COMPUTING FRÉCHET DISTANCE OF TWO PATHS ON A CONVEX POLYHEDRON

By

Jiehua Yi

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

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

August 2004

© Copyright

2004, Jiehua Yi

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97461-8

Our file Notre référence

ISBN: 0-612-97461-8

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

Abstract

Fréchet distance is a well-known metric that measures the similarity of two curves in an arbitrary Euclidean vector space. This thesis addresses the similarity of two paths on the surface of a convex polyhedron with respect to Fréchet distance. Fréchet distance is measured by Euclidean shortest path distance on the convex polyhedron. We present an algorithm for computing Fréchet distance between two paths by building a free space diagram for the two paths. The free space diagram is composed of the parameterization of the edges on the two paths. We operate on the free space diagram for a solution.

This thesis provides an algorithm for computing Euclidean shortest path between any two points on a pair of edges on the surface of a convex polyhedron. We build an edge sequence tree and a visibility diagram for the two edges, such that the shortest path problem in 3-dimension can be solved in a 2-dimensional graph.

Acknowledgements

I would like to thank Anil Maheshwari, my supervisor, for his ideas, guidance, comments, questions, provisions of useful references, patience in all of the drafts and his support. It is because of him that many of the ideas in this thesis have been developed to this extent.

I would like to thank the members on my examining board: Stefan Dobrev, Mark Lanthier and John Oommen. I really do appreciate their willingness to be on the committee, and the time they spent reading my thesis.

Table of Contents

Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Figures	vii
1 Introduction	1
1.1 Related Work on Shortest Path Problems	3
1.2 New Results in the Thesis	7
2 Preliminaries	12
2.1 Notation and Preliminaries	12
2.2 Shortest Paths Between Two Vertices in a Polygon	13
2.3 Shortest Paths in 3-D	15
2.4 The Visibility Diagram of a Polygon in 2-D	16
2.5 Computing the Overlay of a Set of Subdivisions	20
2.6 The Separator of Two Polygons in 2-D	22
2.7 Finding a Monotone Path in a Trapezoidal Map	29
3 Shortest Paths on a Convex Polyhedron	36
3.1 Outline of the Algorithm	36
3.2 Details of the Algorithm and its Correctness	38

4	Fréchet Distance on a Convex Polyhedron	52
4.1	Fréchet Distance Between Two Polygonal Curves	52
4.2	Fréchet Distance of Two Paths on a Convex Polyhedron	59
5	Conclusion	67
	Bibliography	70

List of Figures

1.1	Edge sequences and the dual graph	6
1.2	The trick in adding a node in the edge sequence tree rooted at s	7
1.3	(a): The parameterization of two segments; (b): The parameterization of two paths. P has p segments and Q has q segments.	8
1.4	(a): A sequence of edges and faces that a shortest path from $p \in e_s$ to $q \in e_t$ passes through; (b): The planar unfolding relative to the edge sequence; (c): The visibility diagram of e_s and e_t	10
2.1	The inward-convex polygonal chain	14
2.2	The hyperbolic curves between P'_ξ and P'_ϕ produced by the vertices of P' .	18
2.3	The edge data representation	21
2.4	(a): Two polygons with the same edges e_s and e_t ; (b): The overlay of the two polygons and the edge e_s is identified. e_t corresponds to e_{t_1} and e_{t_2} in the overlay.	22
2.5	The property of a single Voronoi cell	23
2.6	The different kinds of separator curves	24
2.7	The parameter $direction = 0$	28
2.8	(a): The trapezoidation of a set of polygons (b): The free space area of the trapezoidation of a set of polygons (without the areas inside the polygons)	30

2.9	(a)(b)(c): Placing nodes in the trapezoidal map. (d): Add arcs between the nodes to form a directed graph.	31
2.10	The cases with/without a monotone path inside a trapezoid	33
2.11	(a): No monotone path from T' to T through p' , p does not exist; (b): The monotone path from p' to p in T	34
3.1	The projections of the image of e_s on two edges e_5 and e_6	39
3.2	The bold path is a real shortest path from one endpoint of e_s to one endpoint of e_t , which is located in the edge sequence from e_1 to e_2 , not in the paths starting from e_s and ending at e_t in the edge sequence tree of e_s	40
3.3	An edge sequence after planar unfolding	43
3.4	A polygon with two shortest paths on its boundary	44
3.5	Several cases for the visible polygon (bold areas)	45
3.6	One visible polygon P_v with two original labels ξ_1 and ξ_2 . The separator curve of ξ_1 and ξ_2 . The different values of the parameter <i>direction</i> of the separator curve	48
4.1	The parameterization of a polygonal curve	53
4.2	The parameterization of two polygonal curves	54
4.3	The free space for two segments	55
4.4	The free space for two polygonal curves	57
4.5	The values that must be calculated	58
4.6	The monotone free space of two polygonal curves	59
4.7	The trapezoidation of a free space after union	61
4.8	An example of critical values	65

Chapter 1

Introduction

Methods used to match geometric patterns have been studied extensively in the past years, including their applications ranging from machine vision and image databases to computational chemistry and drug design, for example, applications of measuring the dissimilarity between two signatures recorded by an electronic pen, or finding the most similar path on the real map compared to the path recorded by a GPS receiver. There are several ways to measure the quality of the approximation: Hausdorff-distance, Fréchet-distance, uniform distance [24], and a “distance-preserving” measure [6]. Alt and Godau [8, 17] proposed that Fréchet distance is one of the most fundamental measures to compute the dissimilarity between two polygonal curves.

Fréchet distance is often referred to as the dog-leash distance. This reference bases on the analogy that a person is walking a dog and the person is constrained to walk on a curve and the dog on another curve; both the person and the dog are allowed to control their speed independently, but are not allowed to go backwards; then, Fréchet distance of the curves is the minimal length of the leash so that the dog and the person can traverse their respective sequences from beginning to end, while respecting the constraint that at any moment of time, the distance between them cannot exceed the leash length. The smaller

the length of the leash, the more similar the two curves. If the leash length is 0, then the two polygonal curves are exactly the same curve. For a given polygonal curve and an embedded graph, whose edges are line segments, the problem of finding a path in the graph, such that Fréchet distance between the curve and the path is minimized has been studied in Alt et al. [7] and [1]. In all of these references, Fréchet distance is measured by Euclidean distance.

This thesis focuses on the following question: How can we use Fréchet distance to measure the similarity of two paths Z and Z' on the surface of a convex polyhedron P ? Without loss of generality, assume that each segment on the two paths is an edge of P . A person is walking on the path Z and a dog is walking on the path Z' , the leash between them is not the direct Euclidean distance in 3-dimension. The leash must be measured along the surface of P . The shortest distance between the person and the dog at any point is the shortest path distance between them. The shortest path on a convex polyhedron has three properties: triangle inequality, reflexivity and symmetry. These properties determine that shortest path can be a metric. Therefore, the length of the leash is measured by the length of shortest path on P . That is, Fréchet distance is measured by Euclidean shortest path distance on the surface of the convex polyhedron P . Thus, this thesis also explores the computation of shortest path distance between any two points on a pair of edges.

Since computing shortest path distances is the preprocessing step, the next section will introduce the shortest path problems, as well as list and compare previous work in this area. Our algorithms will make use of some of the results and techniques presented in this part.

1.1 Related Work on Shortest Path Problems

Shortest paths in Euclidean space amidst polyhedral obstacles arise in planning optimal collision-free paths for a given robot, and have been widely studied. In Mitchell's [18] survey paper, shortest path problems are categorized as the following:

1. different dimension: shortest paths on planar graphs (2-D), convex polyhedron (3-D), or non-convex polyhedron (3-D) etc.
2. different metric: unweighted, Euclidean shortest path (only consider Euclidean distance of the path) or weighted shortest path (the cost of the travel varies between regions), or link distance as the metric.
3. amount of queries: *single pair problem*, computing the shortest path from a given source point to a given destination point; *single source problem*, computing the shortest path from a given source point to any point of the object; *all pairs (edge-edge) problem*, computing the shortest path of any pair of points on two edges; etc.
4. accuracy: computing optimal shortest paths or computing approximate shortest paths.
5. the availability of the queries: static or dynamic.

One of the most widely studied special cases is computing shortest paths on the surface of a convex polyhedron. The shortest path problem in this thesis is related to the optimal, unweighted *edge-edge* shortest paths on a convex polyhedron.

In 3-dimension, Canny and Reif have shown that the general shortest path problem is NP-hard and that only exponential time algorithms are known. Sharir and Schorr [20]

first present an $O(n^3 \log n)$ time algorithm for the single source shortest path problem on a convex polyhedron, which is subsequently improved by Mitchell et al. [12] to $O(n^2 \log n)$ time by using the continuous Dijkstra technique, whose algorithm is also suitable for the nonconvex case (n is the number of faces of the polyhedral surface). Chen and Han [9] present an $O(n^2)$ time algorithm for convex and nonconvex cases. All of these methods find the subdivision on the surface of a given convex polyhedron according to one fixed source point, such that any point in the same region has the same shortest path edge sequence to this source point. After building the subdivision, the shortest path problem can be transformed into a standard point location problem and the shortest path from the fixed source point to a given query point can be computed in time $O(k + \log n)$ where k is the number of edges in the corresponding shortest path edge sequence.

Hwang et al. [25] present an algorithm, which is quite different from the former ones, running in $O(n^6 \log n)$ time that can find all of the shortest path edge sequences on the convex polyhedron P . Hwang et al.'s algorithm can report shortest path between any pair of points on any two edges respectively on P from the corresponding shortest path edge sequence.

First, they use the brute force approach to build a tree T named *edge sequence tree* for all of the possible shortest path edge sequences rooted at the starting edge e_s . Each node in this tree is an edge of P . Next they build a *shortest path edge sequence tree* T' , obtained from deleting some nodes of T , such that for every node of T' , the path from the root to this node is a shortest path edge sequence.

Lemma 1.1.1. [25] *If an edge sequence $\xi = \{e_1 e_2 \dots e_{n-1} e_n\}$ is a shortest path edge sequence, then its subsequence $\xi_1 = \{e_1 e_2 \dots e_{n-1}\}$ is also a shortest path edge sequence.*

This lemma is obtained from the following property of shortest paths: If a path is a shortest path, then any of its subpaths must also be a shortest path. This lemma also implies that once a shortest path edge sequence ξ_1 is found, it is very likely that ξ would be another shortest path edge sequence. Thus, the process to find new shortest path edge sequences can be considered as the expansion of edge sequence trees. The process is the following: Start from the edge e_s , add the edges that share the same face with e_s as the children of e_s , then iteratively select a leaf e_x — the edge sequence ξ from e_s to e_x is a shortest path edge sequence on P , until all of the shortest path edge sequences are found.

To be a shortest path edge sequence, ξ must satisfy: For two points $p \in e_s$ and $q \in e_t$, p and q are *visible* to one another in the unfolded edge sequence and p and q have the smaller distance in the edge sequence than in any other edge sequences. All of the pairs of points that correspond to the same shortest path edge sequence are organized in one common region in the visibility diagram of e_s and e_t . The details of the visibility and the visibility diagram are introduced in Chapter 2. They are also important techniques adopted in this thesis.

Comparatively, the complexity of the edge-edge shortest path problem is higher than that of Chen and Han's fixed source point shortest path problem. The reason is that the number of the shortest path edge sequences in the edge sequence tree rooted at one source edge is bounded to $O(n^3)$ by Mount [3], where n is the number of faces on the convex polyhedron.

The proof of this bound is based on the *dual graph* G of the convex polyhedron P . The dual graph G is an embedded planar graph whose vertices are the faces of P and whose edges are the pairs of faces that share a common edge. First they show that for two paths p and q in G arising from two edge sequences in P , p and q can only, at the most, cross once.

Figure 1.1: Edge sequences and the dual graph. Paths p_1, p_2 and p_3 correspond to ϕ_1, ϕ_2 and ϕ_3 respectively.

Refer to Figure 1.1. This fact implies that the number of distinct non-crossing paths in G , originating from a common origin and terminating at a common destination, is linearly bounded. Next, they apply the edge contraction operation on G — removing one edge at a time from the graph G . This process may cause previous distinct edge sequences to become equal. They begin with a finite set of paths in G travelling from an arbitrary source edge to an arbitrary destination edge. They show that the set of paths that become equal when a contraction is performed has essentially the same structure as the set of non-crossing paths emanating from a common origin, and hence $O(n)$ paths become equal with each contraction. By repeating the contraction on all of n edges, they reduce the set of distinct paths by $O(n)$ each time until a trivial set of $O(n)$ paths remain. Since $O(n^2)$ paths are removed in the entire process, it follows that the original number of distinct paths between the two edges was $O(n^2)$, thus, the bound $O(n^3)$ for a fixed origin node is obtained as desired.

Chen and Han use a clever technique to build the edge sequence tree T rooted at the source point s , the number of the edge sequences on the tree T is bounded by $O(n^2)$. They start from the source point s , and add the edges that share the same face with s as the

Figure 1.2: The trick in adding a node in the edge sequence tree rooted at s . (a): One unfolded edge sequence. I is the image of s . e' is the only child of e on T . s is visible to the points on the bold segment of e' ; (b): Two unfolded paths on T where e is the corresponding edge of two nodes n_1 and n_2 . I_1 and I_2 are the images of s in the two unfolded paths. s is visible to the points on the bold segments both in the two unfolded edge sequences. n_1 has one child e_1 and n_2 has two children e_1, e_2 on T since $|I_1A| > |I_2A|$.

children of s . Then they add edge e' that shares the same face with node n' (on edge e) as node n' 's child if s is visible to some points on e' inside the unfolded edge sequence from s to e' . Refer to Figure 1.2 (a). If two nodes n_1 and n_2 on the tree T are on the same edge e of face f , then, at the most, only one of them can have two children. Refer to Figure 1.2 (b). Because $|I_1A| > |I_2A|$, n_1 has one child e_1 and n_2 has two children e_1 and e_2 on T . Since there are $O(n)$ leaves in each level on T , it takes $O(n)$ time to build the next level's leaves (one angle one split). Thus, it takes $O(n^2)$ time to generate the tree.

