2 & = & (q_{x_2}, q_{y_2}) & \in\ Q \end{array}$$

so we can rewrite the coordinates for t_1 and t_2 as:

$$\begin{array}{rcl} x_1 & = & p_{x_1} + q_{x_1} \\ y_1 & = & p_{y_1} + q_{y_1} \\ x_2 & = & p_{x_2} + q_{x_2} \\ y_2 & = & p_{y_2} + q_{y_2} \end{array}$$

$$\Rightarrow \left\{\begin{array}{rcl} x_0 & = & (p_{x_1} + q_{x_1}) + (p_{x_2} + q_{x_2} - p_{x_1} - q_{x_1})\,\tau_0 \\ y_0 & = & (p_{y_1} + q_{y_1}) + (p_{y_2} + q_{y_2} - p_{y_1} - q_{y_1})\,\tau_0 \end{array}\right.$$

$$\Rightarrow \left\{\begin{array}{rcl} x_0 & = & [p_{x_1} + (p_{x_2} - p_{x_1})\,\tau_0] + [q_{x_1} + (q_{x_2} - q_{x_1})\,\tau_0] \\ y_0 & = & [p_{y_1} + (p_{y_2} - p_{y_1})\,\tau_0] + [q_{y_1} + (q_{y_2} - q_{y_1})\,\tau_0] \end{array}\right.$$

Now, let

$$p_0 = ((p_{x_1} + (p_{x_2} - p_{x_1})\tau_0)\,, (p_{y_1} + (p_{y_2} - p_{y_1})\tau_0))$$

$$q_0 = ((q_{x_1} + (q_{x_2} - q_{x_1})\tau_0), (q_{y_1} + (q_{y_2} - q_{y_1})\tau_0))$$

We have $t_0 = p_0 + q_0$. Now, p_0 is a point on the line segment (p_1, p_2), and $p_1, p_2 \in P$. Furthermore, q_0 is a point on the line segment (q_1, q_2), and $q_1, q_2 \in Q$. Since P and Q are convex, we have $p_0 \in P$ and $q_0 \in Q$. Therefore, $t_0 = p_0 + q_0 \in P \oplus Q$. This contradicts our assumption: $P \oplus Q$ is therefore convex. ∎

Theorem 5.5.2 *Given two convex polygons P and Q, the vertices of $P \oplus Q$ are vector sums of vertices of P and Q.*

Proof: We have established that $R = P \oplus Q$ is a convex polygon. Let $R_H = CH(R)$, $P_H = CH(P)$, $Q_H = CH(Q)$. The vertices of R are the same as the vertices of R_H. Take a vertex $r = (x_r, y_r)$ of R. Then $\exists\ p \in P, q \in Q$ such that $r = p + q$. Assume that p (say) is not a vertex of P.

Create a new axis δ along the plane (we will say that a point x in the plane have a certain "δ-value", denoted by x^δ). We pick δ such that:

$$\forall x \in R, r^\delta < x^\delta$$

In other words, of all points of R, let δ be such that r has the smallest δ-value. See Figure 5.5. For example, δ could be the angle bisector at

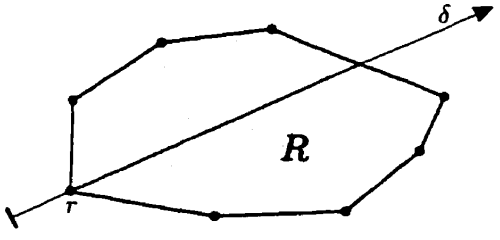


Figure 5.5: Illustrating the proof of Theorem 5.5.2.

r. Furthermore, and without loss of generality, we assume that δ is not

perpendicular to any edge of P or Q (it is always possible to find δ to satisfy this condition).

Now, getting back to our assumption: if p is not a vertex of P, then it either:

1. belongs on an edge of P
2. is an interior point of P.

Let us analyze each case:

1. Ifp belongs to an edge e, then consider the δ-value of the points on this edge. Since e is not perpendicular to δ, then one of its endpoints must have minimum δ-value. Since p is not an endpoint of e (because p is not a vertex), then pick the endpoint p' of e with smallest δ-value. See Figure 5.6: part of polygon P is shown, with edge e is shown as a thick line. Let $r' = p' + q$. We have a new point $r' \in P \oplus Q = R$

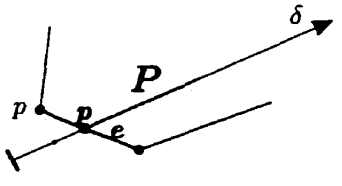


Figure 5.6: Illustrating the proof of Theorem 5.5.2.

but $r'^{\delta} < r^{\delta}$, which contradicts our original assumption.

2. If p is an interior point of P, construct a line ℓ through p and parallel to δ. ℓ intersects the P in a segment $[a, b]$. See Figure 5.7 Along $[a, b]$,

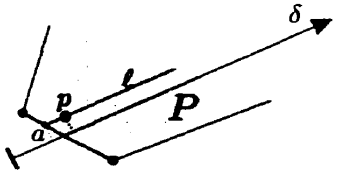


Figure 5.7: Illustrating the proof of Theorem 5.5.2.

the δ-value of points increase monotonically. Furthermore, since $p \neq a$ and $p \neq b$ (p being an interior point), p does not have the minimum δ-value along $[a, b]$. As in case 1, we can construct $r' = a + q$, $r' \in R$ with $r'^{\delta} < r^{\delta}$. This leads to the same contradiction of our assumption.

Therefore, we have that p must be a vertex of P. By symmetry of argument, q must also be a vertex of Q. Since r was chosen arbitrarily, the vertices of $P \oplus Q$ are all sums of vertices of P and Q. ∎

Theorem 5.5.3 *Given two convex polygons P and Q, the vertices of $P \oplus Q$ are sums of co-podal pairs of P and Q.*

Proof: Take a vertex r of $R = P \oplus Q$. Similarly to the proof of theorem 5.5.2, it is possible to construct an axis δ in the plane (with δ not perpendicular to any edge of P or Q) such that of all the points in R, r has minimal δ-value. Since $r \in R$, there exist $p \in P$ and $q \in Q$ such that $r = p + q$. Now, p and q must be the points with minimal δ-value for P and Q respectively. Furthermore, by theorem 5.5.2, p and q must be vertices of P and Q respectively.

