OBSTACLE-AVOIDING SIMILARITY METRICS AND SHORTEST PATH PROBLEMS

APPROVED BY SUPERVISING COMMITTEE:

_____
Carola Wenk, Ph.D., Chair

_____
Tom Bylander, Ph.D.

_____
José Iovino, Ph.D.

_____
Jianhua Ruan, Ph.D.

_____
Weining Zhang, Ph.D.

Accepted: _____
Dean, Graduate School

DEDICATION

This dissertation is dedicated to my father and mother and to the best advisor in the galaxy. Words cannot express my gratitude for the encouragement, humor, and inspiration you provided.

OBSTACLE-AVOIDING SIMILARITY METRICS AND SHORTEST PATH PROBLEMS

by

ATLAS F. COOK IV, B.S.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
December 2009

ACKNOWLEDGMENTS

I am eternally grateful to my advisor Carola Wenk for countless meetings and proofreading sessions. Our meetings always left me excited about research, and I would do well to emulate your cheerful and ever-optimistic demeanor. Thank you for inspiration. Special thanks also go to Tom Bylander, José Iovino, Jianhua Ruan, and Weining Zhang for their insights regarding this dissertation. This work has been graciously supported by the National Science Foundation grant NSF CAREER CCF-0643597 and the 2009 University of Texas at San Antonio Presidential Dissertation Fellowship.

Chapter 3 has not been previously published. The material in Chapter 4 has appeared as [31, 32, 33, 36]. Chapter 5 has appeared as [34, 37]. Chapter 6 has appeared as [35, 37]. Chapters 7 and 8 won the *WADS 2009 Best Paper Award* and have appeared as [38, 39, 40, 41]. Chapter 9 has been presented at the 2009 *Fall Workshop on Computational Geometry* [42] and has been submitted to the *9th Latin American Theoretical Informatics Symposium*.

"This Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Doctoral Dissertation must still conform to all other requirements explained in the "Guide for the Preparation of a Master's Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio." It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Doctoral Dissertation attest to the accuracy of this statement."

December 2009

OBSTACLE-AVOIDING SIMILARITY METRICS AND SHORTEST PATH PROBLEMS

Atlas F. Cook IV, Ph.D.
The University of Texas at San Antonio, 2009

Supervising Professor: Carola Wenk, Ph.D.

Similarity metrics are functions that measure the similarity of geometric objects. The motivation for studying similarity metrics is that these functions are essential building blocks for areas such as computer vision, robotics, medical imaging, and drug design. Although similarity metrics are traditionally computed in environments without obstacles, we use shortest paths to compute similarity metrics in simple polygons, in polygons with polygonal holes, and on polyhedral surfaces. We measure the length of a path either by Euclidean distance or by the number of turns on the path.

We also compute shortest paths that steer a medical needle through a sequence of treatment points in the plane. This technique could be used in biopsy procedures to take multiple tissue samples with a single puncture of the skin. Such an algorithm could also be applied to brachytherapy procedures that implant radioactive pellets at many cancerous locations. Computing shortest paths for medical needles is a challenging problem because medical needles cut through tissue along circular arcs and have a limited ability to turn. Although optimal substructure can fail, we compute globally optimal paths with a wavefront propagation technique.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The ability to recognize and distinguish between objects is a crucial ability for the survival of most species. While the youngest lamb is nurtured by its mother, it must flee the lion. While one berry provides sustenance, another berry is poison. This need to distinguish between objects is neither limited to the animal kingdom nor to primitive survival instincts. Today's society requires many humans to distinguish between red and green traffic lights and to navigate based on landmarks. Without these abilities, much of today's society would not exist.

Machines are increasingly used to automate laborious tasks once carried out by humans. Today, these tasks range from washing clothes to the electronic mail courier. In the future, computerized tasks could conceivably include automatically navigating vehicles and fully automated medical procedures. For machines to carry out these increasingly complicated tasks, they (like humans) must learn to recognize and distinguish between objects.

Similarity metrics are functions that allow computers to measure the similarity of objects in order to recognize and distinguish between them. The motivation for studying similarity metrics is that these functions are essential building blocks for areas such as computer vision, computer aided design, robotics, medical imaging, signature recognition, and drug design. Although similarity metrics are traditionally computed in environments without obstacles, we use shortest paths to compute similarity metrics in simple polygons, in polygons with polygonal holes, and on polyhedral surfaces.

We measure the length of a path either by Euclidean distance or by the number of turns on the path. Minimizing the number of turns on a path has a variety of applications in robotic motion, wireless communications, geographic information systems, VLSI, solid modeling, and even water pipe placement. These applications benefit from minimizing the number of turns on a path because turns are costly while straight line movements are inexpensive.

We also compute shortest paths that steer a medical needle through a sequence of treatment points in the plane. This technique could be used in biopsy procedures to take multiple tissue sam-

ples with a single puncture of the skin. Such an algorithm could also be applied to brachytherapy procedures that implant radioactive pellets at many cancerous locations. Computing shortest paths for medical needles is a challenging problem because medical needles cut through tissue along circular arcs and have a limited ability to turn. Although optimal substructure can fail, we are able to compute globally optimal paths with a wavefront propagation technique.

Chapter 3 has not been previously published and improves the runtime of the Fréchet distance in two specific scenarios. Most of the remaining material in this dissertation is from our four technical reports [31, 34, 35, 38]. Chapter 4 discusses similarity metrics in a simple polygon and has been published in the 2007 *Fall Workshop on Computational Geometry* [32] and the 2008 *Symposium on Theoretical Aspects of Computer Science* [33] (27% acceptance rate). It has also been accepted into the ACM journal *Transactions on Algorithms* [36]. Chapters 5 and 6 discuss similarity metrics and shortest path problems in a polygonal domain. These results have been published in the 2009 *Algorithmic Aspects in Information and Management* [37] and were submitted to the journal *Computational Geometry: Theory and Applications* on January 30, 2009.

Chapters 7 and 8 discuss similarity metrics and shortest path problems on a polyhedral surface. These results won the *WADS 2009 Best Paper Award* and have been published at the 2009 *European Workshop on Computational Geometry* [40], at the 2009 *Dagstuhl Seminar Proceedings #09111* [39], at the 2009 *Algorithms and Data Structures Symposium (WADS)* [41] (39% acceptance rate), and were submitted to the journal *Algorithmica* on April 1, 2009. All of this material is the primary work of Atlas F. Cook IV and his doctoral advisor Carola Wenk. A notable collaborator is Helmut Alt who discussed a few of the initial ideas of the second technical report [34] with Atlas F. Cook IV during a research visit to Berlin. We are grateful for his input.

The needle steering material in Chapter 9 is the primary work of Atlas F. Cook IV and his doctoral advisor Carola Wenk. Steven Bitner, Yam K. Cheung, Ovidiu Daescu, and Anastasia Kurdia discussed some of the initial ideas of this work with Atlas F. Cook IV when he used his 2009 Presidential Dissertation Fellowship funds to visit the University of Texas at Dallas. The needle steering material has been presented at the 2009 *Fall Workshop on Computational Geometry*

[42] and has been submitted to the *9th Latin American Theoretical Informatics Symposium*.

1.1 Related Work in the Plane

Most related work assumes an environment without obstacles where distances between points are measured using an L_p metric. Alt and Godau [10] show that a similarity metric called the Fréchet distance[1] can be computed between polygonal curves A and B in arbitrary dimensions for environments without obstacles in $O(N^2 \log N)$ time, where N is the total number of vertices defined by A and B. Buchin et al. [20] show that the Fréchet distance takes $\Omega(N \log N)$ time to compute. Rote [73] computes the Fréchet distance between piecewise smooth curves. Buchin et al. [21] show how to compute the Fréchet distance between two simple polygons. Wenk et al. [80] apply the Fréchet distance to the practical domain of map matching. All of these works measure distances between points using an L_p metric.

Recent work has focused on measuring distances between points using shortest paths that avoid a set of obstacles. Maheshwari and Yi [65] show how to compute the Fréchet distance for polygonal curves A and B on the surface of a convex polyhedron in $O(N^3k^4 \log(kN))$ time, where k is the complexity of the convex polyhedron and N is the complexity of A and B. Chambers et al. [23] compute the Fréchet distance in $O(N^4k^3 \log kN)$ time inside a polygonal domain (a polygon with polygonal holes) by only considering shortest paths that can be continuously deformed into each other (i.e., homotopy classes). Efrat et al. [52] use the Fréchet distance to morph a polygonal curve A into a polygonal curve B in $O(N^2 \log^2 N)$ time. Their method considers the curves A and B to be obstacles that all shortest paths must go around. They also require that A and B are simple and disjoint. Our work in Chapter 4 can handle the more general case of morphing an arbitrary polygonal curve A into an arbitrary polygonal curve B with respect to any simple polygon obstacle.

Guibas et al. [57] explore Euclidean shortest paths in a simple polygon with k vertices. Using structures called funnels and hourglasses, they show that the length of a shortest path between any two points can be reported in $O(\log k)$ time after $O(k)$ preprocessing.

[1] Chapter 2.4 formally defines the Fréchet distance.

Shortest paths have also been explored in a polygonal domain with k vertices (i.e., a polygon with polygonal holes). In this setting, the length of any shortest path from a *fixed source point* can be computed in $O(\log k)$ time after $O(k \log k)$ time and space preprocessing [61, 66]. This preprocessing produces a *shortest path map* that has $O(k)$ size and partitions the plane into regions such that all points in a region have the same combinatorial shortest path to the fixed source point. It is a much more difficult problem to compute shortest paths in a polygonal domain when the source point is not fixed. The best approach to compute shortest paths between *any two points* in a polygonal domain currently uses $O(k^{11})$ time and space preprocessing to support $O(\log k)$ time queries [28].

An alternative to computing Euclidean shortest paths is to measure path length by the number of edges on the path. A *link path* is a polygonal path that connects two points using the minimum number of obstacle-avoiding edges. The *length* of a link path is the number of edges on the path, and this length is sometimes referred to as the *link distance* between the two points. Link paths are fundamentally different from Euclidean paths and have a wealth of applications including robotic motion, wireless communications, geographic information systems, VLSI, solid modeling, and even water pipe placement. These applications are naturally modeled by link paths because turns are costly while straight line movements are inexpensive.

In a simple polygon with k vertices, Suri [77] defines a data structure called a *window partition* that can answer link distance queries from a fixed source in $O(\log k)$ time after $O(k)$ preprocessing. This window partition divides a simple polygon into regions of equivalent link distance from the fixed source. Thus, a window partition is essentially a link-based shortest path map for a simple polygon.

