Minimum Perimeter Convex Hull of a Set of Line Segments: An Approximation

by

Farzad Hassanzadeh

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
November 2008

Abstract

The problem of finding the convex hull of a set of points in the plane is one of the fundamental and well-studied problems in Computational Geometry. However, for a set of imprecise points, the convex hull problem has not been thoroughly investigated. By imprecise points, we refer to a region in the plane inside which one point may lie. We are particularly interested in finding a minimum perimeter convex hull of a set of imprecise points, where the imprecise points are modelled as line segments. Currently, the best known algorithm that solves the minimum perimeter convex hull problem has an exponential running time in the worst case [14]. It is still unknown whether this problem is NP-hard. We explore several approximation algorithms for this problem. Finally we propose a constant factor approximation algorithm that runs in $O(n \log n)$ time.

Acknowledgements

I would like to dedicate this work to my family who has supported me through all stages of my life.

I am grateful to my supervisor, David Rappaport, who has provided support and guidance throughout my education at Queen's. I would also like to thank my friends, without whom, I would not have had such an amazing experience in Kingston.

Last but not least, I would like to thank the Queen's community for providing me with this great opportunity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

There are many problems in Computational Geometry that deal with a set of points as the input to the problem, for example the Travelling Salesman Problem (TSP), the Minimum Spanning Tree (MST) problem, Voronoi Diagrams, the Convex Hull problem, etc. Here is a brief definition for these problems: Given a number of cities and the costs of travelling from any city to any other city, the TSP problem is to find the least-cost round-trip route that visits each city exactly once and then returns to the starting city. Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree and connects all the vertices together. A minimum Spanning Tree (MST) is a spanning tree with weight less than or equal to the weight of every other spanning tree. Given a set of points SP $= \{p_1, p_2, ..., p_n\}$, the Voronoi Diagram of SP is a partitioning of the plane into regions around each point p_i, such that all the points in the region around p_i are closer to p_i than any other point in SP. We talk about Voronoi Diagrams in Section 4.3.1. Given a set of points P,

the Convex Hull of P is the smallest convex polygon that encloses all the points in P. The Convex Hull problem is discussed in detail in Section 2.2. A more detailed description of the formerly mentioned problems is available in [15]. In all the above problems, an output is generated according to the input set of points. This works perfectly in the theoretical world. But when these algorithms are implemented with ordinary floating-point arithmetic, the results may be unreliable [7]. Independent of floating-point arithmetic error, the data acquisition methods, for example a satellite orbiting around the Earth, may also introduce errors. We use the term *data* to refer to the points in the plane and the term *imprecise point* to refer to a region in the plane inside which one point may lie.

One way of dealing with imprecision is using stochastic or fuzzy models. For instance, Fortune [5] presented an $O(n \log n)$ time algorithm for computing approximate convex hulls using floating-point arithmetic. The drawback of Fortune's convex hull algorithm is that the resulting convex hull may not be exactly convex. More precisely, Fortune's convex hull algorithm, which is a slightly modified version of Graham's Algorithm [6], constructs an *ϵ-weakly convex* polygon, which means the vertices of the resulting polygon may have to be perturbed by as much as much as ϵ in order for the polygon to be convex. These types of algorithms are also used to construct Voronoi diagrams [16] or triangulate planar point sets [5]. These approaches have numerical errors, but the algorithms would not face topological inconsistency and thus the given task will be completed [16].

In order to deal with imprecise data, we need algorithms that can deal with imprecision or approaches that can provide us with lower and/or upper bounds on the output. For example if we are given the amount of the error for the input points,

we are interested in finding out what the largest and smallest TSP tour for that set of imprecise input points is, or what the lower and upper bounds for the area of the convex hull of the input points are. The classic algorithms that solve the TSP or the convex hull problem do not provide us with the required outputs.

Consider the problem of finding the convex hull of a set of imprecise points. First of all, we need to know how the imprecision is modelled. The regions can be modelled as discs, squares, circles, line segments, etc. After deciding which model we are going to use, we need to know what to minimize or maximize. It can either be minimizing or maximizing the area of the convex hull, minimizing or maximizing the perimeter of the convex hull, etc. The classic convex hull algorithms, such as Graham's scan [6] and the gift wrapping algorithm [9], do not solve this type of problems. This motivates us to look for new approaches that can deal with imprecise points for the convex hull problem.

1.2 Problem Definition

Suppose we are given a set of line segments S. We assume that there exists at least three line segments in the set, the line segments do not overlap and they do not meet at the end points. We can construct a set of points by taking one point from each line segment $s_i \in S$. The convex hull of this set of points can be computed in $O(n \log n)$ time [6]. By taking different points from each line segment $s_i \in S$, an infinite number of such sets can be constructed. Fig. 1.1(A) and Fig. 1.1(B) show two different sets of points, taken from the same set of line segments. As it is shown in the figures, the convex hulls of the points are not the same. We are concerned with finding a set of points that yields the minimum perimeter convex hull.

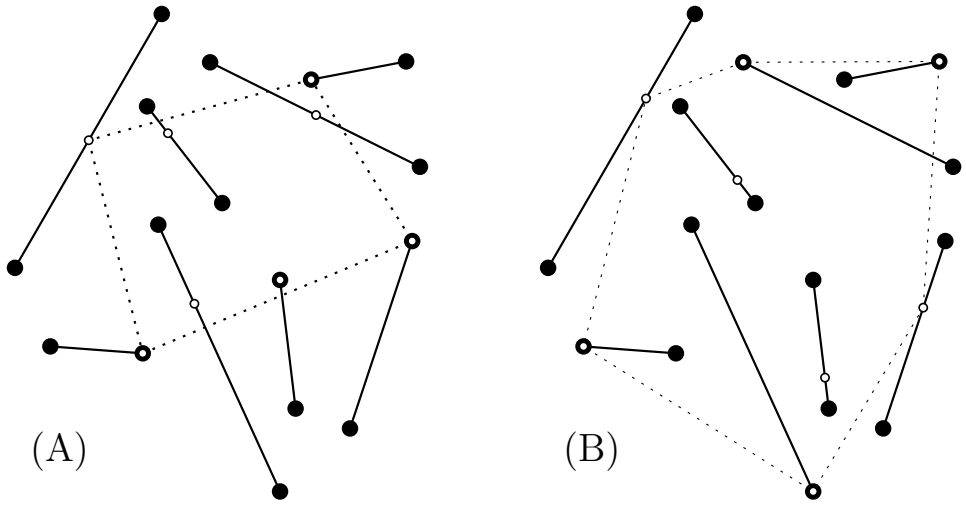

Fig. 1.1: Convex hulls of a set of line segments

We are interested in the problem of finding the minimum perimeter convex hull (MPCH) of a set of line segments S. Imagine that we are given two sets of line segments S_1 and S_2. It may not be possible to draw a straight line between S_1 and S_2 such that the two sets are completely separated. Now consider the minimum perimeter convex hull of S_1 and the minimum perimeter convex hull of S_2. If these two convex hulls are separable by a straight line, then we may call S_1 and S_2 separable. This can be generalized to n sets of line segments. So the minimum perimeter convex hull could be a used as a measure of separability.

This problem is equivalent to finding a convex polygon of minimal perimeter that spans all the line segments in S. A polygon spans a set of line segments S, if every line segment $s_i \in S$ has a non-empty intersection with the polygon. In the following, we refer to this problem as the minimum perimeter convex hull problem for a set of line segments. Observe that the minimum perimeter convex hull of a set of line segments may not be unique.

1.3 Objective

Throughout Chapter 3, we will explain the currently best known algorithm that computes the minimum perimeter convex hull of a set of line segments S [14]. This algorithm in the worst case has an exponential running time of $O(3^n n \log n)$. We will show that this algorithm does not result in the optimal solution in some special cases. Then we propose an $O(4^n n \log n)$ time algorithm that extends the original algorithm and covers the special cases.

The main objective of this thesis is to find an approximation algorithm to compute a convex polygon C', such that C' spans all the line segments in S and the perimeter of C' is bounded. We expect this approximation algorithm to have polynomial running time.

1.4 Contributions

We implemented the currently best known algorithm for computing the minimum perimeter convex hull of a set of non-overlapping line segments, proposed by Rappaport [14]. This helped us to obtain empirical results, as well as an approximation approach, called the FIFO approach (Section 4.2). The main contribution of this thesis to the problem is an approximation algorithm called the Minimum Spanning Circle (MSC) approach (Section 4.3). The final result of this thesis is summarized as Theorem 4.3.8. According to this theorem, for a set of line segments S with a minimum perimeter convex hull C^*, the MSC approach finds a convex polygon C' in $O(n \log n)$ time such that C' spans all the line segments in S and $\frac{\text{perimeter of } C'}{\text{perimeter of } C^*} \leq \frac{\pi}{2}$. Note that the MSC approach also works for a set of overlapping line segments.

1.5 Organization of Thesis

In Chapter 2, we talk about previous works and the problems related to the minimum perimeter convex hull problem.

In Chapter 3, we explain in details, the steps to find a minimum perimeter convex hull of a set of non-overlapping line segments, as proposed in [14]. Although in Section 3.3.3, we add an extension to the original algorithm to cover a case that was missed in [14]. This extension will increase the running time of the algorithm from $O(3^n n \log n)$ to $O(4^n n \log n)$ in the worst case.

Section 3.3 explains how we can find a minimum perimeter convex hull of a set of non-overlapping line segments. The material introduced in this section will be used in Chapter 4, where we introduce two different approaches to approximate a minimum perimeter convex hull.

The first approach, the FIFO approach, uses the algorithms described in Chapter 3. This approach has a polynomial running time, but unfortunately, the output of the FIFO approach differs greatly from the optimal result. However the second approach, the Minimum Spanning Circle approximation, uses a completely different technique to solve the MPCH problem. We provide proofs on feasibility and bounds of the MSC approximation. The main result of this thesis is summarized as Theorem 4.3.8, where the bounds for the Minimum Spanning Circle approximation are presented.

Finally in Section 5, we summarize our results and we talk about future work and open problems.

Chapter 2

Background

2.1 Imprecise Points

Epsilon Geometry, introduced by Guibas *et al.* [8] is a framework for robust computation for imprecise points. In another paper [7], using Epsilon Geometry, they introduce $(-\epsilon)$-convex polygons. For some fixed $\epsilon \geq 0$, a polygon is $(-\epsilon)$-convex if it remains convex under any perturbation of ϵ or less, to its vertices.

Löffler and van Kreveld [13] look at different Computational Geometry problems for imprecise points. For a set of regions $\mathcal{L}$ and a Computational Geometry problem that takes a set of points as input and returns a real number R, they are interested in finding a way to place one point in each region of $\mathcal{L}$ such that the resulting set of points minimizes or maximizes R. The imprecision for a point can be modelled in different ways. A disc or a square can represent a point such that the point can be anywhere inside or on its boundary. The imprecision can also be modelled as a line segment, in which the point can be anywhere on the line segment. Löffler and van Kreveld [13] study the problem of minimizing and maximizing the diameter,

smallest bounding box, smallest enclosing circle, etc, for regions modelled as squares and discs. They propose polynomial time algorithms for some of the problems and prove NP-hardness for the others.

2.2 The Convex Hull Problem

One of the basic problems of Computational Geometry is the problem of finding the convex hull of a set of points. Given a set of points P, we want to find the smallest convex polygon that encloses all the points in P. We can think of the convex hull of a set of points (in the plane) as a rubber band wrapped around the outside points. The convex hull algorithm has applications in computer graphics and gaming, where the convex hull of an object can replace the bounding box of that object. There are many Computational Geometry problems that use the convex hull algorithms as part of their solution.

One of the first efficient convex hull algorithms was introduced by Graham and is commonly known as *Graham's scan* [6]. The Graham's scan begins by designating a pivot point and sorts the rest of the points in angular order about the pivot. Using this ordering the convex hull is constructed by a linear scan. Thus Graham's scan runs in $O(n)$ time after sorting which results in an $O(n \log n)$ time algorithm.

There are a variety of algorithms for constructing the convex hull of a set of points. Some of the algorithms are output sensitive, i.e. their running time depends on the number of points that are located on the convex hull. The gift wrapping algorithm [9] introduced by Jarvis is a very simple output sensitive algorithm to find the convex hull of a set of points in the plane. The algorithm starts from point A, the point with the smallest Y coordinate. Then it adds an edge between A and another point B, if

all the other points were located on the left side of $\overline{AB}$. Point C is chosen such that all the points are located on the left side of $\overline{BC}$ and so on. The algorithm continues until it reaches point A again. The computational complexity of the gift wrapping algorithm is $O(nh)$, were n is the number of points in the input set and h is the number of points on the convex hull. Chan's Algorithm is another output sensitive algorithm which runs in $O(n \log h)$ time. For convex hulls in 2-D, Chan's Algorithm combines Graham's scan and Jarvis's algorithm to obtain an $O(n \log h)$ running time, where n is the number of input points and h is the number of points on the convex hull.

2.2.1 Convex Hull of Imprecise Points

The problem of finding the minimum perimeter convex hull of a set of non-overlapping line segments was studied by Rappaport [14]. He showed that for a set of n line segments with k different orientations, the problem of finding the minimum perimeter convex hull can be solved in $O(3^k n \log n)$ time. In the worst case $k = n$.

Löffler and van Kreveld [12] studied different variations of the convex hull problem with different constraints. They studied the problem of finding the maximum and minimum area/perimeter convex hull of a set of imprecise points modelled as line segments or squares. They proved that finding the maximum area/perimeter convex hull of a set of line segments is NP-hard. They also introduce an $O(n \log n)$ time algorithm to solve the minimum perimeter convex hull of a set of parallel line segments, which is a special case of the work done by Rappaport (for $k = 1$) [14].

Some of the convex hull problems mentioned above are proved to be NP-hard and the best known algorithms for them have exponential running time. Löffler and

van Kreveld [11] introduce an approach with $O(2^n n \log n)$ running time to solve the NP-hard problem of finding the maximum area convex hull of an arbitrary set of line segments. Then they define the concept of *Core-Set* for the problem and come up with an approximation that runs in $O(n)+2^{O(\eta^2)}$ time where n is the input size and $\eta^{-1} = \epsilon$ is the required precision of the answer.

There is no known polynomial-time algorithm that solves the problem of finding a minimum perimeter convex hull of a set of non-overlapping line segments. At the same time, there in no proof of NP-hardness available for this problem.

Chapter 3

Preliminaries

In this chapter, we will discuss the MPCH algorithm for a set of non-overlapping line segments which was proposed in [14]. We describe the algorithms in detail, however, we will not provide all the proofs and analyses for the algorithms. All the proofs for the theorems and analyses for the algorithms are provided in the original paper [14]. During experiments, we noticed that, in some special cases, the algorithm does not result in the optimal solution. So we added an extension to the original algorithm [14] in order to cover all the cases. In Section 3.3.3 we will discuss the missed case and explain how we can extend the algorithm to cover that case.

3.1 Line segment categories

In order to break down the problem, we categorize the line segments. By categorizing the line segments, we can treat different categories of line segments differently, since each category has different properties. There are two levels of categorizations. At the first level, we identify the necessary line segments from unnecessary line segments.

In the second level, we distinguish between different line segments in the *necessary* category.