Construct the two directed lines L_p and L_q, parallel to δ and having same direction as δ. Construct the directed lines $L_p^{\perp}$ and $L_q^{\perp}$, perpendicular to δ, and passing through p and q, respectively. As for the direction of these lines, let us have the $+\infty$ of δ lie to the right of $L_p^{\perp}$ and $L_q^{\perp}$. See Figure 5.8.

Now, since p and q are the points of P and Q with minimal δ-value, we have $L_p^{\perp}$ and $L_q^{\perp}$, two directed lines passing through exactly one point of P and Q, and such that P and Q lie to the right of $L_p^{\perp}$ and $L_q^{\perp}$, respectively.

Therefore, $L_p^{\perp}$ is a line of support for P at p, and $L_q^{\perp}$ is a line of support for Q at q. Since $L_p^{\perp}$ and $L_q^{\perp}$ are parallel and have the same direction, then p and q are co-podal. ∎

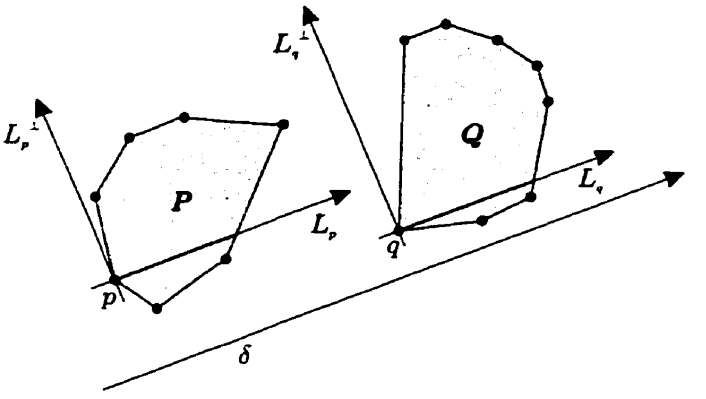


Figure 5.8: Illustrating the proof of Theorem 5.5.3.

Theorem 5.5.4 *Given two convex polygons P and Q with m and n vertices respectively, $P \oplus Q$ has no more than $m + n$ vertices.*

Proof: We have $P = \{p_1, \ldots, p_m\}$ and $Q = \{q_1, \ldots, q_n\}$. Let us represent these polygons using star-diagrams, showing (only) the direction of their edges.

P has m edges $e_1, \ldots, e_m$, so we may represent P as shown in Figure 5.9.

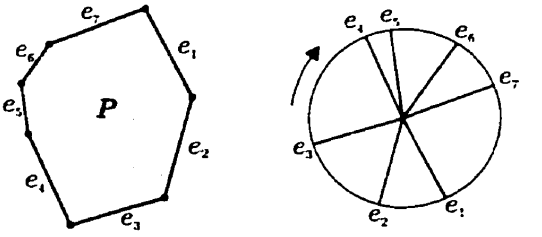


Figure 5.9: (a) Polygon P and its associated star diagram (b).

The vertices of P lie at the intersection of edges. Thus p_1 lies at the intersection of e_m and e_1, p_2 lies at the intersection of e_1 and e_2, and so on. A given vertex p_i accepts lines of support in a range of directions determined by the edges e_{i-1} and e_i — except for p_1 (for which the lines of support determined by e_m and e_1). Refer to Figure 5.10 (a). If we extend

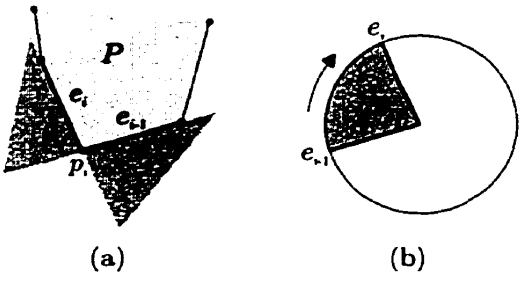


Figure 5.10: Illustrating the proof of Theorem 5.5.4.

edge e_{i-1}, the new-formed line makes an angle θ_i with edge e_i. Vertex p_i can only have lines of support in this range of directions. (Symmetrically, we could extend e_i and it would also make an angle θ_i with e_{i-1}. The same could be said in that case).

So to each vertex p_i is associated two edges e_{i-1} and e_i, and each pair of consecutive edges determines an angle (θ_i), which in turn determines the possible line of support at that vertex.

In the star-diagram representation of P, the disc is divided into sectors, each sector determined by two edges. In essence, each sector is associated to a vertex. The angle of the sector determines the possible lines of support for that vertex (refer again to Figure 5.10 (b)). In the case of P, there are m sectors, with angles $\theta_1, \ldots, \theta_m$, satisfying:

$$\sum_{i=1}^{m} \theta_i = 2\pi$$

Now, by theorem 5.5.3, the vertices of $P \oplus Q$ are sums of co-podal vertices of P and Q. Suppose p_i, q_j, ($p_i \in P$, $q_j \in Q$) form a co-podal pair between P and Q. Therefore, p_i and q_j admit parallel lines of support. Consider the star-diagrams for P and Q: if p_i and q_j admit parallel lines of support, then the sectors for p_i and q_j must intersect. Therefore, each co-podal pair between P and Q must correspond to an intersection of sectors for P and Q. How many sector intersections are there? Sector intersections are obtained by "superimposing" the two star-diagrams, in

other words, by adding the sectors of one polygon in the star diagram to the other. Suppose we take the star-diagram for P and add Q's sectors: the first time a sector is added, a maximum of two new sectors are created. Every subsequent time a sector of Q is added, one new sector is created. However, the last sector to be added is already present in the combined star diagram. Hence, the total number of combined sectors in the combined star diagram is bound by:

$$m + 2 + \underbrace{1 + 1 + \ldots + 1}_{n-2} = m + n$$

Since there can only be up to $m+n$ intersections, the number of co-podal pairs between P and Q is bound by $m+n$. Therefore, $P \oplus Q$ can have at most $m + n$ vertices. ∎

Finally, the following theorem enables the incremental construction of $P \oplus Q$. Refer to Figure 5.11.

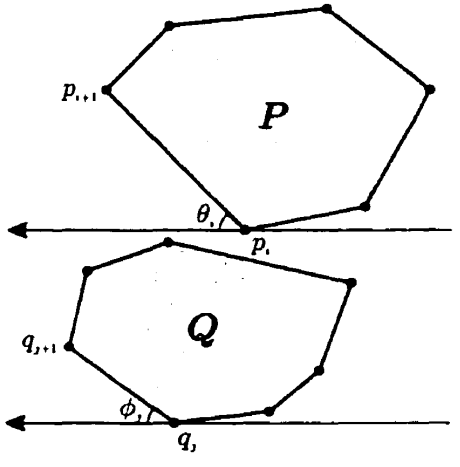


Figure 5.11: Illustrating the proof of Theorem 5.5.5.