When the fixed source is a line segment $\overline{ab}$, Suri's window partition [77] always returns $\min_{s \in \overline{ab}} d_L(s,t)$ (i.e., the link distance from the query point t to a nearest point on $\overline{ab}$). By contrast, the work of Arkin, Mitchell, and Suri [14] supports $O(\log k)$ time link-based queries between any two points in a simple polygon. Their more general approach works by building a set of $O(k^2)$ combinatorially distinct shortest path maps in $O(k^3)$ time and space preprocessing. These bounds

have also been matched in [51] by a plane sweep approach.

Link paths in a *polygonal domain* with k vertices (i.e., a polygon with polygonal holes) have been studied by Mitchell, Rote, and Woeginger [68]. Their shortest path map structure supports fixed-source queries in a polygonal domain in $O(\log k)$ time after $O(k^4)$ time and space preprocessing, and they show that this bound is tight for an explicit representation of a shortest path map.

Link distance problems are usually more difficult to solve than equivalent Euclidean shortest path problems because optimal paths that are unique under the Euclidean metric need not be unique with respect to link distance. Another difficulty is that Euclidean shortest paths only turn at obstacle vertices while link-based paths can turn anywhere [14, 64].

1.2 Related Work on a Polyhedral Surface

Two questions are invariably encountered when dealing with shortest path problems. The first question is how to represent the combinatorial structure of a shortest path. In the plane with polygonal obstacles, a shortest path can only turn at obstacle vertices, so a shortest path can be combinatorially described as a sequence of obstacle vertices [57]. On a polyhedral surface, a shortest path need not turn at vertices [67], so a path is often described combinatorially by an *edge sequence* that represents the sequence of edges encountered by the path [1].

The second commonly encountered shortest path question is how to compute shortest paths in a problem space with M vertices. The following preprocessing schemes compute combinatorial representations of all possible shortest paths. On a *convex* polyhedral surface, Mount [69] shows that $\Theta(M^4)$ combinatorially distinct shortest path edge sequences exist, and Schevon and O'Rourke [75] show that only $\Theta(M^3)$ of these edge sequences are *maximal* (i.e., they cannot be extended at either end without creating a suboptimal path). Agarwal et al. [1] use these properties to compute the $\Theta(M^4)$ shortest path edge sequences on a convex polyhedral surface in $O(M^6 2^{\alpha(M)} \log M)$ time and the *diameter* in $O(M^8 \log M)$ time, where $\alpha(M)$ is the extremely slowly growing inverse Ackermann function (see Chapter 2 and [4]). The diameter is the largest shortest path distance be-

tween any two points on the surface. Despite recent efforts by Chandru et al. [24] to improve the runtimes of Agarwal et al. [1], these runtimes have not improved since 1997. One of our main results is to improve the edge sequence and diameter algorithms of [1] by a linear factor. We achieve this improvement by combining the star unfolding technique of [1] with the kinetic Voronoi diagram structure of Albers et al. [6]. A kinetic Voronoi diagram is a nearest neighbor data structure that allows its defining point sites to move (see Section 2.1 for an introduction to Voronoi diagrams).

A popular alternative to precomputing *all* combinatorial shortest paths in a problem space is to precompute a shortest path map structure SPM(s) that describes all shortest paths from a fixed source s. In the plane with polygonal obstacles, Hershberger and Suri [61] use the continuous Dijkstra paradigm to support all queries from a fixed source after $\Theta(M \log M)$ preprocessing. On a (possibly non-convex) polyhedral surface, Mitchell, Mount, and Papadimitriou [67] use the continuous Dijkstra paradigm to construct SPM(s) by propagating a wavefront over a polyhedral surface in $O(M^2 \log M)$ time and $O(M^2)$ space. Chen and Han [27] solve the same polyhedral surface problem in $O(M^2)$ time and space by combining unfolding and Voronoi diagram techniques. Schreiber and Sharir [76] use the continuous Dijkstra paradigm to construct an *implicit* representation of a shortest path map for a *convex* polyhedral surface in $O(M \log M)$ time and space. In addition to the exact algorithms above, there are also various efficient algorithms to compute approximate shortest paths on weighted polyhedral surfaces, see for example Aleksandrov et al. [7, 8].

Another popular variation on shortest path problems is to consider different methods of measuring the length of a shortest path. For example, suppose the "length" of a path is measured not by its Euclidean distance but by the number of edges on the path. A *link path* is a polygonal path that connects two points using the minimum number of obstacle-avoiding edges. The *length* of a link path is the number of edges on the path, and this length is sometimes referred to as the *link distance* between two points [14, 51, 64, 68, 77]. Surprisingly, we appear to be the first to consider link distance problems on a polyhedral surface.

One reason that shortest path and link distance problems are of crucial importance is that they often serve as building blocks for higher level problems such as the Fréchet distance. The Fréchet distance measures the similarity of continuous shapes [10, 15, 21, 52, 73] by calculating a distance-based value that represents the similarity of the shapes. Although the traditional Fréchet distance operates in a space that is free of obstacles, recent works have realized the potential of the Fréchet distance on surfaces. For example, Maheshwari and Yi [65] compute the Fréchet distance between polygonal curves on a convex polyhedral surface. Buchin, Buchin, and Wenk [21] show how to compute the Fréchet distance between two simple polygons. Cook, Sherette, and Wenk [30] have recently given a fixed-parameter tractable algorithm that computes the Fréchet distance between two triangulated polyhedral surfaces.

1.3 Related Work for Needle Steering

We now discuss the medical problem of steering a bevel-tip needle through the soft tissue of a patient. A bevel-tip needle has an asymmetric tip that cuts through soft tissue along a circular arc. Each rotation of the needle during its insertion into soft tissue causes the needle to start moving along a new circular arc that is tangent to the previous circular arc [12]. The goal of our work is to use a series of rotations to steer the needle along a *needle path* so that it visits a sequence of treatment points. A needle path is a connected sequence of directed circular arcs that satisfies the following properties. First, all circular arcs have the same radius and lie on tangent circles. Second, if the current circular arc travels clockwise, then the next circular arc travels counter-clockwise (and vice versa). Third, the *length* of a needle path is optimal in that it equals the smallest possible number of circular arcs that can be used to visit the sequence of treatment points.

The path traveled by a bevel-tip needle in soft tissue is theoretically equivalent to a restricted type of Dubins path [48] that can move along circular arcs but cannot move straight. Such *needle paths* also have similarities to touring problems [47], curvature-constrained shortest path problems [2, 18, 48], and link distance problems [14, 37, 64, 68, 77]. Since curvature-constrained paths are NP-Hard [2] to compute in the plane with polygonal obstacles, several works [2, 18] have explored

curvature-constrained paths in convex polygons and narrow corridors.

Alterovitz et al. [13] consider steering a needle in a plane with polygonal obstacles toward a single target point. They calculate an optimal entry point for a needle and assume that no rotations are performed during the procedure. Subsequent work in the plane by Alterovitz et al. [12] permits the needle to rotate during its insertion into soft tissue. Deflections of the needle during the procedure are accounted for by periodically taking snapshots of the needle's position during surgery. Their dynamic programming approach uses a Markov model to maximize the probability of successfully reaching a single target point without bumping into an obstacle.

Several studies have steered a needle in three dimensions. Duindam et al. [49] guide a needle to a single target point in a three-dimensional setting that contains no obstacles. Xu et al. [81] use a probabilistic roadmap technique to reach a single target point in a three-dimensional space that contains spherical obstacles.

Related work for needle paths also exists in the realm of robotic motion. Robots typically have a much easier time moving straight than turning, so the "length" of a path is typically measured by the number of turns on the path instead of the Euclidean length of the path [64]. Needle paths are similar in that we minimize the number of rotations of the needle because each rotation inherently complicates the path for the physician.

1.4 Our Results in the Plane

Chapters 4, 5, and 6 show how to compute both the Euclidean and link-based Fréchet distance inside a simple polygon and inside a polygonal domain (a polygon with polygonal holes). We also speed up the computation of the Fréchet distance by replacing the traditional parametric search optimization technique (see Section 2.5) with a red-blue intersection algorithm. As part of this investigation of the Fréchet distance, shortest path maps are developed that support queries from any source point on a fixed *line segment*. Such shortest path maps have applications beyond the Fréchet distance because they encode more information than a traditional shortest path map (which typically supports queries from a fixed source *point*).

Table 1.1: **Our Euclidean shortest path results in the plane.** P is a simple polygon with k vertices; D is a polygonal domain with k vertices. N is the complexity of any objects that are not obstacles. The shortest path map SPM($\overline{ab}$,$\overline{cd}$) supports queries from $s \in \overline{ab}$ to $t \in \overline{cd}$. The Fréchet distance lower bounds apply to the complexity of the free space diagram. $\lambda_s(k)$ is a near-linear function that is defined by the length of a Davenport-Schinzel sequence [4].

		Time	Space
Fréchet Distance	P	$O(k + N^2 \log Nk \log N)$	$O(N^2 + k)$
	D	$O(N^2k^4 \log Nk \log k)$, $\Omega(N^2k^2)$	$O(Nk + k^4)$
Weak Fréchet Distance	D	$O(N^2k^3\lambda_6(k) \log Nk)$, $\Omega(N^2k^2)$	$O(Nk + k^4)$
SPM($\overline{ab}$,$\overline{cd}$)	D	$O(k^4\lambda_6(k))$, $\Omega(k^2)$	$O(k^5)$

Two definitions are central to the Fréchet distance. A *free space cell* is the parameter space defined by a pair of line segments $\overline{ab}$ and $\overline{cd}$. *Free space* in this cell consists of all points (s, t) in the parameter space whose shortest path distance $d(s, t)$ is less than or equal to some threshold value ε. Our motivation for studying the Fréchet distance in the presence of obstacles is that teaming up two people for safety reasons is common practice in many real-life situations, ranging from scouts in summer camp, to fire fighters and police officers, and even to astronauts exploring the moon. In all of these applications, two team members need to coordinate their movement to stay within "walking distance" so that fast assistance can be offered in case of an emergency. The Fréchet distance is an ideal model for this scenario.

Table 1.1 summarizes our Euclidean shortest path results in $\mathbb{R}^2$. In Chapter 4, we compute the Fréchet distance between two polygonal curves inside a simple bounding polygon in $O(k + N^2 \log kN \log N)$ expected time, where N is the larger of the complexities of A and B and k is the complexity of the simple polygon. The expected runtime is only a logarithmic factor larger than the traditional $O(N^2 \log N)$ Fréchet algorithm [10] (without obstacles) and is almost a quadratic factor in k faster than the straightforward approach, similar to [52], of partitioning each cell into $O(k^2)$ subcells with combinatorially distinct shortest paths. It is notable that our randomized algorithm also applies to the traditional Fréchet distance in arbitrary dimensions.