1.2 New Results in the Thesis

Two problems are addressed in this thesis.

Given a convex polyhedron P , and two paths Z and Z' on the surface of P ,

1. Design a data structure, shortest paths of all of the pairs of points on two edges $e_s \in Z$ and $e_t \in Z'$ on P can be computed from this data structure;
2. Compute Fréchet distance of Z and Z' on P .

Figure 1.3: (a): The parameterization of two segments; (b): The parameterization of two paths. P has p segments and Q has q segments.

We use the structure of segment parameterization for computing Fréchet distance between two paths, and computing the shortest paths between any two points on a pair of edges. Figure 1.3 shows two segments P and Q and their parameterization — a unit square; two paths P and Q and their parameterization — a combination of unit squares. Each point in the parameterization figure corresponds to a pair of points on two segments (the parameterization of two paths is introduced in Chapter 4). To make it simple, we will use

the term “the pair of points in the unit square” when referring to the point in the unit square. For a given value ϵ , the parameterization figure is divided into white areas and black areas, the distance of each pair of points in the white areas is shorter than ϵ , while the distance of each pair of points in the black areas is bigger than ϵ . This figure is called a *free space diagram*, refer to Figure 1.3. In 2-dimension, Fréchet distance of two paths P and Q is the smallest Euclidean distance ϵ of two points on P and Q that can make a monotone path (which always goes up and goes to the right) exist in the *white area* of the free space diagram [8]. Figure 1.3 (b) shows one monotone path in the free space area starting from $(0,0)$ and ending at (p,q) . Similarly in the case of convex polyhedron, we need to find the smallest Euclidean shortest path distance that can make a monotone path exist in the free space diagram of Z and Z' on P .

We define crucial values (the possible Fréchet distance) to be shortest path distances of some special pairs of points on the two paths, then according to each crucial value, we construct the free space diagram for the two parameterization paths. The pairs of points in the white area in the free space diagram have shorter distances than the crucial value. If there exists a monotone path in the white area, then this crucial value is a candidate for a Fréchet distance value. Following this process, the smallest value among all of the candidates is computed, which is Fréchet distance of the two paths. Trapezoidation and breadth first search are used in each unit cell of the free space diagram in order to find a monotone path inside the cell, which is the necessary condition for the existence of the successive monotone path in the free space diagram. We present the algorithm in Chapter 4 with the running time $O(zz'n^2 \log n \log(zz'n^2))$, where z and z' are the numbers of edges on the two paths Z and Z' respectively and n is the number of faces of P .

The preprocessing part of our algorithm computes shortest paths for any pair of points

Figure 1.4: (a): A sequence of edges and faces that a shortest path from $p \in e_s$ to $q \in e_t$ passes through; (b): The planar unfolding relative to the edge sequence; (c): The visibility diagram of e_s and e_t

on two segments. For two segments e_s and e_t of the two paths on P , the shortest path of a pair of points on e_s and e_t passes through a sequence of faces (edges) on P , refer to Figure 1.4 (a). For a pair of points $p \in e_s, q \in e_t$, the sequence of edges (e_s, e_2, e_3, e_t) that the shortest path from p to q crosses is called the *shortest path edge sequence* of p and q . The faces f_1, f_2 and f_3 are *unfolded* into the plane, refer to (b) (unfolding is introduced in Chapter 2). The shortest path between p and q — $\Pi(p, q)$, becomes a straight line in the plane after unfolding. The pairs of points on e_s and e_t may pass through different shortest path edge sequences. We divide the pairs of points into several parts in the parameterization figure of e_s and e_t (named the *visibility diagram*), each part corresponds to a shortest path edge sequence. Thus shortest path of any pair of points can be computed in its corresponding edge sequence. In Figure 1.4 (c), all of the pairs of points correspond to one edge sequence

in the visibility diagram.

The shortest path algorithm is based on the common geometric concept of *unfolding*. Basically, we modify the algorithm of Chen and Han [9] which builds the edge sequence tree T rooted at the starting edge e_s . We then select all of the edge sequences starting from edge e_s and ending at edge e_t . Then for two points $p \in e_s$ and $q \in e_t$, all of the edge sequences with *geodesic* paths from p to q ($\overline{pq}$ is a straight line in the interior of the unfolded edge sequences) can be found. Using the visibility diagram, the shortest path between p and q among all of the geodesic paths can be computed. The techniques of visibility in a polygon, line parameterization and application of the property of the single cell Voronoi diagram facilitate the construction of an efficient algorithm for the shortest path problem; it runs in $O(n^4 \log n)$ time where n is the number of faces on P .

This thesis is organized as the following. Chapter 2 formalizes the terminology and lists some of the basic properties of shortest paths, visibility in polygons, the visibility diagram and techniques used to find a monotone path inside a trapezoidal map. Chapter 3 presents a solution to the shortest path problem between two edges on a convex polyhedron. Chapter 4 addresses the algorithm of computing Fréchet distance between two paths on P . In the last chapter we conclude and present some open problems.

Chapter 2

Preliminaries

Most of the geometric and graph theoretic terminologies used in this thesis are standard. This chapter will begin by reviewing some of the main geometric techniques and graph theoretic terminologies used in this thesis.

Also, major techniques which have been originally introduced in several papers, are reviewed in this chapter. These techniques include, unfolding [9], computing shortest path inside a polygon [16], the visibility diagram [25] and computing a path in a trapezoidal map [13].

2.1 Notation and Preliminaries

A *simple polygon* is a simply connected subset of the 2-dimensional plane whose boundary is a closed chain of line segments. As we are only dealing with simple polygons, we will refer to them as polygons for the remainder of the thesis. A subset S of the 2-dimensional plane is called *convex* if and only if for any pair of points $p, q \in S$ the line segment $\overline{pq}$ is completely contained in S . (In this thesis, we denote a straight line from p to q as $\overline{pq}$; and we denote Euclidean distance from p to q as $|pq|$.) In $\mathbb{R}^3$, a *Polyhedron* is bounded

by planar polygons, with each edge shared by exactly two polygons. In 3-dimensional Euclidean space, if each face of a polyhedron is a convex polygon, and all the diagonals (segments joining two vertices, that are different from the sides or the edges) are in the interior, it is known as a *convex polyhedron*.

Two points p and q in a polygon are said to be *visible* if $\overline{pq}$ does not intersect the exterior of the polygon.

Let P be a convex polyhedron with a set of n -triangular faces in 3-dimension. The face set is denoted as $F = \{f_1, f_2, \dots, f_i, \dots\}$. The edge set is denoted as $E = \{e_1, e_2, \dots, e_i, \dots\}$. Let e_s and e_t be two edges on P and let $p \in e_s$ and $q \in e_t$ be two points, respectively. A *path* from point p to q is denoted as $\pi(p, q)$. In a convex polyhedron, $\pi(p, q)$ intersects the faces and edges located between p and q , and the intersection of $\pi(p, q)$ with a face is a line segment. The length of the path is the sum of Euclidean distances of the intersected segments.

A *geodesic path* is a path, which is locally optimal and cannot be shortened by slight perturbations. A path $\Pi(p, q)$ is said to be a *shortest path* if and only if the length of the path is the smallest among all paths from point p to q .

The edge sequence that the shortest path $\Pi(p, q)$ passes through is denoted as $\xi(\Pi(p, q))$. It is denoted as the *shortest path edge sequence*.

2.2 Shortest Paths Between Two Vertices in a Polygon

Relevant properties and algorithms about shortest paths in 2-dimension are given in [4], [5] and [23]. Related concepts in [4] are introduced below.

Figure 2.1: The inward-convex polygonal chain

Definition 2.2.1. A *polygonal chain* $\overline{q_1 q_2 \dots q_k}$ is a sequence of points $q_i (i = 1, 2, \dots, k)$, in which each pair of adjacent points q_i and q_{i+1} denotes a segment, and no two non-consecutive segments intersect.

Definition 2.2.2. A polygonal chain $\overline{q_1 q_2 \dots q_k}$ of a polygon P' is said to be an *inward-convex* polygonal chain if $\overline{q_1 q_2 \dots q_k q_1}$ is a convex, and the convexity faces towards the interior of polygon P' .

Figure 2.1 is an example of two inward-convex polygonal chains in a polygon and there are at least two points on e_s and e_t that can see one another.

Lemma 2.2.1. Let e_s and e_t be two edges of a polygon P' . Let $L_1\{p_1, p_2, \dots, p_n\}$ and $L_2\{q_1, q_2, \dots, q_m\}$ be two polygonal chains, $e_s = \overline{p_1 q_1}$, $e_t = \overline{p_n q_m}$, and there is at least one pair of points $p \in e_s, q \in e_t$ that can see one another in this polygon. Then the shortest paths between p_1 and p_n , q_1 and q_m are inward-convex polygonal chains.

Proof. According to Lee [5], the shortest path between two vertices of a polygon is a straight line or an inward-convex polygonal chain. □

2.3 Shortest Paths in 3-D

Relevant properties of shortest paths have been shown in [3] and [19, 20]. They are summarized as the following.

Property 2.3.1. *A shortest path on a convex polyhedron cannot cross any face or any edge more than once.*

Property 2.3.2. *A shortest path cannot pass more than n faces, where n is the total number of faces of the polyhedron.*

Property 2.3.3. *Two shortest paths starting from the same source point (or ending at the same destination point) cannot intersect one another unless at the source point (or at the destination point).*

According to the properties, a shortest path $\Pi(s, t)$ on P is identified uniquely by its endpoints and the sequence of edges that it passes through. Each edge and each face that $\Pi(s, t)$ crosses in the shortest path edge sequence $\xi(\Pi(s, t))$ are unique. This edge set is denoted as $E = \{e_1, e_2, \dots, e_k\}$ and the face set that the shortest path passes through is denoted as $F = \{f_1, f_2, \dots, f_{k+1}\}$ where $e_i = f_i \cap f_{i+1}$ ($1 \leq i \leq k$). Face f_1 contains the source point s and f_{k+1} contains the target point t .

We can rotate f_1 into the coordinate system of f_2 around the common edge e_1 of f_1 and f_2 . This process is called *unfolding*. All of the faces in the edge sequence can be unfolded into the same plane in the coordinate system of f_{k+1} . Unfolding is widely used in the algorithms for computing shortest paths and has been studied in [21, 11, 15].

Algorithm 1. *Unfolding [9]*

```

 $F := f_1;$ 
for  $i := 1$  to  $k$  do /*together  $k + 1$  faces between  $p$  and  $q$ .*/
    Rotate  $F$  around  $e_i$  until  $F$  and  $f_{i+1}$  are co-planar and
        lay on different sides of  $e_i$ ;
     $F := F \cup f_{i+1}.$ 
endfor.

```

— End of Algorithm —

Property 2.3.4. [12] *If π is a geodesic path which connects the edge sequence ξ , then the planar unfolding of π along the edge sequence ξ is a straight line segment.*

Property 2.3.4 implies that the shortest path in an unfolded edge sequence is a straight line segment.

2.4 The Visibility Diagram of a Polygon in 2-D

The visibility diagram has been used in the algorithm of Hwang et al. [25]. It is also an important data structure in our algorithm. In this section, we will start by introducing the definition of the visibility diagram as in [25] and the lemma for the visibility diagram that Hwang et al. used. We will also present the proof for this lemma which is not presented in [25]¹.

¹Since Hwang et al. [25] is a conference paper, the proofs for their lemmas were omitted. For completion of this thesis, we provide some of the proofs for those lemmas.

Let P' be a polygon whose boundary is composed of edges e_s , e_t and two polygonal chains joining the endpoints of e_s and e_t . Refer to Figure 2.2 (a).

Let domain $z = e_s \times e_t$ be a unit square, refer to Figure 2.2 (b). The horizontal line of the unit square corresponds to edge e_s and the vertical line of the unit square corresponds to edge e_t , i.e., this domain is the affine mapping of e_s and e_t on $[0, 1]$. Each point in this domain corresponds to a pair of points, one on e_s and one on e_t . As stated in Chapter 1, we use the term “the pair of points” when referring to a point in z . The unit square can be partitioned into at most two regions, such that the pairs of points in a region are visible to one another, while in the other region, the pairs of points cannot see one another.

Definition 2.4.1. Let P' be a polygon composed of edge e_s , edge e_t and two polygonal chains joining the endpoints of e_s and e_t respectively. Let $p \in e_s$ and $q \in e_t$. The function $f : e_s \times e_t \longrightarrow P' \cup \{\emptyset\}$ is defined by:

$f(p, q) = P'_\xi$ (P' is written as P'_ξ) if and only if p and q are visible to one another in P' .

$f(p, q) = P'_\emptyset$ ($\emptyset$ is written as $P'_\emptyset$) if and only if p, q can not see one another in P' .

For the pair of edges e_s and e_t , the *visibility diagram* is a partition on domain $z = e_s \times e_t$ defined by f .