Theorem 5.5.5 *Given two convex polygons P and Q. Consider the vertex $z_k = p_i \oplus q_j$ of $P \oplus Q$. Then, the succeeding vertex z_{k+1} is given by:*

$$z_{k+1} = \begin{cases} p_{i+1} \oplus q_j & \theta_i < \phi_j \\ p_i \oplus q_{j+1} & \phi_j < \theta_i \\ p_{i+1} \oplus q_{j+1} & \theta_i = \phi_j \end{cases}$$

where θ_i and ϕ_j are the angles of the edges at p_i and q_j, respectively.

Proof: It has already been determined in Theorem 5.5.3 that z_k and z_{k+1} must be sums of co-podal pairs of $P \oplus Q$. Therefore, p_i and q_j must be co-podal. Construct parallel lines of support L_p and L_q at p_i and q_j, and consider the angles θ_i and ϕ_j, as in Figure 5.11. There are three cases to be considered:

1. $\theta_i < \phi_j$. See Figure 5.12. In this case, p_{i+1} and q_j are co-podal.

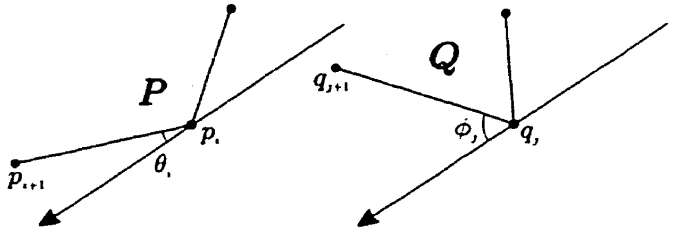


Figure 5.12: Illustrating the first case of Theorem 5.5.5.

2. $\phi_j < \theta_i$. See Figure 5.13. In this case, p_i and q_{j+1} are co-podal.

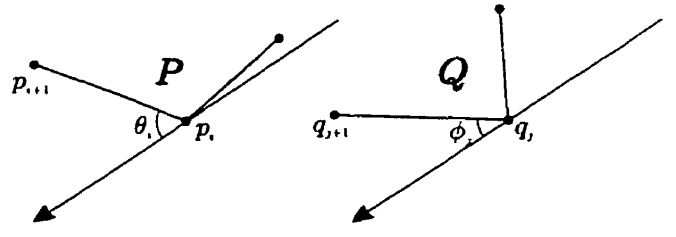


Figure 5.13: Illustrating the second case of Theorem 5.5.5.

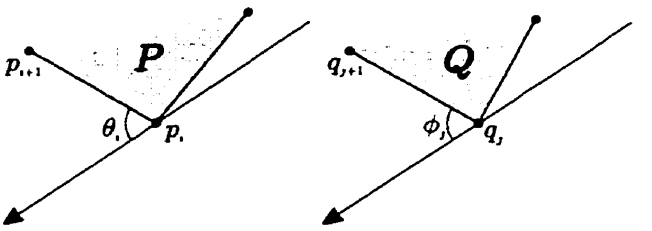


Figure 5.14: Illustrating the third case of Theorem 5.5.5.

3. $\phi_j = \theta_i$. See Figure 5.14. In this case, p_{i+1} and q_j, p_i and q_{j+1}, p_{i+1} and q_{j+1} all form co-podal pairs, and the edges of P and Q considered are parallel.

Now, in cases 1 and 2, when we are not dealing with parallel edges for P and Q, we claim that if $p_a \in P$ and $q_b \in Q$ form a co-podal pair then $p_a \oplus q_b$ must be a vertex of $P \oplus Q$.

Proof: p_a and q_b form a co-podal pair. Therefore, p_a and q_b admit parallel lines of support in the same direction. Assume without loss of generality that we have two directed lines L_a and L_b, lines of support for P and Q at p_a and q_b, respectively, and such that P and Q both lie to the right of their respective lines of support. Construct the directed lines $L_a^{\perp}$ and $L_b^{\perp}$, perpendicular to L_a and L_b, passing through p_a and p_b, respectively, and directed such that their $+\infty$ side lies to the right of L_a and L_b. Thus $L_a^{\perp}$ is parallel to $L_b^{\perp}$. Finally, construct an axis δ, parallel to and in the same direction as $L_a^{\perp}$ and $L_b^{\perp}$. See Figure 5.15. As in the proof of theorem 5.5.2, we shall consider the δ-value of various points. Now, p_a and q_b are points of minimal δ-value for their respective polygons. Let $r = p_a \oplus q_b$. Then r must the point of minimal δ-value for $P \oplus Q$. Again it is assumed that no edge of any polygon considered is perpendicular to δ, a condition that can always be satisfied.

Now, by theorem 5.5.1, $P \oplus Q$ is a convex polygon. As seen in the proof of Theorem 5.5.2, a given minimum δ-value point of a convex polygon can only be one of the vertices of the polygon. Therefore r is a vertex of

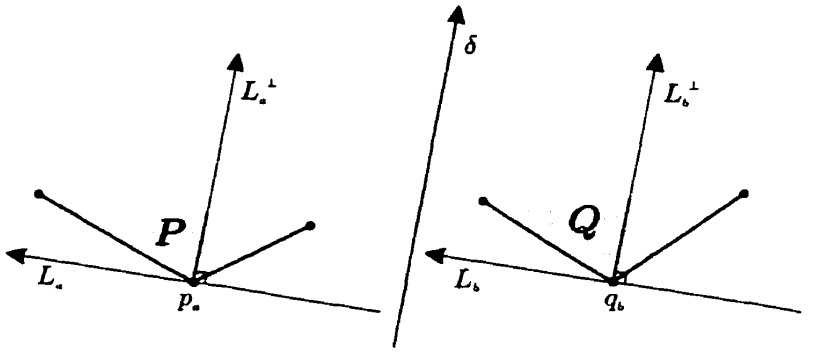


Figure 5.15: Illustrating the proof of the first two cases of Theorem 5.5.5.

$P \oplus Q$. Q.E.D.