We compute the Fréchet distance in a simple polygon by showing that the free space in a

free space cell is always x-monotone, y-monotone, and connected. These properties allow us to calculate the Fréchet distance by only considering the boundaries of free space cells (i.e., the interior points of cells can be ignored). Our approach uses dynamic programming to propagate paths through the free space cells. We also employ a randomized algorithm that is based on red-blue intersection counting in lieu of the standard parametric search approach [10] (see Section 2.5) to compute the exact Fréchet distance. Palazzi and Snoeyink [71] have previously explored red-blue intersections for linear functions using a slab-based approach, but they require that all red segments are disjoint and all blue segments are disjoint. Our approach is more general because it applies to monotone functions, and we have no disjointness requirement.

In addition to investigating problems inside a simple polygon, we also explore problems in a *polygonal domain* (a polygon with polygonal holes). In Chapter 5, we show how to compute the Fréchet distance in the plane between polygonal curves A and B in the presence of k line segments obstacles in $O(N^2k^4 \log Nk \log k)$ expected time and $O(Nk + k^4)$ space, where N is the complexity of A and B. The polygonal curves A and B are not considered to be obstacles, and distances between points are measured by shortest paths. Unlike the $O(N^4k^3 \log kN)$ time Fréchet distance algorithm of [23], we measure distances using shortest paths without ensuring that these shortest paths conform to any specific homotopy class. We also show how to compute a variant of the Fréchet distance called the weak Fréchet distance, and we present several lower bounds. It would be interesting to close the gap between our lower and upper bounds.

We use shortest path structures called dynamic and static spotlights (see Sections 5.1 and 5.2) to encode all shortest paths between two line segments $\overline{ab}$ and $\overline{cd}$ in a polygonal domain. These spotlights are used not only to compute the Fréchet distance but also to compute a shortest path map SPM($\overline{ab}, \overline{cd}$). Unlike traditional shortest path maps that only support shortest path queries from a fixed *point*, our SPM($\overline{ab}, \overline{cd}$) structure can be used to compute shortest paths from a *continuous* set of source points $s \in \overline{ab}$ to any point $t \in \overline{cd}$. A related result by Chiang and Mitchell [28] constructs a two-point shortest path map that supports queries from any source point in the plane to any target point in the plane after $O(k^{11})$ time and space preprocessing, where k is the complexity of the

polygonal domain obstacle. Our SPM($\overline{ab}, \overline{cd}$) structure differs in that we restrict the source and target points to lie on line segments instead of being allowed to lie anywhere in the plane. This restriction allows us to achieve optimal $O(\log k)$ query time after preprocessing in $O(k^5 \log k)$ expected time and $O(k^5)$ space.

It is also possible to measure path length by the number of edges on the path (as opposed to the Euclidean length of the path). Such "link paths" are fundamentally different from Euclidean paths and have a wealth of applications including robotic motion, wireless communications, geographic information systems, VLSI, solid modeling, and even water pipe placement. These applications are naturally modeled by link paths because turns are costly while straight line movements are inexpensive.

Table 1.2 summarizes our link distance results in $\mathbb{R}^2$. In Chapter 6, we present *tight* bounds and efficient algorithms to compute two types of link-based Voronoi diagrams in the plane. See Section 2.1 for a description of Voronoi diagrams. Surprisingly, this seems to be the first time that link-based Voronoi diagrams have been studied.

We also define a link-based shortest path map that allows the source point to be specified as any point on a line segment. See Section 2.2 for a description of shortest path maps. Although there are related *Euclidean* results [28, 57] that support queries from more than one point in the plane, we are not aware of any related work that supports *link distance* queries from a continuous set of source points in a polygonal domain.

For link distance similarity metrics, we develop exact and approximate algorithms for the Hausdorff distance between point and line segment sets in either a simple polygon or a polygonal domain. See Section 2.3 for a description of the Hausdorff distance. Although the Hausdorff distance is traditionally computed with Voronoi diagrams, Voronoi diagrams encode more information than is necessary, and we are able to compute the link-based Hausdorff distance more quickly by using shortest path map techniques. We also have approximation algorithms that are accurate to within one, two, or three links of the optimal result.

We compute the link distance Fréchet distance exactly and approximately in a simple polygon

Table 1.2: **Our link distance results in the plane.** P is a simple polygon with k vertices; D is a polygonal domain with k vertices. N is the complexity of any objects that are not obstacles. The shortest path map SPM($\overline{ab}$,$\mathbb{R}^2$) supports queries from $s \in \overline{ab}$ to $t \in \mathbb{R}^2$; SPM($\overline{ab}$,$\overline{cd}$) supports queries from $s \in \overline{ab}$ to $t \in \overline{cd}$. An asterisk * indicates that in addition to the exact runtimes that are shown, we also have approximation algorithms. The Fréchet distance lower bound applies to the complexity of the free space diagram. $\alpha(k)$ is the extremely slowly growing inverse Ackermann function, and $\lambda_s(k)$ is a near-linear function that is defined by the length of a Davenport-Schinzel sequence (see Chapter 2).

		Time	Space
Site-Based Voronoi Diagram	P	$O(N^2 k \log Nk)$	$\Theta(N^2 k)$
Distance-Based Voronoi Diagram	P	$O(N(N+k)\log N \log k)$	$\Theta(N(N+k))$
SPM($\overline{ab}$,$\mathbb{R}^2$)	P*	$O(k^2)$	$O(k^2)$
SPM($\overline{ab}$,$\overline{cd}$)	D*	$O(k^6 \lambda_6(k))$	$O(k^7)$
Hausdorff distance for points	P	$O(kN + N^2)$	$O(k+N)$
	D	$O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2 k)$	$O(k+N)$
Hausdorff distance for line segments	P	$O(kN + N^2 \log N)$	$O(kN)$
	D*	$O(N^2 k^2(\alpha(k)\log^2 k + \log Nk))$	$O(Nk^2)$
Fréchet Distance	P*	$O(kN + N^2)$	$O(k+N^2)$
	D*	$O(N^2 k^7 \log kN)$, $\Omega(N^2 k^4)$	$O(Nk^2 + k^4)$

and a polygonal domain. See Section 2.4 for a description of the Fréchet distance. A novelty of our approach is that it allows the link distance Fréchet distance to be computed in the same time and space as the decision problem. This is intriguing because the only other known scenarios where it is possible to shave off the logarithmic Fréchet optimization factor are the weak Fréchet distance and discrete Fréchet distance (see Section 2.4). In these cases, the free space diagram can be treated as a planar graph [10], and any linear-time shortest path algorithm (e.g., [58]) can be applied to this planar graph to solve the optimization problem. We also give a lower bound for the complexity of the link-based free space diagram in a polygonal domain.

1.5 Our Results on a Polyhedral Surface

Table 1.3 summarizes our Euclidean shortest path results on a polyhedral surface (see Chapter 7). For a *convex* polyhedral surface, Agarwal et al. [1] give algorithms to compute the diameter and either the exact set or a superset of all $\Theta(M^4)$ shortest path edge sequences. All three of these algorithms are improved by a linear factor in Sections 7.1 and 7.2. Our improvement takes advantage of small combinatorial changes between adjacent events and combines star unfolding and kinetic Voronoi diagram techniques.

Section 7.3 contains an algorithm to compute the Fréchet distance between polygonal curves on a convex polyhedral surface, and this algorithm is a linear factor faster than the algorithm of Maheshwari and Yi [65]. In addition, Section 7.3 contains the first algorithm to compute the Fréchet distance between polygonal curves on a *non-convex* polyhedral surface. Our motivation for studying the Fréchet distance on a polyhedral surface is that teaming up two people for safety reasons is common practice in many real-life situations, ranging from scouts in summer camp, to fire fighters and police officers, and even to astronauts exploring the moon. In all of these applications, two team members need to coordinate their movement in order to stay within "walking distance" so that fast assistance can be offered in case of an emergency. The Fréchet distance is an ideal model for this scenario. Section 7.3 also demonstrates that a subset of the star unfolding called the "core" can overlap itself for a non-convex polyhedral surface. Section 7.4 describes shortest path maps

Table 1.3: **Our Euclidean shortest path results on a polyhedral surface.** In Chapter 7, $\mathcal{P}$ (resp. $\mathcal{P}_N$) indicates a convex (resp. non-convex) polyhedral surface with M vertices. The shortest path maps SPM($\overline{ab}$,$\mathcal{P}$) and SPM($\overline{ab}$,$\mathcal{P}_N$) support queries from any point $s \in \overline{ab}$ to any point on the surface. The Fréchet distance lower bound applies to the complexity of the free space diagram. The constant ϵ can be chosen to have any value greater than zero.

		Time	Space
Star Unfolding maintained over all edges	$\mathcal{P}_N$	$O(M^4)$	$O(M^4)$
Kinetic Voronoi Diagram maintained over all edges	$\mathcal{P}$	$O(M^5 \log M)$	$O(M^5)$
Edge Sequences (superset)	$\mathcal{P}$	$O(M^5)$	$O(M^5)$
Edge Sequences (exact)	$\mathcal{P}$	$O(M^5 2^{\alpha(M)} \log M)$	$O(M^4)$
Diameter	$\mathcal{P}$	$O(M^7 \log M)$	$O(M^4)$
Fréchet Distance	$\mathcal{P}$	$O(M^6 \log^2 M)$	$O(M^2)$
	$\mathcal{P}_N$	$O(M^7 \log^2 M)$, $\Omega(M^4)$	$O(M^3)$
SPM($\overline{ab}$,$\mathcal{P}$)	$\mathcal{P}$	$O(M^4 \log M)$	$O(M^4)$
SPM($\overline{ab}$,$\mathcal{P}_N$)	$\mathcal{P}_N$	$O(M^{9+\epsilon})$	$O(M^9)$

that support queries from any source point on a line segment.

Surprisingly, we appear to be the first to consider link distance problems on a polyhedral surface. In Chapter 8, we define $\mathcal{C}$ as a special type of polyhedral surface such that every face of $\mathcal{C}$ is convex and no two adjacent faces are coplanar. We define $\mathcal{N}$ as a special type of polyhedral surface such that no two adjacent faces are coplanar. $\mathcal{N}$ can be created from any polyhedral surface by triangulating the surface and merging adjacent triangles that are coplanar. The essential property of both $\mathcal{C}$ and $\mathcal{N}$ is that a path must turn whenever it enters a new face. The difference between $\mathcal{C}$ and $\mathcal{N}$ is that the faces of $\mathcal{C}$ are convex while the faces of $\mathcal{N}$ need not be convex and can contain holes.

The link distance between s and t is denoted $d_L(s,t)$, and $\pi_L(s,t)$ denotes a polygonal path with $d_L(s,t)$ edges that connects s and t. We define M as the complexity of $\mathcal{C}$ and $\mathcal{N}$. Table 1.4 summarizes all of our link distance results on a polyhedral surface.