3.1.1 Level 1: necessary and unnecessary line segments

As not every point contributes to construction of the convex hull of a set of points, not every line segment contributes to construction of the convex hull of a set of line segments. We can see the convex hull of a set of random points in Fig. 3.1. In this figure, the points which are located inside the convex hull do not affect the convex hull, i.e. if we remove these points from the set of points, the resulting convex hull will not change. The same argument is valid about the minimum perimeter convex

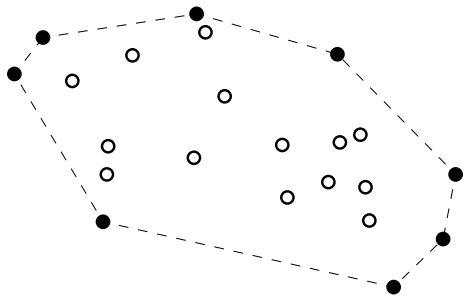

Fig. 3.1: Convex hull of a set of points

hull of a set of line segments. There may be line segments that do not play any role in the construction of the minimum perimeter convex hull.

The notations that we are about to introduce are the same as the ones used in [14]. Assume that L is a directed line. Let $H^l(L)$ be the closed left half-plane bounded by line L. For a set of line segments S, line L is an *extreme* line, if $s_i \cap H^l(L)$ equals an end point of s_i or $s_i \cap L = s_i$ and also $\forall s_j \in S, j \neq i,\ s_j \cap H^l(L) \neq \emptyset$. The end point of line segment s_i that lies on the extreme line L is called a *critical* point. A line segment with one or two critical points is a *critical* segment. An extreme line is

a *critical* extreme line, if it passes through at least two critical points of two different line segments. Fig. 3.2 shows the critical extreme lines for a random set of line

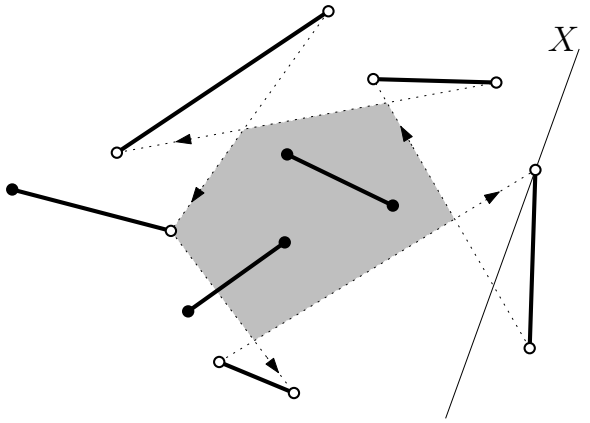

Fig. 3.2: Critical extreme lines

segments S. In this figure, the directed dashed lines represent the critical extreme lines. Line X is an extreme line. The critical points are shown as white end points. The *secure* polygon, the grey area in Fig. 3.2, is the intersection of all the $H^l(L)$ for all the critical extreme lines L. According to the definition of the critical extreme lines, any convex hull of a random set of points representatives of each line segment in S, includes the secure polygon, otherwise there is at least one line segment that is not covered by the convex hull. Since the secure polygon is included in any convex hull of S, any line segment $s_i \in S$ that intersects the secure polygon is also included in any convex hull. In Fig. 3.2 there are two such line segments. These are the line segments that do not contribute to the construction of a minimum perimeter convex hull and thus they can be ignored. The line segments that intersect the secure polygon are not critical, i.e. they have no critical end points. Using the critical extreme lines, we can divide the line segments $s_i \in S$ into two general categories: necessary line segments and unnecessary line segments. From now on, when we talk about the line segments

$s_i \in S$ we only mean the *necessary* line segments (unless explicitly mentioned).

3.1.2 Level 2: nice rim segments, normal line segments and pesky rim segments

In this section, we categorize the necessary line segments from level 1. We categorize the line segments using the critical points. Recall that an end point of a line segment is critical, if an extreme line passes through that end point.

Let L be a line, collinear to a line segment $s_i \in S$. We choose the direction of L such that there is at least one line segment on the left side of L. By CP_i we denote the set of all the critical points in $S - \{s_i\}$. In other words, CP_i is the set of all the critical points in the set of line segments S, except for the critical points located on the line segment s_i.

- If $\forall cp_j \in \mathrm{CP}_i$, $cp_j \cap H^l(L) \neq \emptyset$, then s_i is a *nice rim segment*.
- If $\exists s_j \in S$, $s_j \cap H^l(L) = \emptyset$ then s_i is a *normal line segment*.
- If $\forall s_j \in S$, $s_j \cap H^l(L) \neq \emptyset$, but $\exists cp_j \in \mathrm{CP}_i$, $cp_j \cap H^l(L) = \emptyset$ then s_i is a *pesky rim segment*.

Here are the informal definitions for these three categories. We call a line segment s_i a *nice rim segment* if all the critical points are located on the left side of s_i. Nice rim segments are the boundaries of the minimum perimeter convex hull of S, since there are no critical points located on the right side of the nice rim segments (according to the definition of nice rim segments). A vertex of a minimum perimeter convex hull that is located on a nice rim segment is called a *reflection vertex*, since any edge of the convex hull that intersects a nice rim segment will bounce off of it. According to

the definition of the normal line segments, there is at least one line segment s_l that is completely located on the left side and there is at least one line segments s_r that is completely located on the right side of any normal line segment. This means that the boundary of a minimum perimeter convex hull of S may have to *cross* a normal line segment in order to go from a point on s_l to a point on s_r. In the last case, the pesky rim segments, according to the definition, there is at least one point of each line segment on the left side of the pesky rim segment, which means a pesky rim segment can be treated as a nice rim segment. But at the same time, there is at least one critical point which is located on the right side of the pesky rim segment, which means the pesky rim segment can be treated as a normal line segment.

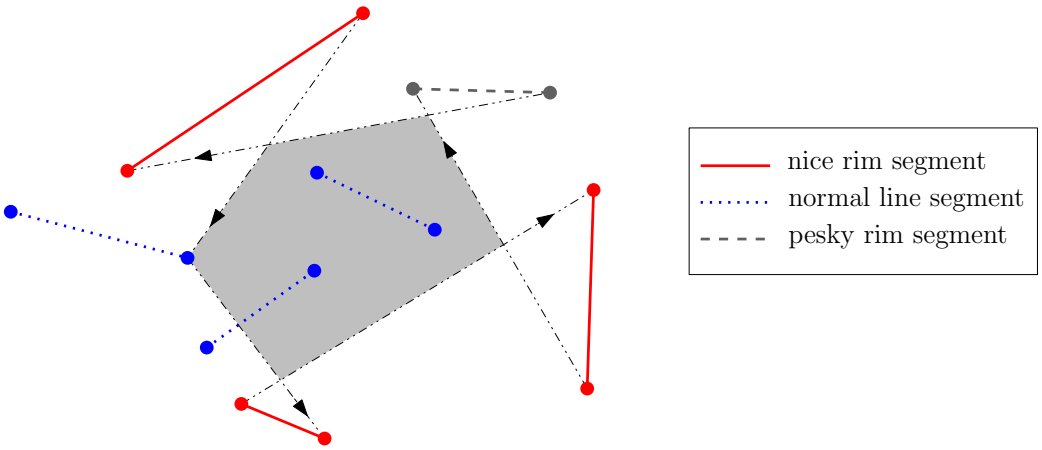

Fig. 3.3: Nice rim segments, normal line segments and pesky rim segments

From here on in the figures, the nice rim segments are shown as the solid line segments. The normal line segments are the dotted line segments and the dashed line segment represent the pesky rim segment. In Fig. 3.3, there are three nice rim segments, three normal line segments and one pesky rim segment. Note that two of the normal line segments are categorized as the unnecessary line segments. In fact

any unnecessary line segment (in the level 1 categorization) is a normal line segment. This does not imply that any normal line segment is an unnecessary line segment.

These are the three categories of the line segments. In the next sections we will see how the different properties of each of these three categories will contribute to the construction of the minimum perimeter convex hull.

3.2 Feasible polygons

As mentioned before, pesky rim segments can play the role of both the nice rim segments and the normal line segments. In the following sections we will see how this property of pesky rim segments can make the problem more complex. So for now we assume that there are no pesky rim segments in the set of line segments S.

According to the definition of nice rim segments, the minimum perimeter convex hull of a set of line segments S will be located on the left side of any nice rim segment. We can use this property of nice rim segments to construct the minimum perimeter convex hull of S.

In [14] it is proved that for any nice rim segment $s_i \in S$, there is a reflection vertex that lies on s_i. This guarantees that we will have to pick a point on each of the nice rim segments. We can sort the critical extreme lines according to their slope. After sorting the critical extreme lines, starting at any nice rim segment, we can choose a point (reflection vertex) on that nice rim segment, move to the next nice rim segment (using the sorted critical extreme lines [14]), and meanwhile, cross (or include) all the normal line segments in between. In this traversal, two nice rim segments are called *neighbours* if there is no nice rim segment between them.

Consider two neighbour nice rim segments NRS_i and NRS_{i+1}. We want to make

a path $P_{i,i+1}$ from a point on NRS_i to a point on NRS_{i+1} such that, $P_{i,i+1}$ includes at least one point of all the normal line segments that are located between NRS_i and NRS_{i+1} on its left side. $P_{i,i+1}$ is a *feasible* path between NRS_i and NRS_{i+1}. Using the list of sorted critical line segments, we can define a simple polygon between any two neighbour nice rim segments NRS_i and NRS_{i+1}, such that any path from NRS_i to NRS_{i+1} which is located inside that polygon is a feasible path [14]. We refer to these polygons as *feasible polygons*, since any feasible path between two neighbour nice rim segments will lie inside or on the boundary of these polygons.

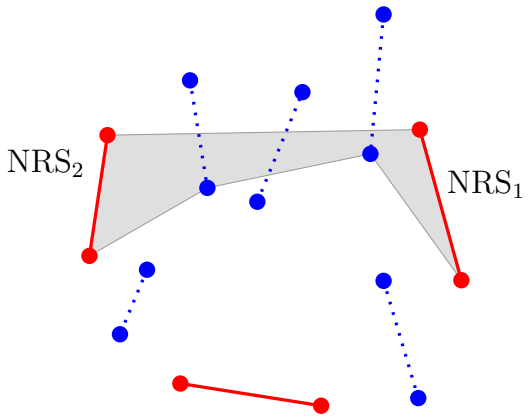

Fig. 3.4: The feasible polygon between NRS_1 and NRS_2

Fig. 3.4 shows the feasible polygon for two neighbour nice rim segments. It is obvious that any path from NRS_1 to NRS_2 will include at least one point of each normal line segment between NRS_1 and NRS_2 on its left side.

There is one unique feasible polygon between any two neighbour nice rim segments [14]. If we have k nice rim segments, we will have k feasible polygons. Each nice rim segment is common between two feasible polygons and thus the feasible polygons overlap. For a set of line segments S with k nice rim segments, if we move from one nice rim segment to its neighbour nice rim segment and move inside the feasible

polygons, after going through k feasible polygons, we will make a *feasible cycle*.

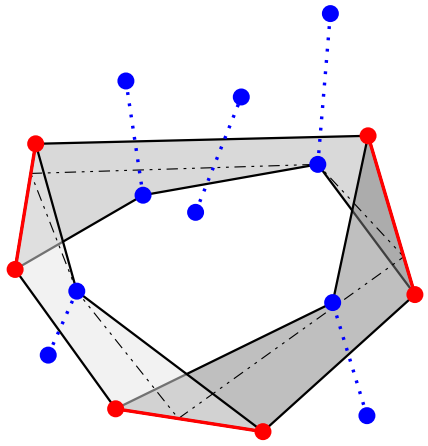

Fig. 3.5: A feasible cycle inside the feasible polygons

Fig. 3.5 shows the feasible polygons for a set of line segments. A possible feasible cycle is also shown as a dash-dotted cycle. Since this cycle is located inside the feasible polygons, it is guaranteed that the cycle includes at least one point of each line segment inside or on its boundary. This feasible cycle is a convex hull for the set of line segments S. For any set of feasible polygons, there exist an infinite number of feasible cycles and thus infinite number of convex hulls. Our final goal is to find a convex hull that has the minimum perimeter.

3.2.1 Flattening the feasible polygons

Any feasible cycle inside the feasible polygons is a convex hull of the set of points S. The smallest cycle, a cycle with the minimum length, is a minimum perimeter convex hull of S. The problem of finding the minimum cycle in the feasible polygons is convertible to a special case of the problem of finding the shortest path inside a polygon. We convert the overlapped polygons into a single polygon, since it is easier

to work with only one polygon.

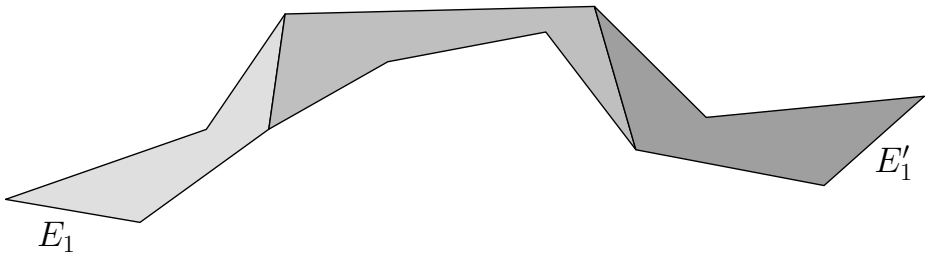

Fig. 3.6: Flattened feasible polygons

Consider two feasible polygons that have a common edge. We call them *neighbour* feasible polygons. The common edge is a nice rim segment. In Fig. 3.5 we can see the common edge between any two neighbour feasible polygons. Note that any two neighbour feasible polygons overlap. We define a *flip* operation as the separation of two neighbour feasible polygons, using their common edge as a hinge. Notice that the neighbour feasible polygons form a cycle. We break this cycle at one of the common edges. Given n feasible polygons, we can flip the neighbour polygons one by one and after $n-1$ flips we obtain one polygon. We call this procedure *flattening*. The result of flattening the feasible polygons in Fig. 3.5 is shown in Fig. 3.6.

The flattened polygon may or may not be simple, but this is of no consequence [14]. However, in our experiments, we observed that there exists at least one common edge, such that breaking the cycle at this edge, and subsequently flipping, produces a simple polygon.

In Fig. 3.6, edge E_1' is a duplicate of edge E_1, as a result of flipping the polygons. Originally, E_1 and E_1' were the common nice rim segment between the first and the last feasible polygon. It is safe to say that any cycle inside the feasible polygons corresponds to a path in the flattened polygon. So instead of finding a minimum perimeter cycle in feasible polygons FP, we can flatten FP into a simple polygon and

then find a shortest path in the flattened FP.

A shortest path in a polygon (Fig. 3.6) which corresponds to a minimum cycle inside feasible polygons (Fig. 3.5) starts at a point on edge E_1 and ends at a point on edge E_1'. Since E_1' is a duplicate of E_1, the end point on E_1' should also be a duplicate of the start point on E_1. So if a path starts at point X on edge E_1, it has to end at point X' on edge E_1', and X' is the point corresponding to X. This is a special case of finding the shortest path inside a polygon, since the starting edge and the ending edge (E_1 and E_1') always have the same length and the ending point (X') depends on the starting point (X). In Section 3.3.1 and Section 3.3.2, we demonstrate the required algorithms [4] to solve this special case of the shortest path problem inside a polygon.

Odd and even polygons

Any set of feasible polygons is convertible to a simple polygon. Depending on the number of the feasible polygons, the resulting simple polygon has a different property. We call a simple polygon P *odd*, if an odd number of feasible polygons were used to construct P. We call a simple polygon P *even*, if an even number of feasible polygons were used to construct P. We treat odd and even polygons differently, because finding the shortest path is slightly different for odd and even polygons.