Observation 1. If a pair of points ($p \in e_s, q \in e_t$) are located on the boundary between P'_ξ and $P'_\emptyset$ in the visibility diagram of e_s and e_t , then $\overline{pq}$ must pass through a vertex of polygon P' .

Figure 2.2 is an example of a polygon P' with its visibility diagram. The boundary between P'_ξ and $P'_\emptyset$ is defined by three vertices of P' . The pair of points (p, q) are located on the boundary of $P'_\emptyset$ and P'_ξ defined by vertex q_1 in P' .

Figure 2.2: The hyperbolic curves between P'_ξ and P'_ϕ produced by the vertices of P'

Lemma 2.4.1. *The boundary between P'_ξ and P'_ϕ produced by the vertices of P' is defined by hyperbolic curves.*

Proof. Let e_s be defined by the function: $y = a_1x + b_1$ and the coordinates of the endpoints of e_s be: $(p_{1a}, p_{1b}), (p_{2a}, p_{2b})$. Then we have,

$$a_1 = \frac{p_{2b} - p_{1b}}{p_{2a} - p_{1a}}, b_1 = \frac{p_{2a}p_{1b} - p_{1a}p_{2b}}{p_{2a} - p_{1a}}$$

The coordinates of the point on e_s can be written as: $(x, a_1x + b_1)$. Using the coordinate system of the affine of e_s , $x' \in [0, 1]$, the coordinates of a point on e_s are:

$$((p_{2a} - p_{1a})x' + p_{1a}, (p_{2b} - p_{1b})x' + p_{1b}).$$

Let e_t be defined by the function: $y = a_2x + b_2$ and the coordinates of the endpoints of e_t be: $(q_{1a}, q_{1b}), (q_{2a}, q_{2b})$. Then we have,

$$a_2 = \frac{q_{2b} - q_{1b}}{q_{2a} - q_{1a}}, b_2 = \frac{q_{2a}q_{1b} - q_{1a}q_{2b}}{q_{2a} - q_{1a}}$$

The coordinates of the point on e_t can be written as: $(y, a_2y + b_2)$. Using the coordinate system of the affine of e_t , $y' \in [0, 1]$, the coordinates of a point on e_t are:

$$((q_{2a} - q_{1a})y' + q_{1a}, (q_{2b} - q_{1b})y' + q_{1b}).$$

Suppose that line $\overline{pq}$ passes through vertex $r(r_a, r_b)$ of P' and intersects e_s and e_t at points

$$p((p_{2a} - p_{1a})x' + p_{1a}, (p_{2b} - p_{1b})x' + p_{1b}) \text{ and } q((q_{2a} - q_{1a})y' + q_{1a}, (q_{2b} - q_{1b})y' + q_{1b})$$

respectively.

Let line pq be defined by the function: $y = ux + w$. Because the vertex $r(r_a, r_b)$ is on $\overline{pq}$, we have: $w = r_b - ur_a$; Since p and q are on $\overline{pq}$, we have:

$$\begin{cases} (p_{2b} - p_{1b})x' + p_{1b} = u[(p_{2a} - p_{1a})x' + p_{1a}] + r_b - ur_a \\ (q_{2b} - q_{1b})y' + q_{1b} = u[(q_{2a} - q_{1a})y' + q_{1a}] + r_b - ur_a \end{cases}$$

From the first equation, we obtain:

$$u = \frac{(p_{2b} - p_{1b})x' + p_{1b} - r_b}{p_{2a} - p_{1a}x' + p_{1a} - r_a}$$

By putting u into the second equation, we obtain:

$$\begin{aligned} & [(p_{2b} - p_{1b})(q_{2a} - q_{1a}) - (q_{2b} - q_{1b})(p_{2a} - p_{1a})]x'y' + \\ & [(p_{1b} - r_b)(q_{2a} - q_{1a}) - (p_{1a} - r_a)(q_{2b} - q_{1b})]y' + \\ & [(q_{1a} - r_a)(p_{2b} - p_{1b} - q_{1b}(p_{2a} - p_{1a}))x' + (q_{1a} - r_a)(p_{1b} - r_b) - (p_{1a} - r_a)(q_{1b} - r_b)] = 0 \end{aligned}$$

This equation is a hyperbolic curve of the form: $c_1xy + c_2x + c_3y + c_4 = 0$. □

2.5 Computing the Overlay of a Set of Subdivisions

The definition of the overlay of two subdivisions has been given in [2], together with the algorithm for overlaying two subdivisions S_1 and S_2 ($Overlay(S_1, S_2)$). In this section, we will introduce the definition of overlay as in [2], and modify the structures of the subdivisions to adapt to the data structures used in the visibility diagram. (The overlay algorithm is used in the construction of the visibility diagram). We will also present the outline of the overlay algorithm in this section.

The overlay of two subdivisions S_1 and S_2 is defined to be the subdivision $O(S_1, S_2)$, such that there is a face f in $O(S_1, S_2)$, if and only if there are faces f_1 in S_1 and f_2 in S_2 , such that f is a maximal connected subset of $f_1 \cap f_2$ [2]. This statement means that, the overlay is the subdivision of the plane induced by the edges from S_1 and S_2 . In the algorithm $Overlay(S_1, S_2)$, which we will use in this thesis, the *subdivision* is a polygon whose boundary is composed of hyperbolic curves and line segments.

According to the definition of the overlay, all of the intersections of the edges in S_1 and the edges in S_2 need to be computed. The *Bentley – Ottmann* algorithm [10] is a typical algorithm used to compute the intersections of a set of line segments using the plane sweep technique. In our situation, the line segments are substituted by hyperbolic curves. For the faces of S_1 and S_2 , the process of storing a face as a set of hyperbolic curves and line segments, and reporting the boundary of the overlay, is complicated. The data structure — the doubly connected edge list is more reliable for this type of operation. The doubly connected edge list incorporates structural and topological information, such as, which curves bound a given region, which regions are adjacent, etc..

For each subdivision, three kinds of data need to be stored in three different lists. They are the *edge data*, the *vertex data* and the *face data*. For the edge data, it is convenient to

Figure 2.3: The edge data representation

view the different sides of an edge as two distinct *half-edges*, called *twins*. So that each half edge bounds only one face defined as *incident face* and has a unique *next* half-edge and a *previous* half-edge. We also store the *origin* point of each half edge in the half edge data. Refer to Figure 2.3.

The vertex data stores the *coordinates* of the vertex v and a pointer *IncidentEdge* to an arbitrary edge originating at v .

The face data stores a pointer *OuterComponent* to any half edge on its outer boundary. For the unbounded face this pointer is nil.

The $Overlay(S_1, S_2)$ algorithm combines the Bentley-Ottmann algorithm with the above data structures. This combination means that, during the plane sweep, the edge data, the vertex data and the face data are updated accordingly when an intersection is computed. This algorithm is in [2] and runs in $O((n + k) \log n)$ time where n is the complexity of the two subdivisions and k is the complexity of the overlay.

The overlay algorithm can be used for various other applications, such as performing the *union* operation on two polygons P'_1 and P'_2 . We can extract the faces in the overlay that are labeled with P'_1 or P'_2 . The complexity of the union is the same as the overlay.

Figure 2.4: (a): Two polygons with the same edges e_s and e_t ; (b): The overlay of the two polygons and the edge e_s is identified. e_t corresponds to e_{t_1} and e_{t_2} in the overlay.

Overlaying two subdivisions can be extended to the overlay of a set of polygons. Each polygon is stored in a doubly connected edge list and all of these lists are copied into a single list as the input. Each face in the output of the overlay is labeled with the names of the original polygons that contain it. Yet in this thesis, we label the faces with the names of the edge sequences instead because each polygon is obtained from its corresponding unfolded edge sequence.

2.6 The Separator of Two Polygons in 2-D

Suppose that there are two different polygons P'_1 and P'_2 with the same edge e_s and the same edge e_t in the plane. The two polygons are obtained from two unfolded edge sequences both with the same starting edge e_s and the same ending edge e_t . (Chapter 3 shows how to generate the two polygons from the unfolded edge sequences.) Each point $u(v)$ on e_s (e_t) in P'_1 corresponds to one point $u(v)$ on e_s (e_t) in P'_2 . Refer to Figure 2.4 (a).

Suppose that a pair of points ($u \in e_s, v \in e_t$) are visible to one another both in P'_1

Figure 2.5: The property of a single Voronoi cell. The perpendicular bisector of $\overline{pq}$ separates the plane into two halves. r is located on the left half plane, $|\overline{rp}| < |\overline{rq}|$.

and P'_2 . How can we determine that (u, v) has a shorter Euclidean distance in P'_1 or P'_2 ? Hwang et al. [25] proposed a lemma, stating that a hyperbolic curve between P'_1 and P'_2 can determine which polygon has a shorter distance for (u, v) . However, there are no solutions for the following questions in [25]: How to compute the separator curve; or how to store the data structures of the separator curve. This section will address these questions, and present a modified lemma and a proof for the hyperbolic curve.

We begin with the problem: For the above polygons P'_1 and P'_2 , which one has the shorter Euclidean distance for the pair of points $(u \in e_s, v \in e_t)$. This problem is solved in the following way. The two polygons are put together and the two start edges e_s are allowed to be identified as a single edge in the plane. As a result, there is one start edge e_s and two end edges e_t in the plane. Then allow the two end edges e_t to correspond to e_{t_1} and e_{t_2} and point v to correspond to $v_1 \in e_{t_1}$ and $v_2 \in e_{t_2}$, respectively. Refer to Figure 2.4 (b).

Figure 2.5 illustrates the property of a single Voronoi cell. For two given points p and q in the plane, the perpendicular bisector of the line segment $\overline{pq}$ splits the plane into two

Figure 2.6: The different kinds of separator curves: The edges between $e_s \in P_1'$ and $e_{t_1} \in P_1'$, $e_s \in P_2'$ and $e_{t_2} \in P_2'$ are omitted.

half-planes. Point r is in the open half-plane where p is located, if and only if $|\overline{rp}| < |\overline{rq}|$.

According to this principle, suppose that point v_1 has a smaller x -coordinate than that of point v_2 (v_1 is on the left side of the bisector l of $\overline{v_1v_2}$, refer to Figure 2.6 (a)). Then points located on the right side of l are closer to v_2 than to v_1 and points located on the left side of l are closer to v_1 than to v_2 . If point $u \in e_s$ is on the left side of l , then $|uv_1| < |uv_2|$, the shorter path from u to v exists in the polygon P_1' . On the contrary, if point $u' \in e_s$ is on the right side of l , then the shorter path from u' to v exists in the polygon P_2' .

Recall that the domain $z = e_s \times e_t$ is a unit square and its boundary is the affine mapping of e_s and e_t on $[0, 1]$. Domain z can be partitioned into two parts: One part stands for that the pairs of points inside the domain have shorter distances in P_1' than in

P'_2 , and the other part stands for that the pairs of points inside the domain have shorter distances in P'_2 than in P'_1 . Thus we can mark the part inside where each pair of points have the shorter distance in P'_1 with P'_1 and mark the other part with P'_2 . For the former example, (u, v) stays in the part labeled with P'_1 and (u', v) stays in the part labeled with P'_2 .

The two parts in domain z are separated by a curve. The following lemma, modified from Hwang et al. [25], indicates the shape of the separator curve.

Lemma 2.6.1. *Let $u \in e_s$ and $v_1 \in e_{t_1}, v_2 \in e_{t_2}$ (v_1 and v_2 correspond to the same point on e_t). The shorter path from u to v in P'_1 or P'_2 depends on a hyperbolic curve in the domain $z = e_s \times e_t$. This curve is determined by the pair of points (r, v) , $r \in e_s$ and r is the intersection of the perpendicular bisector of the line segment $\overline{v_1 v_2}$ and e_s .*

Proof. Let e_s be defined by the function: $y = a_1x + b_1$ and the coordinates of the endpoints of e_s be: $(p_{1a}, p_{1b}), (p_{2a}, p_{2b})$. Then,

$$a_1 = \frac{p_{2b} - p_{1b}}{p_{2a} - p_{1a}}, b_1 = \frac{p_{2a}p_{1b} - p_{1a}p_{2b}}{p_{2a} - p_{1a}}$$

The coordinates of point $r \in e_s$ (r is the intersection of the bisector of $v_1 v_2$ and e_s) can be written as: $(x, a_1x + b_1)$. Using the coordinate system of the affine of e_s , $x' \in [0, 1]$, the coordinates of $r \in e_s$ are:

$$((p_{2a} - p_{1a})x' + p_{1a}, (p_{2b} - p_{1b})x' + p_{1b}).$$

Let e_{t_1} be defined by the function: $y = a_2x + b_2$ and the coordinates of the endpoints of e_{t_1} be: $(q_{1a}, q_{1b}), (q_{2a}, q_{2b})$. Then,

$$a_2 = \frac{q_{2b} - q_{1b}}{q_{2a} - q_{1a}}, b_2 = \frac{q_{2a}q_{1b} - q_{1a}q_{2b}}{q_{2a} - q_{1a}}$$