Sections 8.1, 8.2, and 8.3 describe algorithms to compute link-based shortest path maps on a

Table 1.4: **Our link distance results on a polyhedral surface.** $\mathcal{C}$ is a *convex* subdivision of a polyhedral surface such that no two adjacent faces are coplanar, and $\mathcal{N}$ is an arbitrary polyhedral surface. We define M as the complexity of $\mathcal{C}$ and $\mathcal{N}$. The shortest path map SPM($\overline{ab}$, $\mathcal{C}$) supports $d_L(s,t)$, $\pi_L(s,t)$ queries from any point $s \in \overline{ab} \subset \mathcal{C}$ to any point $t \in \mathcal{C}$, SPM(s, $\mathcal{N}$) supports $d_L(s,t)$, $\pi_L(s,t)$ queries from a fixed source point s to any point $t \in \mathcal{N}$, SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) supports $\min_{s \in \overline{ab}} d_L(s,t)$ and $\min_{s \in \overline{ab}} \pi_L(s,t)$ queries to any point $t \in \mathcal{N}$, and SPM($\overline{ab}$,$\overline{cd}$) supports $d_L(s,t)$, $\pi_L(s,t)$ queries between any points $s \in \overline{ab} \subset \mathcal{N}$ and $t \in \overline{cd} \subset \mathcal{N}$. The Fréchet distance lower bounds apply to the complexity of the free space diagram. An asterisk * indicates that, in addition to the exact runtimes that are shown, we also have approximation algorithms.

		Time	Space
SPM($\overline{ab}$, $\mathcal{C}$)	$\mathcal{C}$	$\Theta(M)$	$\Theta(M)$
SPM(s, $\mathcal{N}$), SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$)	$\mathcal{N}$*	$\Theta(M^4)$	$\Theta(M^4)$
SPM($\overline{ab}$,$\overline{cd}$)	$\mathcal{N}$	$O(M^6 \lambda_6(M))$, $\Omega(M^4)$	$O(M^7)$
Link-Based Diameter	$\mathcal{C}$	$O(M^2)$	$O(M)$
	$\mathcal{N}$	$O(M^{\frac{19}{3}} \log^{3.11} M)$	$O(M^3)$
Voronoi Diagram	$\mathcal{C}$	$\Theta(M)$	$\Theta(M)$
	$\mathcal{N}$	$O(M^6)$, $\Omega(M^4)$	$O(M^6)$
Hausdorff Distance	$\mathcal{C}$	$O(M \log M)$	$O(M)$
for points	$\mathcal{N}$	$O(M^{\frac{10}{3}} \log^{3.11} M)$	$O(M^2)$
Hausdorff Distance	$\mathcal{C}$	$O(M \log M)$	$O(M)$
for line segments	$\mathcal{N}$*	$O(M^4 \alpha(M) \log^2 M)$	$O(M^3)$
Fréchet Distance	$\mathcal{C}$	$O(M^2)$, $\Omega(M^2)$	$O(M^2)$
	$\mathcal{N}$*	$O(M^9 \log M)$, $\Omega(M^6)$	$O(M^4)$

polyhedral surface and use these structures to compute Voronoi diagrams, link-based diameters, and the Hausdorff distance. Section 8.4 describes link-based algorithms for the Fréchet distance on a polyhedral surface. Surprisingly, the Fréchet distance can be computed on $\mathcal{C}$ a logarithmic factor *faster* than the traditional Fréchet distance without obstacles [10].

1.6 Our Needle Steering Results

Table 1.5 illustrates our salient results for bevel-tip needles. In Chapter 9, we show how to steer a bevel-tip needle along circular arcs so that it passes through a sequence of treatment points in the plane. We assume that asymmetric forces cause the needle to travel on circular arcs and that the radius of these arcs can be controlled to some extent by varying the stiffness of the needle. Although deflections can in practice lead to deviations from the ideal path, the needle position could be periodically sampled during a procedure to account for these deflections.

We develop needle steering algorithms in problem spaces without obstacles and show how to compute several data structures that can quickly return a needle path. Given one start position for the needle in $\mathbb{R}^d$, our shortest path map can quickly return a needle path to any target point in $\mathbb{R}^d$. Given multiple candidate needle start positions in $\mathbb{R}^d$, a traditional Voronoi diagram can efficiently identify the best needle start position to use to reach any given destination in $\mathbb{R}^d$. Our main bevel-tip needle result is an algorithm to optimally steer a needle through a sequence of treatment points in the plane. When consecutive points in the sequence are sufficiently separated, optimal substructure holds, and the path can be computed in roughly quadratic time. We also have a fixed-parameter tractable algorithm that can compute paths for an arbitrary sequence of points in the plane (see Table 1.5).

To the best of our knowledge, no previous algorithm exists to compute needle paths that visit *multiple* treatment points. However, related work has explored Euclidean shortest paths that tour a sequence of polygons [47].

Table 1.5: **Our needle steering results in the plane.** In Chapter 9, we steer a needle along circular arcs so that it visits a sequence of treatments points in the plane. Our first algorithm returns an optimal path for a sequence of n treatment points such that all consecutive treatment points are sufficiently separated. Our second algorithm is fixed-parameter tractable and assumes that m pairs of consecutive treatment points in the sequence are positioned arbitrarily while the remaining $O(n)$ pairs of consecutive treatment points are sufficiently separated. The returned path has complexity K.

	Time	Space
Needle Path (first algorithm)	$O(n^2 + K)$	$O(n^2)$
Needle Path (second algorithm)	$O(2^m n + n^2 + K)$, $\Omega(2^m)$	$O(2^m n + n^2)$

1.7 Overview

The remainder of this dissertation is organized as follows. Chapter 2 defines several fundamental data structures including Voronoi diagrams and similarity metrics such as the Fréchet distance and the Hausdorff distance. Chapter 3 improves the runtime of the Fréchet distance in two specific scenarios. Chapter 4 shows how to compute the Hausdorff and Fréchet distance inside a simple polygon. Chapter 5 describes how to compute the Hausdorff and Fréchet distance in a polygonal domain (a polygon with polygonal holes). Chapter 6 explores the *link-based* Hausdorff and Fréchet distance inside a simple polygon or polygonal domain. Chapter 7 investigates Euclidean shortest path problems on convex and non-convex polyhedral surfaces. Chapter 8 investigates link distance problems on polyhedral surfaces. Chapter 9 presents algorithms to guide a bevel-tip needle through a sequence of treatment points in the plane. Chapter 10 concludes and presents directions for future work.

CHAPTER 2: DEFINITIONS AND TERMINOLOGY

Let A and B be sets of polygonal shapes in some problem domain. For example, A and B could be polygonal curves in a plane that contains polygonal obstacles. Let $\pi(s,t)$ and $d(s,t)$, respectively, represent the Euclidean shortest path and Euclidean shortest path distance between s and t in the problem domain. We let $\|s - t\|$ denote the Euclidean distance between s and t. We use $\circ$ to denote concatenation of polygonal paths; for example, $\pi(s,o) \circ t$ is the shortest path from s to o concatenated with the line segment $\overline{ot}$ from o to t. A *link path* $\pi_L(s,t)$ is a polygonal path between two points s and t that avoids a set of obstacles and has the fewest possible edges. We use $d_L(s,t)$ to denote the number of edges on the link path $\pi_L(s,t)$.

As defined by Mitchell [66], a *polyhedral surface* is a "connected union of a finite number of polygonal faces, with any two polygons intersecting in a common edge, a common vertex, or not at all," and each edge belonging to at most two polygons. Examples of polyhedral surfaces include terrains, convex polyhedra, and non-convex polyhedra.

A subset $S \subseteq \mathbb{R}^2$ is *x-monotone* if every vertical line intersects S in at most one connected interval. A subset $S \subseteq \mathbb{R}^2$ is *y-monotone* if every horizontal line intersects S in at most one connected interval. A *semi-algebraic set* is a subset that is defined by polynomial inequalities.

An *arrangement* of n shapes is the set of faces, edges, and vertices that are defined by overlaying the shapes in a space such as $\mathbb{R}^\nu$. We will frequently refer to two functions when we discuss the complexity of arrangements. The inverse Ackermann function $\alpha(n)$ grows so slowly that $\alpha(n) \leq 4$ for all n less than or equal to an exponential tower $2^{2^{\cdot^{\cdot^{2}}}}$ with $65,536$ two's. Consequently, $\alpha(n)$ is often referred to as a "near-constant" function. The second function $\lambda_s(n)$ is the near-linear function that represents the length of a Davenport-Schinzel sequence [3, 4]. In our case, s is a constant that represents the maximum number of times that a pair of shapes can intersect, and n is the total number of shapes. Some known values for $\lambda_s(n)$ are $\lambda_1(n) = n$, $\lambda_2(n) = 2n - 1$, $\lambda_3(n) \in O(n\alpha(n))$. For more details, we refer the interested reader to [4].

2.1 Voronoi Diagram

A *Voronoi diagram* is a data structure for answering nearest neighbor queries. Given a set of shapes $A \in \mathbb{R}^\nu$, a Voronoi diagram is a partition of $\mathbb{R}^\nu$. In this case, ν denotes the dimension of our problem domain. Each face in this partition is associated with one shape $a \in A$ such that all points in the face for a are closer to a than to any other shape $a' \in A$ [17]. As an example, Figure 2.1 illustrates a two-dimensional Voronoi diagram for a set of points.

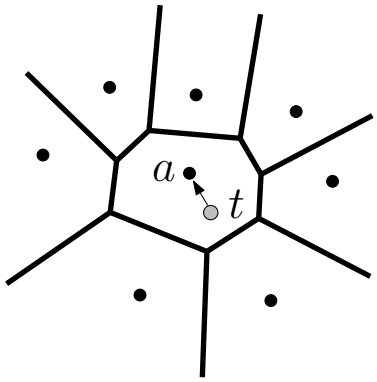


Figure 2.1: A Voronoi diagram for a set A of heavily shaded points. The lightly shaded query point t lies in the face for $a \in A$ and is closer to a than to any other heavily shaded point.

2.2 Shortest Path Map

Now assume that we want to compute shortest paths in a plane that contains impassable polygonal obstacles. A traditional *shortest path map* is a partition of this plane into a set of faces such that all points in a given face have the same combinatorial shortest path to a fixed source [66]. This means that all shortest paths to a given face have the same vertices except possibly for the start and end points of the path. See Figure 2.2. When the fixed source is a line segment $\overline{ab}$, the distance to a query point t is traditionally returned as the *shortest* distance from t to any point on $\overline{ab}$. Our work in Section 5.4 is different because we support queries from any desired source point $s \in \overline{ab}$.