Fig. 3.7 shows an example of odd and even polygons. Edge E_1 is the starting edge in order to find the shortest path in the polygon. Edge E_1' is the duplicate of edge E_1 and the shortest path ends at some point on edge E_1'. For any odd polygon, the direction of edge E_1 is the opposite of the direction of edge E_1'. For any even polygon the direction of edge E_1 is the same as the direction of edge E_1'. The arrows

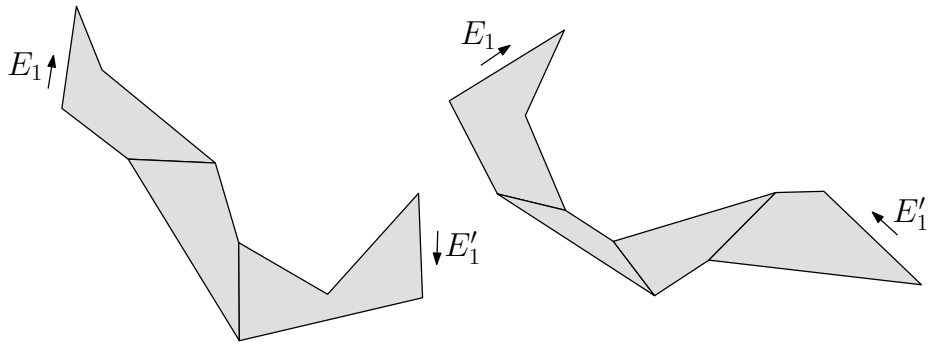

Fig. 3.7: Odd polygon (left) and even polygon (right)

in Fig. 3.7 show the direction of the edges E_1 and E_1'. For an odd polygon, E_1 and E_1' have *opposite correspondence*. If the polygon is an even polygon, E_1 and E_1' have *direct correspondence*.

3.3 Shortest path inside the flattened polygon

In Section 3.1, we explained the three different categories of line segments, nice rim segments, normal line segments and pesky rim segments. In Section 3.2, we introduced feasible polygons and their property. Then we showed that by flattening the feasible polygons, the problem of finding a minimum perimeter convex hull is equivalent to the problem of finding a shortest path inside the flattened feasible polygons. The flattened polygon has two edges, E_1 and E_1', that are identical. E_1 and E_1' are directed edges. We want to find the shortest path inside the polygon from E_1 to E_1', with the condition that the starting point X on E_1 is corresponding to the ending point X' on E_1'. Point X on E_1 is corresponding to point X' on E_1' if and only if the distance from the first end point of E_1 to X is equal to the distance from the first end point of E_1' to X'.

Some parts of E_1 and E_1' may be correspondingly visible and some parts may be correspondingly invisible. The two parts, P_1 and P_2, of two directed same-length edges, E_1 and E_1', are correspondingly visible (invisible), if for any point X on P_1, its corresponding point X' on P_2 is visible (invisible) to it. Given two directed corresponding edges in a simple polygon, we want to determine the correspondingly visible and the correspondingly invisible parts of the two edges. It is obvious that in a simple polygon, only concave vertices may block the visibility between two points inside the polygon. So in order to find the visibility of two corresponding edges, we should find the parts of the edges that are blocked by concave vertices. The parts that are not blocked are visible.

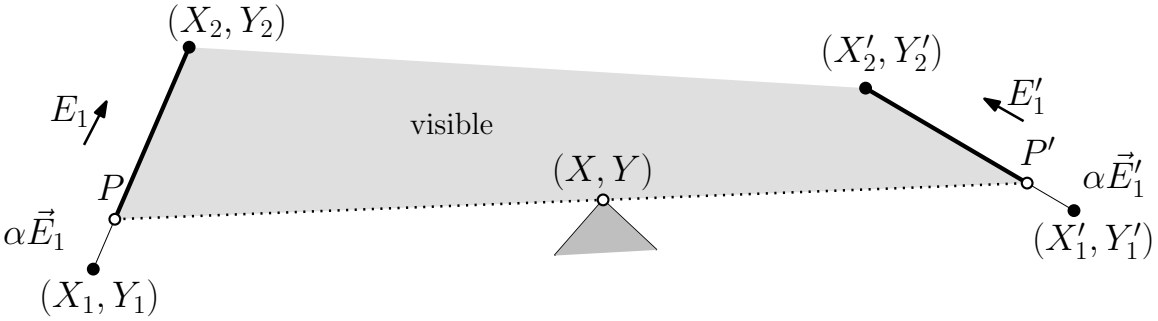

Fig. 3.8: Correspondingly visible parts

In Fig. 3.8, we want to find a line that goes through the concave vertex (X, Y) and intersects E_1 and E_1' such that $\alpha\vec{E_1}$ is equal to $\alpha\vec{E_1'}$. In other words, the line that goes through (X, Y) should intersect E_1 and E_1' such that the distance from the intersection point P on E_1 to the end point (X_1, Y_1) is equal to the distance from the intersection point P' on E_1' to the end point (X_1', Y_1').

Points P and P' are defined as:

$$P = (X_1 + \alpha(X_2 - X_1), Y_1 + \alpha(Y_2 - Y_1)), \quad 0 \leq \alpha \leq 1 \tag{3.1}$$

$$P' = (X_1' + \alpha(X_2' - X_1'), Y_1' + \alpha(Y_2' - Y_1')), \quad 0 \leq \alpha \leq 1 \tag{3.2}$$

The slope of the line that goes through P and (X, Y) is equal to the slope of the line that goes through P' and (X, Y). See Equation 3.3.

$$\frac{Y - Y_1 + \alpha(Y_2 - Y_1)}{X - X_1 + \alpha(X_2 - X_1)} = \frac{Y - Y_1' + \alpha(Y_2' - Y_1')}{X - X_1' + \alpha(X_2' - X_1')} \tag{3.3}$$

If we solve Equation 3.3 for α :

$$A\alpha^2 + B\alpha + C = 0 \qquad \text{where } A \text{ and } B \text{ and } C \text{ are:} \tag{3.4}$$

$$A = (Y_2 - Y_1)(X_2' - X_1') - (Y_2' - Y_1')(X_2 - X_1) \tag{3.5}$$

$$B = (X_2' - X_1')(Y - Y_1) + (Y_2 - Y_1)(X - X_1') - (X_2 - X_1)(Y - Y_1') - (Y_2' - Y_1')(X - X_1) \tag{3.6}$$

$$C = (Y - Y_1)(X - X_1') - (Y - Y_1')(X - X_1) \tag{3.7}$$

$$\Delta = B^2 - 4AC \tag{3.8}$$

In Equation 3.8, if $\Delta < 0$, there exists no valid value for α.

If $\Delta = 0$, it is possible that there exists a valid value for α.

$$\alpha = \frac{-B}{2A} \tag{3.9}$$

If $0 \leq \alpha \leq 1$, α is valid. This means that there exists a line that goes through vertex (X, Y) and intersects edges E_1 and E_1' such that the intersection points, P and P' are corresponding points (see Fig. 3.8).

If $\Delta > 0$, there exist two possibly valid values for α.

$$\alpha_1 = \frac{-B + \sqrt{\Delta}}{2A} \quad , \quad \alpha_2 = \frac{-B - \sqrt{\Delta}}{2A} \quad , \quad \phi = \{\alpha_1, \alpha_2\} \tag{3.10}$$

For any $\alpha \in \phi$ such that $0 \leq \alpha \leq 1$, we can calculate the intersection points P and P'.

$$P = (X_1 + \alpha(X_2 - X_1), Y_1 + \alpha(Y_2 - Y_1)) \tag{3.11}$$

$$P' = (X_1' + \alpha(X_2' - X_1'), Y_1' + \alpha(Y_2' - Y_1')) \tag{3.12}$$

In the case of opposite correspondence (odd polygons), α can have two valid values at the same time. In Fig. 3.9 there are two valid values for α at point (X, Y) which means there are two lines that go through (X, Y). In Fig. 3.9, segment $\overline{P_1P_2}$ is correspondingly visible to segment $\overline{P_1'P_2'}$.

In cases which there is more than one concave vertex that blocks the visibility, we determine all the invisible parts separately. The union of all the invisible parts determines the final invisible parts.

3.3.1 Shortest path for correspondingly visible parts

As mentioned before, two parts of edges E_1 and E_1' are correspondingly visible, if for any point X on the first part, its corresponding point X' on the second part is visible to it. An example has been shown in Fig. 3.10.

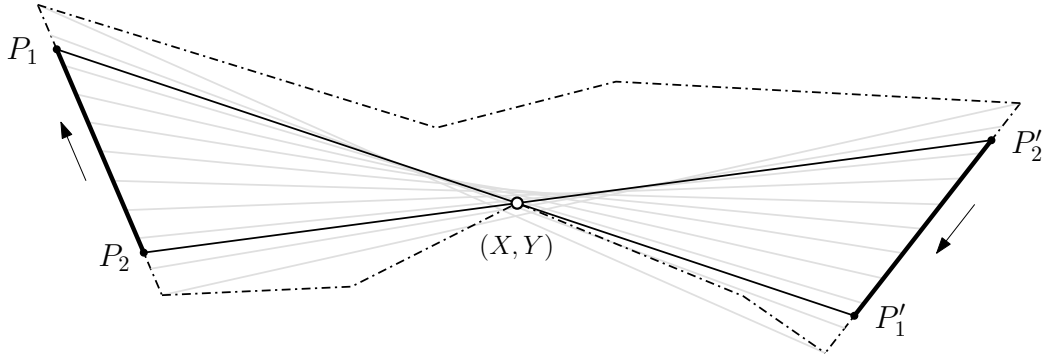

Fig. 3.9: Two valid values for α in an odd polygon

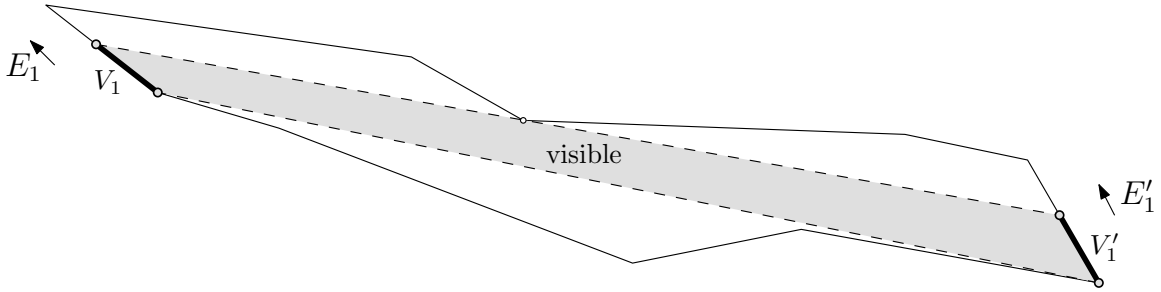

Fig. 3.10: Direct corresponding visibility

In Fig. 3.10 there are no (concave) vertices in the hatched area which means segment V_1 on edge E_1 is visible to its corresponding segment V'_1 on edge E'_1. The rest of the edge E_1 is invisible to its corresponding segment on E'_1. In Fig. 3.10 edges E_1 and E'_1 have direct correspondence, since the polygon in this figure is an even polygon. So V_1 and V'_1 have *direct corresponding visibility*. Fig. 3.11 shows an example of the *opposite corresponding visibility* in an odd polygon.

The approach to find the shortest path between two visible parts is the same for direct and opposite corresponding edges. Without loss of generality we compute the shortest path between direct corresponding visible parts of the two edges E_1 and E'_1 (see Fig. 3.10).

Fig. 3.12 shows the correspondingly visible parts of Fig. 3.10. Equation 3.13 and

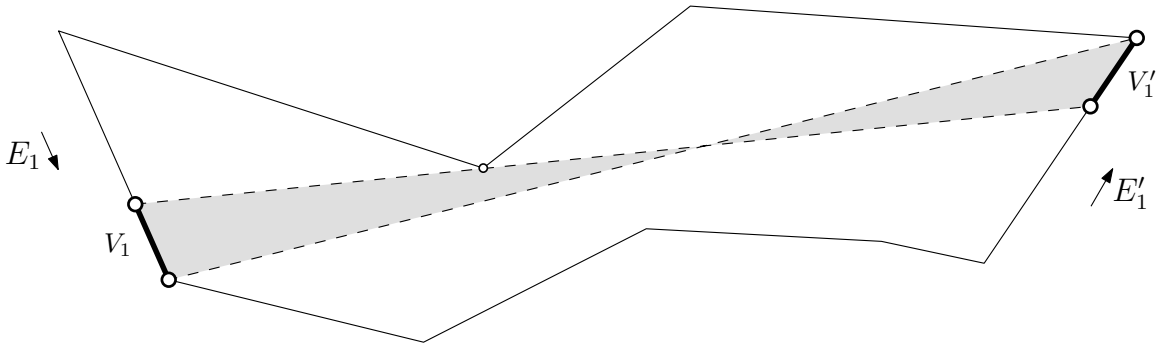

Fig. 3.11: Opposite corresponding visibility

Equation 3.14 formulate the points (X, Y) and (X', Y') respectively.

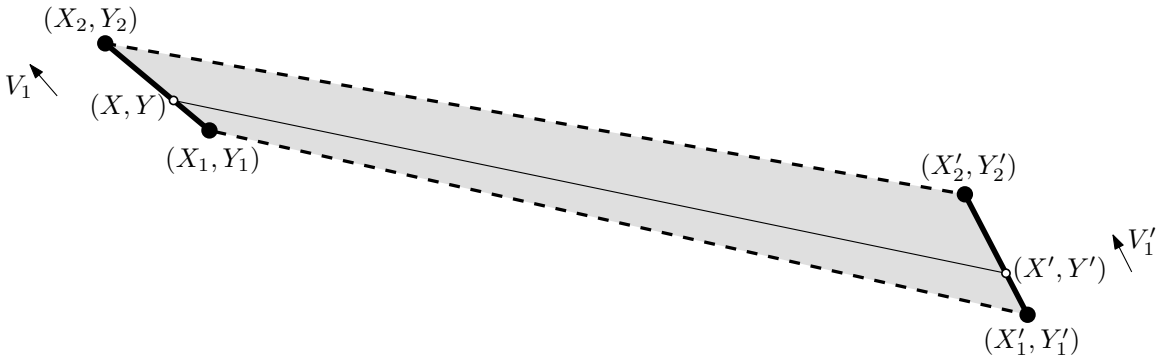

Fig. 3.12: Direct corresponding visible parts from Fig. 3.10

$$(X, Y) = (X_1 + c(X_2 - X_1), Y_1 + c(Y_2 - Y_1)), 0 \leq c \leq 1 \tag{3.13}$$

$$(X', Y') = (X_1' + c'(X_2' - X_1'), Y_1' + c'(Y_2' - Y_1')), 0 \leq c' \leq 1 \tag{3.14}$$

We also know that $c = c'$, since point (X', Y') is corresponding to point (X, Y).

