Cutting a Polygon with a Line

By

Ervin Ruci

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

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

January 2008

Your file Votre référence
ISBN: 978-0-494-36841-1
Our file Notre référence
ISBN: 978-0-494-36841-1

Abstract

Given a simple polygon P and an integer $K > 1$, we want to compute the set of straight lines in the Cartesian plane that cut this polygon into exactly K simple polygons. We call this set of lines a K-separator and call this problem the K-separator problem.

We present an algorithm that finds the K-separators of an n-vertex simple polygon, for all $K > 0$, in $O(n^2)$ total time.

We prove that the decision problem *given an integer $K > 2$ and an edge of the polygon, is there a line through this edge that cuts the polygon in exactly K pieces?*, is 3SUM-HARD. For the special case when $K = 2$, we show that the decision problem can be solved in $O(n \log(n))$ time.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I would especially like to express my sincere gratitude to my thesis supervisor Michiel Smid for the guidance and encouragement he has provided in helping me explore the field of Computational Geometry.

I would also like to thank Prosenjit Bose for the support and advice he provided in developing the ideas presented in this work.

Contents

List of Figures

Chapter 1

Introduction

This chapter presents some preliminaries, the motivation, related work, scope of the research and organization of this thesis.

1.1 Preliminaries

This section presents some basic concepts and the terminology we are going to use throughout this thesis. A simple polygon is a closed curve on the plane consisting of three or more line segments (sides) connected such that no two segments cross. Each segment of the polygon has two vertices, each of which is shared with exactly one other segment of the closed boundary. These segments are generally referred to as the edges of the polygon, and the area of the plane that they enclose is referred to as the interior of the polygon.

A polygon is convex if and only if any segment whose two endpoints are on the polygon boundary, is completely contained in the polygon.

For the purpose of our study, we will be referring to a *simple polygon* P. The number of vertices will be denoted by n.

The convex hull of a simple polygon is the smallest convex polygon that completely contains the simple polygon. Cutting a polygon P is the operation of partitioning the polygon into a number of sub-polygons whose union is P. The notion of studying the different ways of cutting a polygon P is referred to as the *polygon cutting problem.* There are several aspects of the polygon cutting problem depending on whether chords, diagonals or straight lines are used to cut the polygon. A *diagonal* is a straight line segment joining two vertices of the polygon, that is completely contained in the interior of the polygon. A *chord* is a line segment joining any two points on the boundary of P that is completely contained in the interior of P.

1.2 Motivation

The original motivation for this thesis was a problem [18] involving manufacturing of large 3D objects. It was proposed to find a plane that cuts a simple polyhedron into exactly two pieces, which would then make it possible to build the object involving a simpler manufacturing process. Finding all such planes became the problem that was the original inspiration for this work. This thesis addresses this problem in 2D.

1.3 Related Work

Over two decades ago Chazelle's technique of polygon cutting [5] focuses on partitioning polygons via diagonals or chords while maintaining certain properties. Chazelle's original theorem dealt with finding a diagonal in a simple polygon that partitions the polygon into two subpolygons with approximately the same number of vertices (no more than $2n/3$ and at least $n/3$) in $O(n)$ time.

Bose et al [2] studied various optimization problems in the context of polygon cutting using chords. They showed how to cut a simple polygon P in $O(n)$ time using $O(\log(n))$ cuts to obtain a given ratio of the area of the cuts, as well as other specific optimization problems such as finding the shortest *chord* that cuts off α times the area of P.

Another aspect of polygon cutting is the so called *stock cutting* problem. Overmars and Welzl [23] considered the problem of cutting out a polygon from another polygon in the cheapest possible way using guillotine cuts. Several other studies followed in that vein such as Contreras [7] who studied the problem of cutting pieces from a convex polygon with given areas. Approximation algorithms for cutting out polygons with line cuts and ray cuts were presented by Daescu [8] in the context of providing cutting sequences within a certain factor of approximation. They approximate the polygon shape and area. This type of work mainly concentrated on convex polygons, and the type of shapes that could be cut out of them.

The problem of bisecting the area of a polygon using a straight line in relation to the Ham-sandwich [26] problem was studied by Diaz and O'Rourke [9]. This gave rise to a number of papers on cutting polygons under various area constraints. Shermer [24] presented a linear algorithm for bisecting a convex polygon, Stojmenovic [25] studied the problem of bisecting two disjoint convex polygons with a straight line in linear-time. Boehringer et al [1] studied the problem of area bisectors for simple polygons (possibly with holes) and showed that there can be $\Omega(n^2)$ combinatorially distinct area bisectors for a simple polygon with n vertices. *Combinatorially distinct* means that two *combinatorially distinct* lines induce different partitions of the vertices of the polygon.

Van Kreveld et al [28] studied the problem of cutting a simple polygon into two pieces such that the pieces have a minimum bounding square. There is also some work in data structures for cutting queries in convex polygons and generalizations to polyhedra [17].

The few articles on simple polygon cutting using a straight line revolve around the problem of finding the area (of possibly non-contiguous pieces) of a simple polygon left out on one side of a line [19], without taking into consideration the number of pieces that the polygon is cut into.

This thesis will build on this body of research by focusing on a generalized solution and analysis of the Polygon Cutting problem using a straight line and taking into account only the discrete number of contiguous polygonal pieces that are cut by a straight line.

1.4 Scope of the Research

Given a simple polygon P and a straight line ℓ in the plane, ℓ either does not intersect the interior of P, or it intersects the interior of P and separates it into 2 or more simple polygons.

The number of simple polygons that the intersection of this line with the polygon produces, will be referred to throughout this thesis as K, and the problem of computing the set of lines that cause the polygon to be separated into exactly K pieces, will be called the K-Separator problem.

We find that no prior work has been done with regard to a number of questions arising from the line-polygon intersection problem:

Problem 1. *Given a simple n-vertex polygon P, compute a partitioning of the set of* *all lines into groups, such that for each group, the lines in this group cut P into the same number of pieces. For each group of the partition, compute the number of pieces that P is cut into by the lines in this group.*

We present a simple $O(n^2)$ time algorithm based on arrangements to solve this problem. We also show that this is worst-case optimal.

Problem 2. *Given a simple n-vertex polygon P, an edge e of P and an integer K, is there a line that intersects e and cuts the polygon into exactly K pieces?*

We show that this problem can be solved in $O(n^2)$ time. We also show that the problem of determining if such a line exists which intersects a fixed edge of P, and cuts P into $K > 2$ pieces is 3SUM-HARD. [13]

For 3SUM-hard problems the best known algorithmic solutions need $\Omega(n^2)$ time. [13]

For the special case when $K = 2$, i.e, determining if a line exists that intersects a fixed edge of P, and cuts P into exactly 2 pieces, we show that it can be solved in $O(n \log(n))$ time.

This research is different from previous work on polygon cutting in the sense that we are only concerned with the discrete number of cuts, i.e, *the parameter K*, not the shape and type of the cuts themselves or the collective area of the cuts on either side of the line. We also focus on polygon cutting caused by a line as opposed to chords or diagonals. Thus, our problem is succinctly defined as the study of questions arising from the intersection of two simple geometric objects: a straight line and a simple polygon.

1.5 Organization of the Thesis

The rest of this thesis is organized as follows:

In Chapter 2, we focus on developing an algorithm that solves Problem 1 as defined in Section 1.3.

In Chapter 3, we provide a solution to Problem 2 using a reduction from a known 3SUM-hard problem [13] to prove that the problem of finding a line through a given edge of P, with the property of cutting P into K pieces, where $K > 2$, is 3SUM-hard.

In Chapter 4, we deal with the special case of Problem 2 when $K = 2$. We use the hourglass data structures introduced in [15] to show that the same existence question stated in Chapter 3, can be answered in sub-quadratic time for $K = 2$.

In Chapter 5, we conclude this thesis and discuss future work.

Chapter 2

Enumerating combinatorially distinct lines

This chapter presents the methodology and the algorithm used to enumerate all sets of K-separator lines.

2.1 Preliminaries

Given a simple polygon P with n vertices (see Figure 2.1) our goal is to characterize all lines that intersect this polygon and decompose it into a fixed number of simple polygons (which will be referred to as *pieces*). We assume that no three vertices of P are on the same line.

Throughout the rest of this chapter we will refer to the simple n-vertex polygon as P, and the generic line as ℓ.

Claim 3. *If ℓ intersects P and ℓ does not contain a vertex of P, then ℓ intersects an even number of edges of P.*

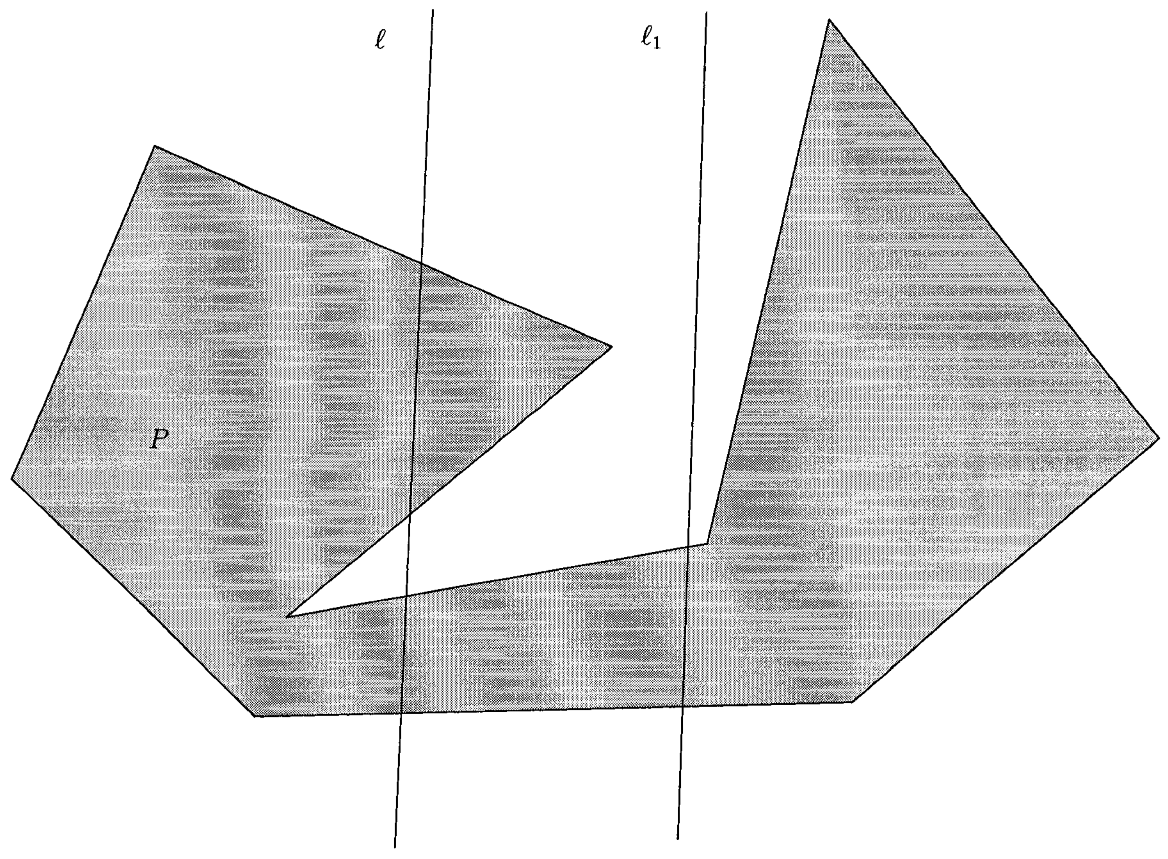


Figure 2.1: A Simple Polygon P. The line ℓ cuts P into three pieces, whereas the line ℓ_1 cuts P into two pieces.

Proof. Imagine walking along the line ℓ starting at infinity. Each point p where ℓ enters the interior of P has a counterpart point q where ℓ leaves the interior of P with the open segment pq completely contained in the interior of P. Therefore, since all entry and exit points caused by the intersection of ℓ with P come in pairs, there is an even total number of intersections of ℓ and P. □

We will treat the special cases of lines containing one or two vertices of P separately at the end of this chapter.

Notation 4. *We will denote the number of edge intersections between ℓ and P by k, and the number of pieces that ℓ cuts P into, by K.*

Claim 5. *A straight line which intersects k edges of a polygon P and does not pass through a vertex of P, cuts P into $K = 1 + k/2$ pieces.*

Proof. Each intersection forms a vertex of exactly two pieces of P. The segments defined by two adjacent intersection points form chords which if applied sequentially *cut off* one piece of the polygon at a time. If we count the number $k/2$ of chords, we count the number of sub-polygons we chop off, until all chords have been applied and there is only one piece left. Thus there are $1 + k/2$ pieces. □

Corollary 6. *To find a solution to the problem of cutting the polygon into K pieces we need to find all lines that intersect exactly $2(K - 1)$ edges of P and then classify the lines that pass through vertices of P.*

Definition 7. *Two lines ℓ_1 and ℓ_2 are combinatorially equivalent if they induce the same partitioning of the vertices of P. Otherwise, they are combinatorially distinct.*

This definition says that all lines which intersect exactly the same edges of P are combinatorially equivalent. The size of this set of lines is obviously quadratic, even when we are dealing with a convex polygon. In terms of enumerating this set, we can reduce its size, as in the case of a convex polygon when this set would have just one member. (Because all lines cut a convex polygon in exactly two pieces we can describe all such lines as one element).