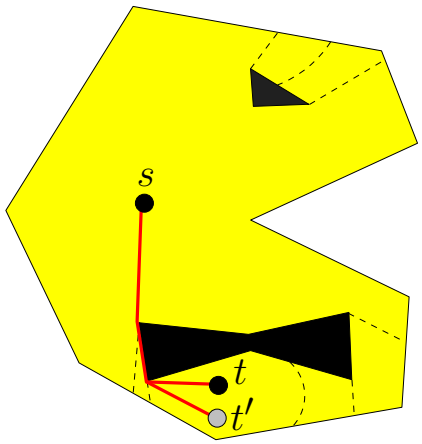


Figure 2.2: A shortest path map partitions a problem space such as the plane into a set of regions. Region boundaries are shown dashed. Every point inside a fixed region has the same combinatorial shortest path to the fixed source point s. For example, every shortest path from the source point s to a point in the region containing t will overlap the leftmost edge of Pac-Man's bow tie.

2.3 Hausdorff Distance

The *Hausdorff distance* is a similarity metric commonly used to compare sets of points or sets of higher-dimensional objects such as line segments or triangles. The *directed* Hausdorff distance is defined as $\tilde{\delta}_H(A, B) = \sup_{a \in A} \inf_{b \in B} d(a, b)$, where A and B are compact sets (see [10, 11]). The (*undirected*) Hausdorff distance is the larger of the two directed distances: $\delta_H(A, B) = \max(\tilde{\delta}_H(A, B), \tilde{\delta}_H(B, A))$.

Intuitively, the Hausdorff distance can be computed between *point* sets A and B as follows. Find for every point $a \in A$ the nearest neighbor distance $\min_{b \in B} d(a, b)$ and also compute for every point $b \in B$ the nearest neighbor distance $\min_{a \in A} d(a, b)$. The largest of these nearest neighbor distances equals the Hausdorff distance.

For points sets in the plane, the Hausdorff distance can be computed in $O(N \log N)$ time as follows [9]. Construct a Voronoi diagram V_A for the set A and a Voronoi diagram V_B for the set B in $O(N \log N)$ time [44]. For each $a \in A$, point location in V_B determines $\min_{b \in B} d(a, b)$ in $O(\log N)$ time. For each $b \in B$, point location in V_A determines $\min_{a \in A} d(a, b)$ in $O(\log N)$ time. The largest of these nearest neighbor distances is the Hausdorff distance. See Figure 2.3. A

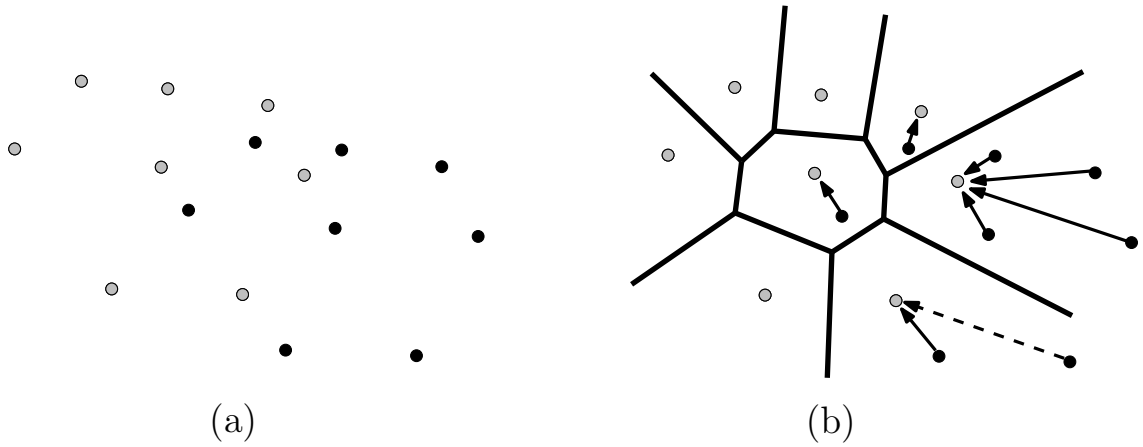


Figure 2.3: (a) The directed Hausdorff distance from the black point set to the gray point set can be computed as follows. Compute a Voronoi diagram of the gray point set as in (b). Use this Voronoi diagram to determine for each black point its nearest neighbor gray point. The largest of these nearest neighbor distances (shown dashed) is the directed Hausdorff distance.

similar technique can also be applied to sets of line segments in the plane [9]. For problem spaces where Voronoi diagrams are impractical to compute, the Hausdorff distance can be computed in brute-force fashion using $O(N^2)$ distances: one distance $d(a, b)$ for each pair $a \in A$, $b \in B$.

2.4 Fréchet Distance

The Fréchet distance is a similarity metric for continuous shapes (such as curves or surfaces) which is defined using reparameterizations of the shapes. The Fréchet distance is generally a more appropriate distance measure for continuous shapes than the often used Hausdorff distance because it takes the continuity of the shapes into account. See Figure 2.4.

Formally, the *Fréchet distance* for two curves $A, B : [0, 1] \to \mathbb{R}^\nu$ is defined as

$$\delta_F(A, B) = \inf_{f,g:[0,1]\to[0,1]} \sup_{t\in[0,1]} d(A(f(t)),\ B(g(t)))$$

where f and g range over continuous non-decreasing reparameterizations, and d is a distance metric for points (usually the L_2 distance).

The Fréchet distance is often illustrated by a person walking a dog on a leash [10]. The idea

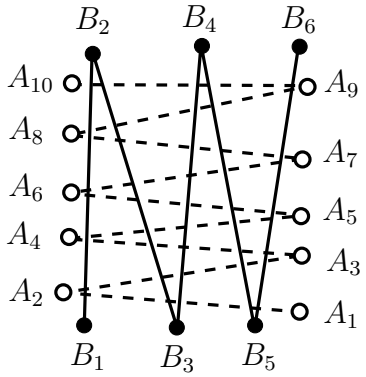


Figure 2.4: The Fréchet distance is a more accurate similarity measure than the Hausdorff distance because it takes the continuity of the shapes into account. For example, the Hausdorff distance between the solid polygonal curve and the dashed polygonal curve is small because every point on the solid curve is close to a point on the dashed curve and every point on the dashed curve is close to a point on the solid curve. However, the more accurate Fréchet distance between these two curves is large.

is that the person walks forward on one curve, and the dog walks forward on the other curve. As the person and dog walk along their respective curves, a leash is maintained to keep track of the separation between them, and the maximum separation attained during a walk defines a leash length.

By controlling their speeds, the person and dog can jointly walk over their curves in many different ways. Each of these joint walks corresponds to a reparameterization and defines a leash length that represents the maximum separation attained between the person and dog during that walk. The length of the *shortest* possible leash length over all possible reparameterizations defines the Fréchet distance. A short Fréchet distance (i.e., leash length) means the curves are similar; a large Fréchet distance means the curves are different.

To actually compute the Fréchet distance, an encoding of all possible reparameterizations for the person and dog is needed. The standard representation in [10] maps the two curves onto the axes of a parameter space called the *free space diagram*. For polygonal curves, the free space diagram is partitioned into *cells* by cutting at every position where a curve has a vertex. See Figure 2.5.

All *legal* reparameterizations for the Fréchet distance correspond to monotone paths from the

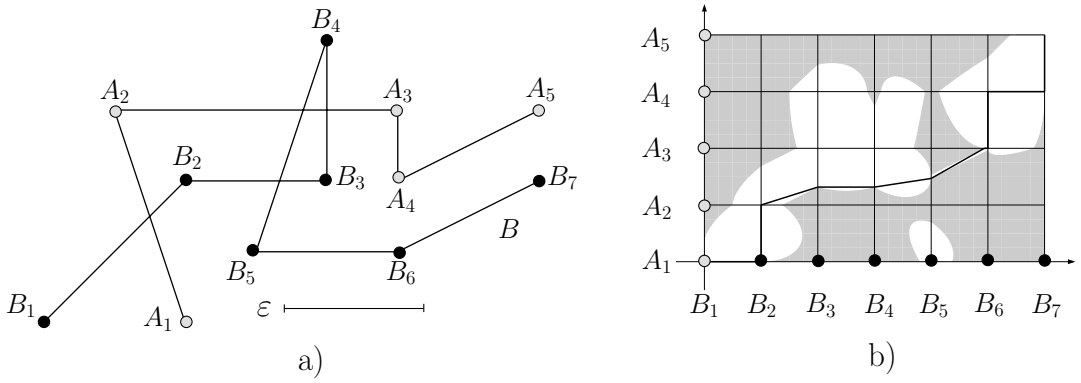


Figure 2.5: (a) Two polygonal curves can be mapped into (b) a free space diagram for a given constant $\varepsilon \geq 0$. This illustration was taken from [10] and modified slightly.

bottom-left corner of the free space diagram to the upper-right corner of the free space diagram. This follows for two reasons. First, the path must be monotone because the Fréchet distance is defined for non-decreasing reparameterizations (i.e., the person and dog are actually only allowed to move *forward* over their curves). Second, legal paths always begin at the bottom-left corner of the free space diagram and end at the upper-right corner because these positions force the person and dog to start moving at the beginning of both curves and to stop moving at the end of both curves.

Although the Fréchet distance requires non-decreasing reparameterizations, a variant called the *weak Fréchet distance* does not require the reparameterizations to be non-decreasing. Another variant called the *discrete Fréchet distance* only considers distances between the vertices of the shapes.

Notice that any point (s, t) in the free space diagram corresponds to a point s on the person curve and a point t on the dog curve. Consequently, (s, t) can be associated with the distance $d(s, t)$ between s and t. Assume a threshold value ε is fixed. We say that *free space* consists of all points (s, t) in the free space diagram such that $d(s, t) \leq \varepsilon$. Intuitively, free space corresponds to positions during the reparameterizations where the person and dog are close together. Alt and Godau [10] show that the free space is convex in a cell when the Fréchet distance is computed between two polygonal curves with respect to Euclidean distance.

The free space diagram provides a means of solving the *decision problem* for the Fréchet distance. For a given fixed value ε, the decision problem returns true if the Fréchet distance is at most ε. That is, the decision problem returns true if and only if a leash of length ε is long enough to permit some legal person and dog reparameterization through the free space. Alt and Godau [10] show that the decision problem can be solved for polygonal curves with dynamic programming. Their approach propagates reachability information in a left-to-right and bottom-to-top fashion through the cells of the free space diagram. Recall that each cell is the parameter space for a pair of line segments. *Reachable space* is the set of free space points that are reachable by a legal monotone path through the free space that originates from the bottom-left corner of the free space diagram. The decision problem returns true if and only if the upper right corner of the free space diagram is reachable.

After solving the decision problem, we need to calculate the shortest leash length ε^* such that the decision problem is true. This length ε^* equals the Fréchet distance. To guarantee that the exact value of ε^* is found in a continuous search space, parametric search with Cole's sorting trick (see [29]) is used to test a discrete set of critical values.

2.5 Parametric Search

Parametric search [5, 29, 79] is an optimization technique that can be used to compute the smallest value ε^* that satisfies a decision problem $\mathcal{D}(\varepsilon)$. This decision problem is required to be monotone in the sense that if $\mathcal{D}(\varepsilon_0)$ is true for ε_0, then $\mathcal{D}(\varepsilon)$ is true for all $\varepsilon \geq \varepsilon_0$.