Hence, for cases 1 and 2, we have:

1. if $\theta_i < \phi_j$ (p_{i+1} and q_j co-podal) $\Rightarrow z = p_{i+1} \oplus q_j$ is a vertex.
2. if $\theta_i > \phi_j$ (p_i and q_{j+1} co-podal) $\Rightarrow z = p_i \oplus q_{j+1}$ is a vertex.

In either case, we still have $z_k = p_i \oplus q_j$ a vertex of $P \oplus Q$ as well. Now, there cannot be any other possible sum of co-podal pairs forming a vertex between z_k and z, therefore $z = z_{k+1}$ in both cases.

Case 3: $\theta_i = \phi_j$. We have (p_i, q_j), (p_{i+1}, q_j), (p_i, q_{j+1}), (p_{i+1}, q_{j+1}) all forming co-podal pairs. Therefore each pair's sum can be considered as a potential vertex for $P \oplus Q$. Now, we have:

$$[p_i, p_{i+1}] \parallel [q_j, q_{j+1}]$$

$$\Rightarrow [p_i \oplus q_j, p_{i+1} \oplus q_j] \parallel [p_i \oplus q_j, p_i \oplus q_{j+1}] \parallel [p_i \oplus q_j, p_{i+1} \oplus q_{j+1}]$$

Therefore, $p_i \oplus q_j$, $p_i \oplus q_{j+1}$, $p_{i+1} \oplus q_j$, and $p_{i+1} \oplus q_{j+1}$ are all collinear. Furthermore, they all lie on the $[p_i \oplus q_j, p_{i+1} \oplus q_{j+1}]$ segment. The order of the two other points depends on the lengths of the edges $[p_i, p_{i+1}]$ and $[q_j, q_{j+1}]$. See Figure 5.16.

These four collinear points all belong to $P \oplus Q$. A closer look reveals that $p_i \oplus q_{j+1}$ and $p_{i+1} \oplus q_j$ cannot be vertices of $P \oplus Q$ since they belong to the interior of a segment of $P \oplus Q$ (the segment being $[p_i \oplus q_j, p_{i+1} \oplus q_{j+1}]$).

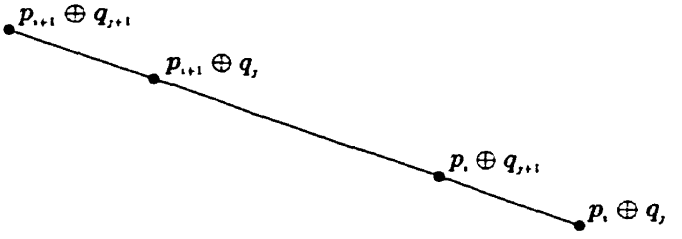


Figure 5.16: Illustrating the proof of the third case of Theorem 5.5.5.

We claim that $p_{i+1} \oplus q_{j+1}$ must be a vertex of $P \oplus Q$. First, note that p_i, p_{i+1}, and p_{i+2} cannot be collinear. The same can be said for q_j, q_{j+1}, and q_{j+2}. Hence p_{i+1} and q_{j+1} are not only co-podal, but they can have parallel lines of support in a range of directions. Therefore, it is (as above) possible to pick a direction δ in which p_{i+1} and q_{j+1} are minimal $\delta-values$. Hence $z = p_{i+1} \oplus q_{j+1}$ is the point with minimal δ-value for $P \oplus Q$, and (as seen above), this implies that z is a vertex.

Finally, we know now that $z_k = p_i \oplus q_j$ and $z = p_{i+1} \oplus q_{j+1}$ are both vertices. But since all co-podal pairs between the pairs (p_i, q_j) and (p_{i+1}, q_{j+1}) have already been considered, then there cannot be any vertices between z_k and z. Therefore, $z_{k+1} = z$. ∎

Theorem 5.5.5 describes a straightforward incremental algorithm for computing the vector sum of two convex polygons P and Q. The result suggests the use of the Rotating Calipers. All that is needed is a starting point, which can be the extrema of the polygons in a given direction.

Consider then the following algorithm. The input is two convex polygons P (m vertices) and Q (n vertices) given in standard form. The output is the list of vertices of $P \oplus Q$.

Algorithm VECTOR-SUM

1. Compute the vertices with minimum y-coordinate for P and Q. Denote these $y_{\min}(P)$ and $y_{\min}(Q)$. If these are not unique, pick the leftmost ones. Let these

vertices be our starting points, with two horizontal lines of support (calipers) through $y_{\min}(P)$ and $y_{\min}(Q)$. In other words let $p_i := y_{\min P}$ and $q_j := y_{\min}(Q)$.

2. Let $k = 1$. z_k, the first vertex of $R = P \oplus Q$ is equal to $p_i + q_j$.

3. Compute the angles θ_i and ϕ_j determined between the calipers and the edges of the polygons. Let $\delta = \min(\theta_i, \phi_j)$.

4. Rotate the lines by angle δ, thus 'hitting' a new vertex. The new co-podal pair then determines a new vertex of R. In other words, if a vertex of P was hit, then $z_{k+1} = p_{i+1} + q_j$. The two other cases are as described in Theorem 5.5.5.

5. Update k, p_i, q_j to the new 'current' vertices.

6. Repeat 3–5, until $y_{\min}(P)$ and $y_{\min}(Q)$ are both 'hit' again.

7. Output $R = \{z_1, \ldots, z_k\}$.

We conclude with the following:

Theorem 5.5.6 *Given convex polygons P and Q with m and n vertices respectively, algorithm VECTOR-SUM computes $P \oplus Q$ in $O(m + n)$ time.*

Proof: The correctness follows from Theorem 5.5.5, and from the fact that the first vertex output is the sum of extreme points (and is therefore a vertex of the vector sum).

The time complexity analysis is straightforward. Step 1 consists of two operations of $O(m)$ and $O(n)$ respectively, along with some constant time initialization. The total running time for this step is therefore $O(m + n)$. Step 2 has $O(1)$ cost. Steps 3–5 make up the main loop. Since there is a fixed number of simple computations for one iteration of these three steps, we can say one iteration takes $O(1)$ time. Since at each iteration a new vertex of either polygon is hit, and the loop stops when the original

vertices are reached, the loop is iterated $O(m+n)$ times. Therefore, steps 3–5 take $O(1)$ time and step 6 takes $O(m+n)$ time. Finally, the vector sum is output (with a unit time cost) in step 7. We conclude the total running time of the algorithm, dominated by steps 1 and 6 is $O(m+n)$. ■

Chapter 6

Thinnest Transversals

6.1 Thinnest Strip Transversal

The thinnest strip transversal is essentially a facility location problem, in which a number of "facilities" exist for a given number of "customers". The goal is to find the best location for the facilities to minimize for instance the maximum distance from a customer to a facility, or to minimize the sum of the customer-facility distances [31].