The distance between (X, Y) and (X', Y') is

$$D = \sqrt{(X - X')^2 + (Y - Y')^2} \tag{3.15}$$

$$D = \sqrt{(X_1 - X_1' + c(-X_1 + X_2 + X_1' - X_2'))^2 + (Y_1 - Y_1' + c(-Y_1 + Y_2 + Y_1' - Y_2'))^2} \tag{3.16}$$

Calculate the first derivative of D, equate it to zero and solve for c:

$$c = -\frac{(X_1 - X1')(-X_1 + X_2 + X_1' - X_2') + (Y_1 - Y_1')(-Y_1 + Y_2 + Y_1' - Y_2')}{(-X_1 + X_2 + X_1' - X_2')^2 + (-Y_1 + Y_2 + Y_1' - Y_2')^2} \tag{3.17}$$

Using Equation 3.17 we can calculate c and using that, we can calculate (X, Y) and (X', Y'):

- If $c < 0$ then $(X, Y) = (X_1, Y_1)$ and $(X', Y') = (X_1', Y_1')$
- If $c > 1$ then $(X, Y) = (X_2, Y_2)$ and $(X', Y') = (X_2', Y_2')$
- If $0 \leq c \leq 1$ then use Equation 3.13 and Equation 3.14

So for any two correspondingly visible segments, V_1 and V_1', if we calculate (X, Y) and (X', Y') as explained above, then the line segment connecting (X, Y) and (X', Y') is the shortest line segment between any two corresponding points. For the case of opposite correspondence (odd polygons), we can use the same equations as above.

3.3.2 Shortest path for correspondingly invisible parts

If two parts of two directed edges are correspondingly invisible, there is no straight line that connects any two corresponding points on these invisible parts. This means that the line connecting a point X on edge E_1 to its corresponding point X' on edge E_1' has to go through one or more vertices of the polygon. Fig. 3.13 shows such an example. V_1 and V_1' are the correspondingly visible parts and the remaining parts of the edges E_1 and E_1' are correspondingly invisible.

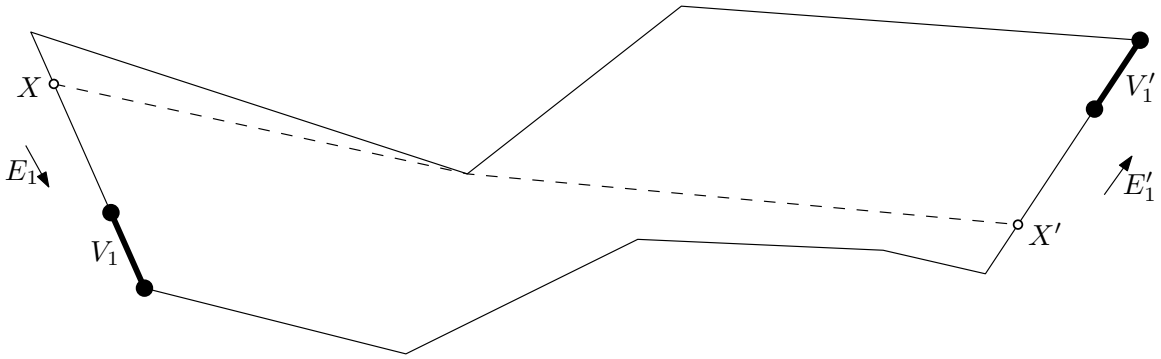

Fig. 3.13: Shortest path between two correspondingly invisible points

It is obvious that the shortest path between any two invisible points in a polygon always bends at the concave vertices. The proof simply follows from the fact that only concave vertices may block the visibility inside a simple polygon. So the shortest path from a part of edge E_1 to its correspondingly invisible part of edge E_1' has to go through some concave vertices. Since for any valid path from E_1 to E_1', the ending point X' is dependent on the starting point X (corresponding points), the problem is to find a start point X on E_1 such that the shortest path from X to its corresponding point X' on E_1' has the minimum length.

Subdividing the corresponding invisible parts

The shortest path from a point X on E_1 to its correspondingly invisible point X' on E_1' has to go through one or more concave vertices. The first vertex after the starting point X, on the shortest path, and the last vertex before the end point X', on the shortest path are called the *anchors* of X and X' [4]. Fig. 3.14 shows the anchors $\mathcal{A}_X$ and $\mathcal{A}_{X'}$ for point X and its corresponding point X'. We can find the anchor for any point on E_1 and E_1'.

Some points on E_1 have the same anchor. In fact, there is a part of edge E_1

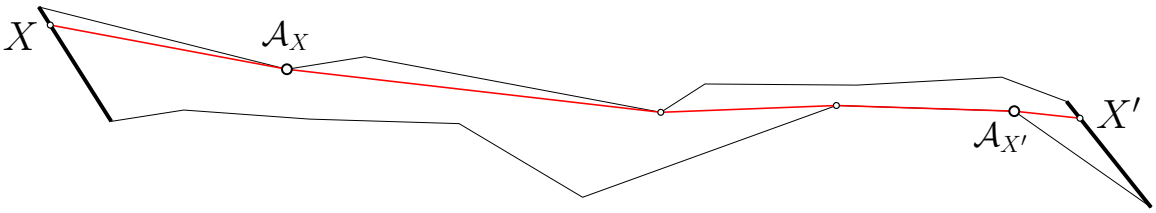

Fig. 3.14: Anchors of X and X'

such that all the points on that part have the same anchor. In this section, we will explain a method [4] to subdivide the edges E_1 and E'_1 such that all the points on each subdivision have the same anchor. We will also see how subdivision can help us in solving the problem of finding the shortest path between two corresponding points inside a polygon.

Consider the illustrative example shown in Fig. 3.15. In order to subdivide the edges E_1 and E'_1, first we find the shortest path between (X_1, Y_1) and (X'_1, Y'_1) and also the shortest path between (X_2, Y_2) and (X'_2, Y'_2). The polygon in Fig. 3.15 is an even polygon. For an odd polygon, we find the shortest path between (X_1, Y_1) and (X'_2, Y'_2) and also the shortest path between (X_2, Y_2) and (X'_1, Y'_1). Then we extend the edges of the two shortest paths, as well as the tangents between them (if any). For any of the extensions that intersects E_1 and/or E'_1, we keep the intersection point. After finding all the intersection points, we also find the corresponding points to those intersection points. This results in an equal number of intersection points on E_1 and E'_1.

Every two consecutive intersection points define a subdivision. There are six subdivisions in Fig. 3.15. The shortest path between (X_1, Y_1) and (X'_1, Y'_1) and the shortest path between (X_2, Y_2) and (X'_2, Y'_2) is shown in bold. There are three dotted lines that are the extensions of the shortest paths. There is also one tangent line

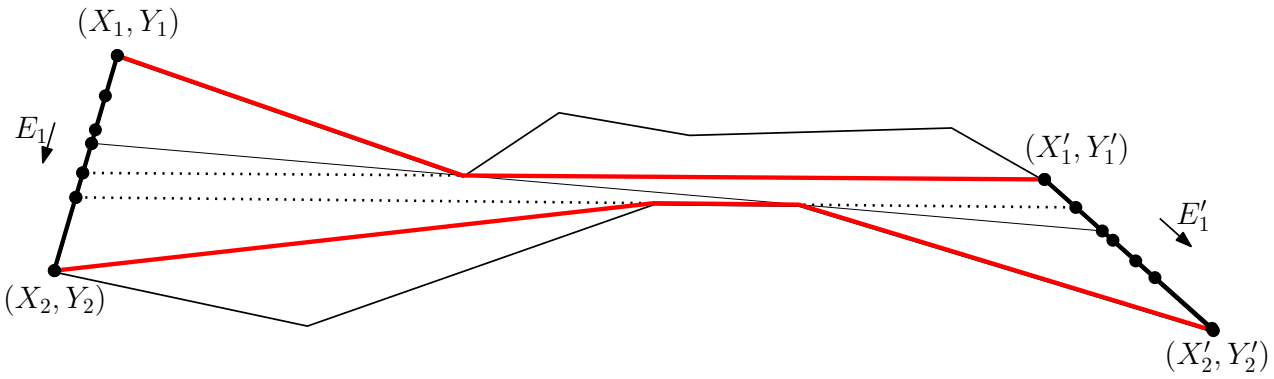

Fig. 3.15: Subdivisions of E_1 and E_1' in an even polygon

between the two shortest paths, intersecting E_1 and E_1'. The points on both edges E_1 and E_1' are either an intersection point or a point corresponding to an intersection point. As mentioned before, all the points on each subdivisions have the same anchor. For any subdivision $\mathcal{S}_i$, we can find a point x_i on $\mathcal{S}_i$ such that the length of the shortest path from x_i to its corresponding point x_i' on $\mathcal{S}_i'$ is minimum [4].

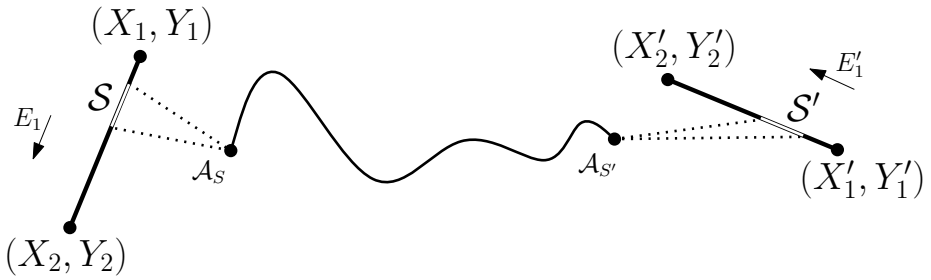

Fig. 3.16: Subdivision $\mathcal{S}$ and $\mathcal{S}'$ and their anchors

In Fig. 3.16, the shortest path from any point x on subdivision $\mathcal{S}$ to its corresponding point x' on subdivision $\mathcal{S}'$ goes through anchor $\mathcal{A}_S$ and anchor $\mathcal{A}_{S'}$. We want to find a point x on $\mathcal{S}$ such that the shortest path from x to x' has minimum length. The shortest path between $\mathcal{A}_S$ and $\mathcal{A}_{S'}$ is a fixed path. So in order to minimize the

length of the total path, we should minimize the following equation:

$$Distance(x, \mathcal{A}_S) + Distance(x', \mathcal{A}_{S'}) \tag{3.18}$$

In order to find point x, reflect, if necessary, and then translate and rotate the triangle defined by $\mathcal{S}'$ and $\mathcal{A}_{S'}$ until $\mathcal{S}'$ is aligned with $\mathcal{S}$ and $\mathcal{A}_S$ and $\mathcal{A}_{S'}$ are on the opposite sides of $\mathcal{S}$ (Fig. 3.17) [4]. The shortest path from $\mathcal{A}_S$ to $\mathcal{A}_{S'}$ intersects subdivision $\mathcal{S}$ (and $\mathcal{S}'$) at point x (and x'). This point can either be one of the end points of subdivision $\mathcal{S}$ (Fig. 3.17(A)) or a point in the interior of subdivision $\mathcal{S}$ (Fig. 3.17(B)) [4]. In either case we denote the intersection point on $\mathcal{S}$ (and $\mathcal{S}'$) as ip.

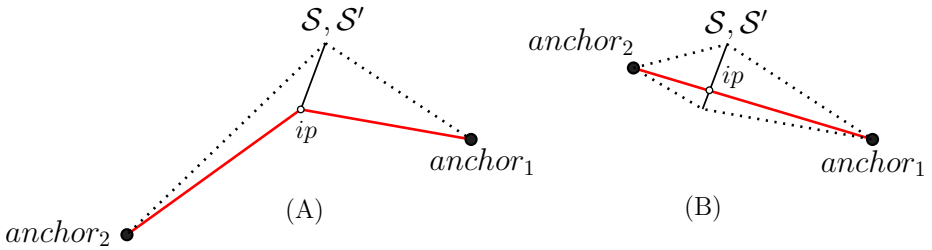

Fig. 3.17: The shortest path between two anchors

For all subdivision $\mathcal{S}_i$, the shortest path from ip_i to ip'_i has the minimum length. There are at most a linear number of subdivisions. In constant time, we can compute the shortest paths for each subdivision $\mathcal{S}_i$, using ip_i [2]. The shortest path with the minimum length among all the subdivisions is the shortest path between two corresponding point inside polygon P.

After finding the shortest path inside the flattened polygon P, by reversing the flattening procedure done in Section 3.2.1, we can fold back polygon P as well as the minimum length shortest path inside it. The flattened polygon P will transform into the overlapping feasible polygons and the minimum length shortest path will

transform into a cycle inside the feasible polygons. This cycle has the minimum length among all the feasible cycles inside the feasible polygons, thus the minimum cycle is a minimum perimeter convex hull of the set of line segments S.

3.3.3 Shortest path with pesky rim segments

What we talked about so far (finding the feasible polygons, flattening them into a single polygon and finding the shortest path inside the flattened polygon) only works for the cases in which no pesky rim segments exist in the set of line segments S. In this section, we will explain how a pesky rim segment p^* can be resolved, i.e. replaced by 4 different candidates, such that each candidate is not a pesky rim segment and at the same time, the union of these candidates is equivalent to p^*. More generally, we will explain how a set of line segments that includes k pesky rim segments is convertible to 4^k sets of line segments such that in each of the 4^k sets, each of the k pesky rim segments is replaced by 1 of its 4 candidates and none of the 4^k sets includes any pesky rim segments [14]. Then using the MPCH algorithm introduced in the previous sections for the set of line segments that do not include any pesky rim segments, we can solve each of the 4^k sets of line segments individually.

Converting the pesky rim segments

Recall that in a set of line segments S, for every pesky rim segment, there is at least one point of each line segment $s_i \in S$ on the left side of the pesky rim segment and there is at least one critical point which is located on the right side of the pesky rim segment. According to this property of pesky rim segments, we can convert a pesky rim segment into a non-pesky segment in different ways, i.e. we can replace a pesky

rim segment with one of its candidates.

There are cases in which a pesky rim segment p^* is included in the minimum perimeter convex hull, even if we overlook the existence of p^*. Fig. 3.18(A) shows a set of line segments in which p^* is a pesky rim segment. In Fig. 3.18(B), p^* is removed from the set of line segments and the minimum perimeter convex hull of the new set is computed. In Fig. 3.18(C), p^* is added back to the set of line segments. The convex hull in Fig. 3.18(C) includes p^*. This means that p^* doesn't affect the minimum perimeter convex hull of the set of line segments shown in Fig. 3.18(A). So the pesky rim segment p^* can be ignored.

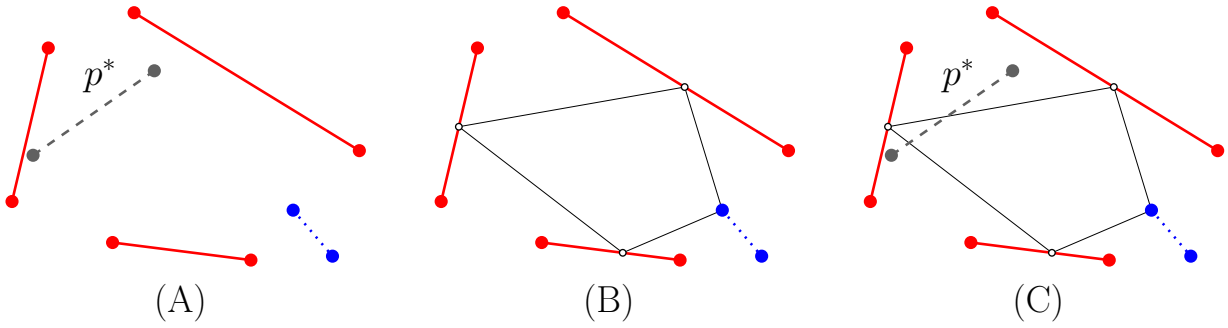

Fig. 3.18: Case 1: Ignoring a pesky rim segment (valid)

But ignoring a pesky rim segment is not always a possible choice. There are cases in which by ignoring p^* and calculating the minimum perimeter convex hull, p^* will not be included in the minimum perimeter convex hull. In cases like Fig. 3.19, the minimum perimeter convex hull is not valid, since p^* is not included in the convex hull.