Therefore to prove a lower bound for our problem we are going to use a slightly different definition of combinatorial equivalence, which we will call strict.

Definition 8. *Two lines ℓ_1 and ℓ_2 are strictly combinatorially equivalent if there exists a continuous transformation from ℓ_1 to ℓ_2 such that at any moment during the transformation, the polygon is cut into the same number of pieces. Otherwise, they are strictly combinatorially distinct.*

The meaning of this definition implies that given a straight line ℓ_1 that cuts P into a specific number of pieces, we can move this line to ℓ_2 while still cutting P into the same number of pieces, hence obtaining new lines within the same *strict combinatorially equivalent* group of lines.

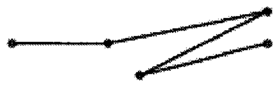
As soon as we move our line to the point that the number of pieces of P that are cut changes, then we enter another group of *strictly combinatorially distinct* lines.

We are going to use the strict definition of combinatorial equivalence when proving a lower bound of the size of the set of separator lines. This will show the size of this set is quadratic under strict equivalence, meaning that the algorithm we are going to develop is worst case optimal in all cases.

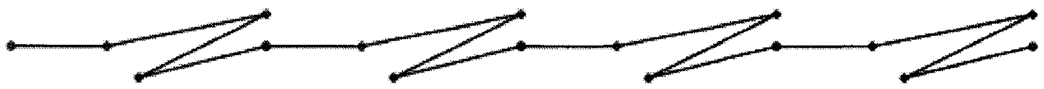
The set of *strict combinatorially equivalent* group of lines is a subset of the set of *combinatorially distinct* set of lines. For the lower bound argument we are going to be talking about the set of *strict combinatorially equivalent* lines whereas throughout the rest of this thesis we will refer to each *combinatorially distinct* set of lines as *separator lines.* In the context of the K-separator problem, we will refer to such sets of lines as K-separator *lines* for any $K > 1$.

Claim 9. *There exists a simple n-vertex polygon P for which the number of* strictly combinatorially distinct *2-separator lines is $\Omega(n^2)$.*

Proof. Define the following zig-zag polyline:



Take m copies of this polyline and join them together in a chain as shown below for $m = 4$:



We create two copies of this chain on two horizontal lines and connect the two leftmost vertices by a vertical edge. After adding two horizontal segments at the right end of each chain we connect the two rightmost vertices by a vertical edge.

This defines P as shown in Figure 2.2 with n vertices where $n = 2(4m+1)+2 = 8m+4$.

We will show that this polygon P has a quadratic number of edge pairs that have unique *strictly combinatorially distinct* 2-separator *lines*.

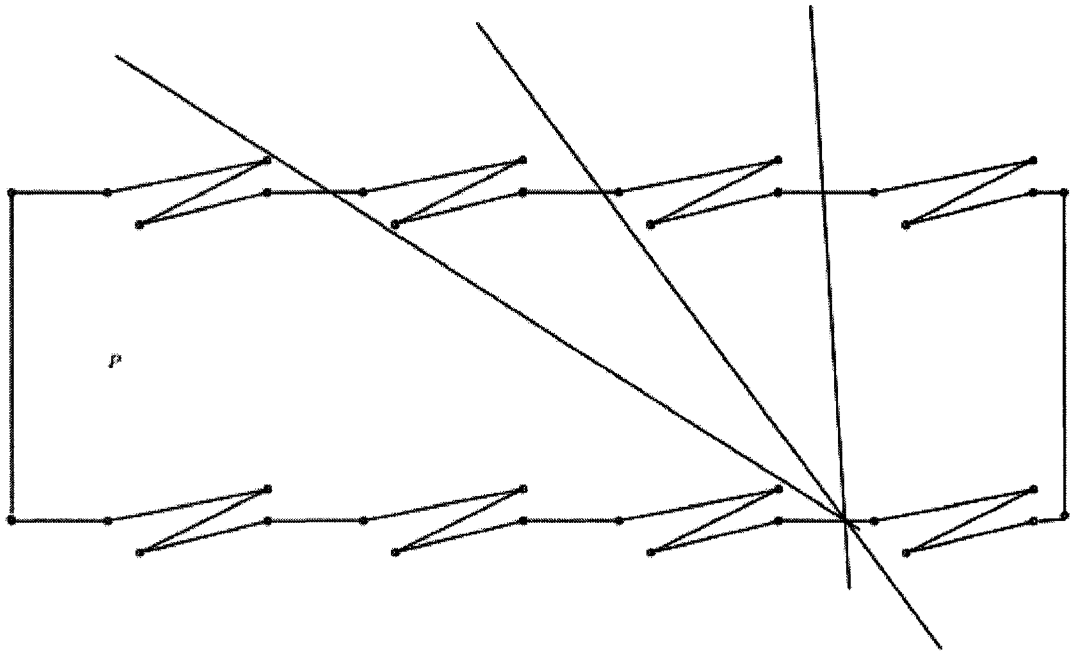


Figure 2.2: Special Polygon Demonstrating Lower Bound

Any horizontal edge on the bottom polygonal line of P (there are $(n+4)/8$ such edges) can be paired with any horizontal edge on the top polygonal line of P. A line through these two edges is a unique 2-separator line (if we move this line it will intersect the neighboring 3-edge zig-zag polygonal line hence become a 4-separator line).

Hence the number of *strictly combinatorially distinct* 2-separator *lines* is at least $((n+4)/8)^2$, which is $\Omega(n^2)$.

$\square$

In order to enumerate all groups of *combinatorially distinct* lines that intersect exactly k edges of the polygon for an even number $k > 0$, we use duality [6]. Duality transformations are a useful tool in Computational Geometry. Our geometric transform T maps a point p in the plane to a non-vertical line and a non-vertical line back to a point:

$$T : p = (a, b) \rightarrow T(p) : y = ax + b$$

$$\ell : y = kx + d \rightarrow T(\ell) = (-k, d)$$

Fact 10. *[21] T retains the relative positions of points and lines:*

p lies on ℓ if and only if $T(\ell)$ lies on $T(p)$,

p lies below ℓ if and only if $T(\ell)$ lies above $T(p)$,

p lies above ℓ if and only if $T(\ell)$ lies below $T(p)$.

Definition 11. *The dual of an edge e of P is called the double-wedge of e and is denoted by $dw(e)$. It is the union of the dual lines of all points on e. See Figure 2.3.*

Fact 12. *Since all points on e are collinear, the dual lines of all points on e pass through the same point which is the dual point of the line containing e.*

Definition 13. *The dual point of the line containing e, is called the <u>double-wedge vertex</u> of e.*

Remark 14. *Let $e = (p, q)$ where p and q are the vertices of e. The double-wedge of e is the collection of all points in the dual which either lie*

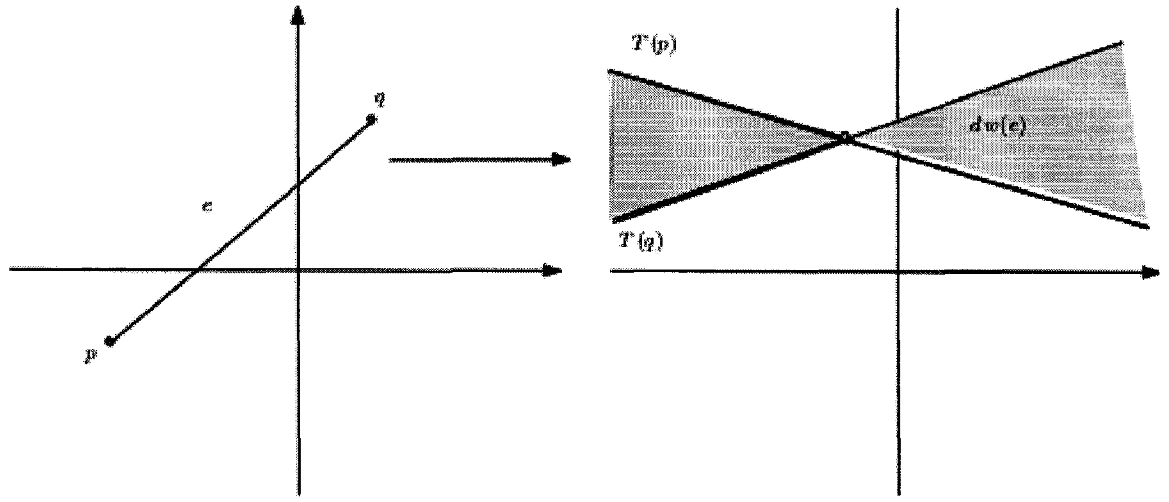


Figure 2.3: Double-Wedge

above $T(p)$ and below $T(q)$

or

below $T(p)$ and above $T(q)$.

Lemma 15. *Let e be an edge of P and let L_e be the set of all lines that intersect e. Then $T(L_e) = dw(e)$.*

Proof. Let ℓ be a line that intersects $e = (p, q)$. Then ℓ is between p and q and thus $T(\ell)$ is between $T(p)$ and $T(q)$. Therefore $T(\ell) \in dw(e)$.

Similarly, let x be a point in $dw(e)$. Then x is between $T(p)$ and $T(q)$. Then $T(x)$ is between p and q, hence the line $T(x)$ intersects (p, q). Therefore $T(x) \in T(L_e)$.

□

The set of double-wedges of all edges of P define an arrangement of lines which we denote as $\mathcal{A}(\mathcal{P})$.

This arrangement consists of cells, edges and vertices. The cells are open (meaning the boundary does not belong to the cell) and the edges are open (the endpoints do not belong to an edge). Therefore, each point in the plane is either in a cell, or on an edge, or is a vertex.

2.2 Algorithm

We obtain a solution for Problem 1 by computing the value of K for every cell, edge, and vertex of $\mathcal{A}(\mathcal{P})$.

As a first step we compute the arrangement of the duals of all edges of P (by computing the dual lines to all vertices of P).

We can compute the complete arrangement of the duals of the polygon vertices in $O(n^2)$ time using the algorithm described in the textbook [21]. This gives us the cells, edges and vertices of $\mathcal{A}(\mathcal{P})$ which we want to classify in order to solve Problem 1. The arrangement of n lines in the plane consists of $O(n^2)$ edges, vertices and cells [21].

Remark 16. *A cell in $\mathcal{A}(\mathcal{P})$ defines the duals of a set of* combinatorially equivalent *separator lines.*

Remark 17. *There are two special cases we must consider:*

1. *A vertex in $\mathcal{A}(\mathcal{P})$ defines a separator line which contains exactly two vertices of P (because it is the intersection of the duals of two vertices of P).*

2. *The points on an edge in $\mathcal{A}(\mathcal{P})$ define a set of separator lines that contain exactly one vertex of P (because the edge is part of the dual of a vertex of P).*

We will treat these special cases separately at the end of this chapter.

Definition 18. *A cell in $\mathcal{A}(\mathcal{P})$ is said to have depth d, if it is contained in exactly d double-wedges.*

Remark 19. *The cells of the arrangement of the double-wedges of the edges of P, having depth k, define the duals of all lines that intersect exactly k edges of P.*

This means that to solve our problem we first need to enumerate all cells that map to *combinatorially equivalent* K-separator lines for all values of K by computing the parameter k which is the depth of a cell. Henceforth we will derive K using the formula $K = 1 + k/2$. Then we will treat the special case of determining the parameter K for the edges and vertices of $\mathcal{A}(\mathcal{P})$.

$\mathcal{A}(\mathcal{P})$ has an upper envelope (the cell whose points are above all the lines of the arrangement) and a lower envelope (the cell whose points are below all the lines of the arrangement).

Claim 20. *Any simple polygon P has a non-empty set of* 2*-separator lines.*

Proof. This is evident by picking a cell in $\mathcal{A}(\mathcal{P})$ that is a neighbor of the upper (or lower) envelope cell. A point x in this cell is above all lines in the arrangement but one (the one line we had to cross below the envelope). Hence x is in exactly two double wedges, which means that the line $T(x)$ intersects exactly two edges of P. Using our formula for computing the number of pieces based on the number of intersections $K = 1 + k/2$, where K is the number of pieces and k is the number of edge intersections in P, we get $K = 1 + 2/1 = 2$. Therefore, $T(x)$ is a 2-separator line. $\square$

Definition 21. *The dual graph of $\mathcal{A}(P)$ denoted by D, is the graph whose vertices are the cells of $\mathcal{A}(P)$ and each edge of D joins two cells of $\mathcal{A}(P)$ which share an edge.*

To classify all cells according to their depth we will perform a type of depth-first search [27] in the graph D defined by $\mathcal{A}(P)$ starting at the upper envelope (with *depth* $= 0$, i.e, defining lines strictly above the polygon) and ending at the lower envelope (with *depth* $= 0$, strictly below).