Such problems occur often in transportation and management sciences. Many versions of the problem exist in the field of computational geometry. The customers can be points, line segments, or simple convex objects, as can the facilities. Furthermore, the customers can also have a weight associated to them, so that the minimum sum of distances to customers becomes a weighted sum. In the case of the facility being a line, the problem is analogous to that of linear approximation, which has application in statistics, computer vision, computer graphics, and pattern recognition [67]. Algorithms exist to suit each version.

The specific problem we focus on in this work is as follows: the facility is a line on the plane and the customers are convex polygons. Thus the problem can either be viewed as a linear approximation problem, or a facility location one. In either

case, the algorithm studied is one presented by Robert and Toussaint [66, 67]. Their papers study the problem from both points of view, along with some different versions mentioned above.

The goal in this specific problem is to find a line l minimizing the maximum distance from l to any polygon. We define this distance as follows: suppose we have a polygon P and a line l lying on the plane. Then

$$\text{dist}(P, l) = \min_{\forall p \in P} \{\text{dist}(p, l)\}.$$

Robert and Toussaint show this problem is equivalent to finding the *strip* — the region in the plane bounded by parallel lines — of minimum width intersecting all polygonal customers. Once the strip has been found, the *medial axis* of this strip is the desired line l. The *medial axis* of a polygon is the set of points in the interior that have more than one closest point to the boundary. In the case of a strip, the medial axis is the line equidistant from the strip's boundaries.

We present the authors' algorithm, which involves the use of the Rotating Calipers. Before we proceed, however, we must define some concepts:

Definition 6.1.1 *Given a line l in the plane, defined by the equation $ax + by + c = 0$ (constrained by either $b > 0$ or $a = -1$). l determines the upper half-plane $H_u(l) = \{p = (p_x, p_y) | ap_x + bp_y + c \geq 0\}$ and the lower half-plane $H_l(l) = \{p = (p_x, p_y) | ap_x + bp_y + c \leq 0\}$.*

The constraints on a and b in the above definition are to ensure that:

1. the equation is valid, e.g. we cannot have $a = b = 0$ and $c \neq 0$.
2. the concepts of *upper* and *lower* half planes are well-defined in case the line is vertical (i.e., when $b = 0$). In that case, the *upper* half-plane is defined as the region containing the negative infinity side of the x-axis.

Definition 6.1.2 *Given two parallel lines l_1 and l_2, the* strip *S is defined as the region of the plane bounded by l_1 and l_2.*

In other words, supposing that $l_2 \subset H_u(l_1)$, $S = H_u(l_1) \cap H_l(l_2)$.

Definition 6.1.3 *The* width *of a strip S bounded by lines l_1 and l_2 is the orthogonal distance between l_1 and l_2. The* orientation *of S is defined as the angle between l_1 and the positive x-axis.*

Definition 6.1.4 *Given a polygon P, for a given orientation θ, the* lower tangent *of P is the line $t_l(P,\theta)$ with angle θ such that $t_l(P,\theta) \cap P \neq \emptyset$ and $P \subset H_u(t_l(P,\theta))$. The* lower point *$p_l(P,\theta)$ of P is a point p belonging to $t_l(P,\theta) \cap P$. We similarly define the* upper tangent *of P, $t_u(P,\theta)$, such that $t_u(P,\theta) \cap P \neq \emptyset$ and $P = H_l(t_l(P,\theta))$ and the* upper point *$p_u(P,\theta)$, belonging to $t_u(P,\theta) \cap P$.*

In our case, we are dealing with a set of polygons, called a *family*. For a given family F, the set of *lower points* and *upper points* of all members of F are denoted $LP(F,\theta)$ and $UP(F,\theta)$, respectively. The following result is the key to finding the minimum width strip for a family of polygons.

Lemma 6.1.1 *Given a family F of convex polygons, and an orientation θ, a strip $S = H_u(l_1) \cap H_l(l_2)$ of width greater than 0 is the minimum width strip for F if and only if:*

$$\exists P, Q \in F \text{ such that } P \cap H_u(l_1) \subseteq l_1 \text{ and } Q \cap H_l(l_2) \subseteq l_2$$

Proof: $\Rightarrow$: Suppose we are given a strip S' that is the minimum width strip intersecting all members of F. See Figure 6.1.

Suppose then that such polygons P and Q do not exist. Since S' intersects all members of F, it must be that at least one polygon farthest from S'

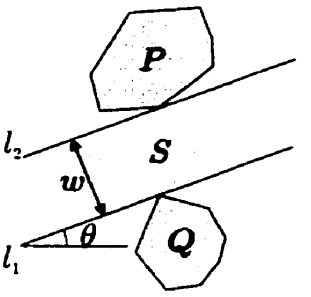


Figure 6.1: Illustrating Lemma 6.1.1.

intersect the interior of the strip (i.e., the region of the strip excluding its boundary lines). But then the width of the strip can be decreased, while maintaining the intersection. This implies S' is not the minimum width strip.

$\Leftarrow$: Suppose a strip S is defined such that two polygons P, Q, exist satisfying the conditions in Lemma 6.1.1. Then (keeping in mind the orientation is fixed), since P and Q only intersect the strip at its boundary, decreasing the width of the strip implies that $S \cap P = \emptyset$ or $S \cap Q = \emptyset$ or both. In any case, this implies a strip of smaller width cannot be found to intersect all members of F, including P and Q. Therefore, S must be the minimum width strip. ∎

From Lemma 6.1.1 it can deduced that:

$$l_1 = t_l(CH(UP(F,\theta)),\theta)$$

and

$$l_2 = t_u(CH(LP(F,\theta)),\theta)$$

See Figure 6.2. Given F and an orientation θ, the optimum strip is obtained by lines $l_1 = t_l(CH(UP(F,\theta)),\theta)$ and $l_2 = t_u(CH(LP(F,\theta)),\theta)$. With this construction, $S = H_u(l_1) \cap H_l(l_2)$ is the minimum-width strip with orientation θ for F.