Removing a pesky rim segment p^* and calculating the minimum perimeter convex hull of the rest of the line segments is the best way to resolve p^*, if the minimum perimeter convex hull includes p^* (Fig. 3.18). If the convex hull includes p^*, we do not

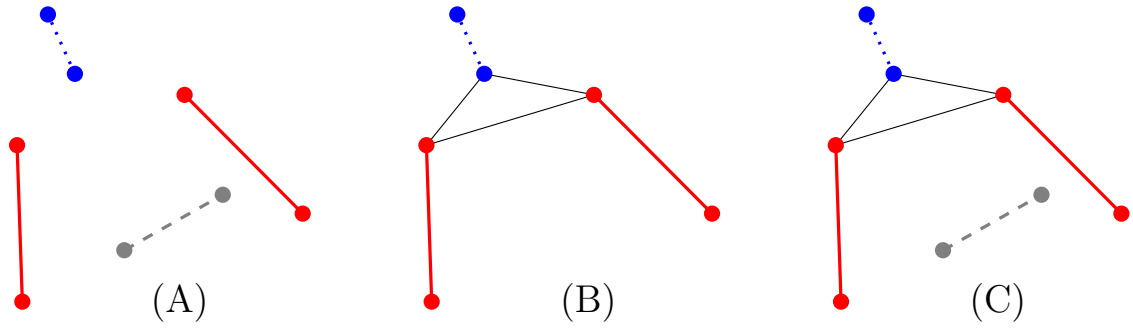

Fig. 3.19: Case 1: Ignoring a pesky rim segment (not valid)

need to check the other candidates. Otherwise (Fig. 3.19) we should check the other candidates in order to resolve p^*. Case 1 refers to checking to see whether ignoring p^* is a valid option or not.

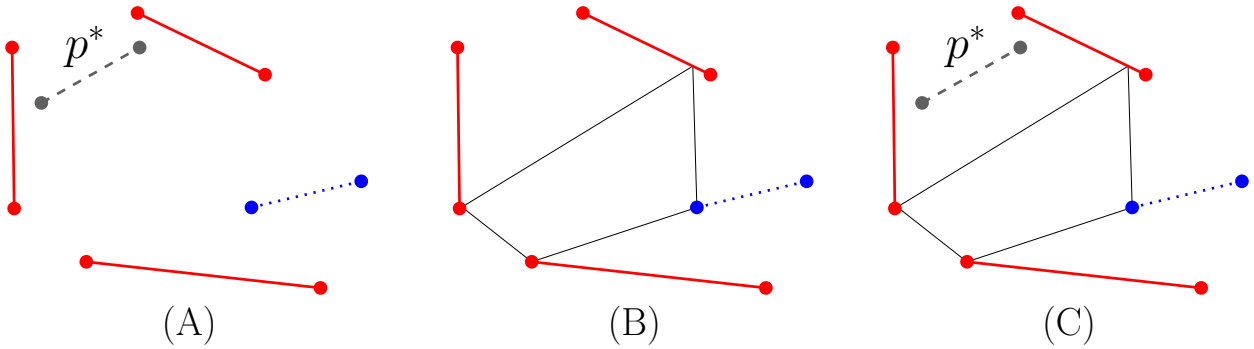

Fig. 3.20: An example where case 1 in not valid

Consider the pesky rim segment $p^* \in S$ in Fig. 3.20(A). Case 1 is not valid for the line segments in this figure. So we need to check the other candidates in order to resolve p^*.

According to the definition of pesky rim segments, it is possible that the minimum perimeter convex hull is located completely on the left side of p^* (case 2). In this case, a pesky rim segment p^* can be converted into a nice rim segment by drawing a line l which is collinear to p^* and then removing everything that is located on the

right side of line l [14]. On the left side of l, there is at least on point of each line segment $s_i \in S$. The result of *trimming* the line segments in Fig. 3.20 is shown in Fig. 3.21. The pesky rim segment p^* is converted into a nice rim segment in the new figure.

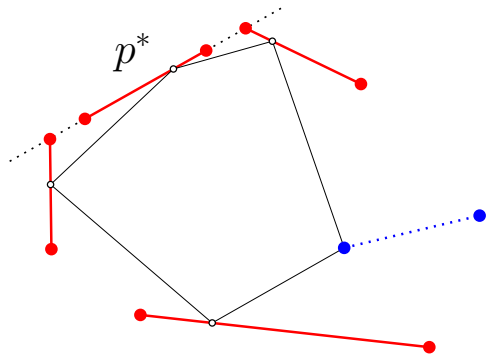

Fig. 3.21: Case 2: Converting a pesky rim segment into a nice rim segment

It is also possible that the minimum perimeter convex hull is not completely located on the left side of a pesky rim segment, i.e. the convex hull has to cross the pesky rim segment. In this scenario, the pesky rim segment plays the role of a normal line segment. When the convex hull crosses p^*, it has to go though one of the end points of p^*. In this case, in order to resolve p^*, we replace p^* with one of its end points. Since each pesky rim segment has two end points, we can consider each of the end points to be a candidate to replace p^*.

Fig. 3.22 demonstrates case 3 in which the convex hull crosses the pesky rim segment p^* such that the first end point of p^* is located on the boundary of the convex hull.

Fig. 3.23 demonstrates case 4 in which the convex hull crosses the pesky rim segment p^* such that the second end point of p^* is located on the boundary of the convex hull.

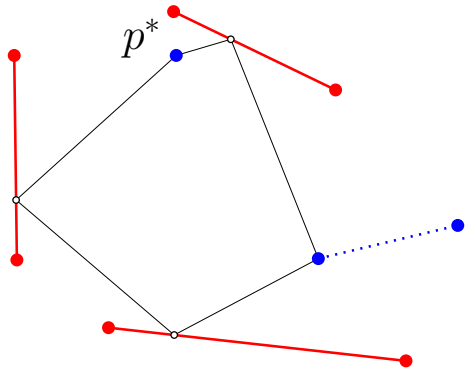

Fig. 3.22: Case 3: Replacing p^* with its first end point

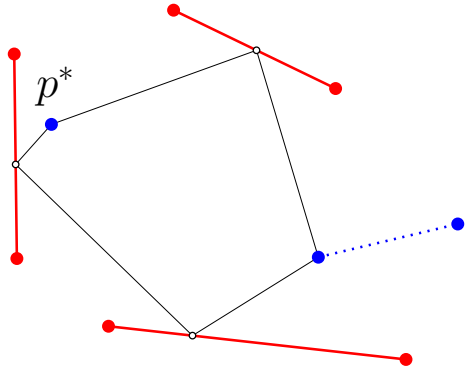

Fig. 3.23: Case 4: Replacing p^* with its second end point

So there are four different candidates that may help in resolving a pesky rim segment p^* such that the minimum perimeter convex hull of the set of line segments S can be computed. The PeskyHandler subroutine will perform the task of resolving a pesky rim segment:

PeskyHandler subroutine

1. Ignore p^* and find the MPCH of $S - \{p^*\}$ (case 1). Then check whether p^* is included in the MPCH of $S - \{p^*\}$. If not included then check the below candidates.

2. Convert p^* into a nice rim segment (case 2) and find the MPCH of S.

3. Replace p^* with its first end point (case 3) and find the MPCH of S.

4. Replace p^* with its second end point (case 4) and find the MPCH of S.

The smallest MPCH among the above is selected as the MPCH of S. Computing a MPCH of a set of line segments that include no pesky rim segments takes $O(n \log n)$ time as described throughout this chapter. At each of the steps in the PeskyHandler subroutine, we compute a MPCH in $O(n \log n)$. The preparation time before computing a MPCH at each step is explained here:
At Step 1, the computational complexity to check to see whether the MPCH intersects p^* is $O(n)$. At Step 2, the computational complexity to trim off the parts of the line segments which are located on the right side of the line that goes through p^* is $O(n)$. Replacing p^* with its end points in Step 3 and Step 4 are trivial and can be done in $O(1)$. The PeskyHandler routine works in $O(n \log n)$ time, dominated by the time to compute the MPCH at each step.

Our Extension Case 1 is not originally proposed in [14]. The original algorithm which is proposed in [14] only considers case 2, case 3 and case 4. During some experiments, we noticed that if we do not consider case 1, we will not get the optimum result. Fig. 3.24 shows an example where the original algorithm (without case 1) would fail. The line segments in Fig. 3.24 are the same as the line segments in Fig. 3.18. The optimum answer (considering cases 1,2,3,4) and the sub-optimum answer produced by the original algorithm [14] (cases 2,3,4) are shown in Fig. 3.24. The optimum answer is shown as the solid convex hull and the sub-optimum answer is shown as the dotted convex hull. The original algorithm impose the condition that either a pesky rim segment p^* should be converted to a nice rim segment (case 2) or

the MPCH should go through one of the end points of p^*. As we explained in case 1 (our extension), there are cases where the MPCH will go through p^*, but it will not go through an end points of p^*, as shown in Fig. 3.24.

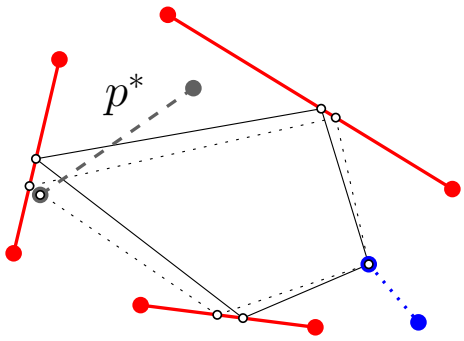

Fig. 3.24: The missed case

So far we talked about sets of line segments which only include one pesky rim segment. Consider a set of line segments S that includes k pesky rim segments $\{p_1, p_2, ..., p_k\}$. There are 4 candidates to resolve p_1, and considering each of the candidates for p_1, there are 4 candidates to resolve p_2 and so on. So in order to resolve all k pesky rim segments in S, there is a total of 4^k combination of candidates in S to be investigated. Recall that there exists no pesky rim segments in the combinations and it takes $O(n \log n)$ time to solve each of the combinations. So given a set of line segments S that includes n line segments such that k line segments in the set are pesky rim segments, it takes $O(4^k n \log n)$ time to find the minimum perimeter convex hull of S.

Before moving on to the next section, let's review what we have done so far. We were given a set of line segments S. We categorized the line segments in S and then we calculated the feasible polygons for S. We explained how to flatten the feasible polygons. Then we demonstrated a method to find a shortest path $\mathcal{P}$ inside

the flattened polygon, such that the starting point and the ending point of $\mathcal{P}$ are corresponding points. If we fold back the flattened polygon as well as $\mathcal{P}$, the folded shortest path $\mathcal{P}$, now a cycle, will correspond to a minimum perimeter convex hull of S. The above procedures can be done in $O(n \log n)$ time [14]. Then we studied the effect of the pesky rim segments on the problem and we showed that it takes $O(4^k n \log n)$ time to find a minimum perimeter convex hull of a set of line segments that includes k pesky rim segments.

Chapter 4

Results: The Approximation Approaches

4.1 Empirical Results

According to the MPCH algorithm described in the previous sections, for a set S including n line segments, in the worst case, it takes $O(4^n n \log n)$ time to construct a minimum perimeter convex hull of S. Fig. 4.1 shows an example when all the line segments are pesky rim segments.

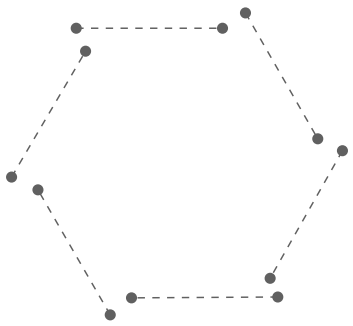

Fig. 4.1: A worst case example with 6 pesky rim segments

An exponential time algorithm takes an intractable amount of time to solve big instances of the MPCH problem. Table 1 shows the result of the experiments for different number of pesky rim segments. In all the experiments, the line segments were positioned similar to the example shown in Fig. 4.1. In order to run these experiments, the MPCH algorithm was coded in Microsoft Visual Studio C# .NET 2005 and the experiments were run on an AMD Athlon 64×2 machine with 1.17×2 GHz CPU and 1 GB of RAM. According to the results of the experiments in Table 1, the algorithm runs for about 50 minutes to solve a problem with 13 pesky rim segments. Obviously for $n > 13$ the running time increases drastically.

n	Running time (seconds)
3	0.015
4	0.031
5	0.156
6	0.620
7	2.034
8	9.484
9	19.015
10	103.472
11	269.595
12	806.234
13	2956.640

Table 4.1: Running time of the MPCH algorithm for n pesky rim segments

If there are no pesky rim segments in the set of line segments S, the problem can be solved in $O(n \log n)$ time [14]. The pesky rim segments force us to check different combinations of the line segments in order to solve the problem. For k pesky rim segments, we should run the basic $O(n \log n)$ algorithm 4^k times which leads to an $O(4^k n \log n)$ time algorithm. This motivates us to approximate the minimum perimeter convex hull, such that a convex hull of S with reasonably small perimeter

can be constructed in polynomial time.

4.2 The FIFO Approach

In this section we introduce a heuristic function that instead of investigating all the 4^k different combinations of the candidates for k pesky rim segments, only investigates a linear number of combinations. Here is a pseudo code of this heuristic function:

1. $S \leftarrow$ set of line segment , $Q \leftarrow \emptyset$
2. Remove the pesky rim segments one by one from S and add them to Q.
3. Remove a pesky rim segment p^* from Q and add it to S.
4. Find the MPCH of S by resolving p^*. This is done using the PeskyHandler subroutine described in Section 3.3.3 in $O(n \log n)$ time.
5. Replace p^* in S with a candidate that results in the best answer.
6. If $Q \neq \emptyset$ go to step 3.
7. At this step, we have the final approximation.

In the above pseudo code, at step 2, we randomly remove one of the pesky rim segments from the set of line segments S. Removing a pesky rim segment from S may affect the other line segments, i.e. a non-pesky segment may convert into a pesky rim segment or a pesky rim segment may convert into a non-pesky segment. In Fig. 4.2, there are two nice rim segments, r_1 and r_2, three pesky rim segments, p_1, p_2 and p_3 and there are two normal line segments n_1 and n_2. If we remove p_1 first, the

rest of the segments do not change. But if we remove p_2 first, p_1 becomes a nice rim segment. If we remove p_3 first, n_2 becomes a pesky rim segment. We call this the *removal effect*.

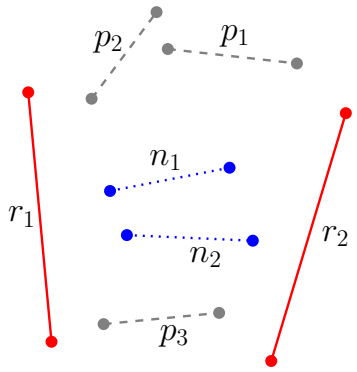

Fig. 4.2: The removal effect as a result of removing p_2 or p_3

At step 2 of the pseudo code above, we remove pesky rim segments one by one, considering the removal effect, until no pesky rim segments are left. All the removed pesky rim segments are stored in a queue Q. At step 3, we remove the pesky rim segment that is located in the front of Q. We call this pesky rim segment p^*. We add p^* back to the set of line segments S. At step 4, S includes one pesky rim segment, p^*. We investigate the four candidates of p^* in order to resolve p^*. This step is done in the PeskyHandler subroutine. At step 5, p^* is replaced with the candidate that results in the smallest minimum perimeter convex hull. Then we go back to step 3 and continue the procedure until all the pesky rim segment are added back to S. At step 7, after adding and resolving all the pesky rim segments, we have the final approximation. The minimum perimeter convex hull of S at the end of step 7 is an approximated convex hull of the original set of line segments, because S includes one combination out of a total of 4^k possible combinations of candidates for k pesky rim segments, and in order to find the optimum solution, all the 4^k combinations should

be investigated. k is the number of pesky rim segments in Q at the end of step 2. Section 4.2.2 explains the computational complexity of the FIFO approach.