Any sequential algorithm $A_{\mathcal{D}(\varepsilon)}$ for $\mathcal{D}(\varepsilon)$ has its control flow governed by a set of comparisons, where each comparison is resolved by testing the sign of a low degree polynomial in ε. The idea is to run $A_{\mathcal{D}(\varepsilon)}$ "generically" on the unknown value of ε^*. To resolve each comparison during this execution, we compute the roots of a polynomial and execute the decision problem on each of these roots. Since the decision problem is monotone, the candidate interval for ε^* must lie between two adjacent roots, and the sign of the polynomial between these two roots must be fixed. Hence, a comparison that is based on the sign of a polynomial at the unknown value of ε^* can be resolved,

and the generic execution of $A_{\mathcal{D}(\varepsilon)}$ can continue. As this execution proceeds, the candidate interval for ε^* shrinks due to repeatedly executing the decision problem on newly calculated roots. When the algorithm completes, the lower endpoint of the final candidate interval equals ε^*. The runtime of the above procedure is proportional to the number of comparisons required for the sequential algorithm $A_{\mathcal{D}(\varepsilon)}$ multiplied by the runtime for the decision problem $\mathcal{D}(\varepsilon)$.

To speed up the runtime of the above generic procedure, a batching technique is used. Instead of calling the decision problem on every root, each step of the algorithm computes a collection of roots (by simulating a parallel algorithm with a sequential algorithm). The monotonicity of the decision problem permits the decision problem to be answered for every root in this collection by calling the decision problem at each step of a binary search. Given the result of the decision problem for every root in the collection, the interval known to contain ε^* is updated and all comparisons associated with the batch can be resolved. The algorithm can then proceed to the next collection of roots. For more details on this process, please see [5].

A commonly used optimization of parametric search is Cole's sorting trick [29]. Cole realized that for many scenarios a sorting algorithm could be used to generate a superset of the roots needed by parametric search in $O(n \log n + T_{\mathcal{D}(\varepsilon)} \cdot \log n)$ time, where $T_{\mathcal{D}(\varepsilon)}$ is the time to solve the decision problem $\mathcal{D}(\varepsilon)$. His approach generically sorts n polynomials in ε that are defined by the control flow of the sequential algorithm $A_{\mathcal{D}(\varepsilon)}$.

A recent approach [79] has replaced the (complicated) sorting network advocated by Cole with the popular Quicksort algorithm. Their implementation improves the hidden constant factors and runs in $O(n \log n + T_{\mathcal{D}(\varepsilon)} \cdot \log^2 n)$ expected time for many scenarios.

As a final note, we comment that parametric search is mostly of theoretical interest because it involves huge constant factors and is usually impractical to implement. In practice, a straightforward binary search on the decision problem can be used to quickly converge to a value that is close to ε^*. Alternatively, there are special scenarios where practical alternatives to parametric search can be applied. For example, see our red-blue intersections technique in Section 4.4.

CHAPTER 3: TWO SPECIAL FRÉCHET DISTANCE SCENARIOS

The information in this chapter has not been previously published. The Fréchet distance is a similarity metric that is useful for comparing two sets of continuous shapes (see Chapter 2). Traditionally, the Fréchet distance is computed in $O(N^2 \log N)$ time for polygonal curves. Here, N is the total number of vertices defining the two input sets A and B that are being compared. The results in this chapter remove a factor of $O(\log N)$ from this runtime for two scenarios. Section 3.1 uses a planar graph to compute the weak Fréchet distance in $O(N^2)$ time. Section 3.2 shows how to compute the traditional Fréchet distance in $O(N^2)$ time when the input sets A and B are a special class of polygonal curves that lie on the boundary of one convex polygon. Related work has shown that the Hausdorff distance equals the Fréchet distance when A and B are *closed* convex curves [11].

3.1 Weak Fréchet distance

Although the Fréchet distance only considers *monotone* paths through the free space diagram, a variant called the *weak Fréchet distance* considers non-monotone paths. We show next how to compute the weak Fréchet distance in $O(N^2)$ time and space. This is an improvement over the traditional parametric search approach of Alt and Godau [10] that requires $O(N^2 \log N)$ time (see Section 2.5).

Theorem 3.1. *The weak Fréchet distance can be computed for two polygonal curves that have N vertices in $O(N^2)$ time whenever (1) the free space in each cell is connected and (2) the minimum distance on any cell boundary can be computed in constant time.*

Proof. The weak Fréchet distance can be computed by building a planar graph $\mathcal{G}$ on the free space diagram and running a shortest path algorithm on $\mathcal{G}$. To construct $\mathcal{G}$, determine the minimum distance on each cell boundary in the free space diagram, and place a node at each of these points. Additionally, place a start node at the bottom-left corner of the free space diagram, and place an

end node at the upper-right corner of the free space diagram. Inside each cell, add edges between the nodes to create a clique, and add Steiner nodes to enforce planarity (see Figure 3.1).

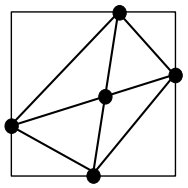


Figure 3.1: A four-clique for a free space cell can be made planar by adding a fifth node in the cell's interior.

Any path in $\mathcal{G}$ can be defined by a sequence of nodes such that each node in the sequence has an associated distance. Let the *weight* of a path in $\mathcal{G}$ be the maximum distance over all nodes in the sequence. Henzinger et al. [58] show that a minimal weight path can be computed in linear time for a planar graph that contains only non-negative distances. Since the planar graph $\mathcal{G}$ has $O(N^2)$ complexity and involves only non-negative distances, a path in $\mathcal{G}$ from the start node to the end node whose weight equals the Fréchet distance can be computed in $O(N^2)$ time and space. □

3.2 Fréchet distance for polygonal curves on a convex polygon

Let C be a convex polygon. Let A be a polygonal curve on the boundary of C that is oriented in the clockwise direction. Let B be a polygonal curve on the boundary of C that is disjoint from A and is oriented in the counter-clockwise direction. Let N be the total number of vertices defining A and B.

Theorem 3.2. *The Fréchet distance for A and B can be computed in $O(N^2)$ time and space.*

Proof. For each of the $O(N^2)$ cells in the free space diagram (see Section 2.4), compute the *minimum* distance on the cell's right and top boundary edges and place a node at these points. This can be done in $O(N^2)$ total time by repeatedly computing the shortest Euclidean distance between a point and a line segment. Additionally, place a start node at the bottom-left corner of the free space diagram, and place an end node at the upper-right corner of the free space diagram. By the

convexity of A and B, nodes always appear in monotone order in any given row or column of the free space diagram. See Figure 3.2.

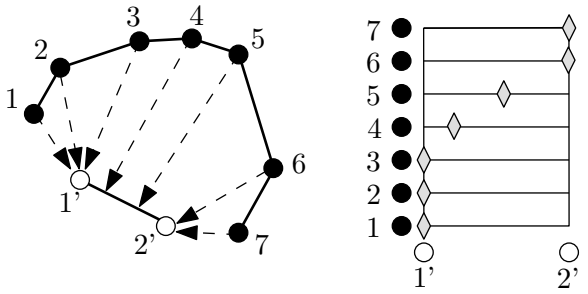


Figure 3.2: The diamond-shaped nodes in any column (or row) in the free space diagram form a monotone-increasing sequence.

Construct a directed acyclic graph G by connecting each node to the nearest node that lies above it and to the nearest node that lies to the right of it (see Figure 3.3). Since nodes always appear in monotone order in any given row or column of the free space diagram, the directed acyclic graph G represents only monotone-increasing paths.

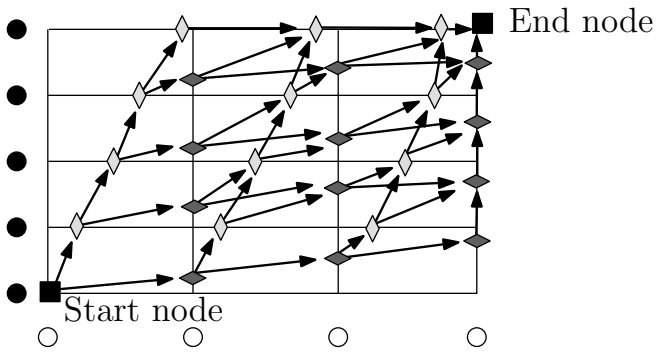


Figure 3.3: A directed acyclic graph for the free space diagram.

As in the previous Theorem, any path in G can be defined by a sequence of nodes, and each node in this sequence has an associated distance. Let the *weight* of a path in G be the maximum distance over all nodes in the sequence. The Fréchet distance equals the minimal possible weight over all paths in G from the start node to the end node. Consequently, a simple depth-first search in G can be used to compute a path whose weight equals the Fréchet distance.

The number of nodes and edges in the directed acyclic graph G is $O(N^2)$. This follows because

there are $O(N^2)$ cells in the free space diagram and $O(1)$ nodes and edges per cell. Hence, the depth-first search takes $O(N^2)$ time, and the Fréchet distance can be computed in $O(N^2)$ time. $\square$

CHAPTER 4: SIMILARITY METRICS INSIDE A SIMPLE POLYGON

This chapter presents algorithms to compute the Hausdorff distance and Fréchet distance inside a simple polygon, where the boundary of the simple polygon is considered to be an obstacle. The Hausdorff distance and Fréchet distance are similarity metrics that are useful for comparing sets of objects (see Chapter 2). Most of the work in this chapter has been published in the 2007 *Fall Workshop on Computational Geometry* [32], the 2008 *Symposium on Theoretical Aspects of Computer Science* [33] (27% acceptance rate), and has been accepted into the ACM journal *Transactions on Algorithms* [36].

Section 4.1 discusses shortest path structures called funnels and hourglasses. Sections 4.2 and 4.3 use funnels and hourglasses to represent all shortest paths needed to compute the Fréchet distance. Section 4.4 presents a practical alternative to parametric search that can be applied to the Fréchet distance. This algorithm is based on a variant of red-blue intersections and is very practical when compared to the traditional parametric search approach of Alt and Godau [10] (see Section 2.5). Section 4.5 applies this randomized algorithm to the Fréchet distance. Section 4.6 describes how to compute the Hausdorff distance.

Throughout this chapter, let k be the complexity of a simple polygon P that contains polygonal curves A and B in its interior. Let $\pi(a, b)$ denote the shortest path inside P between points a and b, and let $d(a, b)$ be the Euclidean length of $\pi(a, b)$. The following chapters frequently discuss bitonic functions. A *bitonic* function has at most one change in monotonicity. A function H is said to be "↓↑-bitonic" when it first decreases monotonically then increases monotonically.