Given this criterion for finding the optimum strip for a given orientation, the algorithm yielding the minimum width strip (independent of direction) directly follows:

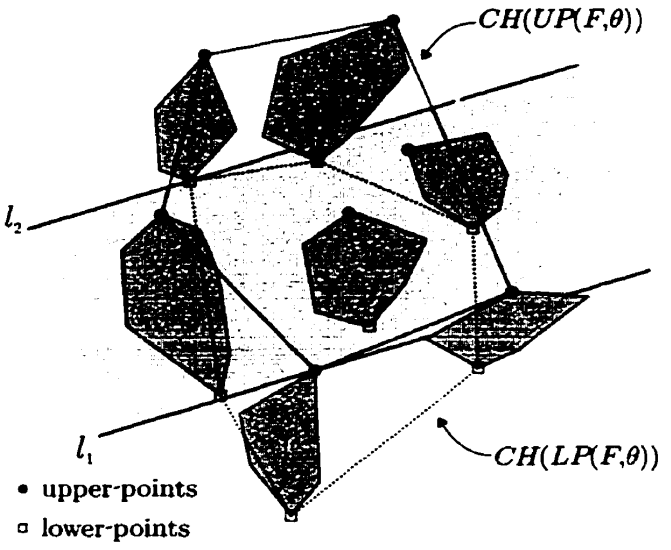


Figure 6.2: The minimum-width strip with orientation θ.

because the number of relevant orientations to be considered is limited, if one finds the optimum strip for each relevant θ, the minimum width strip for F is subsequently found.

Since the focus of this work is the analysis of the role of the Rotating Calipers in the procedure, we do not deal with the correctness and the complexity of the algorithm, but merely explain the idea behind it. The Rotating Caliper algorithm is used essentially to find the sequence of *lower* and *upper* points for a given polygon in the family.

The following is the detailed procedure for finding the lower and upper points sequences for a polygon P.

1. Find the vertices with minimum and maximum y-coordinates, call them p and q, respectively, and set the calipers in horizontal position passing through p and q.

2. Rotate the calipers counterclockwise, say by an angle θ, until a new vertex r is hit. If two vertices are hit (in the case of parallel edges), let $r = p_{next}$, $r' = q_{next}$.

3. Add $[0, \theta]$ to the range of angles for which p and q are lower and upper points, respectively.

4. Update p and q (unless two vertices were hit, only one gets updated, becoming r).

5. Repeat steps 2–4, until we have rotated by an angle greater than π.

Thus, for any direction θ, we know the lower and upper points for P. At this angle, the tangents to P pass through these points. The points are unique unless the tangents coincide with an edge of P. The orientations at which this case occurs are called *critical directions*. In those cases, the lower (or upper) point will be the next tangent point when rotating the calipers counterclockwise.

Two other important factors need to be considered. First, not all orientations need to be considered in order to find the minimum width strip. Consider the sequence of critical directions for all members of F, and take two consecutive orientations θ_1 and θ_2. Within this range of directions, the upper and lower points are the same: they only change at the critical directions. Therefore, given the minimum width strip for orientation θ, with $\theta_1 < \theta < \theta_2$, it is always possible to rotate the strip about its tangent points, keeping θ within that range, and maintaining the strip's validity. Then the strip cannot be optimum for such an orientation, for it is always possible to rotate it such that its width decreases. For this reason, only strips with critical directions should be considered in the algorithm. So, given the strip for θ_1, we rotate it until the orientation is θ_2, at which time we need to update. This is easily done, since only one point will be updated unless we have parallel edges among members of F, and because of this, $CH(LP(F, \theta_2))$ and $CH(UP(F, \theta_2))$ need not be recomputed, but merely updated. Again, if we are not dealing with parallel edges, then the update involves removing one point and adding another.

If these steps are followed for all relevant directions, the optimum strip is output. The complete algorithm, its correctness and complexity analysis are given in [66].

Chapter 7

Conclusion

Throughout this work, many applications of the Rotating Calipers paradigm have been studied. The variety and extent of the various problems suggests that the algorithm exploits key properties that all these problems hold in common. Furthermore, the paradigm's ease of implementation makes it an ideal candidate to solve a problem (or a subproblem) whenever it can be applied. When should one think of using the Rotating Calipers? In many cases — e.g., when anti-podal pairs or co-podal pairs are involved — the algorithm comes to mind due to some characterization of the solution. When this is not the case, e.g., for the spiral triangulation, use of the Rotating Calipers is intuitive because the algorithm can be so easily visualized.

Some problems discussed in this work warrant a more detailed study. A possible connection was suggested between the minimum area enclosing rectangle and the width for a convex convex polygon. Although counterexamples exist, the connection cannot be dismissed; it would be interesting to study the family of polygons for which the same edge-vertex pair determine both the width and the minimum area enclosing box. Can similar parallels be drawn between other problems?

Other areas of study include possible improvements to existing algorithms. Consider the problem of finding the maximum distance between two convex polygons. The Rotating Calipers solution has linear time complexity. Is it possible to improve

this, by further characterizing the solution? In the case of finding the minimum distance between convex polygons, optimal algorithms already exist. We know then that it is possible to improve on the $O(n)$ running time of the algorithm proposed in this work.

Finally, we remind the reader that this work is not an exhaustive collection of all the problems solved using the Rotating Calipers. An equivalent of the paradigm exists in three dimensions, with applications to a variety of problems.

The extent of the paradigm, its conceptual simplicity and its ease of implementation make the Rotating Calipers a highly useful and versatile tool that deserves further research.

Appendix A

Rotating Calipers homepage

As mentioned in the introduction to this work, a summarised version of all the problems has been prepared, and is available on the Internet. The information is in the form of a webpage, and can be found at

http://cgm.cs.mcgill.ca/~orm/rotcal.html

Although proofs and details are missing, this webpage can serve as a pedagogical tool to anyone interested in Computational Geometry, or the Rotating Calipers paradigm specifically.

Appendix B

Rotating Calipers applet

To demonstrate the Rotating Calipers paradigm "at work", an interactive Java applet is provided. The user can input one or two convex polygons on a drawing area, and then select for a problem to be solved using the Rotating Calipers algorithm. The problems are separated for practicality according to whether they are "one-polygon" or "two-polygon" problems.