Since in this approach the pesky rim segments are added back in the order that they were initially removed from S, we call it the *FIFO approach*.

4.2.1 Approximation Bounds

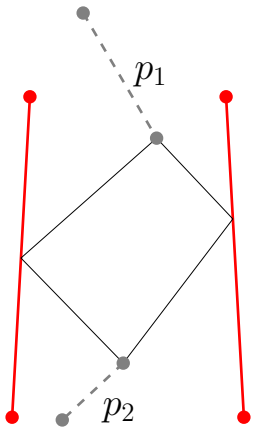

Fig. 4.3: A good FIFO approximation

The precision of the FIFO approximation varies depending on the input set of line segments S and the order in which the pesky rim segments are processed. For example in Fig. 4.3, the approximated convex hull of the line segments is the same as the minimum perimeter convex hull, which means the FIFO approach is capable of producing good results. But we can easily find examples in which the FIFO approach results in arbitrarily bad approximations. Consider the line segments in Fig. 4.4. The minimum perimeter convex hull of the line segments is also shown in this figure. But if we apply the FIFO approach, the result is a poor approximation of the optimal solution.

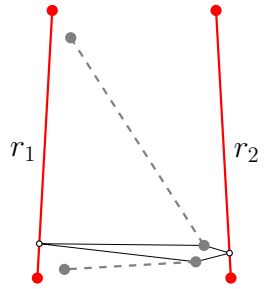

Fig. 4.4: A set of line segments and its MPCH

Fig. 4.5(A) shows the initial set of line segments S. Fig. 4.5(B) shows the line segments after removing the pesky rim segments p_1 and p_2. We assume that p_1 has been removed first, so in Fig. 4.5(C), we add p_1 first and we compute the minimum perimeter convex hull. Replacing p_1 with its first end point results in the best answer, so we replace p_1 with its first end point. Then we add p_2 back to S. We compute the convex hull and the best result is shown in Fig. 4.5(D). Recall that in the FIFO approach, after removing all the pesky rim segments from S, the pesky rim segments are added back to S one at a time, and after resolving each pesky rim segment, we replace that pesky rim segment with its best candidate.

In the example shown in Fig. 4.5(A), line segments r_1, r_2 and p_1 can be arbitrarily long. The perimeter of the approximated convex hull in Fig. 4.5(D) increases proportionally to the length of p_1. This means that the perimeter of the approximated convex hull can be arbitrarily large. Note that the perimeter of the minimum perimeter convex hull in Fig. 4.4 is proportional to the distance between r_1 and r_2. For the line segments in Fig. 4.6, the perimeter of the convex hull approximated by the FIFO approach is about 9 times greater than the perimeter of the optimal solution. We can make this ratio arbitrarily large.

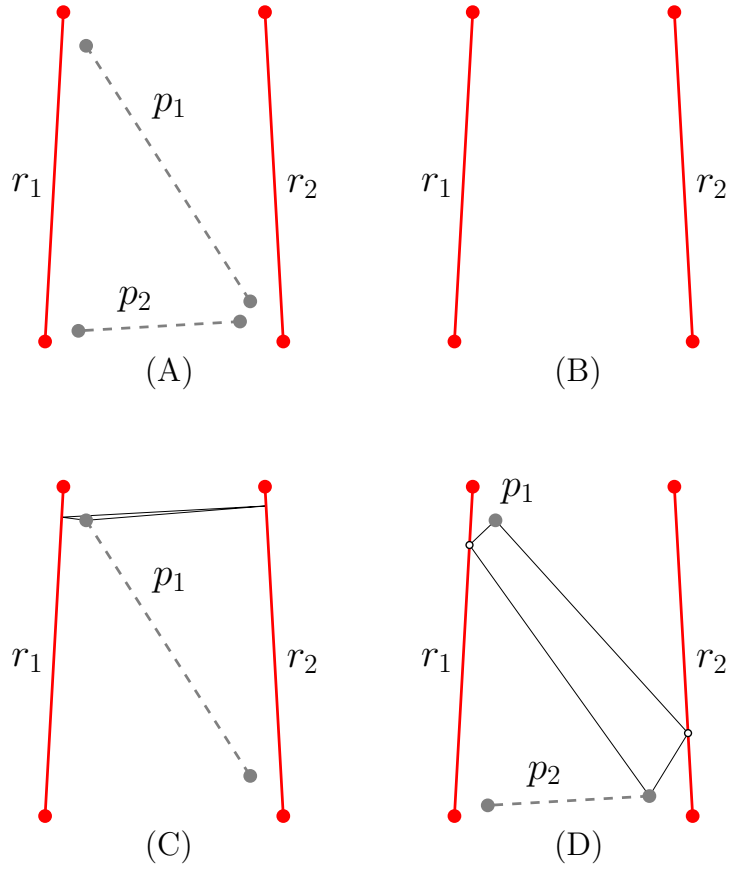

Fig. 4.5: An example where the FIFO approach fails

4.2.2 Computational complexity

In the worst case of the problem, all the line segments in S are pesky rim segments. In each iteration of the heuristic, we add a pesky rim segment to S and then we solve the problem in $O(n \log n)$ time, using the PeskyHandler subroutine described in 3.3.3. In the worst case, the algorithm iterates n times, so the computational complexity of the FIFO approach is $O(n^2 \log n)$.

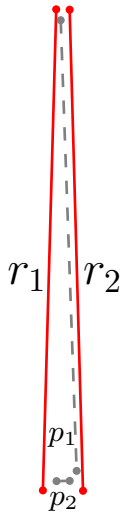

Fig. 4.6: Approximated perimeter $\simeq 9\times$minimum perimeter

4.3 Minimum Spanning Circle Approach

In this section, we propose a completely different approach to approximate the minimum perimeter convex hull of a set of line segments S. In the FIFO approach, we reduced the computational time by avoiding to investigate all the 4^k different combinations of k pesky rim segments in S. In the new approach, we find a part of the X-Y plane that covers all the line segments in S. In other words, this region of the plane includes at least one point of each line segment in S. Knowing the perimeter of this region, we will try to bound the perimeter of the minimum perimeter convex hull. We consider this region to be a circle. So the new approach finds a Minimum Spanning Circle (MSC) of S to approximate the minimum perimeter convex hull. An MSC of a set of line segments is the smallest circle that intersects all the line segments in that set. In Section 4.3.1, we talk about farthest line segment Voronoi diagrams in order to find the MSC of S. The MSC of S can be found in $O(n \log n)$ time [1]. We will prove that using this approach, the result of the approximation is at most $\frac{\pi}{2}$

times greater than the minimum answer.

4.3.1 Voronoi Diagrams

Given a set of points SP $= \{p_1, p_2, ..., p_n\}$, we want to partition the plane into regions around each point p_i, such that all the points in the region around p_i are closer to p_i than any other point in SP. This is called the 1^{st} *order Voronoi diagram.*

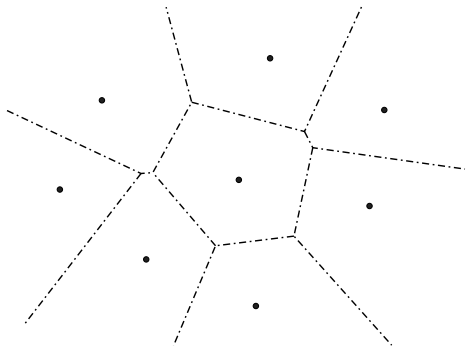

Fig. 4.7: 1^{st} order Voronoi diagram

Fig. 4.7 shows an example of a 1^{st} order Voronoi diagram. In 2^{nd} order Voronoi diagrams, we want to partition the space into regions around the points, such that all the points in the region around p_i and p_j, $i \neq j$, are closer to p_i and p_j than any other point in SP. In $(n-1)^{th}$ order Voronoi diagrams, we want to partition the space into regions, such that all the points in one region are closer to the points in SP $- \{p_i\}$ than point p_i. The $(n-1)^{th}$ order Voronoi diagrams are also called farthest point Voronoi diagrams, because there are regions in the plane that are farthest from point p_i compared to all the other points $p_j \in SP$, $i \neq j$. In Fig. 4.8 we can see the farthest Voronoi diagram of 4 points. For instance all the points in region 2 are farthest from point p_2.

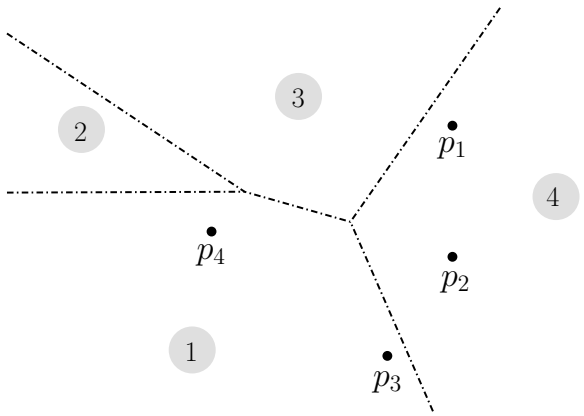

Fig. 4.8: $(n-1)^{th}$ order Voronoi diagram, a.k.a. farthest point Voronoi diagram

We can use similar definitions of Voronoi diagrams for a set of line segments. In order to find the MSC of a set of line segments S, we are interested in finding the farthest line segment Voronoi diagram of S. We can find the farthest line segment Voronoi diagram of S in $O(n \log n)$ time [1]. The algorithm introduced in [1] also works for the cases in which the line segments overlap. Fig. 4.9 shows an example of the farthest Voronoi diagram for six line segments. In this figure we have six different regions. Line segment s_2 has no region, which means there is no point on the plane that is farther from line segment s_2 than any other line segment in S. Some line segments like s_4 can have more than one region.

Each Voronoi diagram consists of some regions, edges and vertices. An edge is the intersection of two regions R_i and R_j, so the points on edge E_{ij} have the properties of both R_i and R_j. A vertex in a Voronoi diagram is the intersection point of three regions R_i, R_j and R_k. So a Voronoi vertex V_{ijk} has the properties of regions R_i, R_j and R_k. For example, in the farthest line segment Voronoi diagram, region R_i is farthest from line segment s_i than any other line segment in S. Vertex V_{ijk} at the intersection point of regions R_i, R_j and R_k, has the same distance from line segments

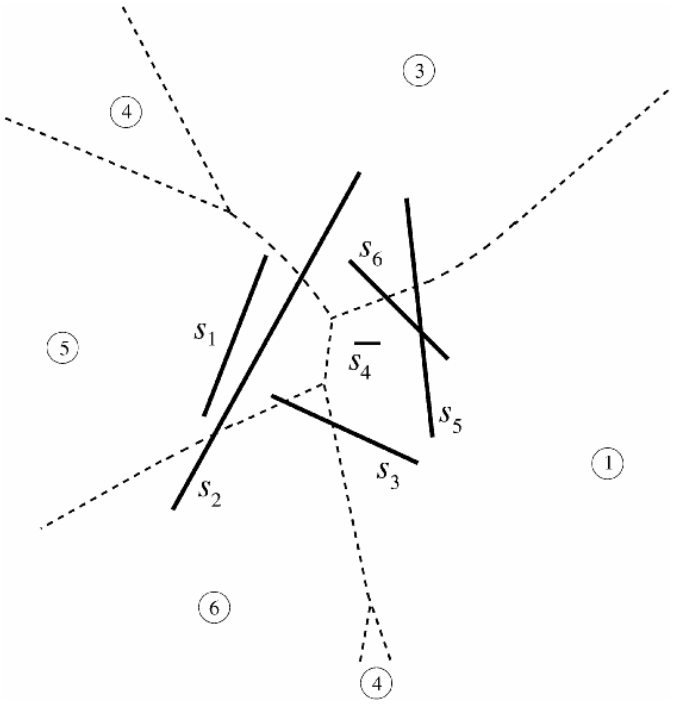

Fig. 4.9: Farthest line segment Voronoi diagram (illustration obtained from [1])

s_i, s_j and s_k, and this distance is more than the distance from V_{ijk} to any other line segment s_t, $t \neq i, j, k$.

4.3.2 Minimum Spanning Circle

Given a set of line segments S, a *Minimum Spanning Circle* (MSC) of S is the smallest circle that includes at least one point of each line segment $s_i \in S$ inside or on its boundary.

Theorem 4.3.1 *A Minimum Spanning Circle of a set of line segments S is defined by two or three defining points on its boundary.*

Proof In the first step, start with a circle with centre c_1 in the plane that completely includes all the line segments inside itself. Without moving the centre c_1, reduce

the radius of the circle until further reduction of the radius results in missing a line segment. In Fig. 4.10(A), the initial (dotted) circle and the reduced circle are shown. Line segment s_1 is tangent to the reduced circle and s_1 will be missed if the reduction continues. x_1 is the intersection point of s_1 with the current circle.

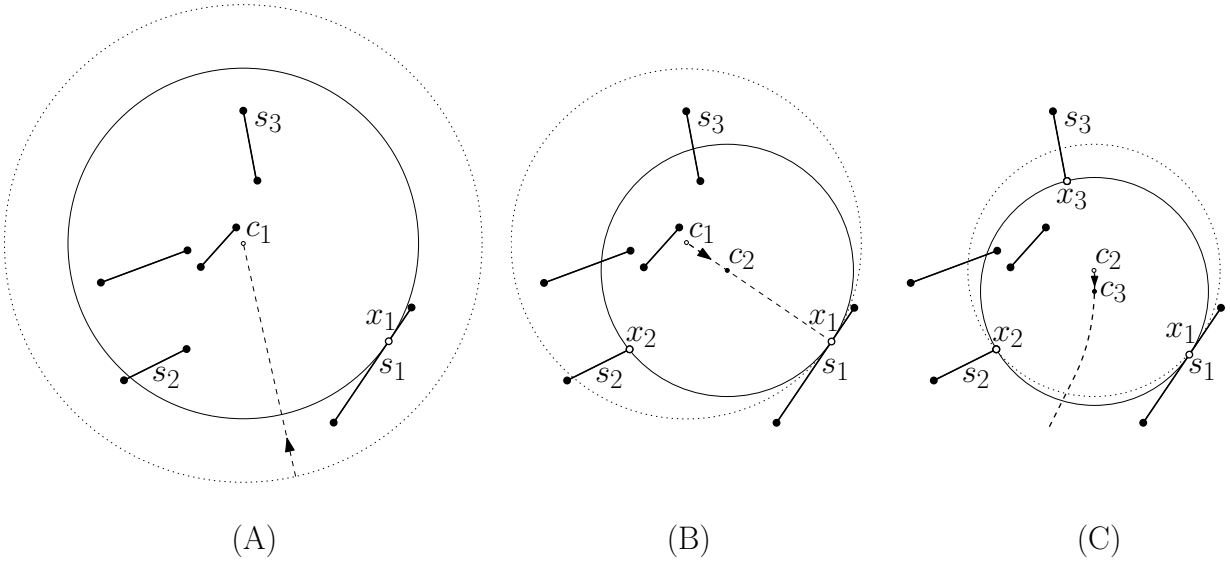

Fig. 4.10: Three steps to find an MSC

In the second step, reduce the circle by moving c_1 towards x_1 until further movement of c_1 results in missing a line segment. In Fig. 4.10(B), point c_2 is the centre of a smaller circle and further movement of c_2 towards x_1 results in missing line segment s_2. x_2 is the intersection point of s_2 with the current circle. In the last step, find a curve such that for any point p on the curve, the distance between p and s_1 is equal to the distance between p and s_2. In the general case, this curve consists of lines and parabolas. Then move c_2 on the curve towards s_1 and s_2 until further movement of c_2 results in missing another line segment (3 defining points) or until $\overline{x_1x_2}$ becomes the diameter of the circle (2 defining points). Note that while we are moving c_2 towards

s_1 and s_2, the intersection points x_1 and x_2 may move on the line segments s_1 and s_2. In Fig. 4.10(C), line segment s_3 will be missed if c_2 goes beyond c_3. The circle with centre c_3 is defined by the three points x_1, x_2 and x_3. This circle is the MSC defined by s_1, s_2 and s_3. Any movement of the centre or reduction of the radius of this circle will result in missing at least one line segment, so it is an MSC. The same argument holds for the case of the circle with two defining points. ∎

According to Theorem 4.3.1, in order to find a Minimum Spanning Circle of a set of line segments, two or three defining points are needed. Next we explain an efficient algorithm to find an MSC of a set of line segments, defined by two or three defining points.