4.1 Funnels and Hourglasses

Let $\overline{ab}$ and $\overline{cd}$ be line segments in a simple polygon P. Let the parameter space for $\overline{ab}$ and $\overline{cd}$ be the cell C. Each point in C is defined by a point $s \in \overline{ab}$ and a point $t \in \overline{cd}$ and is associated with the shortest path distance $d(s, t)$.

A funnel $\mathcal{F}_{p,\overline{cd}}$ [56] describes all shortest paths between a point p and any point on a line

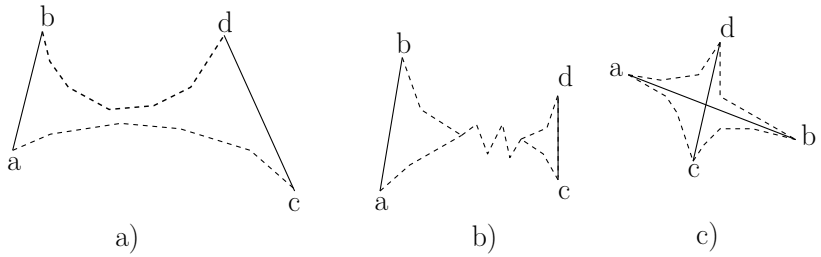


Figure 4.1: (a) An open hourglass, (b) a closed hourglass, and (c) an intersecting hourglass

segment $\overline{cd}$. Consequently, a funnel can be used to represent all shortest paths on a horizontal (or vertical) line segment in any cell C. The boundary of a funnel $\mathcal{F}_{p,\overline{cd}}$ is $\overline{cd} \circ \pi(d,p) \circ \pi(p,c)$, where $\pi(d,p)$ and $\pi(p,c)$ are shortest paths and $\circ$ denotes concatenation (see Figure 4.2a).

An hourglass $\mathcal{H}_{\overline{ab},\overline{cd}}$ [56] describes all shortest paths from any point on a line segment $\overline{ab}$ to any point on a line segment $\overline{cd}$. Consequently, an hourglass can be used to represent all shortest path distances for an entire cell C. For non-crossing segments, the boundary of the hourglass $\mathcal{H}_{\overline{ab},\overline{cd}}$ is $\overline{ab} \circ \pi(a,c) \circ \overline{cd} \circ \pi(d,b)$, and for crossing line segments the boundary is $\pi(a,c) \circ \pi(c,b) \circ \pi(b,d) \circ \pi(d,a)$ (see Figure 4.1). Both funnel and hourglass boundaries have $O(k)$ complexity because shortest paths inside a simple polygon P are simple, polygonal, and only have corners at vertices of P [57].

There are three types of hourglasses: open, closed, and intersecting. An *open hourglass* is defined by non-crossing $\overline{ab}$ and $\overline{cd}$ and two disjoint shortest paths $\pi(a,c)$ and $\pi(d,b)$. A *closed hourglass* has non-crossing $\overline{ab}$ and $\overline{cd}$ and a collapsed interior such that $\pi(a,c)$ and $\pi(d,b)$ share a common polygonal sub-path. An *intersecting hourglass* has crossing $\overline{ab}$ and $\overline{cd}$ and four shortest paths $\pi(a,c)$, $\pi(c,b)$, $\pi(b,d)$, and $\pi(d,a)$. Open, closed, and intersecting hourglasses are illustrated in Figure 4.1.

Assume the cell C is defined by the line segments $\overline{ab}$ and $\overline{cd}$, and assume that $p \in \overline{ab}$. Any horizontal or vertical line segment in C is associated with a funnel's distance function $F_{p,\,\overline{cd}} : [c,d] \to \mathbb{R}$ with $F_{p,\,\overline{cd}}(q) = d(p,q)$. The below three results are generalizations of Euclidean properties and will prove useful in Section 4.2 for analyzing the structure of a free space cell.

Lemma 4.1. *$F_{p,\,\overline{cd}}$ is piecewise parabolic with $O(k)$ complexity and is $\downarrow\uparrow$-bitonic.*

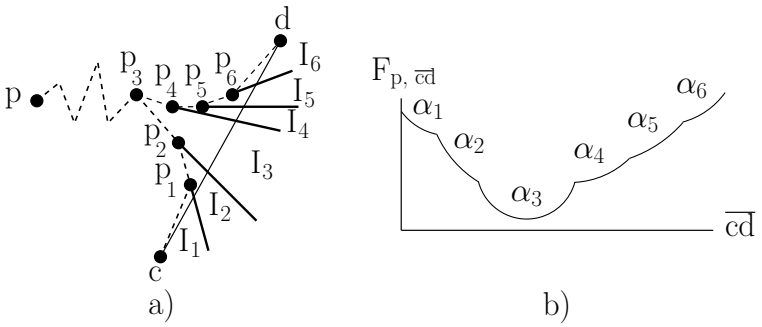


Figure 4.2: (a) A funnel $\mathcal{F}_{p,\overline{cd}}$ can be partitioned into $O(k)$ intervals such that funnel vertex p_j defines interval I_j. (b) The funnel's distance function $F_{p,\ \overline{cd}}$ is piecewise parabolic.

Proof. All shortest paths $\pi(p, q)$ in a simple polygon between a fixed point p and any point $q \in \overline{cd}$ can be described by a funnel $\mathcal{F}_{p,\overline{cd}}$ that is bounded by shortest paths $\pi(p, c)$ and $\pi(p, d)$. These two shortest paths may overlap for a period of time but must eventually diverge at one point (e.g., p_3 in Figure 4.2a). From the point of divergence, it is known that these two shortest paths (e.g., $\pi(p_3, d)$ and $\pi(p_3, c)$ in Figure 4.2a) are convex polygonal curves [56]. By extending all line segments on these convex curves into lines and intersecting these lines with $\overline{cd}$, a partition of $\overline{cd}$ into $O(k)$ intervals $I_1,\ I_2, ...,\ I_R$ is obtained (see Figure 4.2a).

All shortest paths $\pi(p, q)$ from p to any point $q \in I_j$ are polygonal and have either the form $\pi(p, p_i) \circ p_i,\ p_{i+1},\ ...,\ p_j,\ q$ or the form $\pi(p, p_i) \circ p_i,\ p_{i-1},\ ...,\ p_j,\ q$ where $p_i, ...,\ p_j$ are the funnel vertices visited by $\pi(p, q)$ after the point of divergence. For example, all shortest paths from p to $q \in I_1$ in Figure 4.2a have the form $\pi(p, p_3) \circ p_2,\ p_1,\ q$. Now assume that $q \in I_j$. Let L be the length of the path from p to p_j so that $d(p, q) = L + ||p_j - q||$. As q varies along I_j, L and p_j are fixed, so $d(p, q)$ equals the L_2-distance from a point to a line segment (plus the constant L). Consequently, $d(p, q)$ is a parabolic arc α_j as q varies along I_j. Hence, $F_{p,\ \overline{cd}}$ is piecewise parabolic with $O(k)$ complexity.

Observe that the *slopes* of the line segments defining the polygonal curves $\pi(c, p_3)$ and $\pi(p_3, d)$ in order from c to p_3 to d form a monotone sequence (see Figure 4.2a). This follows from the convexity of each curve. We now show that at most one arc of $F_{p,\ \overline{cd}}$ is bitonic, and the remaining arcs are monotone.

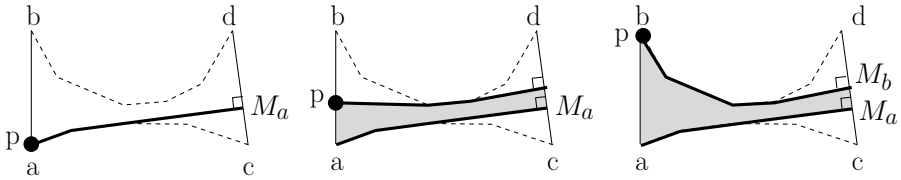


Figure 4.3: Shortest paths in the hourglass $\mathcal{H}_{\overline{ab},\overline{cd}}$ define $H_{\overline{ab},\ \overline{cd}}$.

Each interval I_j for $1 \leq j \leq m$ is defined by two rays R_{j-1} and R_j that originate from the funnel vertex p_j and intersect $\overline{cd}$. Let ζ be the line supported by $\overline{cd}$. Let $\perp_j$ be the line segment that touches p_j and is perpendicular to ζ. Arc α_j is bitonic if and only if $\perp_j$ lies strictly in the interior of I_j. Otherwise, α_j is monotone. Note that the slope μ of any perpendicular to ζ is a constant. Since I_j is defined by two rays $R_{j-1},\ R_j$ from p_j onto ζ, a perpendicular from p_j will only intersect ζ in the interior of I_j when the slope μ lies between the slopes of R_{j-1} and R_j. Since the ray slopes are monotone through the intervals $I_{1\ldots m}$, at most one bitonic arc α_v for $1 \leq v \leq m$ exists (e.g., α_3 in Figure 4.2a), and the remaining arcs are monotone. The ray slopes also ensure that the arcs $\alpha_{1\ldots(v-1)}$ are monotone decreasing and the arcs $\alpha_{(v+1)\ldots m}$ are monotone increasing. Hence, $F_{p,\ \overline{cd}}$ is ↓↑-bitonic. □

Corollary 4.2. *Any horizontal (or vertical) line segment in a free space cell C contains at most one connected set of free space points.*

Proof. A horizontal (or vertical) line segment in a cell C is associated with a funnel's distance function $F_{p,\ \overline{cd}}$, and free space consists of all values less than or equal to a given distance ε. Since Lemma 4.1 ensures that $F_{p,\ \overline{cd}}$ is ↓↑-bitonic, $F_{p,\ \overline{cd}}$ contains at most one connected set of free space points. □

Consider the hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$ in Figure 4.3. As p varies from a to b, the *minimum* distance from p to $\overline{cd}$ defines a function $H_{\overline{ab},\ \overline{cd}} : [a, b] \to \mathbb{R}$ with $H_{\overline{ab},\ \overline{cd}}(p) = \min_{q \in [c,d]} d(p, q)$.