The applet, as well as all information pertaining to its use and installation (where necessary) is provided in the author's Rotating Calipers webpage. The applet can be accessed at the following address:
http://cgm.cs.mcgill.ca/~orm/rotcal.html

Bibliography

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

[2] A.V. Aho and J.D. Ullman. *Foundations of Computer Science.* W.H. Freeman and Company, New York, 1992.

[3] E.M. Arkin, M. Held, J.S.B. Mitchell, and S.S. Skiena. Hamiltonian triangulations for fast rendering. In J. van Leeuwen, editor, *Algorithms - ESA '94*, volume 855 of *Lecture Notes Comput. Sci.*, pages 36–47, September 1994.

[4] E.M. Arkin, M. Held, J.S.B. Mitchell, and S.S. Skiena. Hamiltonian triangulations for fast rendering. *Visual Comput.*, 12(9):429–444, 1996.

[5] D. Avis. Lower bounds for geometric problems. In *Proc. Allerton Conference*, pages 35–40, Urbana, IL, October 1980.

[6] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. *Comput. Vision Graph. Image Process.*, 63, 1996. In press.

[7] G. Barequet and M. Sharir. Piecewise-linear interpolation between polygonal slices. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 93–102, 994.

[8] H. Baumgarten, H. Jung, and K. Mehlhorn. Dynamic point location in general subdivisions. *J. Algorithms*, 17:342–380, 1994.

[9] B.K. Bhattacharya and G.T. Toussaint. Efficient algorithms for computing the maximum distance between two finite planar sets. *Journal of Algorithms*, 4, 1983.

[10] Prosenjit Bose and Godfried Toussaint. Characterizing and efficiently computing quadrangulations of planar point sets. *Comput. Aided Geom. Design*, 14:763–785, 1997.

[11] K.Q. Brown. Geometric transforms for fase geometric algorithms. Technical report, Dep. Comput. Sci., Carnegie-Mellon Univeristy, 1979.

[12] B. Chazelle. *Computational geometry and convexity.* PhD thesis, Carnegie-Mellon University, 1980.

[13] B. Chazelle. Algorithms for computing depths and layers. In *Proc. Allerton Conf. Commun. Control Comput.*, pages 427–436, 1983.

[14] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31:509–517, 1985.

[15] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[16] B. Chazelle and D. P. Dobkin. Detection is easier than computation. In *Proc. 12th Annu. ACM Sympos. Theory Comput.*, pages 146–153, 1980.

[17] B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir. Diameter, width, closest line pair, and parametric searching. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 120–129, 1992.

[18] B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric searching. *Discrete Comput. Geom.*, 10:183–196, 1993.

[19] Bernard Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Trans. Graph.*, 3(2):135–152, 1984.

[20] S. W. Cheng and R. Janardan. New results on dynamic planar point location. *SIAM J. Comput.*, 21:972–999, 1992.

[21] F. Chin and C. A. Wang. Optimal algorithms for the intersection and the minimum distance problems between planar polygons. *IEEE Trans. Comput.*, C-32(12):1203–1207, 1983.

[22] K. L. Clarkson, R. E. Tarjan, and C. J. Van Wyk. A fast Las Vegas algorithm for triangulating a simple polygon. *Discrete Comput. Geom.*, 4:423–432, 1989.

[23] S.A. Cook and R.A. Reckhow. Time bounded random access machines. *J. Computer and Systems Sciences*, 7(4):354–375, 1973.

[24] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Massachusetts, 1990.

[25] A.A.N. DePano. Rotating Calipers revisited: optimal polygonal enclosures in optimal time. In *Proc. ACM South Central Regional Conference*, Lafayette, LA, November 1987.

[26] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[27] H. Edelsbrunner. On computing the extreme distance between two convex polygons. Technical Report F96, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1982.

[28] H. Edelsbrunner, M. H. Overmars, and D. Wood. Graphics in Flatland: A case study. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 35–59. JAI Press, London, England, 1983.

[29] H. Everett, W. Lenhart, M. H. Overmars, T. Shermer, and J. Urrutia. Strictly convex quadrilateralizations of polygons. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 77–83, 1992.

[30] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Trans. Graph.*, 3(2):153–174, 1984.

[31] R. L. Francis and J. A. White. *Facility Layout and Location.* Prentice Hall, Englewood Cliffs, NJ, 1974.

[32] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Comm. A.C.M.*, 18:409–413, July 1975.

[33] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett.*, 7:175–179, 1978.

[34] U. Grenander. *Pattern Synthesis.* Springer-Verlag, New York, 1976.

[35] F. C. A. Groen, P. W. Verbeek, N. DeJong, and J. W. Klumper. The smallest box around a package. *Pattern Recogn.*, 14(1–6):173–178, 1981.

[36] Leonidas J. Guibas and F. F. Yao. On translating a set of rectangles. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 61–77. JAI Press, London, England, 1983.

[37] L.J. Guibas, L. Ramshaw, and J. Stolfi. A kinetic framework for computational geometry. In *Proc. 24th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 100–111, 1983.

[38] E. Heighway. A mesh generator for automatically subdividing irregular polygons into quadrilaterals. *IEEE Trans. Magn.*, 19(6):2535–2538, 1983.

[39] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. In *Proc. 2nd Scand. Workshop Algorithm Theory*, volume 447 of *Lecture Notes Comput. Sci.*, pages 380–392. Springer-Verlag, 1990.

[40] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[41] S. Hertel and K. Mehlhorn. Fast triangulation of simple polygons. In *Proc. 4th Internat. Conf. Found. Comput. Theory*, volume 158 of *Lecture Notes Comput. Sci.*, pages 207–218. Springer-Verlag, 1983.

[42] M. E. Houle and G.T. Toussaint. Computing the width of a set. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(5):761–765, September 1988.

[43] K. Ichida and T. Kiyono. Segmentation of plane curves. *Trans. Elec. Commun. Eng., Japan*, 58-D:689–696, 1975.

[44] H. Imai and M. Iri. Polygonal approximations of a curve: Formulations and solution algorithms. In G.T. Toussaint, editor, *Computational Morphology*. North-Holland, Amsterdam, The Netherlands, 1988.

[45] B. P. Johnston, J. M. Sullivan, and A. Kwasnik. Automatic conversion of triangular finite meshes to quadrilateral elements. *Internat. J. Numer. Methods Eng.*, 31(1):67–84, 1991.

[46] J. Kahn, M. M. Klawe, and D. Kleitman. Traditional galleries require fewer watchmen. *SIAM J. Algebraic Discrete Methods*, 4:194–206, 1983.