The coordinates of $v_1 \in e_{t_1}$ can be written as: $(y, a_2y + b_2)$. Using the coordinate system of the affine of e_t , $y' \in [0, 1]$, the coordinates of v_1 are:

$$((q_{2a} - q_{1a})y' + q_{1a}, (q_{2b} - q_{1b})y' + q_{1b}).$$

Let e_{t_2} be defined by the function: $y = a_3x + b_3$ and the coordinates of the endpoints of e_{t_2} be: $(Q_{1a}, Q_{1b}), (Q_{2a}, Q_{2b})$. Then,

$$a_3 = \frac{Q_{2b} - Q_{1b}}{Q_{2a} - Q_{1a}}, b_3 = \frac{Q_{2a}Q_{1b} - Q_{1a}Q_{2b}}{Q_{2a} - Q_{1a}}$$

The coordinates of $v_2 \in t_2$ can be written as: $(y, a_3y + b_3)$. Using the coordinate system of the affine of t , $y' \in [0, 1]$, the coordinates of v_2 are :

$$((Q_{2a} - Q_{1a})y' + Q_{1a}, (Q_{2b} - Q_{1b})y' + Q_{1b}).$$

According to $|rv_1| = |rv_2|$, we get the equation:

$$\begin{aligned} & [(p_{2a} - p_{1a})y' + p_{1a} - (q_{2a} - q_{1a}x') - q_{1a}]^2 + [(p_{2b} - p_{1b})y' + p_{1b} - (q_{2b} - q_{1b}x') - q_{1b}]^2 \\ &= [(p_{2a} - p_{1a})y' + p_{1a} - (Q_{2a} - Q_{1a}x') - Q_{1a}]^2 + [(p_{2b} - p_{1b})y' + p_{1b} - (Q_{2b} - Q_{1b}x') - Q_{1b}]^2 \end{aligned}$$

That is:

$$\begin{aligned} & 2[(p_{2a} - p_{1a})(q_{2a} - q_{1a} - Q_{2a} + Q_{1a}) + (p_{2b} - p_{1b})(q_{2b} - q_{1b} - Q_{2b} + Q_{1b})]x'y' \\ & + 2[(p_{2a} - p_{1a})(q_{1a} - Q_{1a}) + (p_{2b} - p_{1b})(q_{1b} - Q_{1b})]y' + \\ & 2[(p_{1a} - q_{1a})(q_{2a} - q_{1a}) + (p_{1b} - q_{1b})(q_{2b} - q_{1b}) + \\ & (Q_{2a} - Q_{1a})(Q_{1a} - p_{1a}) + (Q_{2b} - Q_{1b})(Q_{1b} - p_{1b})]x' + \\ & Q_{1a}^2 + Q_{1b}^2 - q_{1a}^2 - q_{1b}^2 + 2(p_{1a}q_{1a} + p_{1b}q_{1b} - p_{1a}Q_{1a} - p_{1b}Q_{1b}) = 0 \end{aligned}$$

This equation is a hyperbolic curve of the form: $c_1xy + c_2x + c_3y + c_4 = 0$. Because the variables x', y' correspond to the x - coordinates of the points v and r respectively, this hyperbolic curve is defined by the pair of points (r, v) . $\square$

We define a simple method $Separator(P'_1, P'_2)$ to store all of the parameters (c_1, c_2, c_3, c_4) in the above hyperbolic curve function. This method is used to locate the four parameters, which represent the hyperbolic curve. (Note, the separator curve may be a straight line, refer to Figure 2.6 (b)). There is also a parameter *direction* stored with the curve, which indicates whether the pairs of points above the curve have shorter distances in P'_1 than in P'_2 (*direction* = 0) or if they have shorter distances in P'_2 than in P'_1 (*direction* = 1), when the curve intersects the interior of the domain $z = e_s \times e_t$ and partitions z into two parts. If the curve does not intersect the interior of z , then set *direction* = 2 that all of the pairs of points have shorter distances in P'_1 than in P'_2 , or set *direction* = 3 that all of the pairs of points have shorter distances in P'_2 than in P'_1 . Thus, the parameter values *direction* = 0 and 1 correspond to the case that the separator curve is partially (or fully) inside z , and divides z into two parts, the pairs of points in one part have shorter distances in one polygon than in the other polygon; and *direction* = 2 and 3 correspond to the case that the separator curve does not intersect z and all of the pairs of points have shorter distances in one polygon than in the other polygon.

From the function of the separator curve we can easily capture whether or not the curve intersects the interior of z . But how do we define the labels for the two parts of z or for the whole area of z ? We only need to check the position of the point (0,0) in z , which corresponds to the pair of points $u \in e_s, v \in e_t$, u and v are the starting endpoints of e_s and e_t . For example, in Figure 2.7, since $|uv_1| < |uv_2|$, u, v has shorter distance in P'_1 than in P'_2 , the part including the point (0,0) in domain z belongs to the polygon P'_1 .

We set the values of *direction* as the following:

1. *direction* = 0 when the separator curve intersects the interior of z and point (0,0) in z belongs to P'_1 ;

Figure 2.7: The parameter $direction = 0$: the part with point $(0,0)$ in the visibility diagram belongs to P'_1 .

2. $direction = 1$ when the separator curve intersects the interior of z and point $(0,0)$ in z belongs to P'_2 ;
3. $direction = 2$ when the separator curve does not intersect the interior of z and point $(0,0)$ in z belongs to P'_1 ;
4. $direction = 3$ when the separator curve does not intersect the interior of z and point $(0,0)$ in z belongs to P'_2 .

Usually, not all of the pairs of points on e_s and e_t can see one another both in P'_1 and P'_2 . Let P' be the common area in the domain z , and inside this area, each pair of points are visible to one another both in P'_1 and P'_2 . Then z is substituted by P' in the method $Separator(P'_1, P'_2)$ and there is no change to the definition of the parameter $direction$.

2.7 Finding a Monotone Path in a Trapezoidal Map

Different from the former sections, which relate to the shortest path algorithm in this thesis, this part introduces the concept of the *trapezoidal map* and the technique for finding a monotone path from a starting point P_{start} to an ending point P_{goal} in this trapezoidal map. There is a solution for finding a collision-free path in a trapezoidal map, which belongs to the robot motion problem, given by Kedem et al. [13] [14]. We modify the part of the *road map* construction, such that a *monotone path* can be computed in this road map. A path is called a *monotone path* if each point on this path always goes to the right and goes up from its left neighboring point on this path. This algorithm is used in the Fréchet-distance algorithm in Chapter 4.

In the next two paragraphs, we present some definitions stated in Kedem et al. [13, 14].

Let S be a set of polygons enclosed in a bounding box R (unit square) and assume that none of them intersects the interior of others. The *trapezoidal map* of S is obtained by drawing two vertical extensions from every vertex of a polygon in S , one extension goes upwards and one goes downwards. The extensions stop when they meet another segment of S or the boundary of R . Refer to Figure 2.8 (a).

Seidel's [22] randomized incremental algorithm is used to construct the trapezoidal map for the set of polygons. Each edge of the polygons is given a sequence number, each time one edge is added into R , and at the same time a search tree structure is maintained in order to complete point location quickly. After building the trapezoidal map, the trapezoids that lay inside the polygons are removed from the map. The remaining trapezoidal region is named *the free space*. Refer to Figure 2.8 (b). A *road map* is a graph embedded in the free space, and paths will always follow the road map. Kedem et al. constructed their road map by placing one node in the center of each trapezoid, placing one node in the middle

Figure 2.8: (a): The trapezoidation of a set of polygons (b): The free space area of the trapezoidation of a set of polygons (without the areas inside the polygons)

of each vertical extension, and joining two nodes by an arc if and only if one node is in the center of a trapezoid and the other node is on the boundary of that same trapezoid.

In order to find a monotone path in the free space area, we construct the road map in a different way from Kedem et al.. We restrict P_{start} to be the intersection of a vertical edge of a trapezoid with the bottom edge of R and P_{goal} to be the intersection of a vertical edge of a trapezoid with the top edge of R . Refer to Figure 2.9 (a). Let l be the left vertical edge of a trapezoid that firstly (from the left side of R) intersects the bottom edge of R . Let p_0 be the source node of the road map and p_0 is located at the intersection of l with the bottom edge of R . The other nodes are placed on the vertical edges of the trapezoids in the following way:

Step 1: If a vertical edge of a trapezoid intersects the top or the bottom edge of R , then place a node at this intersection, marked as *candidate*, which means that such a node stands for the possible location of P_{start} or P_{goal} . Refer to Figure 2.9 (a).

Step 2: Define the first trapezoid T_0 as the one whose left vertical edge contains p_0 .

Figure 2.9: (a)(b)(c): Placing nodes in the trapezoidal map. (d): Add arcs between the nodes to form a directed graph.

Place one node at the lowest endpoint of the right vertical edge of T_0 , mark this node and p_0 with *verstand*, which means that this node is placed on a vertical edge of a trapezoid. Refer to Figure 2.9 (b).

Step 3: If a trapezoid T 's left vertical edge is not marked and the lowest endpoint p' of the left vertical edge of T is higher than the node placed on the right vertical edge of T 's left neighbor, then place a node on p' , and mark p' with *verstand*. Refer to Figure 2.9 (b).

Step 4: If a trapezoid T 's right vertical edge is not marked and at least one of the

endpoints of the right vertical edge of T is no lower than the node p' placed on the left vertical edge of T , then place a node p on the right vertical edge of T with the smallest y – coordinate among those whose y – coordinate is no less than that of p' , and mark p with *verstand*. Go to Step 3. Refer to Figure 2.9 (c).

Step 5: Adding arcs. A directed arc is used to connect the node on the left vertical edge with the node on the right vertical edge of a trapezoid if and only if the two nodes are marked by *verstand*. Directed arcs are used to connect the node marked by *candidate* with its right horizontal neighboring node or with its vertical neighboring node, if the segment between the two nodes is the edge of a trapezoid(s). Similarly, directed arcs are used to connect the node marked by *verstand* with its vertical neighboring node marked by *verstand* if the segment between the two nodes is the edge of a trapezoid(s). Note that the direction of the arcs always goes up and goes to the right. Refer to Figure 2.9 (d).

In this way, the road map is constructed to be a directed graph (ignore the isolated nodes). Refer to Figure 2.9 (d). Then by applying breadth first search in this directed graph, a monotone path from a starting point P_{start} to a goal point P_{goal} can be defined.

We obtain the following lemma.

Lemma 2.7.1. *The path from P_{start} to P_{goal} computed in the new road map is a monotone path.*

Proof. To make the proof easier, assume that there is a path from p_0 passing through any trapezoid mentioned in this proof. Without explicit notation, *path* in the proof refers to one part of the whole path. This proof is by induction.

First, it is obvious that the path in the trapezoid T_0 , which contains p_0 , is monotone.

Figure 2.10: The cases with/without a monotone path inside a trapezoid. (a): No monotone path from T' to T through p' ; (b)(c): Monotone path from p' to p in T .

Next, without loss of generality, suppose that a path passes through trapezoid T right after passing through trapezoid T' , T' is the left neighboring trapezoid of T and there is a part of the monotone path inside T' . We need to prove the monotonicity of the path in T that originated from the right endpoint of the path in T' . Since T' is passed by the monotone path, the node p' placed on the right vertical edge of T' must be on the path according to the node placement. To form a monotone path, the successive part of the path in T starting from p' should always go up and to the right.

We use Figure 2.10 to show that this concept is maintained in all of the cases of the road map construction. Figure (a) represents the case that if the highest endpoint of the right vertical edge of T is lower than p' , then there is no monotone path entering into T through p' , thus no node is added in this case during the road map construction. Figure (b) and (c) illustrate the case that if at least one of the endpoints of the right vertical edge of T is no lower than p' , then there is one monotone path entering into T through p' . So, T does have a path from p' going up and going to the right to p .

Thus, the whole path from p_0 to P_{goal} is monotone. Since P_{start} is on the way from p_0 to P_{goal} , the path from P_{start} to P_{goal} is monotone. $\square$

Figure 2.11: (a): No monotone path from T' to T through p' , p does not exist; (b): The monotone path from p' to p in T

In this thesis, all of the edges of the polygons for trapezoidation are hyperbolic curves or ellipse arcs, some of them degenerate to line segments. The node placement needs to be changed according to the shapes of the non-vertical edges of the trapezoids. Refer to Figure 2.11. There are extra conditions added for placing a node p on the right vertical edge of a trapezoid T . Suppose that the highest point of the lower non-vertical edge of T is q , then the placed node p must be as high as q if p exists.

As in Kedem et al. [13], the complexity of finding a collision-free path inside a trapezoidal map is $O(n)$ with $O(n \log n)$ preprocessing time for building the trapezoidal map, where n is the number of the edges of all the polygons. Although our algorithm has a different road map construction from Kedem's, the time complexity is not changed. Thus,

finding a monotone path also takes linear time, even if the polygons have hyperbolic curves or ellipse arcs.

Chapter 3

Shortest Paths on a Convex Polyhedron

3.1 Outline of the Algorithm

This chapter presents an algorithm for computing shortest paths between any pair of points on two fixed edges e_s and e_t respectively on a convex polyhedron P . We design a data structure that can output shortest paths for each pair of points on e_s and e_t . First, we find all of the edge sequences from e_s to e_t by unfolding; next, for each edge sequence, we construct the visible areas in the visibility diagram of e_s and e_t ; then, we compute the overlay of all of these visible areas in one visibility diagram. Finally, each area in the overlay corresponds to an edge sequence and for each pair of points in this area, their shortest path can be computed in the corresponding edge sequence. The whole algorithm runs in $O(n^4 \log n)$ time, where n is the number of faces on P .

Input: A convex polyhedron P with n faces, two edge segments $e_s(\overline{p_s q_s})$ and $e_t(\overline{p_t q_t})$ on P .