Observation 22. *Let v be a vertex of P and let u and w be the two neighbors of v. The line $T(v)$ is divided into three sections by the two $double - wedge$ vertices of the edges (u, v) and (v, w). In the example shown in Figure 2.4, the line $T(v)$ has two sections on either side of $T(w)$ and $T(u)$, and one section between $T(w)$ and $T(u)$.*

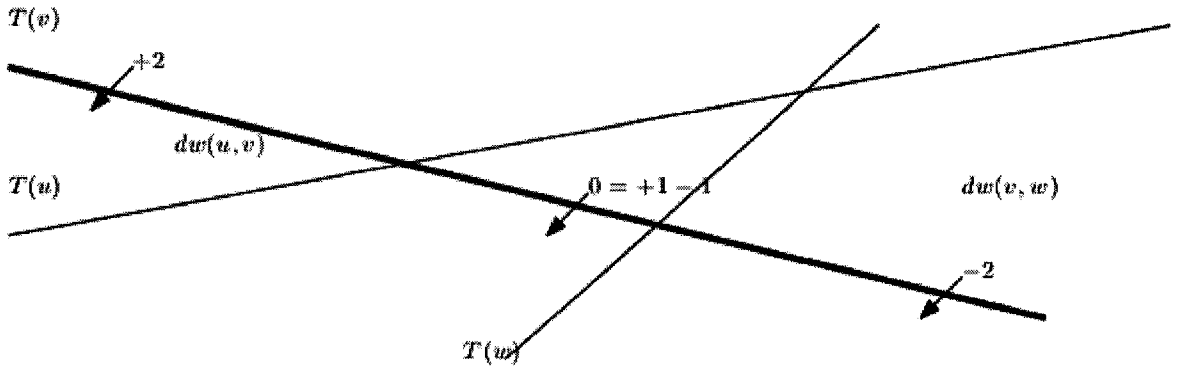


Figure 2.4: An arrangement of two double-wedges

When we walk along an edge in the dual graph D of $\mathcal{A}(P)$ we cross an edge of $\mathcal{A}(P)$. Since no edge of $\mathcal{A}(P)$ is vertical, we can talk about the relative positioning of the cells of $\mathcal{A}(P)$.

Definition 23. *Any two cells of $\mathcal{A}(P)$ that share an edge, can be classified as one being <u>above</u> and the other <u>below</u>. When we <u>cross</u> an edge of $\mathcal{A}(P)$ we walk along an edge in D, starting at the vertex corresponding to the cell that is <u>above</u>, and ending at the vertex of D corresponding to the cell of $\mathcal{A}(P)$ that is <u>below</u>.*

Consider the arrangement of just two double wedges as shown in Figure 2.4. There are two double-wedge vertices on the line $T(v)$. What happens when we cross $T(v)$?

Lemma 24. *If we cross an edge of $\mathcal{A}(P)$ which is in either of the three sections of $T(v)$, then we either enter 2 double-wedges, or leave 2 double wedges, or enter a double wedge and leave another.*

Proof. As illustrated in Figure 2.4, the line $T(v)$ is part of exactly two double-wedges of the edges (u, v) and (v, w).

There are exactly two double-wedge vertices on this line because there are exactly two adjacent polygon vertices, next to every vertex of P. These double-wedge vertices partition the line into three sections. Therefore, if we cross the line at any of these three sections, we enter or leave both of these double wedges, or we enter one of the double wedges and leave the other.

In the example shown in Figure 2.4, the upper section is above both lines $T(w)$ and $T(u)$. If we cross below this section of the line we enter two double-wedges. The middle section of $T(v)$ (which is between $T(w)$ and $T(u)$) is above one line and below the other, hence we enter a double-wedge and leave another. The third section is below both lines hence if we move below that section we leave two double-wedges.

□

This fact will come in handy in the second part of the algorithm which deals with classifying the cells of $\mathcal{A}(P)$ according to their depth. To reiterate, as seen before due to the fact that $\mathcal{A}(P)$ does not contain vertical lines, for any two neighboring cells that share an edge, we can classify one of them as the above cell, and the other as the below cell.

Definition 25. *Each edge e of $\mathcal{A}(P)$ will be associated with a pointer denoted by $flag(e)$. The value of $flag(e)$ is an integer taking one of three values $(+2, 0, -2)$ indicating the number by which the depth of the cell below changes compared to the cell above, when we cross the edge that separates the two cells.*

Let u, v, w be three consecutive vertices of P. For a given edge e of $\mathcal{A}(P)$ which lies on $T(v)$, the value of $flag(e)$ depends on the section of $T(v)$ that e is part of. The value of the flags associated with a given section of $T(v)$ will depend on the slopes of $T(v)$, $T(u)$ and $T(w)$.

Next we will show how we will compute flags associated with the three sections for any line of $\mathcal{A}(P)$.

Notation 26. *We will denote the* slope *of a line ℓ by $SL(\ell)$. The* double-wedge *vertex on $T(v)$ with the smallest x-coordinate is called xmin. The* double-wedge vertex *with the largest x-coordinate on $T(v)$ is called xmax.*

There are three cases for dividing the line $T(v)$ in three sections depending on the slopes of the three lines in question as shown in Figure 2.5. Let u, v, and w be three consecutive vertices of P.

If $SL(T(u)) > SL(T(v))$, and $SL(T(w)) > SL(T(v))$ then: $T(v)$ is divided into three different sections. The <u>upper</u> section $(\infty, xmin)$, the <u>middle</u> section $(xmin, xmax)$ and the <u>lower</u> section $(xmax, \infty)$.

If $[SL(T(u)) > SL(T(v))$, and $SL(T(w)) < SL(T(v))]$ or $[SL(T(u)) < SL(T(v))$, and $SL(T(w)) > SL(T(v))]$ then: $T(v)$ is divided into two different sections. The <u>upper</u> section $(xmin, xmax)$, and two <u>middle</u> sections $(\infty, xmin)$ and $(xmax, \infty)$.

If $SL(T(u)) < SL(T(v))$, and $SL(T(w)) < SL(T(v))$ then: $T(v)$ is divided into three different sections. The <u>lower</u> section $(\infty, xmin)$, the <u>middle</u> section $(xmin, xmax)$ and the <u>upper</u> section $(xmax, \infty)$.

Bearing this in mind we can execute the following simple algorithm for pre-processing $\mathcal{A}(P)$ and associating each edge of $\mathcal{A}(P)$ with a flag.

Algorithm ComputeFlags: For every three adjacent vertices u,v and w of P (v is between u and w) do the following:

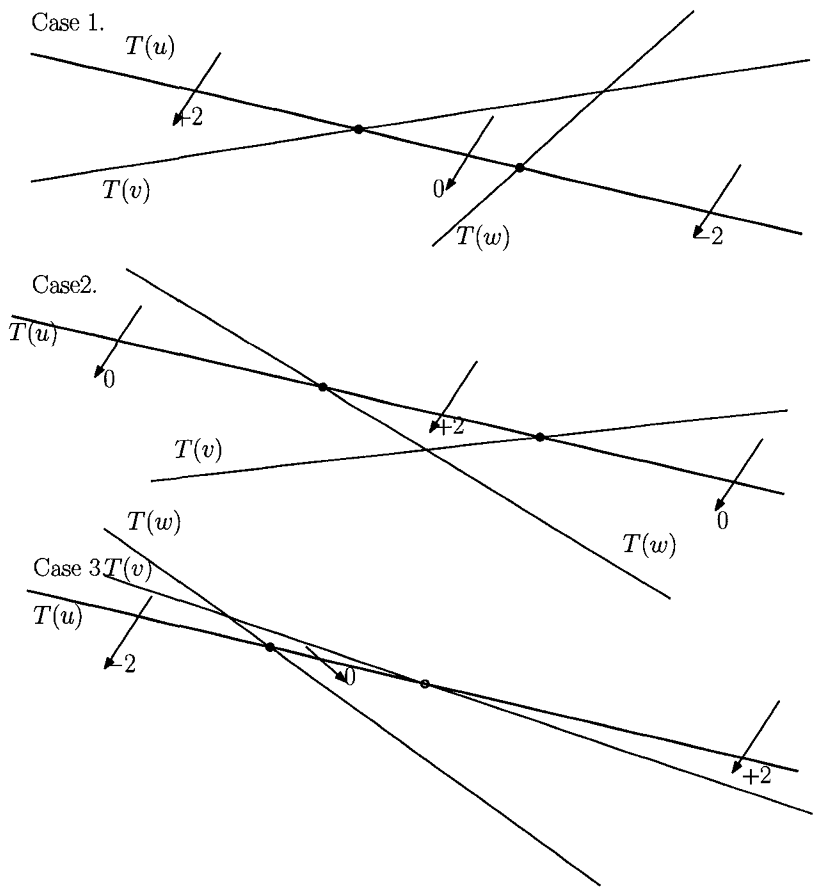


Figure 2.5: Three cases for line crossings

1. Compute the intersections of $T(v)$ with $T(u)$ and $T(w)$.

2. Compute the slopes of $T(v)$, $T(u)$ and $T(w)$.

3. Compare the slopes and determine the upper, middle and lower sections of $T(v)$.

4. Assign a flag with value +2 to all edges of $\mathcal{A}(P)$ which lie on the upper section of $T(v)$.

5. Assign a flag with value 0 to all edges of $\mathcal{A}(P)$ which lie on the middle section of $T(v)$.

6. Assign a flag with value -2 to all edges of $\mathcal{A}(P)$ which lie on the lower section of $T(v)$.

Clearly we can compute the sections of all lines of $\mathcal{A}(P)$, and as a result the flags on every edge of $\mathcal{A}(P)$, in time that is linear in the complexity of $\mathcal{A}(P)$ by going through every line and comparing its slope to the slopes of its two neighbors and then assigning the appropriate flag to every edge on the sections. Thus, the running time of algorithm **ComputeFlags** is $O(n^2)$.

Now we will use the dual graph of $\mathcal{A}(P)$ and the flags we have computed to compute the depths of all cells of $\mathcal{A}(P)$ using depth-first search. The dual graph D (as illustrated in Figure 2.6) in the $\mathcal{A}(P)$ is comprised of the following:

The vertex set of D is the set of all cells in $\mathcal{A}(P)$.

The edge set is the set of all pairs of cells that share an edge, with a direction value from the above cell to the below cell.

In addition we will associate each edge of D with the a flag corresponding to the edge of $\mathcal{A}(P)$ that it crosses. As seen previously the flag is an integer having one of three possible values $(+2, 0, -2)$.

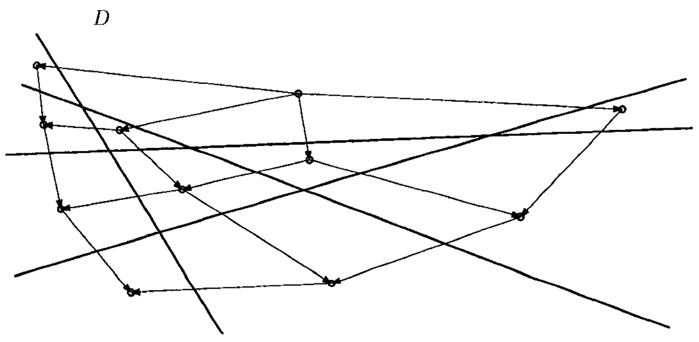


Figure 2.6: The Dual Graph D of the arrangement $\mathcal{A}(P)$

The following algorithm traverses the graph D using depth-first search [27]. It computes the depth of each cell of the arrangement $\mathcal{A}(P)$.

Algorithm-dfs-compute-depth:

Input: Graph D as defined above

Output: The depth of each cell of $\mathcal{A}(P)$

```
1. Array $S$ = empty ($S[d]$ will contain all cells of the same depth $d$)
2. list L = empty
3. $x$ = vertex of $D$ representing the cell of the upper envelope
4. initialize $depth(x) = 0$
5. for each edge $(x, w)$ add edge $(x, w)$ to the end of L
6. while L is not empty do:
        remove edge (v, w) from the end of L
        if w not yet visited then:
         depth(w) = depth(v) + flag(edge(v,w))
          add w to S[ depth(w) ];
          for each edge (w,z) add edge (w,z) to the end of L
          set w to visited
7. return S
```

Claim 27. *This algorithm correctly computes the depth of each cell of the arrangement* $\mathcal{A}(P)$.

Proof. The depth-first-search process visits every cell exactly once (each cell of the arrangement $\mathcal{A}(P)$ is associated with one vertex of D). The depth of a cell differs from the depth of the preceding cell by exactly the quantity of the graph edge flag, as shown above. We recursively use this information to correctly compute the depths of all the cells we visit as vertices of the graph D.

We start at the upper envelope cell of the arrangement $\mathcal{A}(P)$ whose depth is known beforehand (0), and it is clear by the design of the classical depth first search algorithm that we recursively visit the cells immediately below the cell of the upper envelope, and then the cells below it and so on. □

Claim 28. *The running time of this algorithm is $O(n^2)$. The output size for any depth is $O(n^2)$ in the worst case.*

Proof. This is true due to the time required to compute the arrangement and the time it takes to traverse all cells in the arrangement. Both these quantities are $O(n^2)$ as seen before in this chapter. In the worst case we have a quadratic quantity of cells of depth 2 as seen in Claim 8 of this chapter. □

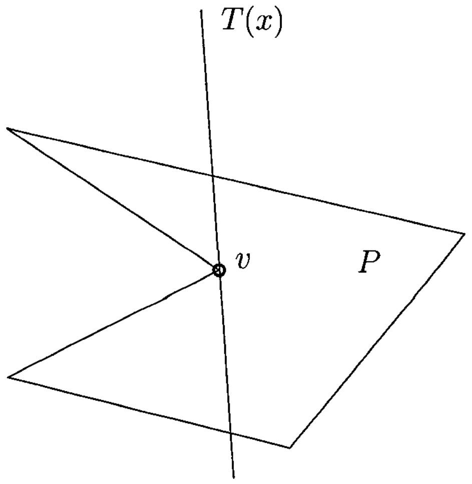
After we have classified each cell according to its depth, we can treat the special cases for separator lines that contain one or two vertices of P.