MSC Defined by Three Points

To find the Minimum Spanning Circle, defined by three points, we use farthest line segments Voronoi diagrams. Recall from Section 4.3.1 that in farthest Voronoi diagram of a set of line segments S, vertex V_{ijk} has distance d from line segments s_i, s_j and s_k, such that $\forall s_t \in S$ $(t \neq i, j, k)$, $d > minDist(V_{ijk}, s_t)$. In other words, in the farthest Voronoi diagram of S, for any Voronoi vertex V, there are three line segments in S that have maximum minimum distance d from V. This means that if we draw a circle C with centre V and radius d, C will be a spanning circle of the set of line segments S. For the set of line segments $S = \{s_1, s_2, ..., s_n\}$, the farthest Voronoi diagram has O(n) Voronoi vertices. This means that there are O(n) candidate spanning circles for S. In O(n) time, we can find an MSC among all the candidate spanning circles by simply comparing their diameters. So using the vertices of the farthest Voronoi diagram for the set of line segments S, we can find an MSC

of S in $O(n \log n)$, for the circles defined by three points.

MSC Defined by Two Points

To find the Minimum Spanning Circle, defined by two points, we use the edges of the farthest Voronoi diagram. A point on an edge of a farthest Voronoi diagram of a set of line segments S, has the property of being farther from two of the line segments in S, compared to the rest of the line segments in S. Suppose there are two line segments $s_i, s_j \in S$ and there is a Voronoi edge e_{ij} which is the edge farthest from s_i and s_j. Any point on e_{ij} has the same distance from s_i and s_j and is farthest from s_i and s_j than any other line segment in S. Consider the shortest line segment that connects s_i to s_j and call it L_{ij}. If L_{ij} intersects e_{ij}, then the intersection point c is the centre of the a spanning circle of the set of line segments S with radius L_{ij}, defined by two points on s_i and s_j. If L_{ij} and e_{ij} do not intersect, no such circle exists. Repeating this process for all $O(n)$ edges of the farthest line segment Voronoi diagram, we can determine all candidate spanning circles delivered by two points.

After finding all the spanning circles defined by two or three points, we choose the circle with the minimum diameter as the MSC of the set of line segments S.

4.3.3 Approximation Bounds

We can find the Minimum Spanning Circle of a set of line segments S in $O(n \log n)$ time [1]. Next we show the relation between the perimeter of the minimum perimeter convex hull of S and the MSC of S.

The following lemmas provide useful direction towards the main result:

Lemma 4.3.2 *For any convex polygon Poly with edges* $E = \{v_1v_2, v_2v_3, ..., v_nv_1\}$, $\forall e_i \in E$, $len(e_i) < \sum len(e_j), j \neq i$.

Proof Our proof uses induction on the number of vertices of the polygon.

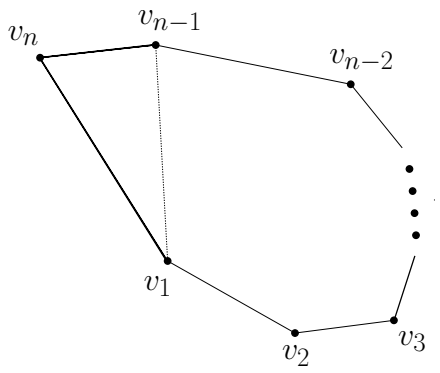

Fig. 4.11: Convex polygon with n edges

For a polygon with $n = 3$ (triangle), the lemma holds by the triangle inequality. Suppose that for a polygon with $n-1$ edges, the lemma holds, so $v_1v_{n-1} < \sum v_iv_{i+1}$, $i = 1, ..., n-2$. Now for a polygon with n edges (see Fig. 4.11) there are two more edges v_1v_n and $v_{n-1}v_n$ in addition to the edges from the previous polygon. According to the triangle inequality

$$v_{n-1}v_n < v_1v_n + v_1v_{n-1} < v_1v_n + \sum v_iv_{i+1}, i = 1, ..., n-2 \tag{4.1}$$

$$v_1v_n < v_{n-1}v_n + v_1v_{n-1} < v_{n-1}v_n + \sum v_iv_{i+1}, i = 1, ..., n-2 \tag{4.2}$$

So the lemma still holds for a polygon with n edges. ∎

Lemma 4.3.3 *For any convex polygon P with diameter D,* $2D < Perimeter(P)$.

Proof The diameter of a convex polygon P is the maximum Euclidean distance between any two vertices of P. The diameter of P divides the polygon into two

smaller convex polygons P_1 and P_2 (See Fig. 4.12). If D is one of the edges of the big polygon P, then $P_1 = P$

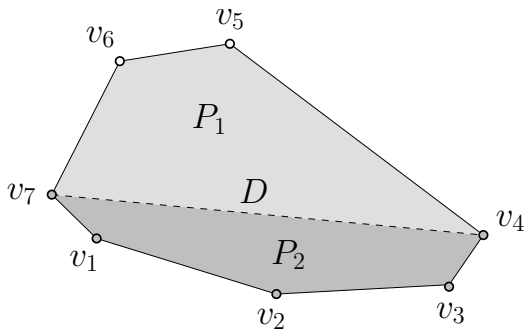

Fig. 4.12: A convex polygon P with diameter D

According to the Lemma 4.3.2,

$$D < Perimeter(P_1) - D \Longrightarrow 2D < Perimeter(P_1) \tag{4.3}$$

$$Perimeter(P_1) \leq Perimeter(P) \tag{4.4}$$

$$\Longrightarrow 2D < Perimeter(P) \tag{4.5}$$

∎

Observation 4.3.4 *For any triangle $\triangle abc$, such that $\overline{ab}$ is the diameter of a circle C, if $\angle acb > \frac{\pi}{2}$, then vertex c is located inside circle C.*

Lemma 4.3.5 *The centre of the Minimum Spanning Circle of a convex polygon P is included inside P.*

Proof We use contradiction to prove this lemma. Suppose that circle C is an MSC of polygon P and the centre of C is not included inside P. Without loss of generality,

we assume that P is a triangle and at least two of the vertices of P are located on the C.

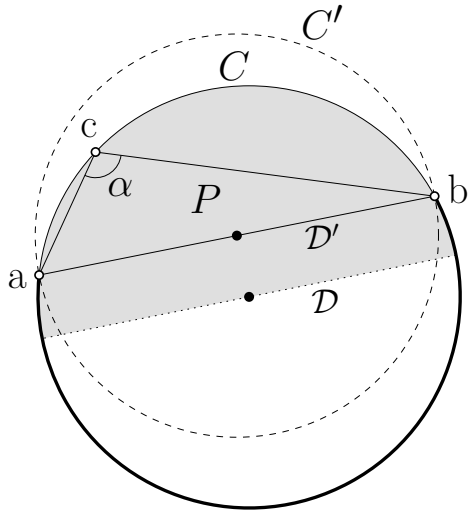

Fig. 4.13: Spanning circles of a triangle

Fig. 4.13 demonstrates such an example. If P does not include the centre of circle C, we can draw a diameter $\mathcal{D}$ of circle C such that P is completely located on one side of $\mathcal{D}$. It is also obvious that angle $\alpha > \frac{\pi}{2}$, since the arc facing α is greater than π. If we draw a circle C' with diameter $\mathcal{D}' = \overline{ab}$, then according to Observation 4.3.4, vertex c is located inside C', so C' is a spanning circle of P. Since $\mathcal{D}' < \mathcal{D}$, circle C' is smaller than circle C, and this contradicts the initial assumption of C being the MSC of P.

If polygon P has more than three vertices, then in the same way explained above for vertex c, we can show that each vertex is included in circle C'. ∎

Theorem 4.3.6 *Let P^* be a minimum perimeter convex hull of a set of line segments S and circle C^* with diameter $\mathcal{D}$ be a Minimum Spanning Circle of S. Then $2\mathcal{D} < Perimeter(P^*) < \pi\mathcal{D}$.*

Proof Because circle C^* is a convex shape, $Perimeter(P^*)$ cannot be larger than the perimeter of C^* (otherwise C^* is a minimum perimeter convex hull of S), so $Perimeter(P^*) < \pi\mathcal{D}$ (proof of the upper bound).

Let circle $\mathrm{C}_{\mathrm{P}^*}$ with diameter $\mathbb{D}$ be the minimum spanning (bounding) circle of P^*. If two vertices a and b of P^* are located on the boundary of $\mathrm{C}_{\mathrm{P}^*}$, $\overline{ab} = \mathbb{D}$ is the diameter of $\mathrm{C}_{\mathrm{P}^*}$ and also the diameter of P^*. So according to Lemma 4.3.3, $2\mathbb{D} < Perimeter(P^*)$. Otherwise, if three vertices of P^* are located on $\mathrm{C}_{\mathrm{P}^*}$, then according to Lemma 4.3.5, the centre of $\mathrm{C}_{\mathrm{P}^*}$ is included inside P^* and triangle $\triangle abc$.

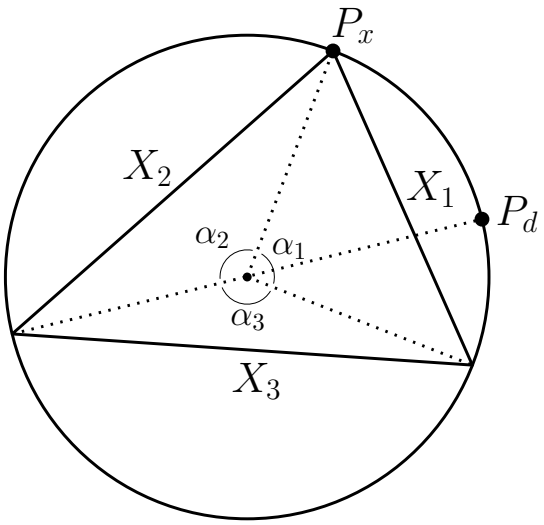

Fig. 4.14: Minimum Spanning Circle of a triangle

In Fig. 4.14, $0 < \alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \pi$. Since the triangle must include the centre of the circumcircle, α_1, α_2 and α_3 cannot be larger than π.

$$x_1 = \mathbb{D}\sin(\frac{\alpha_1}{2}) \quad , \quad x_2 = \mathbb{D}\sin(\frac{\alpha_2}{2})$$

($\mathbb{D}$ is the diameter of the circle)

By moving point P_x towards point P_d on the boundary of the circle, we decrease α_1 and x_1 and increase α_2 and x_2. Note that α_3 and x_3 do not change. We will show that moving P_x towards P_d decreases $x_1 + x_2$ and since x_3 is fixed, the perimeter of the triangle decreases.

The first derivatives of x_1 and x_2 show the growth rate or shrinkage rate of x_1 and x_2, with respect to α_1 and α_2. Since $\alpha_1 + \alpha_2$ is fixed, decreasing α_1 by 1 unit will increase α_2 by 1 unit.

$$\frac{\partial x_1}{\partial \alpha_1} = \frac{\mathbb{D}}{2}\cos(\frac{\alpha_1}{2}) \quad , \quad \frac{\partial x_2}{\partial \alpha_2} = \frac{\mathbb{D}}{2}\cos(\frac{\alpha_2}{2})$$

$\frac{\partial x_1}{\partial \alpha_1}$ *is the shrinkage rate of* x_1 *and* $\frac{\partial x_2}{\partial \alpha_2}$ *is the growth rate of* x_2

$$0 < \alpha_1 \leq \alpha_2 \leq \pi \;\Rightarrow\; 0 < \frac{\alpha_1}{2} \leq \frac{\alpha_2}{2} \leq \frac{\pi}{2}$$
$$\Rightarrow\; \cos(\frac{\alpha_1}{2}) > \cos(\frac{\alpha_2}{2}) \;\Rightarrow\; \frac{\mathbb{D}}{2}\cos(\frac{\alpha_1}{2}) > \frac{\mathbb{D}}{2}\cos(\frac{\alpha_2}{2})$$

which means that the shrinkage rate of x_1 is greater than the growth rate of x_2, so by moving P_x towards P_d, the perimeter of the triangle decreases. We can shrink x_1 until x_2 and x_3 are equal to the diameter of the circle ($\mathbb{D}$). At this point, the perimeter of $\triangle abc$ is equal to $2\mathbb{D}$, so the perimeter of the original triangle $\triangle abc$ and the perimeter of P^* are greater than $2\mathbb{D}$, so $2\mathbb{D} < Perimeter(P^*)$.

Circle C_{P^*} with diameter $\mathbb{D}$ is the MSC of P^*, so C_{P^*} is also a spanning circle of S. Circle C^* with diameter $\mathcal{D}$ is an MSC of S. Since C^* is an MSC, $\mathcal{D} \leq \mathbb{D}$. We also know that $2\mathbb{D} < Perimeter(P^*)$, so $2\mathcal{D} < Perimeter(P^*)$ (proof of the lower

bound). From the first part of the proof we also know that $Perimeter(P^*) < \pi\mathcal{D}$.
$\Rightarrow 2\mathcal{D} < Perimeter(P^*) < \pi\mathcal{D}$. ∎

4.3.4 Approximation Result

In the previous section we proved that the perimeter of a minimum perimeter convex hull of a set of line segments S is within a constant times the diameter of an MSC of S. What remains to be done is to find a convex hull of S such that its perimeter is bounded. We will use the MSC of S in order to find such a convex hull.

Observation 4.3.7 *For any two convex shapes $\mathcal{H}_1$ and $\mathcal{H}_2$ in 2D, $\mathcal{H}_1 \subset \mathcal{H}_2$, the perimeter of $\mathcal{H}_1$ is less than the perimeter of $\mathcal{H}_2$ (Fig. 4.15).*

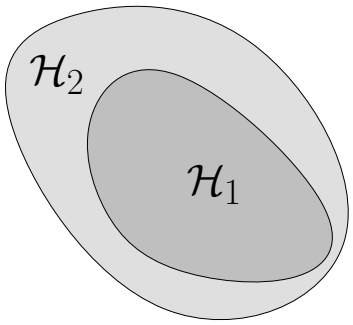

Fig. 4.15: Two convex shapes

According to the definition of a Minimum Spanning Circle C^* of a set of line segments S, we know that there is at least one point of each line segment $s_i \in S$ that is inside or on the boundary of C^*. In our approach, for each $s_i \in S$, we simply choose the point on s_i that is closest to the centre of C^* and we keep the chosen points in S'. The minimum distance from any $s_i \in S$ to the centre of C^* is at most $\frac{\mathcal{D}}{2}$, where $\mathcal{D}$ is the diameter of C^*. Then we compute the convex hull of S' in $O(n \log n)$ [6].