Output: A visibility diagram of e_s and e_t , shortest path for each pair of points on e_s and

e_t can be computed from the visibility diagram.

Algorithm 2. *ShortestPaths*

1. *Use Chen and Han's algorithm [9] to compute the shortest paths from p_s to p_t and q_t , and shortest paths from q_s to p_t and q_t .*
2. *ConstructSequenceTree: Construct the edge sequence tree T of edge e_s .*
3. *Select a set of edge sequences from tree T in which all edge sequences have the same starting edge e_s and the same ending edge e_t .*
4. *VisibilityDiagram: Construct the visibility diagram of e_s and e_t for each edge sequence.*
5. *Overlay: Compute the overlay of these visible polygons in the visibility diagrams. Label each area in the overlay with one edge sequence.*
6. *Output the final visibility diagram. For the pair of points on e_s and e_t , compute their Euclidean distance in its corresponding edge sequence.*

— End of Algorithm —

Note: Shortest paths from p_s and q_s to p_t and q_t , respectively, are computed in Step 1. Without explicit claim, shortest paths computed in the algorithms to follow do not include shortest paths from the endpoints of e_s to the endpoints of e_t . After the computation in the following algorithms, the pairs of endpoints may be located in some regions in the visibility diagram. Thus, shortest paths for the pairs of endpoints can be computed from the corresponding edge sequences of these regions. However, these shortest paths may be different from those shortest paths computed in Step 1. We will eventually take the shortest paths computed in Step 1 as the shortest paths of the pairs of endpoints.

3.2 Details of the Algorithm and its Correctness

The following algorithm builds a tree called an edge sequence tree rooted at e_s . We follow the denoting of a tree node as in Chen and Han [9]. A *node* in this tree other than the root is an edge e that is represented by a triple, $(e, I_e^{e_s}, Proj_e^{I_e^{e_s}})$, where $I_e^{e_s}$ is the image source of e_s on e , $Proj_e^{I_e^{e_s}}$ is the projection of $I_e^{e_s}$ on e . Refer to Figure 3.1 (b). All the steps in Algorithm 3 are the same as the ones in Chen and Han's algorithm, except the last step for the computation of a node's children.

Algorithm 3. *ConstructSequenceTree* [9]

```

(1) root :=  $e_s$ ;

    for each edge  $e$  located in the same face as  $e_s$ ,

        insert  $(e, e_s, e)$  as one of the children of the root.

        /*they form the first level of the tree.*/

(2) for  $i = 1$  to  $n - 1$  do /* $n$  is the number of the faces */

    for all of the leaves  $(e, I_e^{e_s}, Proj_e^{I_e^{e_s}})$  at the  $i$ th level do

        /* $I_e^{e_s}$  is the image of source  $e_s$  and  $Proj_e^{I_e^{e_s}}$  is the projection of  $I_e^{e_s}$  on  $e$ . */

        unfold  $I_e^{e_s}$  about  $e$  to  $\overline{I_e^{e_s}}$ 

        /* $\overline{I_e^{e_s}}$  is co-planar with the shadowed face of  $e$  (refer to Figure 3.1).*/

        for  $e' := e_1, e_2$  do /* $e_1, e_2$  are the two other edges of this shadowed face.*/

            compute  $Proj_{e'}^{\overline{I_e^{e_s}}}$ ;

            if  $Proj_{e'}^{\overline{I_e^{e_s}}}$  nonempty then

                insert  $(e', \overline{I_e^{e_s}}, Proj_{e'}^{\overline{I_e^{e_s}}})$  as a child of  $(e, I_e^{e_s}, Proj_e^{I_e^{e_s}})$ ;

```

— End of Algorithm —

Figure 3.1: The projections of the image of e_s on two edges e_5 and e_6

Theorem 3.2.1. [9][3] *All of the shortest path edge sequences starting from e_s and ending at e_t are among the edge sequence tree computed in Algorithm 3. The algorithm runs in $O(n^3)$ time, where n is the number of faces in the convex polyhedron.*

Proof. When the tree T is built, each *node* (corresponding to an edge of P) in the tree defines an edge sequence, which consists of those edges on the way from root e_s down to this node. For a point p on face f_i such that p is contained in node N' 's shadow, there must be a geodesic path to p from point $q \in e_s$ through the edge sequence defined by node N' . By selecting the node whose edge sequence corresponds to the shortest distance between p and q , and $\overline{pq}$ is a straight line intersecting the interior of the unfolding of the edge sequence, $\Pi(p, q)$ can be defined. According to Property 2.3.1 and Property 2.3.2, no face can be passed by a shortest path more than once and a shortest path cannot pass more than n faces, where n is the total number of faces of the polyhedron, there are n loops together in Step 2.

According to Mount [3], the number of shortest path edge sequences from a fixed starting edge to other edges on P is bounded by $O(n^3)$, and there are $O(n^3)$ nodes in the edge

Figure 3.2: The bold path is a real shortest path from one endpoint of e_s to one endpoint of e_t , which is located in the edge sequence from e_1 to e_2 , not in the paths starting from e_s and ending at e_t in the edge sequence tree of e_s .

sequence tree computed in Algorithm 3. Therefore, the running time of Algorithm 3 is $O(n^3)$. $\square$

There is also a bound on the number of edge sequences from e_s to e_t as stated in [3].

Corollary 3.2.2. [3] *The number of edge sequences from e_s to e_t is $O(n^2)$.*

Note: Shortest paths from the endpoints of e_s to the endpoints of e_t may not exist in the paths from e_s to e_t in the edge sequence tree T of e_s . Refer to Figure 3.2.

Step (3) and Step (4) in Algorithm 2 state that after selecting all edge sequences starting at e_s and ending at e_t , we need to construct the visible area for each edge sequence, and then overlay these areas. The details of this step are explained in the following algorithm.

Input: Unfolded edge sequences $\xi_1, \xi_2, \dots, \xi_K$ with the starting edge e_s and the ending edge e_t ($K = O(n^2)$ according to Corollary 3.2.2).

Output: A list P'_L of polygons. All of the polygons are visible areas of these edge sequences in the visibility diagram of e_s and e_t .

Algorithm 4. *VisibilityDiagram*

(1) for $i := 1$ to K

(a) $P_{\xi_i} := \text{ShortestPathOfEndpoints}(\xi_i)$;

*/*Compute the polygon corresponding to the edge sequence ξ_i .*/*

(b) $P_{v_i} := \text{VisibleArea}(P_{\xi_i})$;

*/*Compute the visible area in the polygon P_{ξ_i} and map the area into the visibility diagram.*/*

(2) $P'_L := \text{OverlayPolygons}(P_{v_1}, P_{v_2}, \dots, P_{v_K})$;

*/*This step is introduced in Chapter 2.*/*

(3) while there is a visible polygon P_{v_j} ($1 \leq j \leq M$) in P'_L with more than one label

/ P_{v_j} is the intersecting face of two or more visible polygons. M is the number of polygons in P'_L after the overlay in Step (2).*/*

(c) $\text{curve} := \text{Separator}(\xi_u, \xi_v)$ ($u \neq v$);

*/*compute separator curves by every two labels of P_{v_j} . ξ_u, ξ_v are any two labels of P_{v_j} .*/*

(d) $P'_L := \text{Overlay}(P'_L, P_{v_j}, \text{curve})$;

*/*compute the divisions of P_{v_j} by this curve and modify the labels of P_{v_j} .*/*

— End of Algorithm —

Step 1(a) computes the polygon P_ξ for the unfolded edge sequence ξ whose boundary is shortest paths; Step 1(b) computes the visible area in the visibility diagram of e_s and e_t for polygon P_ξ .

Step 2 overlays the visible areas (polygons) for all of the edge sequences in one visibility diagram. As stated in Chapter 2, before overlay, each polygon P_{v_l} ($1 \leq l \leq K$) is labeled with an edge sequence ξ_l ; after overlay, each new polygon in P'_L is labeled with one edge

sequence or more (When a polygon P'' is contained in the intersection of several polygons, it means that each pair of points in P'' are visible to one another in all of the polygons. Because each of these intersected polygons corresponds to one edge sequence, we label P'' with these edge sequences). We define the size of P'_L to be M after the overlay.

Step 3 decomposes those polygons that have more than one label, until each of them is labeled with one shortest path edge sequence. Step 3(c) computes the separator curve of two edge sequences (introduced in Chapter 2, here we use the corresponding edge sequences of the two parameter polygons instead of polygons). Step 3(d) divides the polygon P_{v_j} into several parts according to the *curve*, and marks each new part of it with new labels. Following this step, the old P_{v_j} is deleted from P'_L , and the new subdivisions are added to P'_L . The size of $P'_L(M)$ is changed to a new value correspondingly.

After Step 3, each polygon in the visibility diagram is labeled with one shortest path edge sequence. We now illustrate the details of each step.

In Step 1, the edge sequence ξ is composed of e_s , e_t and the edges of the unfolded faces lying between e_s and e_t . The outer boundary of this unfolding forms a polygon, which consists of $e_s(p_s q_s)$, $e_t(p_t q_t)$ and the edges connecting the endpoints of the unfolded face edges. The output is the two shortest paths between p_s and p_t , and between q_s and q_t , refer to Figure 3.3. According to Guibas et al. [16], we obtain the following theorem.

Lemma 3.2.3. *The $ShortestPathOfEndpoints(\xi)$ function can be computed in linear time in the number of edges in ξ .*

The shortest paths together with e_s and e_t form a new polygon. We need to compute the visible area of this polygon.

Figure 3.3: An edge sequence after planar unfolding. (a) is the outer polygon of the sequence; (b) is the resulting polygon after Step 1(a) whose boundary is shortest paths.

Input: Given a simple polygon P' and the starting edge $e_s(p_s q_s)$, the ending edge $e_t(p_t q_t)$, and all of the vertices on the boundary of P' are stored in two lists: list1 $\{p_1, p_2, \dots, p_n\}$; list2 $\{q_1, q_2, \dots, q_m\}$ ($p_1 = p_s, p_n = p_t, q_1 = q_s, q_m = q_t$.) where $p_2, \dots, p_{n-1}$ are the vertices between p_s and p_t along one side of the boundary of P' and $q_2, \dots, q_{m-1}$ are the vertices between q_s and q_t along the other side of the boundary of P' . (Refer to Figure 3.4).

Output: Visible polygon P'_v inside the visibility diagram of e_s and e_t , whose boundary is stored in an edge list L . Each point inside the polygon P'_v corresponds to a pair of points on e_s and e_t that can see one another.

Algorithm 5. *VisibleArea*

```

(1) tangents := computeTangent(list1, list2);
    /*The number of tangents can be 0,1,2.*/
(2) for  $i := 1$  to  $n - 2$  /*Operating on the points of list1.*/
    (a)  $e_1 := \overline{p_i p_{i+1}}$ ;  $e_2 := \overline{p_{i+1} p_{i+2}}$ 
    (b) if the intersection of  $e_1$  with  $e_s$  or  $e_t$  is not in the interior of  $e_s$  or  $e_t$ 
        if  $p_{i+1}$  is one of the endpoints of tangents
    
```

Figure 3.4: A polygon with two shortest paths on its boundary

$e_1 := \text{tangents}$ (if there are two tangents, e_1 is set to the corresponding one);
 else $e_1 := \emptyset$;
 (c) if the intersection of e_2 with e_s or e_t is not in the interior of e_s or e_t
 if p_{i+1} is one of the endpoints of tangents
 $e_2 := \text{tangents}$ (if there are two tangents, e_2 is set to the corresponding one);
 else $e_2 := \emptyset$;
 (d) compute the hyperbolic curve defined by e_1 , e_2 and p_{i+1} if neither e_1 nor e_2 is empty.
 (e) add the curve in the list L .
 /*The operation on the points of list2 is omitted, which is the same as the operation on
 the points of list1.*/
 (3) Adjust L by adding the segment of the outer boundary of the visibility diagram into L
 when the number of tangents $\neq 2$
 /*Before adding a new edge into L , the edges in L cannot form the polygon P'_v .*/
 — End of Algorithm —

Lemma 3.2.4. *Algorithm 5 defines a visible polygon in the visibility diagram, inside which the pairs of points on e_s and e_t can see one another.*

Figure 3.5: Several cases for the visible polygon (bold areas). There are two tangents in (a) and one tangent in (b) and (d) respectively. No hyperbolic curve is in (c); so the visible polygon is the whole square.

Proof. According to Lemma 2.2.1 in Chapter 2, we know that the shortest path between two vertices in a polygon P' is an inward convex polygonal chain. Therefore, the function *computeTangent* computes the tangents between two convex chains. This computation can be done in linear time by a variation of Graham's Scan algorithm and the number of *tangents* can be 0, 1 or 2.

This algorithm maintains the following invariant: Each vertex of the polygon P' defines an edge of the visible polygon P'_v , which is a hyperbolic curve (refer to Lemma 2.4.1 in Chapter 2), or this vertex has no contribution to the visible polygon. There are three cases:

1. This curve is determined by the two edges $p_{i-1}p_i$ and $p_i p_{i+1}$ and both intersect the interior of e_s and e_t ;
2. One of the edges $p_{i-1}p_i$ and $p_i p_{i+1}$ does not intersect the interior of e_s or e_t , and there is a tangent passing through the vertex p_i . This hyperbolic curve is determined by the tangent and the edge, which intersects the interior of e_s or e_t ;

3. A curve is not produced when neither $p_{i-1}p_i$ nor $p_i p_{i+1}$ intersects the interior of e_s or e_t and a tangent does not pass through vertex p_i .