Claim 29. *The duals of all points on an edge of $\mathcal{A}(P)$ are lines through a single vertex of P.*

Proof. The duals of the vertices of P are the lines of $\mathcal{A}(P)$. Since each edge of $\mathcal{A}(P)$ does not contain its endpoints, and there are no vertices on the edge, then any point x on an edge of $\mathcal{A}(P)$ corresponds to a line $T(x)$ which passes through a single vertex of P. (If it passed through more than one vertex then x would be the intersection of the dual lines of two vertices of P.) □

Let e be an edge of $\mathcal{A}(P)$ which separates cells c_1 and c_2. Let $depth(c_1) \leq depth(c_2)$. We will show how to compute the K-separator value for all points x on e.

Case 1: The line $T(x)$ passes through a reflex vertex v of P as shown here:

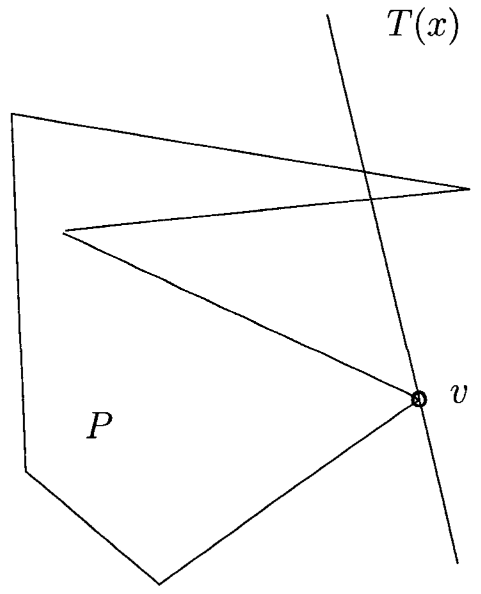


In the example shown in this picture $T(x)$ intersects 4 edges of P and cuts P into 3 pieces.

If we move $T(x)$ slightly toward the interior of P the number of pieces decreases by one and the point x moves in $\mathcal{A}(P)$ from the edge e into the cell c_1. Hence the point x moves into the cell with the smaller depth in $\mathcal{A}(P)$. If we move $T(x)$ in the opposite direction the number of pieces does not change and the point x moves into the cell c_2.

Therefore we set $k = \max(depth(c_1), depth(c_2))$.

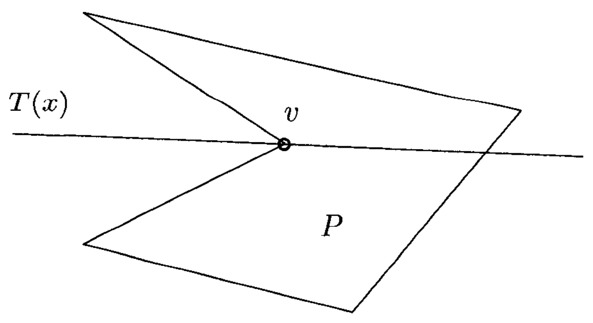
Case 2: The line $T(x)$ passes through a convex vertex v of P as shown below:



In the example shown in this picture $T(x)$ intersects 4 edges of P and cuts P into 2 pieces. If we move $T(x)$ slightly toward the interior of P the number of pieces increases by one and the point x moves in $\mathcal{A}(P)$ from the edge e into the cell c_2. Hence the point x moves into the cell with the larger depth in $\mathcal{A}(P)$. If we move $T(x)$ in the opposite direction the number of pieces does not change and the point x moves into the cell c_1.

Therefore we set $k = \min(depth(c_1), depth(c_2))$.

Case 3: The line $T(x)$ passes through a vertex v and one of the extensions of $T(x)$ enters the interior of P and the other extension does not enter the interior of P as shown below:



In the example shown in this picture the line $T(x)$ intersects 3 edges of P and cuts P into two pieces. This case corresponds in the dual to points on an edge shared by two cells of the same depth. If we move $T(x)$ in any direction away from the vertex v we move x into either c_1 or c_2 and the number of pieces that P is cut into does not change. Therefore we set $k = \mathit{depth}(c_1) = \mathit{depth}(c_2)$.

Assign the parameter $K = 1 + k/2$ to the edge e of $\mathcal{A}(P)$. Then K is the number of pieces of P that results by cutting P using $T(x)$, for any x on e.

The following simple algorithm traverses the arrangement, and using the already computed depth of the cells, computes the parameter K for the edges of $\mathcal{A}(P)$.

```
For every edge $e$ of $\mathcal{A}(P)$ do:
Let $c_1$ and $c_2$ be the two cells that share $e$.
Let $T(e)$ be the vertex of $P$ that corresponds to the edge $e$ of $\mathcal{A}(P)$.
if $\mathit{depth}(c_1) = \mathit{depth}(c_2)$ then assign $K = 1 + \mathit{depth}(c_1)/2$ to $e$.
else if $T(e)$ is reflex then assign $K = 1 + \max(\mathit{depth}(c_1), \mathit{depth}(c_2))/2$ to $e$.
else if $T(e)$ is not reflex then assign $K = 1 + \min(\mathit{depth}(c_1), \mathit{depth}(c_2))/2$ to $e$.
```

Claim 30. *This algorithm correctly classifies all edges of $\mathcal{A}(P)$ according to the parameter K of the K-separator in $O(n^2)$ time.*

Proof. Clearly since we traverse all edges of $\mathcal{A}(P)$ and there are $O(n^2)$ edges in $\mathcal{A}(P)$ the running time is $O(n^2)$. □

After we have computed the parameter K of all the edges of $\mathcal{A}(P)$, we will compute the parameter K for all the vertices of $\mathcal{A}(P)$.

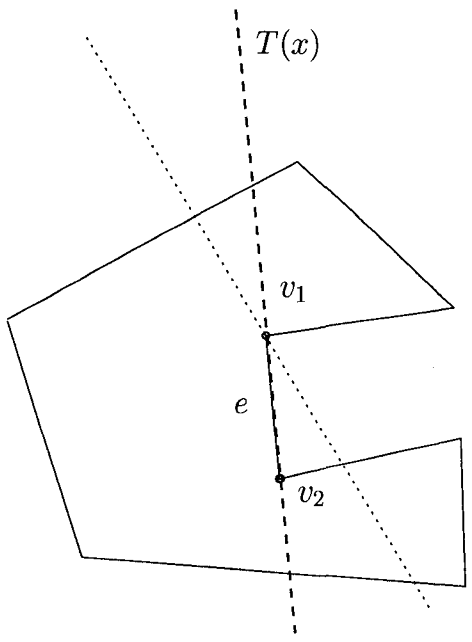
Observation 31. *A vertex in $\mathcal{A}(\mathcal{P})$ has degree 4, because no three vertices of P are on the same line.*

We are going to assign a parameter K to each vertex of $\mathcal{A}(\mathcal{P})$ based on the parameter K we assigned to each of the 4 edges that share the vertex.

There are two types of vertices in $\mathcal{A}(P)$, double-wedge vertices and the rest. The dual of every double-wedge vertex is a line that contains an edge of P. The dual of every non-double-wedge vertex is a line that contains two vertices of P, which are not on the same edge of P.

We deal with the double-wedge vertices first. Let x be a double-wedge vertex in $\mathcal{A}(P)$ and let e be the edge of P that is contained in $T(x)$. There are three cases for computing the parameter K for this separator line:

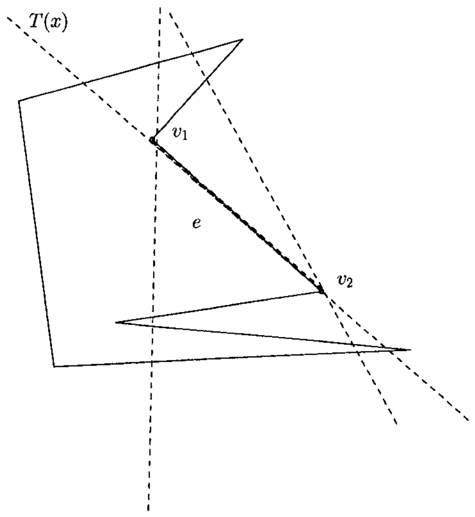
Case 1: The vertices of e are both reflex.



In this case the number of edges of P that are intersected by $T(x)$ is 5, including e, and $T(x)$ cuts P in 4 pieces.

If we rotate the line $T(x)$ around either vertex of e, we cut the same number of pieces of P. The parameter K for x is the same as the larger parameter K on all edges of $\mathcal{A}(P)$ that share x.

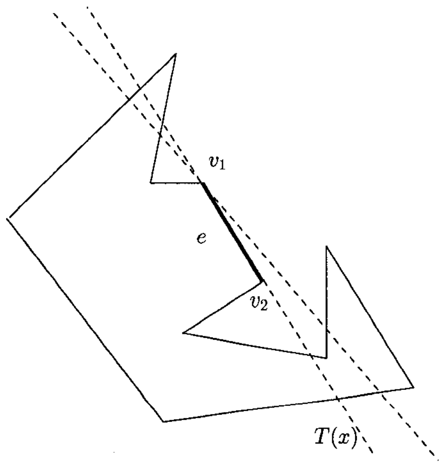
Case 2: One of the vertices of e is reflex and the other is non-reflex.



We can rotate $T(x)$ around either vertex of e. If we rotate it around the reflex vertex we get one more piece of P.

In this case the parameter K of the vertex x is equal to the parameter K of the edge of $\mathcal{A}(P)$ which lies on the line whose dual corresponds to the reflex vertex of e in P.

Case 3: Both vertices of e are convex.

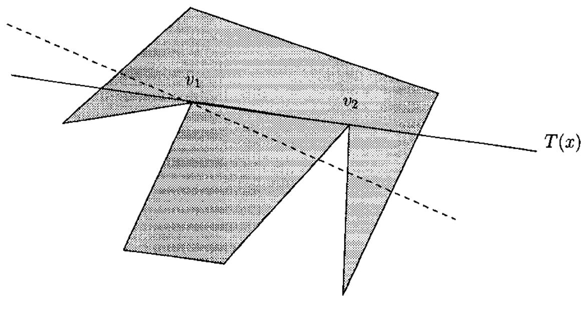


In this case the parameter K of x is the smaller of the parameter K's of all edges of $\mathcal{A}(P)$ that share x.

Now, let x be a non-double-wedge vertex in $\mathcal{A}(P)$. Then $T(x)$ contains exactly two vertices v_1 and v_2 of P, and (v_1, v_2) is not an edge of P.

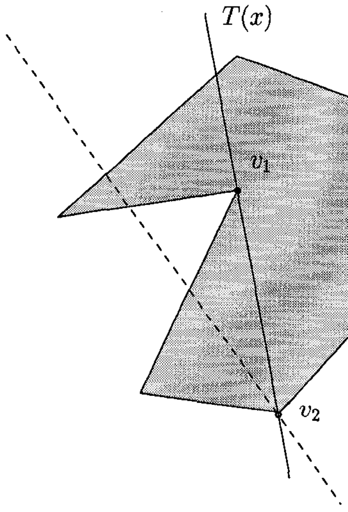
Similar to the previous case of the double-wedge vertices there are three cases for this separator line:

Case 1: The vertices of v_1 and v_2 are both reflex.



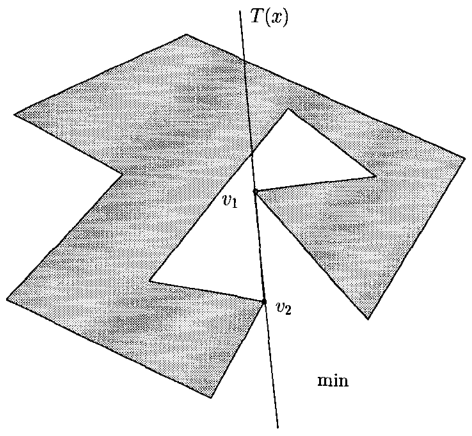
In this case the parameter K of x is equal to the larger parameter K of all edges in $\mathcal{A}(P)$ that share x.

Case 2: Vertex v_1 is reflex and v_2 is convex; or vice versa.



In this case the parameter K of x is equal to the larger parameter K of the edges in $\mathcal{A}(P)$ which lie on the line whose dual is the convex vertex (in this case v_2).

Case 3: Both vertices v_1 and v_2 are convex.



In this case the parameter K of x is the smallest of the parameter K's of all edges in $\mathcal{A}(P)$ that share x.

The following algorithm classifies all vertices of $\mathcal{A}(\mathcal{P})$ according to the parameter K that the separator line dual to the vertex has.

For every vertex x of $\mathcal{A}(\mathcal{P})$ do:

We are going to denote the parameter K of x by $K(x)$. The vertex x is shared by 4 edges of $\mathcal{A}(\mathcal{P})$ which we will call e_1, e_2, e_3, e_4. For $1 \leq i \leq 4$, let $K(e_i)$ be the parameter K assigned to e_i.