[47] J. M. Keil and J.-R. Sack. Minimum decompositions of polygonal objects. In G. T. Toussaint, editor, *Computational Geometry*, pages 197–216. North-Holland, Amsterdam, Netherlands, 1985.

[48] D. G. Kirkpatrick, M. M. Klawe, and R. E. Tarjan. Polygon triangulation in $O(n \log\log n)$ time with simple data structures. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 34–43, 1990.

[49] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, second edition, 1973.

[50] D.E. Knuth. Big omicron and big omega and big theta. *ACM SIGACT News*, 8(2):18–23, 1976.

[51] Y. Kurozumi and W.A. Davis. Polygonal approximation by the minimax method. *Comput. Graphics Image Processing*, 19:248–264, 1982.

[52] D.T. Lee. Personal communication, 1998.

[53] N. J. Lennes. Theorems on the simple finite polygon and polyhedron. *Amer. J. Math.*, 33:37–62, 1911.

[54] W. Lipski, Jr., E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On two-dimensional data organization , Part II. *Fundam. Inform.*, 2:245–260, 1979.

[55] T. Lozano-Pérez and M.A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22:560–570, 1979.

[56] A. Lubiw. Decomposing polygonal regions into convex quadrilaterals. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 97–106, 1985.

[57] M. McKenna and G. T. Toussaint. Finding the minimum vertex distance between two disjoint convex polygons in linear time. *Comp. & Maths. with Appls.*, 11(12):1227–1242, 1985.

[58] T. Ohtsuki. Minimum dissection of rectilinear polygons. In *Proc. IEEE Symp. on Circuits and Systems*, pages 1210–1213, 1982.

[59] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* John Wiley & Sons, Chichester, UK, 1992.

[60] J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM*, 24:574–578, 1981.

[61] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[62] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor. A new linear algorithm for intersecting convex polygons. *Comput. Graph. Image Process.*, 19:384–391, 1982.

[63] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, 1977.

[64] F. P. Preparata and R. Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM J. Comput.*, 18:811–830, 1989.

[65] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, 1985.

[66] J. Robert and G.T. Toussaint. Computational geometry and facility location. Technical Report SOCS 90.20, McGill University, School of Computer Science, Montreal, PQ, 1990.

[67] J.-M. Robert and G. Toussaint. Linear approximation of simple objects. *Comput. Geom. Theory Appl.*, 4:27–52, 1994.

[68] J.-R. Sack and G. T. Toussaint. A linear-time algorithm for decomposing rectilinear polygons into convex quadrilaterals. In *Proc. 19th Allerton Conf. Commun. Control Comput.*, pages 21–30, 1981.

[69] J.-R. Sack and G. T. Toussaint. Guard placement in rectilinear polygons. In G. T. Toussaint, editor, *Computational Morphology*, pages 153–175. North-Holland, Amsterdam, Netherlands, 1988.

[70] W. J. Schroeder and M. S. Shephard. Geometry-based fully-automatic mesh generation and the Delaunay triangulation. *Internat. J. Numer. Methods Eng.*, 26(11):2503–2515, 1988.

[71] J. T. Schwartz. Finding the minimum distance between two convex polygons. *Inform. Process. Lett.*, 13:168–170, 1981.

[72] M. I. Shamos. Problems in computational geometry. Technical report, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1977.

[73] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 17th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 208–215, 1976.

[74] M.I. Shamos. *Computational geometry.* PhD thesis, Yale University, 1978.

[75] 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.

[76] G. T. Toussaint. Computational geometric problems in pattern recognition. In J. Kittler, K. S. Fu, and L. F. Pau, editors, *Pattern Recognition Theory and Application*, pages 73–91. D. Reidel Publishing Company, Boston, MA, 1982.

[77] G. T. Toussaint. An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons. In *Proc. IEEE Internat. Conf. Pattern Recogn.*, pages 465–467, 1984.

[78] G. T. Toussaint. An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons. *Computing*, 32:357–364, 1984.

[79] G. T. Toussaint. A simple linear algorithm for intersecting convex polygons. *Visual Comput.*, 1:118–123, 1985.

[80] G.T. Toussaint. Pattern recognition and geometrical complexity. In *Proc. Fifth International Conference on Pattern Recognition*, pages 1324–1347, Miami Beach, December 1980.

[81] G.T. Toussaint. Solving geometric problems with the 'rotating calipers'. In *Proc. MELECON*, Athens, Greece, 1983.

[82] G.T. Toussaint. Movable separability of sets. In G.T. Toussaint, editor, *Computational Geometry*, pages 335–375. North-Holland, Amsterdam, The Netherlands, 1985.

[83] G.T. Toussaint. New results in computational geometry relevant to pattern recognition in practice. In E.S. Gelsema and L.N. Kanal, editors, *Pattern Recognition in Practice II*, pages 135–146. North-Holland, Amsterdam, Netherlands, 1986.

[84] G.T. Toussaint. An output-sensitive polygon triangulation algorithm. In *Proc. of the 8th Internat. Conf. on Computer Graphics*, pages 443–446, 1990.

[85] G.T. Toussaint and B.K. Bhattacharya. Optimal algorithms for computing the minimum distance between two finite planar sets. In *Proc. Fifth International Congress of Cybernetics and Systems*, Mexico City, August 1981.

[86] G.T. Toussaint and J.A. McAlear. A simple $O(n \log n)$ algorithm for finding the maximum distance between two finite planar sets. *Pattern Recognition Letters*, 1:21–24, October 1982.

[87] T. Wang. A C^2-quintic spline interpolation scheme on triangulation. *Comput. Aided Geom. Design*, 9:379–386, 1992.

[88] Y. F. Wang and J. K. Aggarwal. Surface reconstruction and representation for 3-d scenes. *Pattern Recognition*, 19:197–207, 1986.

[89] S. K. Wismath. Triangulations: An algorithmic study. M.Sc. thesis, Dept. Comput. Inform. Sci., Queen's Univ., Kingston, ON, July 1980. Report 80-106.

[90] I.M. Yaglom and V.G. Boltyanskii. *Convex Figures*. Holt, Rinehart and Winston, New York, 1961.

[91] P. Yoeli. Compilation of data for computer-assisted relief cartography. In J. C. Davis and M.J. McCullagh, editors, *Display and Analysis of Spatial Data*, pages 352–367. Wiley, New York, 1995.