Let's call this convex hull CH′. Since all the points in S' are included in C^*, CH′ is also included inside C^*, so according to Observation 4.3.7

$$Perimeter(\text{CH}') < Perimeter(C^*) \Rightarrow Perimeter(\text{CH}') < \pi\mathcal{D} \tag{4.6}$$

Suppose that P^* is a minimum perimeter convex hull of S. Then the perimeter of CH′ cannot be smaller than the perimeter of P^*.

$$Perimeter(P^*) \leq Perimeter(\text{CH}') \tag{4.7}$$

According to Theorem. 4.3.6, Equation 4.6 and Equation 4.7 we can conclude that

$$2\mathcal{D} < Perimeter(P^*) \leq Perimeter(\text{CH}') < \pi\mathcal{D} \tag{4.8}$$

We summarize the result of this section with the following theorem:

Theorem 4.3.8 *Given a set of line segments S with minimum perimeter convex hull P^*, we can obtain a convex polygon P in $O(n \log n)$ time that spans all the line segments in S and satisfies the inequality*

$$Perimeter(P) \leq \frac{\pi}{2} \times Perimeter(P^*) \tag{4.9}$$

The Worst Cases

Fig. 4.16 illustrates the MSC of a set of line segments. In this figure, the white point on each line segment is the closest point on that line segment to the centre of the circle. The grey polygon is the convex hull of the white points. This convex hull is

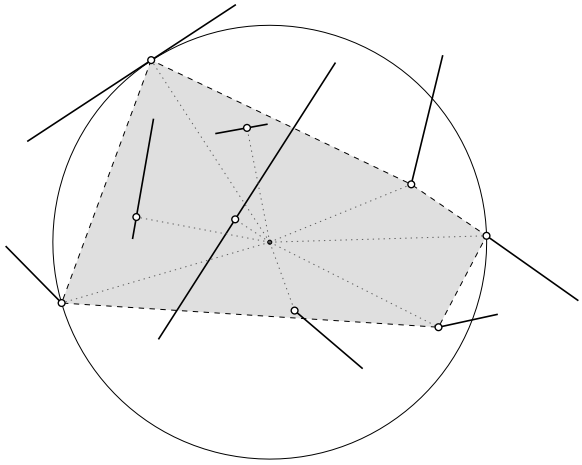

Fig. 4.16: Approximated convex hull

the approximated convex hull for the set of line segments in Fig. 4.16.

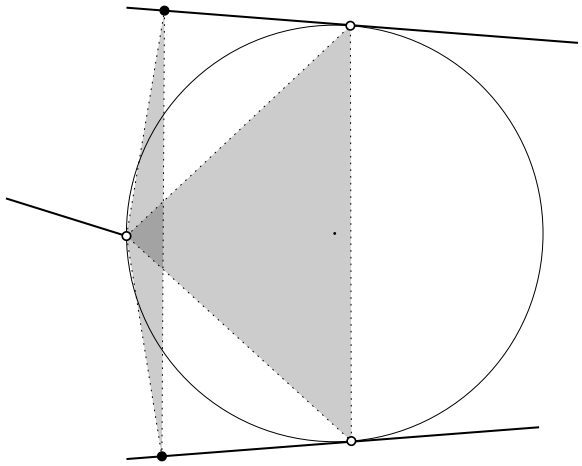

Fig. 4.17: Approximated convex hull and MPCH

An approximated convex hull of S is always completely located inside the MSC of S. But the minimum perimeter convex hull of S can be located outside of the MSC of S (see Fig. 4.17). According to Theorem 4.3.6, the approximated result is at most $\frac{\pi}{2}$ times greater than the optimum result. Fig. 4.18 shows an example in which the result of the approximation is $\frac{\pi+2}{4}$ times greater than the optimum result.

In this figure, all the line segments are pesky rim segments, except for the top and the bottom line segments. The minimum perimeter convex hull is almost a vertical line, crossing all the line segments. The perimeter of the minimum perimeter convex hull is approximately $2d$. Note that according to Theorem 4.3.6, the perimeter of the MPCH in this example cannot be less than $2d$. The Approximated convex hull is almost a half circle and its perimeter is approximately $\frac{\pi d}{2}+d$. The ratio between the approximated answer and the optimum answer is $\frac{\pi+2}{4}$.

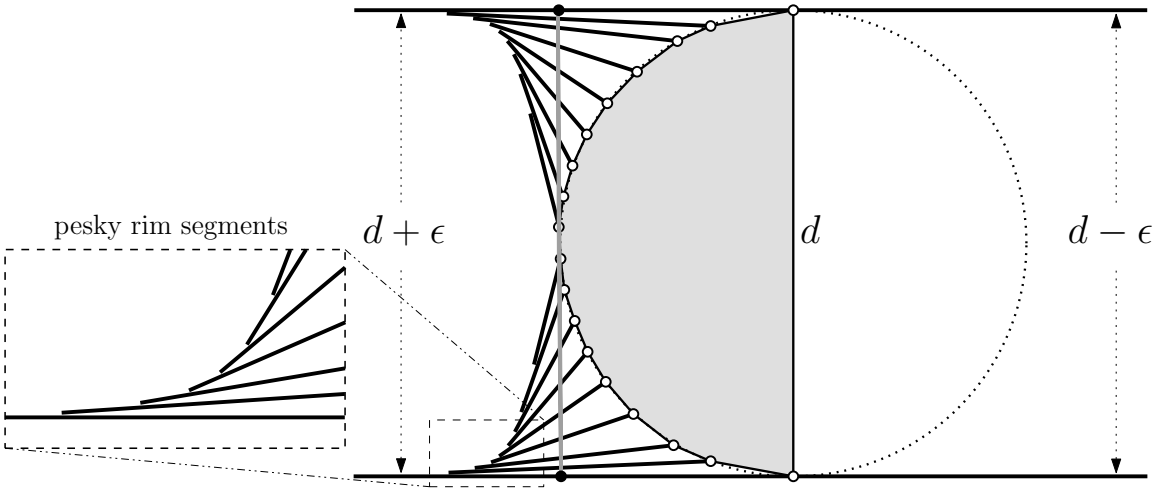

Fig. 4.18: Approximated convex hull and MPCH

In Fig. 4.19 the ratio between the optimum answer and the approximated answer is also $\frac{\pi+2}{4}$, but in this case the minimum perimeter convex hull is located *inside* the MSC. Fig. 4.19(A) shows the set of line segments. Note that these line segments are non-intersecting and except for the top and the bottom line segments, all the other ones are pesky rim segments. The minimum perimeter convex hull of this set of line segments is almost a vertical line connecting the top and the bottom line segments and crossing the rest of the line segments. The perimeter of the minimum perimeter convex hull is approximately $2d$. In Fig. 4.19(B) The white point on the line segment

is the closest point on that line segment to the centre of the MSC (the big dashed circle). For the top half of the line segments, the closest points are located on a circle with diameter $\frac{d}{2}$ which is centred on the diameter of the MSC. For the bottom half of the line segments, there exists a circle similar to the top half. These two circles are shown as dotted circles inside the MSC in Fig. 4.19(B). The convex hull of the white points is shown as a grey polygon in Fig. 4.19(B). The perimeter of this polygon is approximately $\frac{\pi d}{2}+d$.

Although there are no proofs that the examples in Fig. 4.18 and Fig. 4.19 are the worst case examples that can be found, they motivate us to investigate whether the upper bound can be reduced from $\pi\mathcal{D}$ to $\frac{\pi\mathcal{D}}{2}+\mathcal{D}$ or even less.

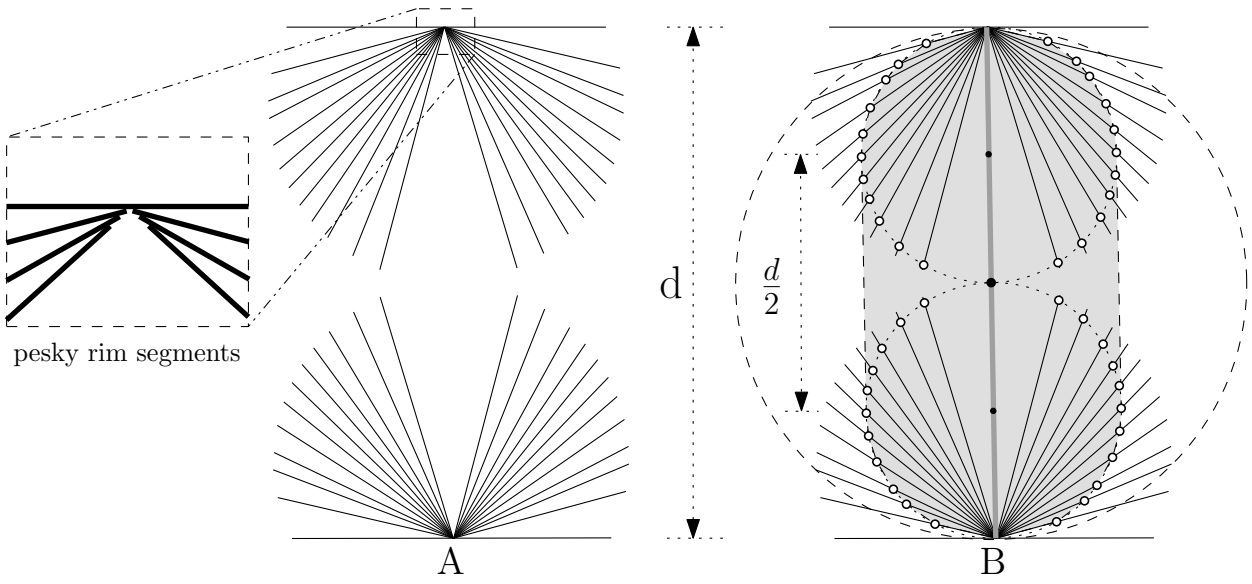

Fig. 4.19: Approximated convex hull and MPCH

Chapter 5

Conclusion and Future Work

5.1 Summary

The problem of finding a minimum perimeter convex hull of a set of line segments is to choose exactly one point on each line segment such that the convex hull of those points has minimum perimeter. The so far best known algorithm to solve this problem is proposed by Rappaport [14], and considering our extension to Rappaport's algorithm to cover the missed case, this algorithm in the worst case works in $O(4^n n \log n)$ time. In Chapter 3, we demonstrated all the step to implement this algorithm, we extended the algorithm by investigating a new case and we also pointed out the reason behind the exponential running time of this algorithm.

In Chapter 4, we proposed an approximation algorithm, the FIFO approach, which is very similar to Rappaport's algorithm, and we showed that the results produced by the FIFO approach cannot be bounded. Then we proposed a completely different approach, the MSC approximation, which uses Voronoi diagrams in order to find the Minimum Spanning Circle of a set of line segments S. We proved that using the MSC

of S, we can compute a convex polygon C' that spans all the line segments in S, such that the perimeter of C' is a constant factor of the perimeter of an MPCH of S.

5.2 Future work

Besides the polynomial running time and the tight bounds of the MSC approximation, another useful property of this approximation is its generality, i.e. for any set of objects O, we can apply the MSC approximation in order to approximate the minimum perimeter convex hull of O. So if we can find an MSC of O, then $\frac{\text{approximated answer}}{\text{optimal answer}} \leq \frac{\pi}{2}$.

Notice that finding the MSC of different objects may need different approaches. For example, finding the MSC of a set of line segments uses Voronoi diagrams and takes $\mathrm{O}(n \log n)$ time. This computational time may be different for a set of triangles or generally, a set of simple polygons. But in any of these cases, after computing an MSC of the set of objects O, the MSC approximation can be applied in $\mathrm{O}(n \log n)$ time and the aforementioned ratio holds.

We may also be able to improve the running time of the algorithm using linear programming. One may come up with a linear programming approach to find the MSC of a set of line segments in linear time. This may reduce the running time of the algorithm. But since after finding the MSC, we compute the convex hull of a set points that are located inside the MSC in $\mathrm{O}(n \log n)$ time, the complexity will still be $\mathrm{O}(n \log n)$.

It is still unknown whether the problem of finding a minimum perimeter convex hull of a set of line segments is NP-hard.

Bibliography

[1] F. Aurenhammer, R. L. S. Drysdale and H. Krasser: Farthest line segment Voronoi diagrams, Information Processing Letters, Volume 100, Issue 6, Pages 220-225, December 2006.

[2] R. E. Burkard, G. Rote, E. Y. Yao, Z. L. Yu: "Shortest polygonal paths in space", Computing, Volume 45, Pages 51-68, 1990.

[3] Timothy M. Chan: "Optimal output-sensitive convex hull algorithms in two and three dimensions", Discrete and Computational Geometry, Volume 16, Pages 361368, 1996.

[4] Jurek Czyzowicz, Peter Egyed, Hazel Everett, David Rappaport, Thomas C. Shermer, Diane L. Souvaine, Godfried T. Toussaint, Jorge Urrutia: The Aquarium Keeper's Problem. SODA 1991: Pages 459-464.

[5] Steven Fortune: Stable Maintenance of Point Set Triangulations in Two Dimensions. Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Pages 494-499, 1989.

[6] R. L. Graham: An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. Information Processing Letters, Volume 1, Issue 4, Pages 132-133, June 1972.

[7] Leonidas Guibas, David Salesin and Jorge Stolfi: Constructing strongly convex approximate hulls with inaccurate primitives. Algorithmica 9, Pages 534560, 1993.

[8] Leonidas Guibas, David Salesin and Jorge Stolfi: "Epsilon Geometry: Building Robust Algorithms from Imprecise Computations", Proceedings of the 5th Annual ACM Symposium on Computational Geometry, Pages 208-217, 1989.

[9] R. A. Jarvis: "On the identification of the convex hull of a finite set of points in the plane". Information Processing Letters 2, Pages 1821, 1973.

[10] Zhenyu Li and Victor Milenkovic: Constructing Strongly Convex Hulls Using Exact or Rounded Arithmetic. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 450, 261-270, 1990.

[11] Maarten Löffler and Marc van Kreveld: Approximating Largest Convex Hulls for Imprecise Points. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 4927, Pages 89-102, 2008.

[12] Maarten Löffler and Marc van Kreveld: Largest and Smallest Convex Hulls for Imprecise Points. Algorithmica 2008.

[13] Maarten Löffler and Marc van Kreveld: Largest Bounding Box, Smallest Diameter, and Related Problems on Imprecise Points. Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 4619, Pages 446-457, 2007.

[14] David Rappaport: Minimum polygon transversals of line segments. Int. J. Comput. Geometry Appl. 5(3): 243-256, 1995.

[15] Steven S. Skiena, The Algorithm Design Manual, Springer 2nd edition (August 21, 2008), ISBN-10: 1848000693, ISBN-13: 978-1848000698

[16] Kokichi Sugihara and Masao Iri: Construction of the Voronoi diagram for one million generators in single-precision arithmetic. Proceedings of the IEEE, Volume 80, Pages 1471-1484, 1992.