If the duals of both lines of $\mathcal{A}(\mathcal{P})$ intersecting at x are reflex vertices in P then $K(x) = \max(K(e_1), K(e_2), K(e_3), K(e_4))$.

If one of the duals of the lines of $\mathcal{A}(\mathcal{P})$ intersecting at x is a reflex vertex in P and the other is not, then let e_1, e_2 be the edges of $\mathcal{A}(\mathcal{P})$ corresponding to the reflex vertex. We set $K(x) = \max(K(e_1), K(e_2))$.

If the duals of both lines of $\mathcal{A}(\mathcal{P})$ intersecting at x are convex vertices in P then $K(x) = \min(K(e_1), K(e_2), K(e_3), K(e_4))$.

This concludes the calculation of the parameter K for the K-separators of the vertices and edges of $\mathcal{A}(P)$.

Theorem 32. *Given a polygon P with n vertices, in $O(n^2)$ time we can compute a partition of the set of all lines into groups and the depth of each group, such that all lines in the same group are combinatorially equivalent.*

Chapter 3

Finding K-separator lines crossing a fixed edge

In this chapter we will analyze the complexity of the problem we are trying to solve using techniques from Gajentaan and Overmars [13] to prove that a simple decision problem defined as a sub-problem of the general line-set enumeration problem is 3SUM-hard for $K > 2$.

3.1 Discussion

The method we employed in the previous chapter is *brute-force* to achieve quadratic running time. Can we do better?

We are going to state an existence question and prove that answering this question most likely requires quadratic time. The existence question will be formulated as follows:

This is a reformulation of Problem 2 on page 4, by replacing the parameter K which signifies the number of pieces of the polygon that are cut by a line, with the parameter k which signifies the number of edges of P that are intersected by a line that cuts P in K pieces.

Problem 33. *Given a simple polygon P, an edge e of P and an even integer k, is there a line intersecting the edge e, which intersects exactly k edges of P?*

This problem is basically only looking at cells that are in the double wedge of e, hence it is a subset of the problem which we solved in time $O(n^2)$ in the previous chapter.

Notation 34. *Throughout the rest of this chapter we are going to refer to the edge e we picked for the existence question, as the* designated *edge.*

A classical problem in computational complexity is 3SUM.

Problem 35. *3SUM: Given a set of n integers, determine if there exist three elements of the set that add up to zero.*

The best known algorithms for the 3SUM problem take quadratic time, and it has been widely accepted that no faster solutions to the 3SUM problem are possible. [13]

Definition 36. *3SUM-hard: If a problem can be reduced in $o(n^2)$ time to the 3SUM base problem, then it is at least as difficult to solve as 3SUM, and it is called 3SUM-hard.*

If we can show a sub-quadratic reduction from a known 3SUM-hard problem to our polygon cutting problem, then we can claim that solving our problem in sub-quadratic time would also mean that it is possible to solve 3SUM in sub-quadratic time.

This should be sufficient to claim based on what we know on the class of 3SUM hard problems, that it is probably not possible to solve our polygon cutting problem in sub-quadratic time.

Throughout the rest of this chapter we will be excluding the special cases of lines that pass through a vertex of P.

Theorem 37. *The problem of determining if there exists a line through a designated edge of the polygon which intersects exactly k edges of P, hence cuts P in $K = 1+k/2$ pieces is 3SUM-hard for any $K > 2$.*

The rest of this section deals with proving this theorem by showing a reduction from a 3SUM-hard problem.

This proof will be divided into two parts.

1. The first part of the proof will show that the 3-separator problem is 3SUM-hard.
2. In the second part of the proof we will generalize the result in Part 1, by showing that for any constant $K > 3$ the K-separator problem is 3SUM-hard.

This will suffice to show that the problem of finding a line through an edge of a particular polygon P with the property that it intersects exactly a fixed number of other edges of P, is at least as hard as the 3SUM problem.

3.2 3-separator problem complexity

For $K = 3$ in the K-separator problem we must have $k = 4$ where k is the number of edges of P that we intersect, since $K = 1 + k/2$.

Theorem 38. *Given a polygon P and an edge e of P, it is 3SUM-hard to decide if there is a line intersecting e which intersects exactly 3 other edges of P.*

The rest of this section deals with proving this theorem.

Definition 39. *[13] Given a set S of n objects in the plane, we call a line ℓ a separator of S if ℓ does not intersect any object in S and both half planes bounded by ℓ contain a non-empty subset of the objects in S.*

Problem 40. *[13] SEPARATOR2: Given a set S of n closed, non-intersecting (nor touching), horizontal line segments on 3 lines $y = 0$, $y = 1$, $y = 2$ with the left-most and right-most segments extended and bounded by a left vertical line and a right vertical line, is there a separator?*

The problem SEPARATOR2 (as illustrated in Figure 3.1) was shown by Gajentaan and Overmars [13] to be equivalent to the base 3SUM problem.

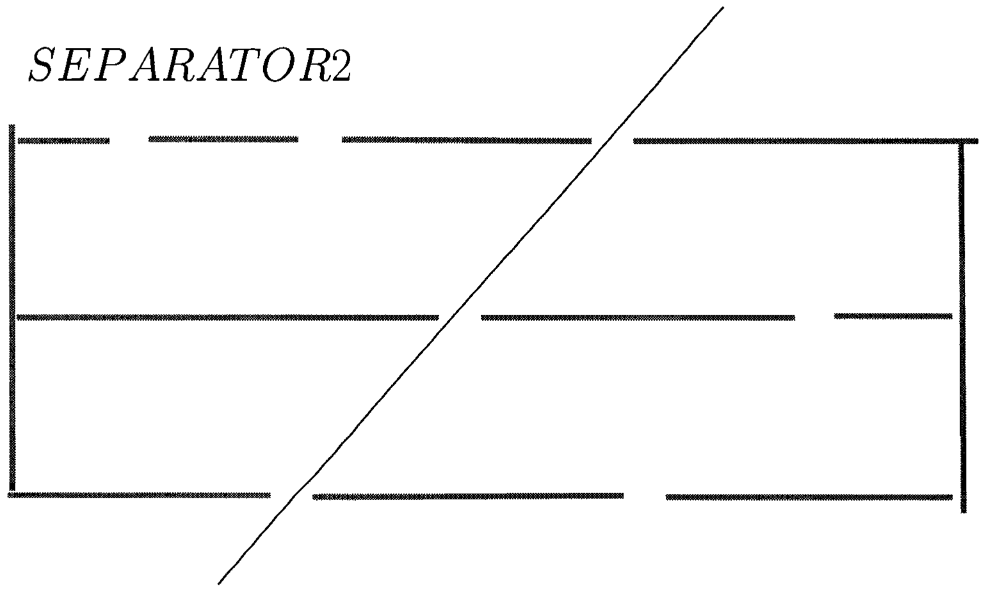


Figure 3.1: Separator2

As seen in Figure 3.1 there are three sets of horizontal line segments, bounded on either end by two (non-symmetrical) vertical segments. These vertical segments do not touch any of the horizontal segments. It was shown in [13] that no separator can run through the holes at the far left or far right in between the vertical bounding lines and the leftmost/rightmost segment.

We will show how to reduce the SEPARATOR2 problem in time $O(n(\log(n)))$ to the problem of a finding a line that intersects a given edge and exactly three other edges of a polygon P.

We are going to construct a polygon P and an edge e of P, with the property that *a line ℓ is a solution of SEPARATOR2, if and only if ℓ intersects e and exactly three other edges of P.*

3.3 Polygon Construction

Consider an input to SEPARATOR2.

To construct our polygon P we are going to build three polygonal chains on top of each of the three horizontal lines in the SEPARATOR2 input. Then we are going to construct a *special* edge of P and complete the polygon construction by joining the three polygonal chains and the special edge as well as use two vertical polygonal chains for each of the 2 vertical segments of the input. The construction will follow the following steps:

1. Sort the x-coordinates of the segment vertices on each line from the smallest to the largest

   Let $a_1, a_2 \ldots, a_m$; $b_1, b_2 \ldots, b_{m'}$; $c_1, c_2 \ldots, c_{m''}$ be the sorted sequences of the vertices where

(a) a_1 is the segment vertex on the top line with the smallest x-coordinate, a_m is the vertex on the top line with the largest x-coordinate,

(b) b_1 is the segment vertex on the middle line with the smallest x-coordinate, $b_{m'}$ is the vertex on the middle line with the largest x-coordinate,

(c) c_1 is the segment vertex on the bottom line with the smallest x-coordinate, $c_{m''}$ is the vertex on the bottom line with the largest x-coordinate.

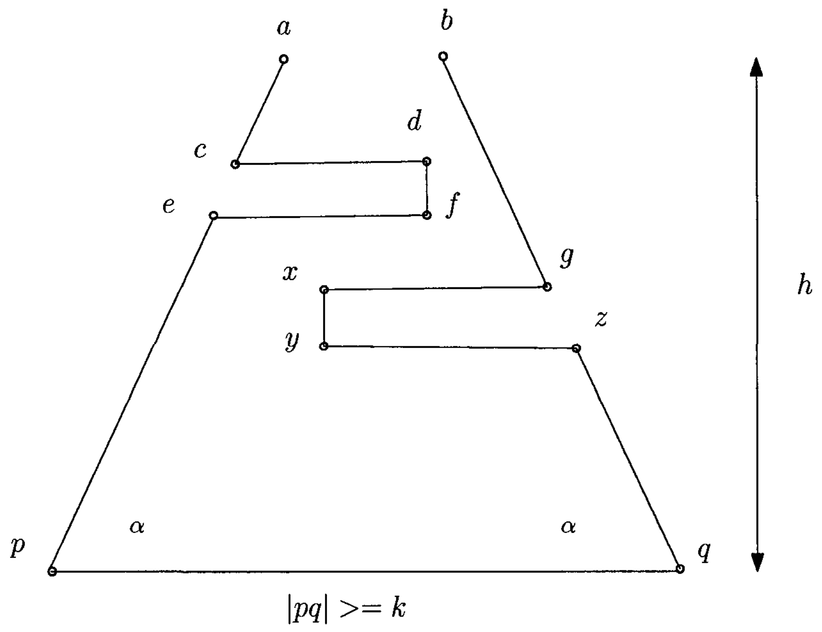
2. Let α_1 be the smaller angle between the x-axis and the line though $c_{m''}$ and a_1, and let α_2 be the smaller angle between the x-axis and the line though a_m and c_1. Let $\alpha = \min(\alpha_1, \alpha_2)$.

   The value of α ensures that there is no line in the solution of SEPARATOR2 which has slope less than α in absolute value.

3. Let k be the length of the shortest segment on all three horizontal lines.

4. Let $h = \min(1/4, (k/4)\tan(\alpha))$.

Definition 41. *For every horizontal line segment (p, q), define the following* gadget *polygonal line:*



The height h ensures that the two polygonal lines $a..p$ and $b..q$ do not intersect and a is to the left of b.

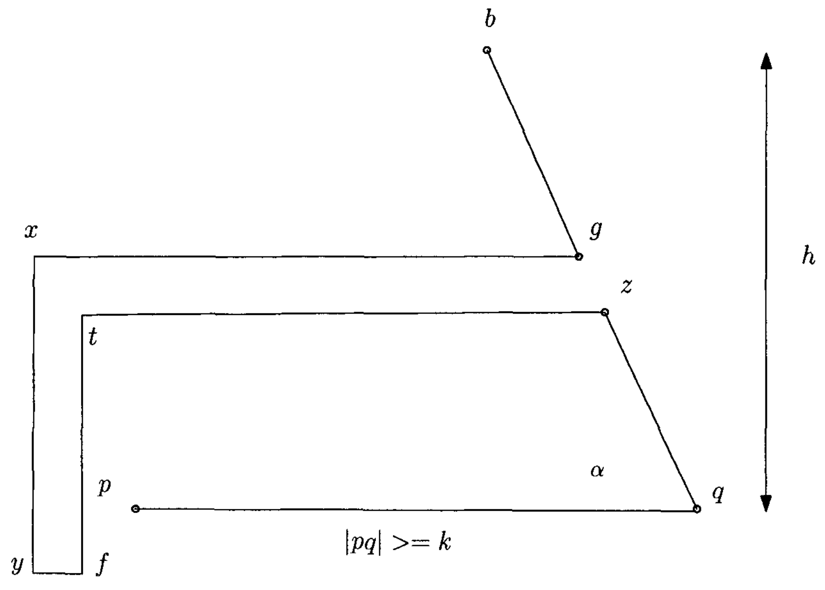
In this gadget, a is to the left of b; (p, e) and (c, a) are on the same line; (q, z) and (g, b) are on the same line; the angle between (p, q) and (p, e) is α in absolute value; the angle between (p, q) and (q, z) is α in absolute value; the vertical distance from y to pq is $h/2$; the vertical distance from e to pq is $2h/3$; the distance between d and f and between x and y are $|d, f| = |x, y| = h/16$; This ensures that the polygonal lines do not collide.

The polygonal line is composed of horizontal and vertical line segments, except for the segments having slope α in absolute value.

This polygonal line will be constructed on top of every line segment which has a neighboring line segment on either side. The same construction will be made on all three horizontal chains of segments.