It is easy to see that the invariant is maintained in the three cases.

Now we prove the last step of the algorithm. Since the number of tangents of the two convex chains can be 0, 1 or 2, there are three cases for the visible polygon. When there are two tangents for the two convex chains, they intersect e_s at a_1, a_2 and intersect e_t at b_1, b_2 . Only the pair of points $a \in e_s$ and $b \in e_t$, where $a_1 \leq a \leq a_2$ and $b_1 \leq b \leq b_2$, have the possibility of being visible to one another. Therefore, the two pairs of points (a_1, b_2) and (a_2, b_1) are the vertices of the hyperbolic curves and the hyperbolic curves themselves form the visible polygon. Thus there is no need to adjust the boundary of the polygon, refer to Figure 3.5(a).

If there is only one tangent between the two convex chains, then at least two endpoints of e_s and e_t are on the hyperbolic curves in the visibility diagram. Thus, the hyperbolic curves intersect the outer boundary of the visibility diagram at no less than two intersections. The polygon is composed of hyperbolic curves and the boundary of the visibility diagram, refer to Figure 3.5(b) and (d).

If the two convex chains have no tangent between them, then all of the endpoints of e_s and e_t are on the hyperbolic curves in the visibility diagram. Therefore, the hyperbolic curves intersect the outer boundary of the visibility diagram at four intersections. The polygon is composed of hyperbolic curves and the boundary of the visibility diagram. Figure 3.5(c) is a special case of this situation.

Thus, if there is one tangent or no tangent between the two convex chains, the visible polygon P'_v is formed by all of the hyperbolic curves and the outer boundary of the visibility diagram. □

Step 3(c) is the method $Separator(\xi_u, \xi_v)$, which stores the parameters of the separator curve function. The separator curve of ξ_u and ξ_v , in the visibility diagram of e_s and e_t , is a hyperbolic curve as stated in Lemma 2.6.1. Because the pairs of points in P_{v_j} can see one another in both ξ_u and ξ_v , the separator curve divides the points of P_{v_j} into two parts if it intersects the interior of polygon P_{v_j} . One part of P_{v_j} is labeled with ξ_u , the pairs of points in this part have shorter distances in ξ_u than in ξ_v ; The other part of P_{v_j} is labeled with ξ_v , the pairs of points in this part have shorter distances in ξ_v than in ξ_u . There is a parameter *direction* that stores the label information in the method. Suppose that after unfolding ξ_u and ξ_v into a common plane and $e_s \in \xi_u$ and $e_s \in \xi_v$ are identified to be one edge, the x -coordinate of the left endpoint of e_t in ξ_u is smaller than that of the left endpoint of e_t in ξ_v , refer to Figure 2.7, e_s and e_{t_1} belong to ξ_u , e_s and e_{t_2} belong to ξ_v . If *direction* = 0, then the part closer to the point $(0,0)$ in P_{v_j} is labeled with ξ_u . If *direction* = 1, the part closer to the point $(0,0)$ in P_{v_j} is labeled with ξ_v ; if *direction* = 2, P_{v_j} is labeled with ξ_u ; if *direction* = 3, P_{v_j} is labeled with ξ_v . Refer to Figure 3.6.

Next, we will describe the algorithm of overlaying the separator curve of ξ_1 and ξ_2 to the visible polygon P_v , which has labels of ξ_1 and ξ_2 in P'_L .

Input: A list P'_L of polygons, one fixed polygon $P_v \in P'_L$ which is labeled with the edge sequence ξ_1 and ξ_2 , and a hyperbolic separator curve l of ξ_1 and ξ_2 .

Output: New list P'_L .

Algorithm 6. $Overlay(P'_L, P_v, l)$

/ P_v is labeled with two edge sequences ξ_1 and ξ_2 . The curve l is the separator of ξ_1 and ξ_2 . */*

Figure 3.6: One visible polygon P_v with two original labels ξ_1 and ξ_2 . The separator curve of ξ_1 and ξ_2 . The different values of the parameter *direction* of the separator curve. Case A, the pairs of points in the area below the separator have shorter distances in ξ_1 , the other part belongs to ξ_2 ; Case B, opposite to Case A; Case C, the whole area belongs to ξ_1 ; Case D, the whole area belongs to ξ_2 .

(1) Compute the divisions of P_v separated by the curve l ;

(2) if l intersects P_v

determine in which divisions the label ξ_2 needs to be deleted,

then in the other divisions the label ξ_1 needs to be deleted.

*(according to the value of the parameter *direction* of the curve l , 0 or 1).*

delete P_v from P'_L , add the new divisions into P'_L .

(3) if l does not intersect P_v

*if the parameter *direction* = 2 then delete label ξ_2 from P_v ;*

*else delete label ξ_1 from P_v . /*direction=3*/*

(4) return P'_L .

— End of Algorithm —

Lemma 3.2.5. *Algorithm 6 computes the subdivisions of P_v correctly in $O(\log n)$ time, where n is the number of faces on P .*

Proof. The polygon P_v is intersected by curve l and divided into at least two parts if l intersects P_v . The curve l has a parameter *direction* to indicate what the new divisions of P_v should be labeled by deleting the corresponding label from the original labels of P_v . The different cases are illustrated in Figure 3.6. Step 1 takes $O(\log n)$ time, and each of the other steps in this algorithm takes $O(1)$ time, thus, time complexity of this algorithm is $O(\log n)$. $\square$

Lemma 3.2.6. *The time complexity of Algorithm 4 is $O(n^4 \log n)$.*

Proof. In Step 1, there are $O(n^2)$ edge sequences (where n is the number of the faces on P), the time spent on computing the corresponding polygon whose boundary is shortest paths for each edge sequence is linear in the number of edges in each edge sequence according to Lemma 3.2.3, and computing the visible area for each polygon takes time that is also linear in the number of edges in each edge sequence. Since the number of edges in each edge sequence is at most n , Step 1 takes $O(n^3)$ time in total;

We analyze the time complexity for Step 2 and Step 3 in the following way. The numbers of polygons before, during and after the overlay are $O(n^2)$, which can not exceed the number of the original edge sequences. After overlay, the regions in the visibility diagram are separated by the hyperbolic curves. Since every pair of polygons can intersect one another, the number of the hyperbolic curves is $O(n^4)$. According to Lemma 3.2.5, the intersections of each of the polygons with one hyperbolic curve take $O(\log n)$ time. Therefore, Step 2 and 3 take $O(n^4 \log n)$ time;

Together, the running time of Algorithm 4 is $O(n^4 \log n)$. $\square$

Theorem 3.2.7. *For every pair of points $p \in e_s$ and $q \in e_t$, Algorithm 2 computes the shortest path $\Pi(p, q)$ correctly.*

Proof. The shortest paths from the endpoints of e_s to the endpoints of e_t are computed in [9]. Below, the pair of points p, q does not include any of the pairs of the endpoints of e_s and e_t .

According to Theorem 3.2.1, the shortest path edge sequence of $\Pi(p, q)$ must be in one of the edge sequences computed in Step (2) of Algorithm 2.

Suppose the shortest path $\Pi(p, q)$ is in the edge sequence ξ_k and $\overline{pq}$ is a straight line in the unfolding of ξ_k . In Step (4), the visible polygon P_{v_k} of ξ_k is computed. Since p and q can see one another, $(p, q) \in P_{v_k}$.

Suppose that after computing all of the regions in the visibility diagram by overlaying all of the visible polygons of the edge sequences, the pair of points (p, q) are located in the polygon p_0 . There are two cases:

1. After the overlay, p_0 has only one label ξ_k . That means (p, q) can only see one another in ξ_k , p and q are not visible in all other edge sequences.
2. After the overlay, p_0 is labeled with several edge sequences, that means (p, q) can see one another in several edge sequences. Let the number of the edge sequences be m and without loss of generality, the edge sequences of the labels are written as: $\xi_1, \xi_2, \dots, \xi_m$. Then corresponding to the m edge sequences, $\overline{pq}$ exists in the visible polygons that can be written as: $\{P_{v_1}, P_{v_2}, \dots, P_{v_k}, \dots, P_{v_m}\} (1 \leq k \leq m)$.

In case 1, since ξ_k is the only label of p_0 , p and q are only visible in ξ_k . Therefore, we can define the shortest path $\Pi(p, q)$ in the edge sequence ξ_k .

In case 2, since $\overline{pq}$ is labeled with m edge sequences, p and q can see one another in m visible polygons. For every two of the m polygons P_{v_i} and P_{v_j} ($j \neq i, 1 \leq j \leq m, 1 \leq i \leq m$), there is a separator curve between them. The curve determines in which corresponding edge sequence, either ξ_i or ξ_j , p and q have a shorter distance. Without loss of generality,

suppose that p and q have a shorter distance in P_{v_i} than in P_{v_j} . Then after overlay the separator *curve* in Step 3(d) of Algorithm 4, (p, q) is located in a subdivision of P_0 and this subdivision is labeled with all of the edge sequences of the labels of P_0 except ξ_j . For the new subdivision, because there is more than one label with the subdivision, Step 3(d) will continue to execute, until (p, q) is labeled with only one edge sequence, which is ξ_k . Then we can define the shortest path $\Pi(p, q)$ in the edge sequence ξ_k .

Thus, in either of the two cases, the shortest path $\Pi(p, q)$ can be computed from the edge sequence ξ_k in Algorithm 2. $\square$

Lemma 3.2.8. *The time complexity of Algorithm 2 is $O(n^4 \log n)$.*

Proof. The complexity of the entire algorithm is determined by:

1. The time required to construct the edge sequence tree in Algorithm 3, which is $O(n^3)$;
2. The time required to construct the visibility diagram in Algorithm 4, which is $O(n^4 \log n)$.

Therefore, the complexity of the entire algorithm is $O(n^4 \log n)$. $\square$

Chapter 4

Fréchet Distance on a Convex Polyhedron

This chapter introduces the metric *Fréchet – distance*, a measure for the resemblance of curves in arbitrary dimensions, which is compatible with parameterizations of the curves. For two paths Z and Z' consisting of z and z' edges, respectively, on the convex polyhedron P , an algorithm of running time $O(zz'n^2 \log n \log(zz'n^2))$ measuring Fréchet-distance between Z and Z' is developed, where n is the number of the faces on P . The algorithm is based on the approach of Alt and Godau [8].

4.1 Fréchet Distance Between Two Polygonal Curves

Parameterization is useful, for situations in which performing mathematical operations on a curve of an arbitrary shape turns out to be difficult. A polygonal curve $Z : [0, z]$ is a continuous and piecewise linear curve made of z connected segments. Such a curve can be parameterized with a parameter $a \in \mathbb{R}$ such that $Z(a)$ refers to a given position on the curve, $Z(0)$ refers to the first vertex of the polygonal curve and $Z(z)$ refers to its last vertex, as shown in Figure 4.1 and Figure 4.2 [17].

Figure 4.1: The parameterization of a polygonal curve

As stated in Chapter 1, Fréchet distance can be illustrated as the dog-leash distance. Suppose that the person is constrained to walk on a polygonal curve Z of length z and that the dog has to walk on a polygonal curve Z' of length z' . The position of the person and the dog on their respective curves can be modelled as a function of t by using two continuous and increasing functions $\alpha(t)$ and $\beta(t)$, where $\alpha(0) = 0, \alpha(1) = z, \beta(0) = 0$ and $\beta(1) = z'$. The position of the person as a function of time is given by $Z(\alpha(t))$ and the position of the dog by $Z'(\beta(t))$. While many functions $\alpha(t)$ and $\beta(t)$ can be used, some of them will cause the distance between the dog and the person to be large at some instances, whereas other functions will not. The process of finding Fréchet distance actually corresponds to finding two parameterized functions $\alpha(t)$ and $\beta(t)$ that minimize the maximum distance between the person and the dog. Mathematically, let V denote an arbitrary Euclidean vector space, Fréchet distance between two curves is defined as:

Definition 4.1.1. [8] A curve is a continuous mapping $f : [a, b] \rightarrow V$ with $a, b \in \mathbb{R}$ and $a < b$. A polygonal curve is a curve $f : [0, n] \rightarrow V$, such that for all $i \in \{0, 1, \dots, n-1\}$ each $f|_{[i, i+1]}$ is affine, i.e. $f(i + \lambda) = (1 - \lambda)f(i) + \lambda f(i + 1)$ for all $\lambda \in [0, 1]$, where $n \in \mathbb{N}$

Figure 4.2: The parameterization of two polygonal curves

is called the length of f .

Definition 4.1.2. [8] Let $f : [a, a'] \rightarrow V, g : [b, b'] \rightarrow V$ be curves. Then $\delta_F(f, g)$ denotes their Fréchet-distance, defined as

$$\delta_F(f, g) := \inf_{\substack{\alpha: [0,1] \rightarrow [a, a'], \\ \beta: [0,1] \rightarrow [b, b']}} \max_{t \in [0,1]} \|f(\alpha(t)) - g(\beta(t))\|$$

where $\|\cdot\|$ denotes Euclidean distance, α, β range over continuous and increasing functions with $\alpha(0) = a, \alpha(1) = a', \beta(0) = b$ and $\beta(1) = b'$ only.