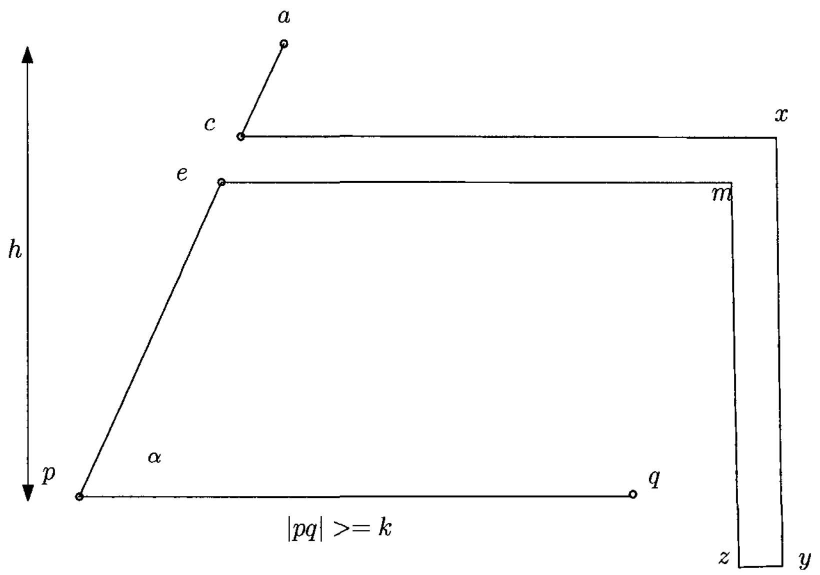
It is left to show the construction we are going to make on the leftmost and rightmost line segments on every horizontal chain.

Definition 42. *For the left-most line segment (p, q) on the top horizontal segment chain, define the following* top-left-gadget *polygonal line:*



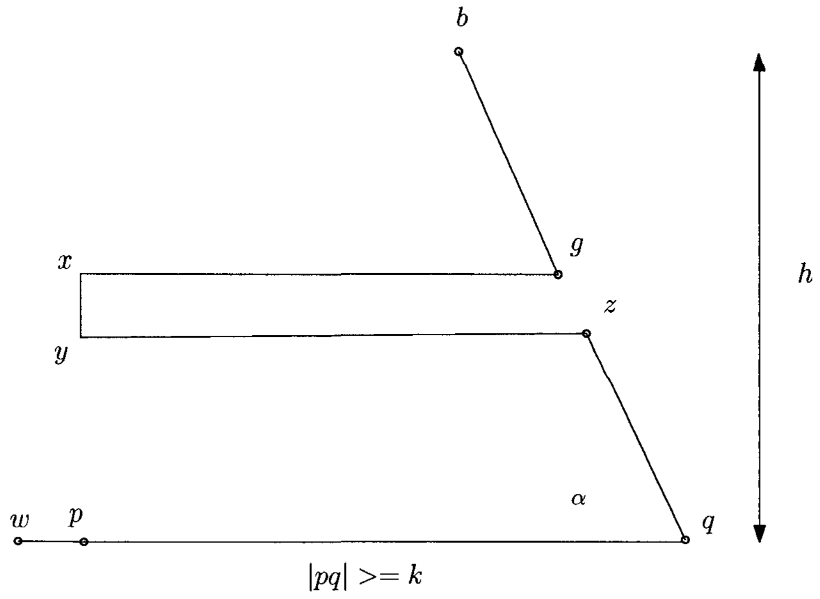
The dimensions of this polygonal chain segments are the same as in the main gadget. $|x, y| = |p, q| + h$ and $|x, g| = h$

Definition 43. *For the right-most top line segment define the following* top-right-gadget *polygonal line:*



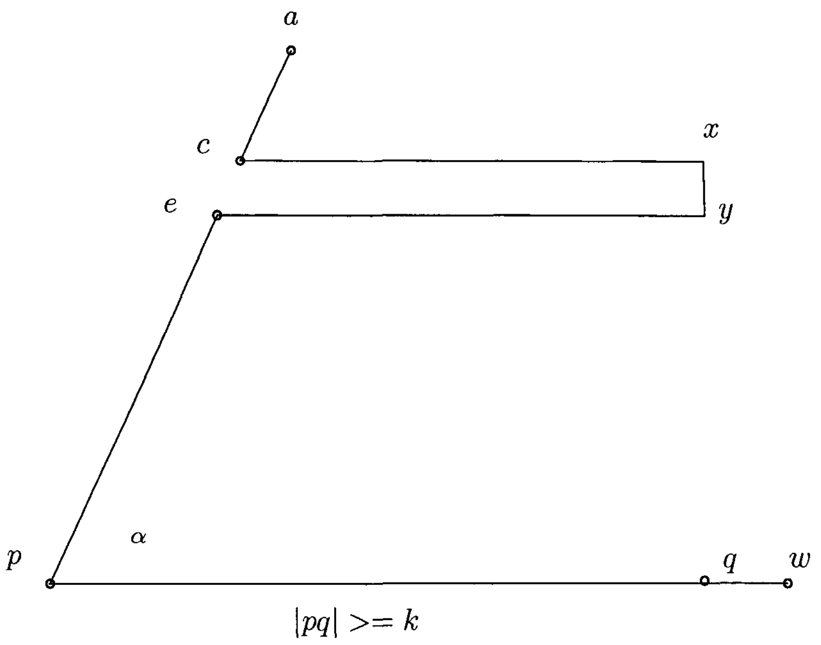
The dimensions of the top-right-gadget polygonal line are the same as those of the top-left-gadget, with the difference being in the direction of the L-shaped polygonal extension.

Definition 44. *For the leftmost segments on the middle and the bottom horizontal segment chain define the following* left-gadget *polygonal line:*



Note that the points x, y and p are on the same vertical line. We add (p, w) with length $h/16$ on the same line as (p, q).

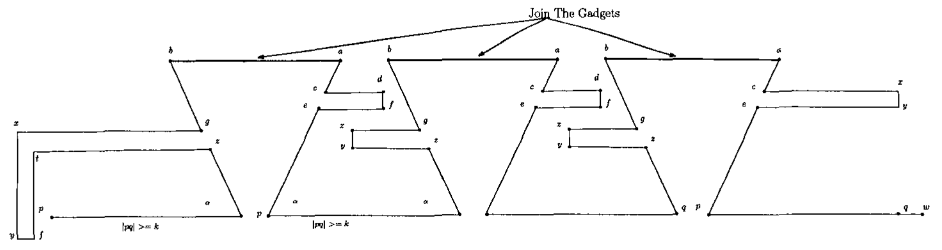
Definition 45. *For the rightmost segments on the middle and the bottom horizontal segment chain define the following* right-gadget *polygonal line:*



Note that the points x, y and q are on the same vertical line. We add (q, w) with length $h/16$ on the same line as (p, q).

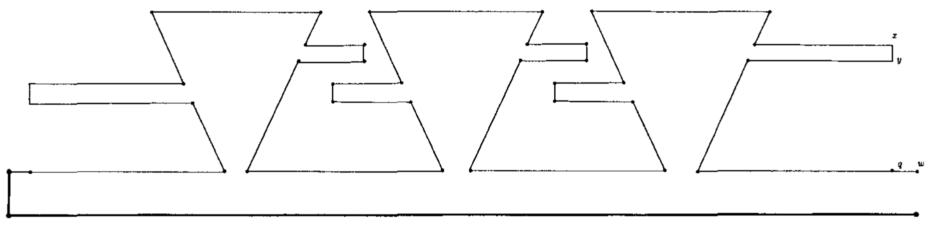
After we have built such a *gadget* on top of every horizontal line segment, we join the gadgets on the same horizontal line into a polygonal line in the following manner:

The top-right vertex of each gadget is connected by a horizontal edge to the top-left vertex of the gadget immediately to its right as shown below:



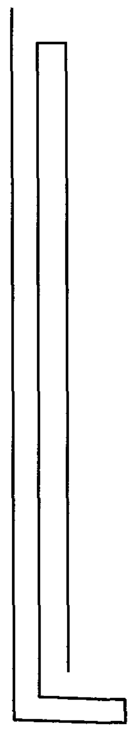
We now have three *polygonal lines* on each of the three horizontal lines of the original SEPARATOR2 construction.

Definition 46. *Starting at the left vertex of the left-most segment of the middle horizontal segment set, we define a polygonal line extension composed of a vertical edge with length $h/4$, and a horizontal edge with length equal to the horizontal length of the polygonal chain of gadgets. This horizontal edge, will be called the designated edge and this construction ensures that any line that is a solution to SEPARATOR2 will intersect this designated edge. This construction is shown below:*

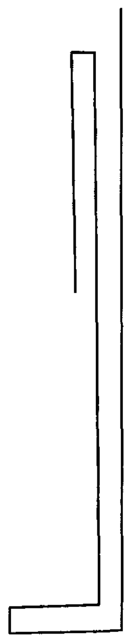


Now we complete the polygon construction by joining the rightmost vertex of the designated edge with the rightmost vertex of the bottom polygonal line using a vertical edge.

To close off the left vertical part of the polygon we draw a left vertical zigzag between the top line and the bottom horizontal line as shown below:



For the right vertical side we construct the following gadget to complete the construction of the polygon:



These two vertical gadgets correspond to the two vertical lines of the SEPARATOR2 input.

This ensures that any line crossing either of the vertical edges of the polygon will intersect at least 6 edges of the polygon.

Finally we have constructed the polygon P on top of the SEPARATOR2 input as shown in Figure 3.2.

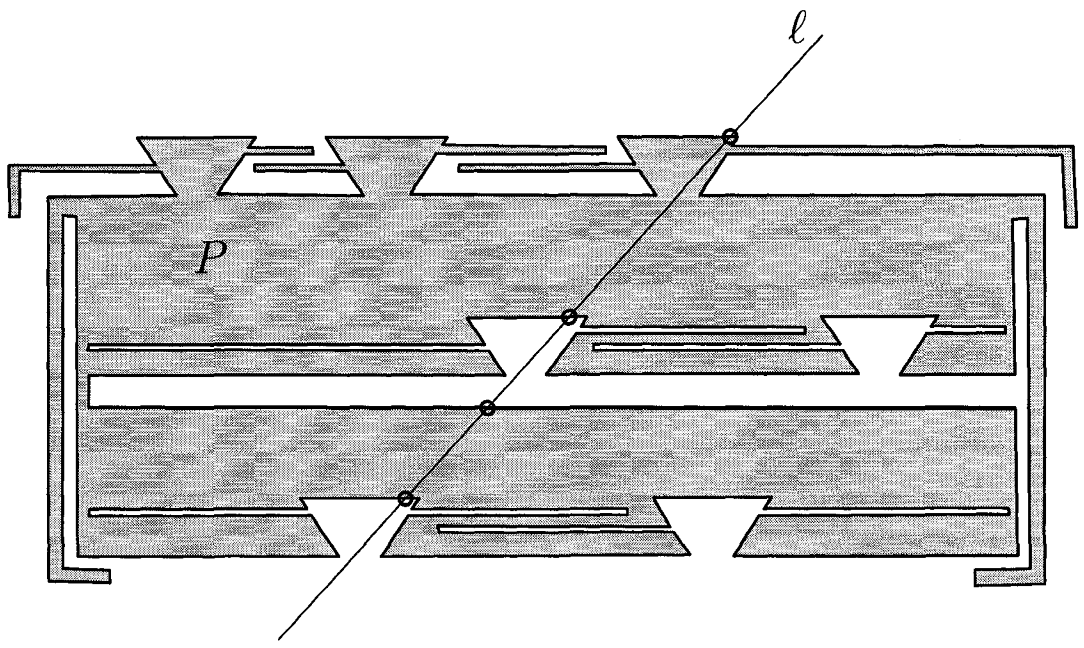


Figure 3.2: The final polygon P

Claim 47. *The polygon P can be constructed in $O(n(\log(n)))$ time.*

Proof. Clearly sorting the vertices of the edge segments takes $O(n(\log(n)))$ time. The computation and addition of the polygon edges takes linear time in the number of segments in the base problem which is $O(n)$. So the sorting step dominates the running time. □

Lemma 48. *In the polygon P defined by this construction, a line ℓ intersects the designated edge, and exactly three other edges of P, if and only if, ℓ is a solution to SEPARATOR2.*

The rest of this section deals with the proof of this lemma.

If the line ℓ intersects any of the bounding vertical edges of P, it is not a solution to SEPARATOR2. So we only need to look at the case when the line intersects each of the three horizontal *polygonal lines* of P plus the designated edge.

Claim 49. *If a line ℓ is a solution to SEPARATOR2, it intersects exactly 4 edges of the polygon P, including the designated edge.*

Proof. Let us call the edges of P that have a *top-left* or *top-right* vertex in either of the gadgets we defined, the trapezoid edges, since they are the only edges of P that are neither vertical nor horizontal. The slope of the trapezoid edges is either α or $-\alpha$, and the angle of any solution line through the gaps must be bigger than α in absolute value (since we picked the minimum possible slope of any line passing through the gaps). Such a line will intersect the trapezoid shape we obtained by joining the gadgets above the gaps, at the top horizontal edge joining a *top-left* to a *top-right* vertex only, which corresponds and is parallel to the gap between two horizontal line segments in SEPARATOR2.