We start from an easier variant of the problem, namely the following decision problem:

Given: polygonal curves Z and Z' and some $\epsilon > 0$

Determine: whether $\delta_F(Z, Z') \leq \epsilon$

First we look at the case when $Z : [0, z]$ and $Z' : [0, z']$ are polygonal curves in 2-dimension, z and z' are the number of edges of Z and Z' , respectively. In order to solve the decision problem, we first consider the case where $z = z' = 1$, i.e. Z and Z' are line segments. The set

$$F_\epsilon = \{(s, t) \in [0, 1]^2 \mid d(Z(s), Z'(t)) \leq \epsilon\} \quad (4.1.1)$$

Figure 4.3: The free space for two segments. As the distance between the two *square* points on the two segments is smaller than ϵ , the corresponding *square* point in the unit square belongs to the free space. Conversely, the *dot* point in the unit square does not belong to the free space, as the distance between the two corresponding *dot* points on the two segments is larger than ϵ .

describes all of the pairs of points on Z and Z' , whose Euclidean distance is at most ϵ . Figure 4.3 shows line segments Z, Z' , a distance $\epsilon > 0$, and F_ϵ being the white area within the unit square; subsequently called the free space diagram.

Lemma 4.1.1. [8] *In the case of line segments Z and Z' , the free space F_ϵ corresponding to Z and Z' is the intersection of the unit square with an ellipse, possibly degenerated to the space between two parallel lines. F_ϵ is convex.*

The proof of this lemma is in [8]. The function of the ellipse is given here since it will be used in the case of Z and Z' being two paths on the convex polyhedron.

Let the coordinates of the endpoints of Z be: $(z_{1a}, z_{1b}), (z_{2a}, z_{2b})$ and the coordinates of the endpoints of Z' be: $(z'_{1a}, z'_{1b}), (z'_{2a}, z'_{2b})$. Let Z be defined by the function: $y = a_1x + b_1$

and Z' be defined by the function: $y = a_2x + b_2$. Therefore,

$$a_1 = \frac{z_{2b} - z_{1b}}{z_{2a} - z_{1a}}, b_1 = \frac{z_{2a}z_{1b} - z_{1a}z_{2b}}{z_{2a} - z_{1a}}$$

$$a_2 = \frac{z'_{2b} - z'_{1b}}{z'_{2a} - z'_{1a}}, b_2 = \frac{z'_{2a}z'_{1b} - z'_{1a}z'_{2b}}{z'_{2a} - z'_{1a}}$$

The points located in the 2-dimensional coordinate system of the parameterization of Z and Z' satisfy: $x' \in [0, 1], y' \in [0, 1]$, thus the coordinates of the point $p \in Z$ are:

$$((z_{2a} - z_{1a})x' + z_{1a}, (z_{2b} - z_{1b})x' + z_{1b}).$$

The coordinates of the point $q \in Z'$ are:

$$((z'_{2a} - z'_{1a})y' + z'_{1a}, (z'_{2b} - z'_{1b})y' + z'_{1b}).$$

According to $d(p, q) \leq \varepsilon$, we have:

$$[(z_{2a} - z_{1a})x' + z_{1a} - (z'_{2a} - z'_{1a})y' - z'_{1a}]^2 + [z_{2b} - z_{1b})x' + z_{1b} - (z'_{2b} - z'_{1b})y' - z'_{1b}]^2 \leq \varepsilon^2$$

which is:

$$\begin{aligned} & [(z_{2a} - z_{1a})^2 + (z_{2b} - z_{1b})^2]x'^2 + [(z'_{2a} - z'_{1a})^2 + (z'_{2b} - z'_{1b})^2]y'^2 - \\ & 2[(z_{2a} - z_{1a})(z'_{2a} - z'_{1a}) + (z_{2b} - z_{1b})(z'_{2b} - z'_{1b})]x'y' + \\ & 2[(z_{1a} - z'_{1a})(z_{2a} - z_{1a}) + (z_{1b} - z'_{1b})(z_{2b} - z_{1b})]x' - \\ & 2[(z_{1a} - z'_{1a})(z'_{2a} - z'_{1a}) + (z_{1b} - z'_{1b})(z'_{2b} - z'_{1b})]y' + \\ & (z_{1a} - z'_{1a})^2 + (z_{1b} - z'_{1b})^2 \leq \varepsilon^2 \end{aligned}$$

Since:

$$(z_{2a} - z_{1a})^2 + (z_{2b} - z_{1b})^2 = |Z|^2; (z'_{2a} - z'_{1a})^2 + (z'_{2b} - z'_{1b})^2 = |Z'|^2$$

Figure 4.4: The free space for two polygonal curves

The function that represents the ellipse is:

$$\begin{aligned}
 &|Z|^2 x'^2 + |Z'|^2 y'^2 - 2[(z_{2a} - z_{1a})(z'_{2a} - z'_{1a}) + (z_{2b} - z_{1b})(z'_{2b} - z'_{1b})]x'y' + \\
 &2[(z_{1a} - z'_{1a})(z_{2a} - z_{1a}) + (z_{1b} - z'_{1b})(z_{2b} - z_{1b})]x' - \\
 &2[(z_{1a} - z'_{1a})(z'_{2a} - z'_{1a}) + (z_{1b} - z'_{1b})(z'_{2b} - z'_{1b})]y' + \\
 &(z_{1a} - z'_{1a})^2 + (z_{1b} - z'_{1b})^2 \leq \epsilon^2
 \end{aligned}$$

Inequation 4.1.1 can be extended to arbitrary polygonal curves Z, Z' , of length z, z' respectively:

$$F_\epsilon = \{(s, t) \in [0, z] \times [0, z'] | d(Z(s), Z'(t)) \leq \epsilon\}$$

The free space corresponding to the curves Z and Z' is the combination of the free spaces of all of the pairs containing one segment of Z and one segment of Z' . Figure 4.4 shows two polygonal curves and their corresponding free space diagram for a given value of ϵ .

Alt and Godau's algorithm [8] is based on the following straightforward observation:

Figure 4.5: The values that must be calculated

Observation 2. For polygonal curves Z and Z' , $\delta_F(Z, Z') \leq \epsilon$, only if there exists a curve within the corresponding free space F_ϵ from $(0, 0)$ to (z, z') that is monotone in both coordinates.

In the person-dog example, the monotonic condition corresponds to the fact that neither the dog nor the person is allowed to go backwards. Figure 4.4 shows such a curve, proving that for the person-dog example, Fréchet distance is almost ϵ .

Let $F_\epsilon(i, j)$, with $1 \leq i \leq z, 1 \leq j \leq z'$, refer to the free space unit cell corresponding to the segment $Z(i-1)Z(i)$ and the segment $Z'(j-1)Z'(j)$. If the free space intersects the edge of a cell, it means that it is possible to enter into that cell through this edge. As the free space of each unit cell is convex, it is an easy matter to determine if there is a monotone curve that goes from $(0,0)$ to (z, z') by computing all of the intersections of the free space with the contour of each cell. Figure 4.5 illustrates the values that must be calculated in order to identify the passages between neighboring cells. $L_{i,j}^F$ refers to the left line segment bounding $F_\epsilon(i, j)$ and $B_{i,j}^F$ refers to the bottom line segment.

Figure 4.6: The monotone free space of two polygonal curves. The paths from $(0, 0)$ to any point in the white area must always go up and go to the right at anytime

Furthermore, there is another function:

$$R_\epsilon = \{(s, t) \in F_\epsilon | \exists \text{ a monotone curve within } F_\epsilon \text{ from } (0, 0) \text{ to } (s, t)\}$$

R_ϵ can easily be computed from F_ϵ . See Figure 4.6. Almost the same notation is used for a single cell as in Figure 4.5 to designate the values that have to be calculated for the monotone free space. That is, $L_{i,j}^R$ refers to the left line segment bounding $R_\epsilon(i, j)$ and $B_{i,j}^R$ refers to the bottom line segment. For a cell of segments $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$, $L_{i,j}^R$ and $B_{i,j}^R$ can be easily computed from $L_{i,j}^F$, $B_{i,j}^F$, $L_{i,j-1}^R$ and $B_{i-1,j}^R$.

4.2 Fréchet Distance of Two Paths on a Convex Polyhedron

When Z and Z' are two paths on the convex polyhedron P , the style of the free space is quite different from that in which when Z and Z' are polygonal curves in 2-dimension. As stated in Chapter 1, a person is walking on the path Z and a dog is walking on the path

Z' , the leash between them is measured by the length of the shortest path on the surface of P .

In Chapter 3, the shortest paths between two edges $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$ on P have been computed, and for any pair of points on the two edges, the shortest path between them is obtained from the corresponding edge sequence of the polygon in the visibility diagram. The outer boundary of the visibility diagram is a unit square and exactly the same square as the one where the free space of the two edges are located. Because the polygons in the visibility diagram belong to different unfolded edge sequences, and each unfolded edge sequence has a different coordinate system from one another in the plane, each polygon needs to compute its own corresponding free space area. Following this computation, we unite all of the free space areas of the polygons (simply introduced in Chapter 2), refer to Figure 4.7 (a).

As in Figure 4.5, four boundary values of each unit cell must be computed. If a monotone path enters into the cell of $Z(i)Z(i+1)$ and $Z'(j)Z'(j+1)$ from $L_{i,j}^F$ or $B_{i,j}^F$, and leaves the cell from $B_{i+1,j}^F$ or $L_{i,j+1}^F$, then at least one pair of $(L_{i,j}^F, B_{i+1,j}^F)$, $(B_{i,j}^F, B_{i+1,j}^F)$, $(L_{i,j}^F, L_{i,j+1}^F)$ and $(B_{i,j}^F, L_{i,j+1}^F)$ must be nonempty. To be concise, the necessary condition for the possibility of the monotone path to exist is in one of the following conditions:

1. None of the four boundary values is empty;
2. $L_{i,j}^F, B_{i+1,j}^F$ are nonempty;
3. $B_{i,j}^F, B_{i+1,j}^F$ are nonempty;
4. $L_{i,j}^F, L_{i,j+1}^F$ are nonempty;
5. $B_{i,j}^F, L_{i,j+1}^F$ are nonempty;

Figure 4.7: The trapezoidation of a free space after union. (a) : The free space, which contains some blank areas. (b) : After trapezoidation and executing BFS (the dashed line is one of the paths), $B_{i+1,j}^R = c_{i+1}^j d_{i+1}^j$.

6. $L_{i,j}^F, B_{i+1,j}^F, B_{i,j}^F$ are nonempty;
7. $L_{i,j}^F, B_{i+1,j}^F, L_{i,j+1}^F$ are nonempty;
8. $L_{i,j+1}^F, B_{i+1,j}^F, B_{i,j}^F$ are nonempty;
9. $L_{i,j}^F, L_{i,j+1}^F, B_{i,j}^F$ are nonempty;

We use the case of $B_{i,j}^F, B_{i+1,j}^F$ nonempty to illustrate how to find the monotone path in this cell.

Inside the union of the free space (refer to Figure 4.7 (a)), there are some blank areas (composed of hyperbolic curves and/or ellipse arc(s), the distances of the pairs of points inside this area are bigger than ϵ). First, we construct the trapezoidal map in the free space (introduced in Chapter 2), by drawing vertical extensions of each vertex of the union (including the blank areas) until it hits the edge(s) of other blank areas, or the boundary of the free space or the outer boundary of the unit square. Second, we modify $B_{i,j}^F$ by

computing the intersection of $B_{i,j}^F$ and the former value (old $B_{i,j}^R$) from the cell of $Z(i)Z(i+1)$ and $Z'(j-1)Z'(j)$. This process will ensure the succession of the monotone path from the cell of $Z(i)Z(i+1)$ and $Z'(j-1)Z'(j)$ to the cell of $Z(i)Z(i+1)$ and $Z'(j)Z'(j+1)$ through $B_{i,j}^F$ if it exists, respectively. Third, we build a road map in the trapezoidal map and find a monotone path from the points on $B_{i,j}^F$ to the points on $B_{i+1,j}^F$. The data structures of the road map and the details of the algorithm were introduced in Chapter 2. P_{start} can be set to be the leftmost point c_i^j of $B_{i,j}^F$ at first, then we use BFS once to find the leftmost point c_{i+1}^j on $B_{i+1,j}^F$ so that the point c_i^j can be reached in the road map. If the point c_{i+1}^j does not exist, then the point P_{start} is moved to its right neighbor on $B_{i,j}^F$, which is the intersection of the vertical edge of the trapezoid and $B_{i,j}^F$. We try BFS again, until both points c_i^j and c_{i+1}^j exist. In the same way, points d_i^j of $B_{i,j}^F$ and d_{i+1}^j of $B_{i+1,j}^F$ can be computed. Thus, $B_{i+1,j}^R = c_{i+1}^j d_{i+1}^j$. Refer to Figure 4.7 (b).

Therefore, the algorithm for solving the decision problem is summarized as the following.

Algorithm 7. *Decision Problem*

for $i = 1$ to z

for $j = 1$ to z'

- (1) Compute each of the free space areas of the polygons in the visibility diagram of the cell $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$, then compute the union of the free space areas. If the former $B_{i,j-1}^R$ and/or $L_{i-1,j}^R$ are nonempty, modify the values of $B_{i,j-1}^F$ and/or $L_{i-1,j}^F$ to be $B_{i,j-1}^R \cap B_{i,j-1}^F$ and/or $L_{i-1,j}^R \cap L_{i-1,j}^F$ accordingly. If $i = j = 1$, set $B_{0,1}^R = B_{0,1}^F$ and $L_{0,1}^R = L_{0,1}^F$.
- (2) If the boundary of the free space satisfies one of the nine conditions above, compute the trapezoidal map and road map of the free space area.