There are three such *top* edges, each corresponding to one of the gaps, so including the designated edge, this makes the number of intersected edges, 4. $\square$

Claim 50. *If a line is not a solution to SEPARATOR2, hence it intersects at least one of the horizontal segments in the original SEPARATOR2 problem, and if this line intersects the designated edge, then it intersects at least 6 edges of P.*

Proof. If a line is not a solution to SEPARATOR2 that means that it does not pass through three gaps, hence it must intersect at least one of the horizontal segments. (we are discarding lines that do not intersect this structure) Clearly any line intersecting any of the horizontal polygonal chains and not passing through a gap, will intersect at least 3 edges in one of the *gadgets* due to our construction. If the same line crosses our designated edge then it must intersect an additional edge to exit the polygon, so the total number of edges intersected is greater than 4. In the best case the number of edges of P which are intersected are at least six. $\square$

To conclude, we have proved that the problem of finding a line in the polygon P, with the property that it intersects a designated edge and exactly three other edges of P, is at least as hard as the SEPARATOR2 problem.

3.4 The K-separator complexity for $K > 3$

Since $K = 1 + k/2$ where k is the number of edges in P that are intersected, for $K > 3$ we must have $k > 4$.

Theorem 51. *For any constant $k > 4$, the problem of finding a line which intersects k edges of P (including the designated edge), is 3SUM-hard.*

The rest of this section generalizes the results obtained in the previous section by proving this theorem.

Our proof for the case when $k = 4$ used the special polygon we constructed around the SEPARATOR2 input.

The special polygon we constructed can be modified so that any solution line to SEPARATOR2 intersects k edges by adding the required even number of horizontal zig-zag polygon edges just below the designated edge as shown in Figure 3.3.

We also double the number of edges in all gadgets we used to build the polygon. This is not shown in Figure 3.3 so as not to clutter the picture.

The additional construction is shown in Figure 3.3 to illustrate how to modify the polygon such that any solution line to the SEPARATOR2 base problem, intersects exactly k edges of P, including the designated edge. Any other line intersecting the polygon P will intersect more than k edges of P due to the increase in the number of edges in the gadgets.

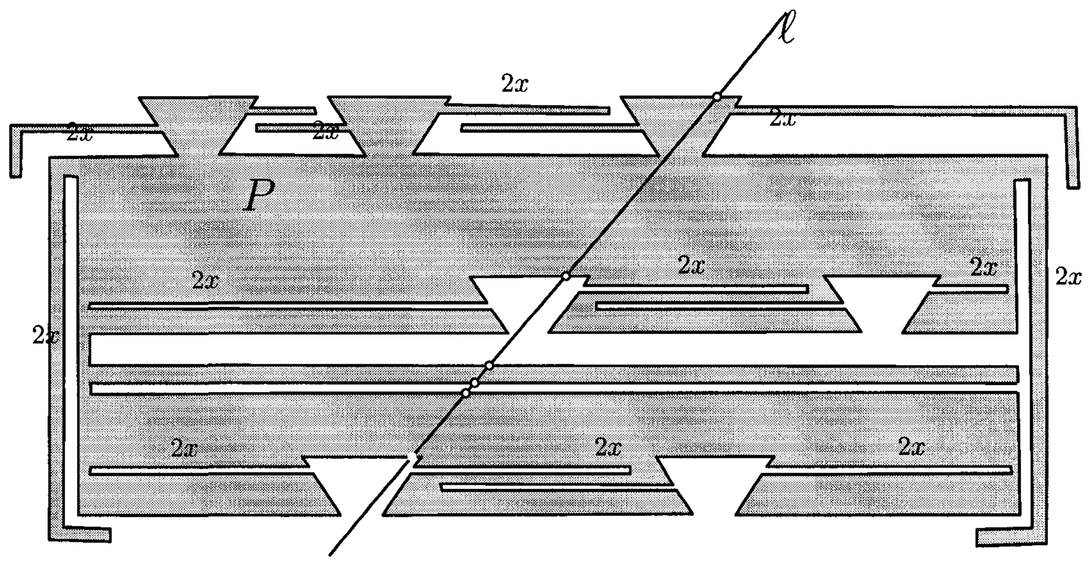


Figure 3.3: Modified Construction for 6-separator

This concludes the proof of the 3SUM-hard complexity of the problem for any constant $k > 2$.

Chapter 4

Finding 2-Separator lines crossing a fixed edge

In this chapter, we will show that the decision problem of finding a line that intersects a fixed edge of P and exactly one other edge of P, can be solved in sub-quadratic time using the hourglass data structures introduced by Guibas and Hershberger [15].

That is, we will prove the following result:

Theorem 52. *Given a simple polygon P with n vertices and an edge e of P, we can decide in $O(n\log(n))$ time if there exists a line through e which intersects exactly one other edge of P. This includes the special cases of lines passing through a vertex of P or lines containing an edge of P.*

4.1 Discussion

The rest of this chapter deals with proving this theorem.

Notation 53. *A pocket of a simple polygon P, consists of an edge of the convex hull of P which is not an edge of P (call it (p, q)), and all other edges of P on the path from p to q. The pocket is a simple polygon inside the convex hull of P, which is not a part of P. The edge (p, q) is called the window of the pocket. See Figure 4.1.*

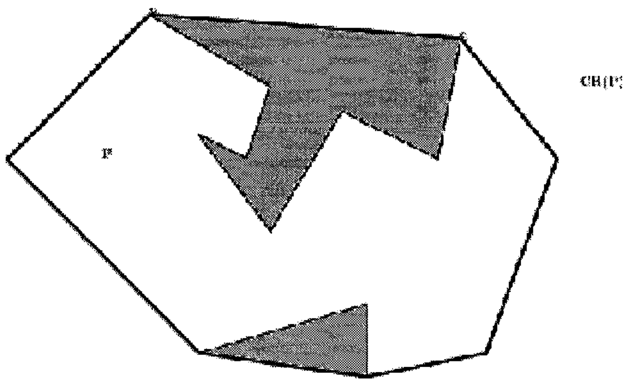


Figure 4.1: Two pockets of P

As seen in Chapter 2, if e is an edge of P that lies on the convex hull of P then there always exists a line through e which intersects exactly one other edge of P, (because convex hull vertices map to lines in the envelope of the arrangement $\mathcal{A}(P)$).

Therefore, it only makes sense to talk about this existence problem, if the edge in question is inside a *pocket* of P.

Notation 54. *The shortest path between two points v and w inside a simple polygon P is denoted as $\pi(v, w)$ and consists of the shortest chain of contiguous segments from v to w which is completely contained inside P.*

Guibas and Hershberger [15] showed how to build a data structure in linear time, such that the shortest path between any two query vertices of the simple polygon can be computed in $O(\log(n))$ time. This shortest path is represented as a balanced binary tree whose height is logarithmic in the length of the path it represents.

Definition 55. *An hourglass between two edges e and e' of P, is the union of all shortest paths $\pi(v, w)$, where v ranges over all points on e and w ranges over all points on e'.*

To compute the hourglass thus defined between two edges $e = (v_1, v_2)$ and $e' = (v_3, v_4)$, we only need to compute the shortest paths among two pairs of the vertices of both edges. These pairs are determined as follows: Let $e = (v_1, v_2)$ and $e' = (v_3, v_4)$ be the two edges in question such that the segments (v_1, v_3) and (v_2, v_4) do not intersect. Then the hourglass is the polygon bounded by the shortest path $\pi(v_1, v_3)$, the edge (v_3, v_4), the shortest path $\pi(v_2, v_4)$ and the edge (v_1, v_2). See Figure 4.2.

Notation 56. *The edges (v_1, v_2) and (v_3, v_4) are called the* lids *of the hourglass, while the shortest paths $\pi(v_1, v_3)$ and $\pi(v_2, v_4)$ are called the* sides.

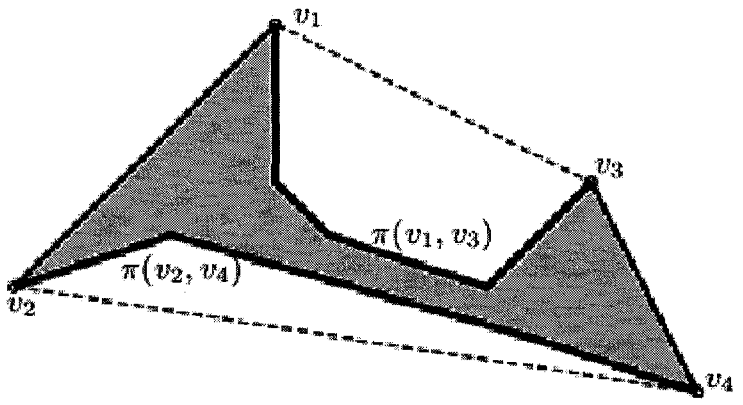


Figure 4.2: Two Shortest Paths - Hourglass

Since we can compute these shortest paths inside P in logarithmic time as shown in [15], it follows that we can compute the hourglasses in logarithmic time as well after a linear amount of pre-processing work.

Definition 57. *If the two shortest paths $\pi(v_1, v_3)$ and $\pi(v_2, v_4)$, defining the sides of an hourglass are disjoint, they are two inward convex chains and the hourglass is* *called* open. *Otherwise, these two shortest paths must have at least one vertex in common, and the hourglass is called* closed. *See Figure 4.3.*

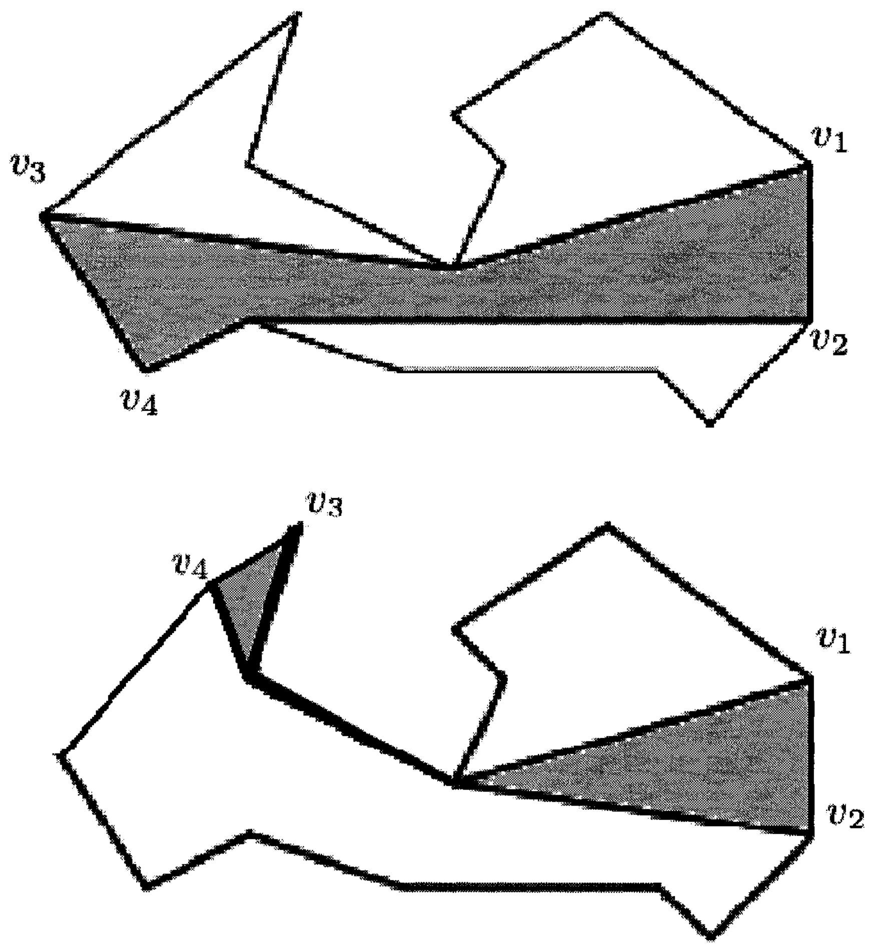


Figure 4.3: Two types of Hourglasses - open and closed

Now we have all the geometric properties needed to analyze the solution of our problem. As we will show now, to solve our problem we only need to look at *open* hourglasses.

Claim 58. *There exists a segment joining some point on one lid of the hourglass, to some point on the other lid of the hourglass, which does not intersect any other part of the hourglass, if and only if, the hourglass is open.*

Proof. If the hourglass is closed then it is obvious that any line joining the two lids must intersect at least one of the sides.

If the hourglass is open, then we have two disjoint inward convex chains that make up the hourglass. These convex chains do not touch.

As shown in Figure 4.2, we can join the end vertices of either chain by a line segment, hence forming two disjoint convex polygons, which can always be separated by a straight line.

Any such line must intersect both lids of the hourglass, because they are two opposite edges of a quadrilateral which contains both convex polygons defined by the convex chains, as well as the open hourglass.

□

Another property of hourglasses we are going to use is concatenation.

Fact 59. *Two disjoint open hourglasses that share a lid are concatenatable, meaning they can be merged into one hourglass that could be either* open *or* closed. *The concatenation of two open hourglasses can be done in the time required for finding a tangent from a point to a convex chain.*

Observation 60. *Finding the common tangents of the convex chains that make up two hourglasses, enables us to concatenate two hourglasses into one.*

Since we are interested in finding lines that intersect exactly two edges of P, we are only interested in hourglasses that are *open.*

Fact 61. *Using the standard algorithms for finding common tangents between two convex chains [22] we can decide in time logarithmic in the number of vertices on the hourglass, whether the hourglass is open or closed.*

Bearing all this in mind, here is an overview of our algorithm:

1. Compute the convex hull of P and identify all pockets of P.

2. Compute the hourglass data structure for P.

3. For each pocket of P, compute the hourglass data structure for this pocket.

4. Let e be the designated edge and assume it is inside some pocket.

5. Compute the hourglass of e to the *window* of e's pocket. Let us call this hourglass $H(e)$. If $H(e)$ is closed then answer our decision problem *NO.* (see claim 62)

6. For every edge f that is not in any pocket, compute the hourglass $H(f, e)$ between f and e and concatenate this hourglass with $H(e)$. If the result is an open hourglass we answer our decision problem *YES.* (see claim 63)

7. For every edge f of P, which lies inside some pocket but not in e's pocket, compute its hourglass $H(f)$ to the *window* of its pocket and compute a second hourglass $H(e, f)$, between the designated edge e and f. Concatenate the hourglasses $H(e)$, $H(e, f)$ and $H(f)$. If the resulting hourglass is open, answer the decision problem *YES* (see claim 64).

8. Answer the decision problem *NO* (see claim 65).

Claim 62. *Let e be the fixed edge of P which lies inside a pocket of P. If $H(e)$ is closed, then any line that intersects e also intersects at least one other edge of the pocket, hence it intersects more than 2 edges of P.*

Proof. The pocket itself is a polygon. Since the hourglass between e and the window of the pocket is closed, there does not exist a line that intersects exactly e and the window of the pocket. Therefore it must intersect another edge of the pocket, hence enter the interior of P in two directions (once through e and the second time through some other edge of e's pocket). Therefore this line must intersect at least two more edges of P. □

Claim 63. *Let e be a fixed edge of P. Let f be an edge that is not in any pocket, let $H(f,e)$ be the hourglass between f and e. If the result of the concatenation of $H(e)$ with $H(f,e)$ is open, then there exists a line ℓ that intersects exactly f and e.*

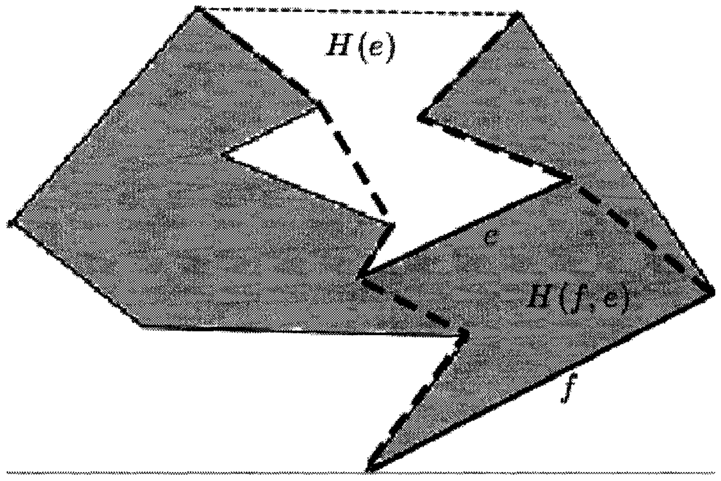


Figure 4.4: $H(f,e)$ concatenated with $H(e)$

Proof. Since the concatenation of $H(e)$ with $H(f,e)$ is open, there exist two disjoint inward convex chains from the window of the pocket of e, to the edge f. Both the window of the pocket of e and f are edges of the convex hull of P. Therefore we can find a line ℓ that separates these two convex chains, by joining some point on f and some point on the window of the pocket of e. This line also intersects e and does not intersect any other edge because e is intersected by both inward convex chains. See Figure 4.4. □

Claim 64. *Let e be the fixed edge of P that is assumed to lie inside a pocket, f an edge of P which lies inside some pocket but not in e's pocket. If the concatenation* *of the hourglasses $H(e)$, $H(e, f)$ and $H(f)$ is open, then there exists a line through e which intersects f and no other edge of P.*

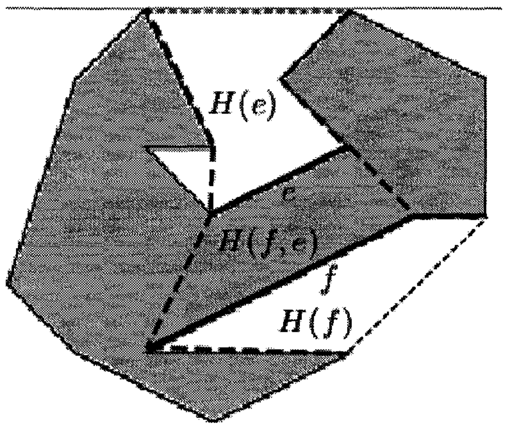


Figure 4.5: $H(e)$, $H(e, f)$ and $H(f)$

Proof. The hourglass obtained by concatenating $H(e)$, $H(e, f)$ and $H(f)$ is composed of two inward convex chains and the window of the pocket of e and the window of the pocket of f. Therefore, we can find a line ℓ that separates both these convex chains formed by joining two points on either window. The edges e and f are intersected by both chains, hence the line also intersects e and f. No other edge is intersected because the line separates the convex chains, hence does not intersect any edge that belongs to either chain. See Figure 4.5 □

Claim 65. *If all concatenated hourglasses computed in steps 6 and 7 of the algorithm are closed, then there does not exist a line through the fixed edge e, which intersects exactly one other edge of P.*

Proof. Steps 6 and 7 of the algorithm examine every edge of P that is not in the pocket of e. If all these concatenated hourglasses are closed then we can be certain that none of these edges can have a line through it that intersects only the fixed edge

e and no other edge of P. If we join any point on an edge that is in the pocket of e with any point on e then the line will intersect at least two more edges of P. This accounts for all pairs of edges composed of an edge of P with the fixed edge e. □

Claim 66. *The running time of this algorithm is $O(n\log(n))$.*

Proof. The first step is finding the convex hull of the simple polygon P, which can be done in $O(n)$ time [14].

Building the data structures for the hourglasses takes linear time in the size of P plus the total size of all pockets of P. This combined size is as most $2n$ since each edge of P is counted at most twice: once in P and once in some pocket of P.

Also, since we fixed an edge, the total number of hourglasses we need to compute is $O(n)$. Each hourglass can be computed in $O(\log(n))$ time as shown in [15], and the concatenation of hourglasses can be done in logarithmic time as shown in [15]. Hence the total running time is $O(n\log(n))$.

This concludes the proof of Theorem 48. □

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have shown a simple algorithm which uses a variation of the classical depth first search algorithm, for computing all sets of lines that cut a simple polygon in a given number of pieces - as defined in Problem 1 of the introduction. This algorithm runs in $O(n^2)$ time and is worst case optimal.

We used the duality transformation to define the solution as sets of lines determined by points inside the cells of the arrangement of double-wedges.

The *depth* of a cell was the quantity we needed to compute in order to determine how many edges of P, the duals of the points of the cell intersect.

We denoted the number of pieces that the polygon P is cut by a line ℓ by K and the number of edges of P that are intersected as a result by k. These two quantities are related and were used interchangingly in our proofs.

Then we have shown that Problem 2 as defined in the introduction, is 3SUM-HARD for $K > 2$ due to a reduction from the SEPARATOR2 3SUM-hard problem in sub-quadratic time. For $K = 2$ the existence problem for a fixed edge can be answered in $O(n \log(n))$ time.

5.2 Future Work

Given the complexity of enumerating a sub-set of lines with the property of intersecting a fixed number of edges of a simple polygon, finding output sensitive algorithms to output these solution sets remains an open problem. There are polygons where the output size is a constant (for example a spiral shaped polygon as shown in Figure 5.1), so an output sensitive algorithm in these cases will produce an optimal result.

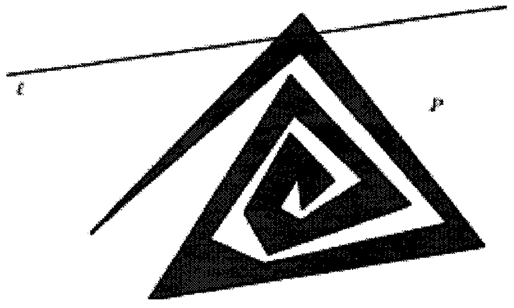


Figure 5.1: A spiral Polygon

A subquadratic solution may still be derived if we simplify the polygon by replacing convex edge chains with a constant number of edges, hence making the running time depend on the number of reflex vertices in P.

Several other complexity results may be obtained. We suspect that the problem of finding the cell of maximum depth is also 3SUM-hard, and as a corollary the problem of identifying the line that cuts the polygon in the maximum possible number of pieces is also 3SUM-hard.

These problems can then be extended to three dimensions. The same questions we asked and answered for the 2D case of a simple polygon in a plane intersected by a line, can be addressed for the 3D case of a simple polyhedron intersected by a plane.

Bibliography

[1] K. F. Boehringer, B. Randall Donald, and D. Halperin. On the area bisectors of a polygon. Annual Symposium on Computational Geometry, Pages: 457 - 459, Nice, France, 1997.

[2] P. Bose, J. Czyzowicz, E. Kranakis, D. Krizanc, and A. Maheshwari. Polygon cutting: Revisited. Proceedings of the Japanese Conference on Discrete and Computational Geometry, Geom., pp. 114-118, 1998.

[3] P. Bose, E. D. Demaine, F. Hurtado, S. Langerman, J. Iacono, and P. Morin. Geodesic ham-sandwich cuts. Discrete and Computational Geometry, accepted, 2006.

[4] P. Bose and M. Van Kreveld. Generalizing monotonicity: On recognizing special classes of polygons and polyhedra by computing nice sweeps. International Journal of Computational Geometry, 15(6), 2005.

[5] B. Chazelle. A theorem on polygon cutting with applications. In proc. of found. of comp. sci. pp 339-349, 1982.

[6] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. BIT 25 (1985), 76-90, 1985.

[7] F. Contreras. Cutting plygons and a problem on illumnation of stages. PhD Thesis - University of Ottawa, Ottawa, Ontario, Canada, 1998.

[8] O. Daescu. Cutting out polygons with lines and rays. Algorithms and Computation pages 669-680, University of Texas at Dallas, Dallas, TX, USA, 2004.

[9] M. Diaz and J. O'Rourke. Ham-sandwich sectioning of polygons. Canadian Conference on Computational Geometry, pages 282-286, 1990.

[10] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl. Implicitly representing arrangements of lines or segments. Proceedings of the fourth annual symposium on Computational geometry, Pages: 56 - 69, Urbana-Champaign, Illinois, United States, 1989.

[11] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. Proceedings of the eighteenth annual ACM symposium on Theory of computing, Pages: 389 - 403, Berkeley, California, United States, 1989.

[12] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. BIT, Springer Netherlands, 1982.

[13] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. Computational Geometry, Volume 5, Issue 3, October 1995, Pages 165-185, 1995.

[14] R. Graham and F. Yao. Finding the convex hull of a simple polygon. Journal of Algorithms 4, 324-331, 1983.

[15] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. Journal of Computer and System Sciences, 39(2):126-152, 1989.

[16] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. Proceedings of the third annual symposium on Computational geometry, Pages: 50 - 63, Waterloo, Ontario, Canada, 1989.

[17] J. Iacono and S. Langerman. Volume queries in polyhedra. Proc. Japan Conf. Discrete Comput. Geom, 2001.

[18] I. Ilinkin, R. Janardan, J. Majhi, J. Schwerdt, M. Smid, and R. Sriram. A decomposition-based approach to layered manufacturing. Computational Geometry: Theory and Applications, volume 23, 2002, pp. 117-151.

[19] S. Langerman. On the complexity of halfspace area queries. Journal Discrete and Computational Geometry pages 639-648, 2004.

[20] S. Chandra Nandy. Shattering a set of objects in 2d. Canadian Conference on Computational Geometry - 1999, Halifax, NS, Canada, 1999.

[21] M. Overmars and M. van Kreveld. Computational geometry - theory and applications. Springer-Verlag, 1999.

[22] M. Overmars and H. van Leeuwen. Maintainance of configurations in the plane. Journal of Computer and System Sciences, vol 23, pp 166-204, 1981.

[23] M. Overmars and E. Welzl. The complexity of cutting paper. Procs. of the 1st Annual ACM Symposium on Computational Geometry. (1985) 316321., 1985.

[24] Thomas C. Shermer. A linear algorithm for bisecting a polygon. Inf. Process. Lett. 41(3): 135-140, 1992.

[25] I. Stojmenović. Bisections and ham-sandwich cuts of convex polygons and polyhedra. Information Processing Letters 38(1), pages 15-21, April 1991.

[26] A. H. Stone and J. W. Tukey. Generalized 'sandwich' theorems. Duke Math. J. 9, 356-359, 1942.

[27] R. Tarjan. Depth-first search and linear graph algorithms. Society for Industrial and Applied Mathematics, Pages 146-160, 1972.

[28] M. van Kreveld and B. Speckmann. Cutting a country for smallest square fit. Lecture Notes in Computer Science / Heidelberg, Volume 2518/2002, Pages 133-167, 2002.