(3) Use breadth first search to find the monotone path in this trapezoidal map and record the values of $B_{i,j}^R$ and $L_{i,j}^R$.

Answer "Yes" if $(z, z') \in L_{z+1, z'}^R$; "No" otherwise.

— End of Algorithm —

Lemma 4.2.1. *Algorithm 7 computes the monotone path in the free space diagram of the two paths Z and Z' correctly. The time complexity of the algorithm is $O(zz'n^2 \log n)$ where z and z' are the number of the edges in Z and Z' , and n is the number of faces on P .*

Proof. To ensure that there exists a monotone path in the free space diagram, each unit cell of the free space that the path passes through should also be monotone.

For a cell of $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$, the corresponding visibility diagram of edges $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$ is composed of a set of polygons. Each polygon corresponds to a shortest path edge sequence. For a given $\epsilon > 0$, the free space area of a polygon is the intersection of this polygon and the free space of the two edges $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$ (which is an ellipse) in the coordinate system of this polygon's corresponding unfolded edge sequence, such that their intersection is the area composed of hyperbolic curves and ellipse curves. In this way, each polygon in the visibility diagram of $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$ can have its own free space. Thus, the union of these free spaces will always exist. Refer to Figure 4.7.

To find the monotone path in the cell of $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$, Step (2) and (3) ensure that the path in this cell always goes up and goes to the right, which has been proved by Lemma 2.7.1 in Chapter 2.

Thus, the cell of $Z(i-1)Z(i)$ and $Z'(j-1)Z'(j)$ has a monotone path. Since the boundary value of $B_{i,j-1}^F$ and $L_{i-1,j}^F$ are modified in the first step of the algorithm, this

modification ensures that the path between two neighboring cells continuously goes up and goes to the right. Thus, the algorithm correctly computes the monotone path.

The complexity of this algorithm: In Step (1), there are $O(n^2)$ polygons in the visibility diagram of $Z(i-1)(i)$ and $Z'(j-1)Z'(j)$. Each of the polygons' free spaces is computed in $O(1)$ time (computing the intersections of the ellipse and the hyperbolic curves), the union of the free spaces is computed in $O(n^2 \log n)$ time, and the values of $B_{i,j}^F$ and $L_{i,j}^F$ can be adjusted in $O(1)$ time. Therefore, together, Step (1) takes $O(n^2 \log n)$ time where n is the number of the faces on P ;

In Step (2), since computing the trapezoidal map of a set of m segments takes $O(m \log m)$ time, and there are $O(n^2)$ polygons in the visibility diagram, computing the trapezoidal map and road map of the free space in a cell takes $O(n^2 \log n)$ time;

Step (3) is a standard BFS algorithm, which takes $O(n^2)$ time.

Thus, the complexity of this algorithm is $O(zz'n^2 \log n)$. $\square$

Next we define the candidate values of Fréchet distance. Assume that we start with $\epsilon = 0$ and continuously increase ϵ . Then the free space becomes larger and larger and we want to determine the smallest ϵ such that it contains a monotone curve from $(0,0)$ to (z, z') . Observe that the smallest ϵ can occur in one of the following cases: [8]

1. ϵ is minimal with $(0,0) \in F_\epsilon$ and $(z, z') \in F_\epsilon$;
2. ϵ is minimal with $L_{i,j}^F$ or $B_{i,j}^F$ becomes nonempty for some pair (i, j) .

Clearly, case 1 states that $\delta_F(Z, Z')$ must be greater or equal to the distance between the starting points of Z and Z' and also greater or equal to the distance between their ending points. Case 2 corresponds to the opening of a new passage inside a cell and between two neighboring cells.

Figure 4.8: An example of critical values. (a) : Two critical values are the perpendicular distances ϵ_1 and ϵ_2 from vertex v to edge e in two unfolded edge sequences, respectively. $v \in e'$, $e' \in Z'$ and $e \in Z$. (b) : If Fréchet distance of Z and Z' is ϵ_1 , then the monotone path passes through point p in the cell of e and e' in the free space diagram.

There are $O(zz'n^2)$ such critical values of ϵ , namely the distance between starting points and endpoints of Z and Z' (case 1), and the distances between vertices of one curve and edges of the other inside all of the corresponding edge sequences (case 2). Refer to Figure 4.8 (a). There are two unfolded edge sequences in this figure, vertex $v \in Z'$ and edge $e \in Z$. In each edge sequence, there is a perpendicular distance between v and e , named ϵ_1 and ϵ_2 respectively. Both ϵ_1 and ϵ_2 are critical values. Suppose that ϵ_1 is Fréchet distance between paths Z and Z' , then point $p \in e$ is passed through by the monotone curve in the free space diagram. Refer to Figure 4.8 (b). Each one of these critical values can be computed in $O(1)$ time. We use the following algorithm in [8] to compute $\delta_F(Z, Z')$.

Algorithm 8. *Computed $\delta_F(Z, Z')$*

1. Determine all of the critical values of ϵ ;
2. Sort them;

3. Do a binary search on the sorted sequence in each search step solving the decision problem, continue with the half that contains the smaller critical values if it has a positive answer and with the half that contains the larger critical values otherwise.

— End of Algorithm —

For each critical value, using the decision algorithm to find if there exists a monotone path from $(0,0)$ to (z, z') takes $O(zz'n^2 \log n)$ time. Since there are $O(\log(zz'n^2))$ critical values that need to be tested (binary search on the sorted sequence of length $O(zz'n^2)$), the final Fréchet distance can be computed in $O(zz'n^2 \log n \log(zz'n^2))$ time. As the shortest paths between the points on each pair of edges on Z and Z' respectively can be computed in $O(n^4 \log n)$ time, the zz' pairs' shortest paths problem can be solved in $O(zz'n^4 \log n)$ time. We summarize this analysis in the following theorem.

Theorem 4.2.2. *For two paths Z and Z' consisting of z and z' edges on the convex polyhedron P , Fréchet-distance between Z and Z' can be computed in $O(zz'n^2 \log n \log(zz'n^2))$ time, with preprocessing time $O(zz'n^4 \log n)$ on computing the shortest paths between each pair of edges on Z and Z' , where n is the number of faces on P .*

Chapter 5

Conclusion

Fréchet distance as a metric for computing the similarity of two curves has been a widely studied topic in recent years. Most of the studies are in 2-dimension. A question then arises: How can Fréchet distance metric be applied to measure the similarity of two paths on a convex polyhedron? This question is the motivation for the study in this thesis.

An algorithm of computing Fréchet distance in 2-D has been given by Alt and Godau [8]. Fréchet distance is computed by building a free space diagram and finding a monotone path in this diagram. This thesis adopted these techniques; Yet there is no regularity for the shape of each cell of the free space diagram. The free space in each cell is the union of a set of polygons whose edges are hyperbolic curves and elliptical arcs. The contribution in this part is:

- We designed an algorithm for computing a monotone path for each unit cell in the free space diagram.

As we already know, Fréchet distance is measured by Euclidean shortest path distance on the surface of the given convex polyhedron. A shortest path between any pair of points on two edges can be extracted from the algorithm of Hwang [25], but with a complicated

tree representation for the edge sequences and lack of the details of the data structures for the visibility diagram, etc.. The contributions in this part are:

- In order to lower the space requirements, we use the variation of the tree construction in Chen and Han [9] to build the edge sequence tree ;
- We provided the data structures for the visibility diagram;
- We redesigned and presented details of each step for the edge-edge shortest path algorithm on a convex polyhedron.

According to the data structures of visibility diagram, shortest paths for each pair of points can be obtained from the visibility diagram quickly, however, the time complexity of the shortest path algorithm is bounded by $O(n^4 \log n)$. This bound is due to the number of edge sequences in the tree. If the edge sequences with the starting edge e_s and ending edge e_t can be computed directly, without extracting them from all of the edge sequences ending at different edges, the time complexity will decrease.

A problem also arises from computing the critical values of Fréchet distance. We considered two cases that can produce the critical values. One is the distance between the two pairs of endpoints of Z and Z' . The other case is the distance — the perpendicular distance inside an unfolded shortest path edge sequence, between a vertex of a path and an edge of the other path. In the second case, the critical values are computed from the edge sequences whose corresponding polygons in the visibility diagram intersect the boundary of the visibility diagram. One interesting question arises: Are there other cases that can generate critical values? For example, do the critical values include the suitable shortest path distances of the pairs of points in the central polygons in the visibility diagram?

Another challenging problem is to do an empirical study of these two algorithms. The intersections between hyperbolic curves, and even intersections of hyperbolic curve and ellipse, relate to a high degree (at the most degree 4) of computations. The approximation values in each step will finally affect the accuracy to some degree of Fréchet distance.

Our objective was to discover a way to determine whether two paths on a convex polyhedron are similar. In this thesis, we used Fréchet distance with respect to the geodesic path metric. There are other ways to measure the quality of the approximation: Hausdorff-distance, uniform distance, and a “distance-preserving” measure. It would be interesting to see whether or not they can be used to measure the similarity of two paths on a convex polyhedron instead of using Fréchet distance.

Bibliography

- [1] H. Alt, A. Efrat, G. Rote, and C. Wenk, *Matching planar maps*, Journal of Algorithms **49** (2003), 262–283.
- [2] M.de Berg, M.van Kreveld, and M.Overmars, *Computational Geometry, Algorithms and Applications*, 2nd edition, Springer-Verlag (2000), 19–42.
- [3] D.M.Mount, *The number of shortest path on the surface of a polyhedron*, SIAM Journal of Computing **19** (1990 August), no. 4, 593–611.
- [4] D.T.Lee, *Proximity and reachability in the plane*, Report R-831, Department of Electrical Engineering and Computer Science, University of Illinois (1978).
- [5] D.T.Lee and F.P.Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks **14** (1984), 393–410.
- [6] J. Gudmundsson, G. Narasimhan, and M. Smid, *Distance-preserving approximations of polygonal paths*, Lecture Notes in Computer Science, Springer-Verlag, Berlin **2914** (2003), 217–228.
- [7] H.Alt, C.Knauer, and C.Wenk, *Matching polygonal curves with respect to the Fréchet distance*, Proceedings 18th International Symposium on Theoretical Aspects of Computer Science, STACS (2001), 63–74.

- [8] H.Alt and M.Godau, *Computing the Fréchet distance between two polygonal curves*, International Journal of Computational Geometry and Applications **5** (1995), no. 12, 75–91.
- [9] J.Chen and Y.Han, *Shortest paths on a polyhedron*, International Journal of Computational Geometry and Application **6** (1996), 127–144.
- [10] J.L.Bentley and T.A.Ottmann, *Algorithms for reporting and counting geometric intersections*, IEEE transactions on Computing **C-28** (1979), 643–647.
- [11] J.O'Rourke, *Folding and unfolding in computational geometry*, Proceeding of Japan Conference on Discrete and Computational Geometry 1998, Heidelberg, Germany: Springer-Verlag (2000), 1258–266.
- [12] J.S.B.Mitchell, D.M.Mount, and C.H.Papadimitriou, *The discrete geodesic problem*, SIAM Journal of Computing **16** (1987 August), no. 4, 647–668.
- [13] K.Kedem and M.Sharir, *An efficient algorithm for planning collision-free translational motion of a convex polygonal object in 2-dimensional space amidst polygonal obstacles*, In Proceeding 1st Annual ACM Symposium on Computational Geometry (1985), 75–80.
- [14] K.Kedem, R.Livne, J.Pach, and M.Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete and Computational Geometry **1** (1986), 59–71.
- [15] L.A.Lyusternik, *The shortest lines, variational problems*, Little Mathematics Library, Mir Publishers. Moscow (1983).
- [16] L.Guibas, J.Hershberger, D.Leven, M.Sharir, and R.E.Tarjan, *Linear algorithms for visibility and shortest path problems inside simple polygons*, Algorithmica **2** (1987), no. 2, 209–233.

- [17] M.Godau, *A natural metric for curves—computing the Fréchet distance for polygonal chains and approximation algorithms*, Proceedigs Symposium on Theoretical Aspects of Computer Science, STACS, Springer Lecture Notes in Computer Science **480** (1991), 127–136.
- [18] J.S.B Mitchell, *Geometric shortest paths and network optimization*, Handbook of Computational Geometry, Jörg-Rudiger Sack and Jorge Urrutia, Elsevier (2000), 633–701.
- [19] M.Sharir, *On shortest paths amidst convex polyhedra*, SIAM Journal of Computing **16** (1987), no. 3, 561–572.
- [20] M.Sharir and A.Schorr, *On shortest paths in polyhedral spaces*, SIAM Journal of Computing **15** (1986), 193–215.
- [21] P.K.Agarwal, B.Aronov, J.O’Rourke, and C.A.Schevon, *Star unfolding of a polytope with application*, SIAM Journal of Computing **26** (1997), 1689–1713.
- [22] R.Seidel, *A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons*, Computational Geometry, Theory and Application **1** (1991), 51–64.
- [23] T.Lozano-Perez and M.A.Wesley, *An algorithm for planning collision-free paths among polyhedral obstacles*, Communications of the ACM **22** (1979), 560–570.
- [24] K. R. Varadarajan, *Approximating monotone polygonal curves using the uniform metric*, Proceedings of the 12th annual symposium on Computational geometry (1996), 311–318.
- [25] Y.H.Hwang, R.C.Chang, and H.Y.Tu, *Finding all shortest path edge sequences on a convex polyhedron*, LNCS, Algorithms and Data Structures Workshop **382** (1989 August), 251–266.