Lemma 4.3. *$H_{\overline{ab},\ \overline{cd}}$ is ↓↑-bitonic.*

Proof. Let the *shortest* distance from a to any point on $\overline{cd}$ occur at $M_a \in \overline{cd}$. Similarly, let the shortest distance from b to any point on $\overline{cd}$ occur at $M_b \in \overline{cd}$ (see Figure 4.3). Observe that $H_{\overline{ab},\ \overline{cd}}$

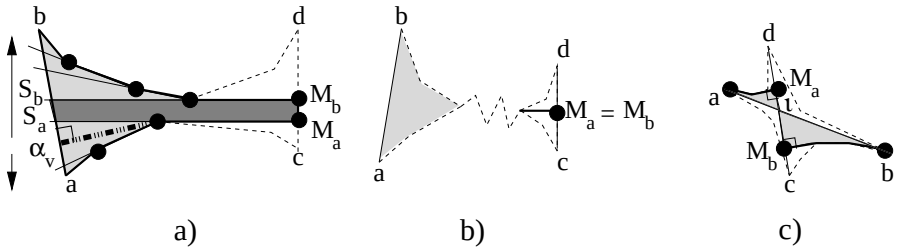


Figure 4.4: (a) An *open* hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$. $\mathcal{F}_{\overline{aS_a},\ M_a}$ and $\mathcal{F}_{\overline{S_bb},\ M_b}$ are lightly shaded; $\mathcal{L}$ is heavily-shaded. (b) A *closed* hourglass always has $M_a = M_b$. (c) An *intersecting* hourglass can be split into two (shaded) open hourglasses.

and $H_{\overline{ab},\ \overline{M_aM_b}}$ are identical functions. We now show that $H_{\overline{ab},\ \overline{M_aM_b}}$ is ↓↑-bitonic regardless of whether the hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is open, closed, or intersecting (cf. Figure 4.1).

Suppose that $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is an open hourglass. If $M_a = M_b$, then observe that the hourglass distance function $H_{\overline{ab},\ \overline{M_aM_b}}$ equals the funnel distance function $F_{\overline{ab},\ M_a}$[1] and is ↓↑-bitonic by Lemma 4.1. When $M_a \neq M_b$ the convexity and disjointness of the open hourglass boundary curves ensures that perpendiculars to $\overline{cd}$ through M_a and M_b intersect $\overline{ab}$ in two points S_a and S_b (see Figure 4.4a). Using S_a and S_b, the hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ can be split into three parts: two funnels $\mathcal{F}_{\overline{aS_a},\ M_a}$, $\mathcal{F}_{\overline{S_bb},\ M_b}$, and a heavily-shaded L_2-section $\mathcal{L}$ that lies in-between the two funnels. These three structures together define a ↓↑-bitonic distance function for any open hourglass by the proof of Lemma 4.1. This follows because the line segment slopes on the boundary curves form a monotone sequence from a to M_a along $\pi(a, M_a)$ and continuing from M_b to b along $\pi(M_b, b)$ (see Figure 4.4a). Since these three structures define $H_{\overline{ab},\ \overline{M_aM_b}}$, it is ↓↑-bitonic.

Suppose that $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is a closed hourglass. *All* shortest paths from $p \in \overline{ab}$ to the closest $q \in \overline{cd}$ will end at the same point M_a as shown in Figure 4.4b. Hence, $H_{\overline{ab},\ \overline{M_aM_b}}$ equals $F_{\overline{ab},\ M_a}$ and is ↓↑-bitonic by Lemma 4.1.

When $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is an intersecting hourglass, $\overline{ab}$ and $\overline{cd}$ will cross at the point ι as illustrated in Figure 4.4c. $H_{\overline{ab},\ \overline{M_aM_b}}$ has the form $H_{\overline{a\iota},\ \overline{\iota M_a}} \circ H_{\overline{\iota b},\ \overline{M_b\iota}}$, where $H_{\overline{a\iota},\ \overline{\iota M_a}}$ and $H_{\overline{\iota b},\ \overline{M_b\iota}}$ are distance functions for *open* hourglasses. By the above arguments on open hourglasses, $H_{\overline{a\iota},\ \overline{\iota M_a}}$ and $H_{\overline{\iota b},\ \overline{M_b\iota}}$

[1]Notice that $F_{\overline{ab},\ M_a}$ uses the *second* subscript for the apex point M_a. This emphasizes that this apex occurs on $\overline{cd}$ instead of on $\overline{ab}$.

are each (at worst) ↓↑-bitonic. It follows from $d(\iota, \iota) = 0$ that $H_{\overline{ab},\ \overline{cd}}$ is ↓↑-bitonic. □

4.2 Free Space Cell Properties

Consider a free space cell C for polygonal curves A and B inside a simple polygon. Let $\overline{ab} \subset A$ and $\overline{cd} \subset B$ be the two line segments defining C. Recall from Section 2 that *free space* consists of all shortest path distances that are less than or equal to a given distance ε.

Lemma 4.4. *For any $\varepsilon \geq 0$, the free space in a cell C is connected, x-monotone, and y-monotone.*

Proof. The monotonicity of the free space in C follows from Corollary 4.2. For connectedness, choose any two free space points $(p_1, q_1), (p_2, q_2)$, and construct a path connecting them in the free space as follows: move vertically from (p_1, q_1) to the minimum distance in C on the vertical line defined by p_1. Similarly, move vertically from (p_2, q_2) to the minimum associated distance in C on the vertical line defined by p_2. By Lemma 4.1, this movement causes the distance to decrease monotonically. By Lemma 4.3, any two minimum points are connected by a ↓↑-bitonic distance function $H_{\overline{ab},\ \overline{cd}}$ (cf. Section 4.1), but as the starting points are in the free space – and therefore have distance at most ε – all points on this constructed path lie in the free space. □

The boundary of a free space cell consists of vertical and horizontal line segments. Each boundary segment can be associated with a funnel $\mathcal{F}_{p,\ \overline{cd}}$ that has a ↓↑-bitonic distance function $F_{p,\ \overline{cd}}$ (cf. Lemma 4.1). Given $\varepsilon \geq 0$, computing the free space on a boundary segment requires finding the (at most two) values t_1, t_2 such that $F_{p,\ \overline{cd}}(t_1) = F_{p,\ \overline{cd}}(t_2) = \varepsilon$ (see Figure 4.5).

Lemma 4.5. *After preprocessing the simple polygon P in $O(k)$ time and space, both the minimum value of $F_{p,\ \overline{cd}}$ and the (at most two) values t_1, t_2 such that $F_{p,\ \overline{cd}}(t_1) = F_{p,}(t_2) = \varepsilon$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time. Hence, the free space on the boundary of a cell can be computed in $O(\log k)$ time.*

Proof. Although an explicit construction of $F_{p,\ \overline{cd}}$ would take $O(k)$ time, realize that it is not necessary to explicitly construct *all* the arcs of $F_{p,\ \overline{cd}}$. Instead, a binary search can find the intersections t_1 and t_2 of $\alpha_{1\ldots m}$ with ε by examining only $O(\log k)$ arcs.

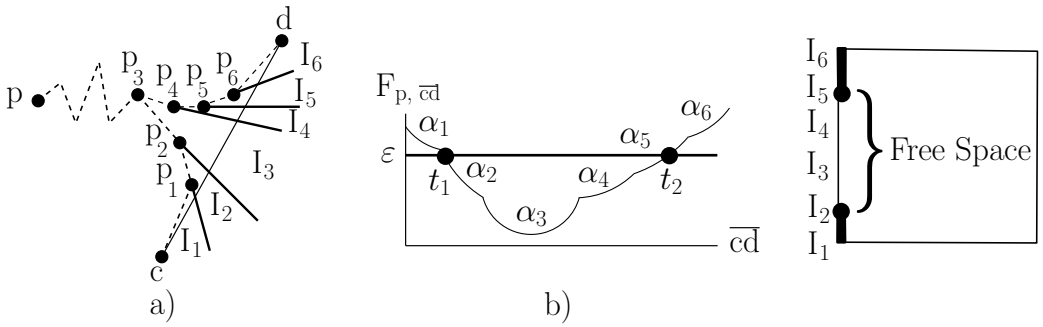


Figure 4.5: (a) A funnel $\mathcal{F}_{p,\,\overline{cd}}$ and its (b) bitonic distance function $F_{p,\,\overline{cd}}$. (c) Free space on a boundary segment of a free space cell is defined by the at most two values t_1, t_2 such that $F_{p,\,\overline{cd}}(t_1) = F_{p,\,\overline{cd}}(t_2) = \varepsilon$.

Any shortest path $\pi(p, q)$ can be represented as a balanced binary tree $\mathcal{T}$ in $O(\log k)$ time (after $O(k)$ preprocessing) by the algorithms of Guibas and Hershberger [55, 59]. $\mathcal{T}$ supports binary searches because it stores shortest path edges at its nodes such that "the edges along the [shortest path], taken in order, are the same as the edges stored in the nodes, taken in symmetric order" [59]. Even though $\mathcal{T}$ can have $O(k)$ complexity, it can be constructed in only $O(\log k)$ time by merging preconstructed trees together at query time [55].

Construct the binary search trees $\mathcal{T}_c$ and $\mathcal{T}_d$ for the two shortest paths $\pi(d, p)$ and $\pi(p, c)$, respectively. These shortest paths together define all arcs α_j of the piecewise parabolic distance function $F_{p,\,\overline{cd}}$, where $1 \le j \le m$. The bitonic arc of $F_{p,\,\overline{cd}}$ (e.g., α_3 in Figure 4.5b) can be found by searching $\mathcal{T}_c$ and $\mathcal{T}_d$ for the arc α_j with the smallest value among $\alpha_{1\ldots m}$. At most one bitonic arc exists because $\mathcal{T}_c$ and $\mathcal{T}_d$ together represent the shortest paths for the funnel $\mathcal{F}_{p,\overline{cd}}$ (cf. Lemma 4.1). After finding the bitonic arc, t_1 and t_2 can be found by binary searches over at most three monotone sequences of arcs defined by $\mathcal{T}_c$ and $\mathcal{T}_d$. □

Corollary 4.6. *$F_{p,\,\overline{cd}}$ can be evaluated at any $\varepsilon \ge 0$ in $O(\log k)$ time.*

Proof. This follows immediately from the balanced binary tree $\mathcal{T}$ representation for $F_{p,\,\overline{cd}}$ that is due to Guibas and Hershberger [55, 59]. □

4.3 Propagating Reachability Information Through a Cell in $O(1)$ Time

Reachability information determines which points on the cell boundary are reachable by a *monotone* path that originates from the bottom-left corner of the free space diagram and only travels through free space. Dynamic programming is traditionally used to propagate reachability information through each cell to solve the Fréchet decision problem [10]. More details on this process can be found in Section 2.4.

Lemma 4.7. *Given the free space on the four boundary segments of a cell C and given reachability information for the bottom and left boundaries of C, reachability information can be propagated to the upper and right boundaries of C in constant time.*

Proof. The standard argument used for convex free space [10] still applies. By Lemma 4.4, the free space is x-monotone, y-monotone, and connected. Hence, if some point on the left boundary is reachable, then the whole top boundary is reachable. If no point on the left boundary is reachable, then project the reachable points from the bottom boundary onto the top boundary. Analogous arguments hold for the bottom and right boundaries. □

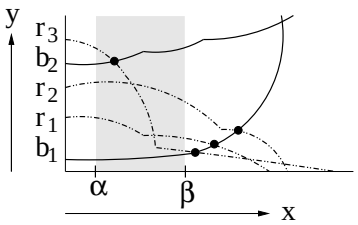
4.4 Red-Blue Intersections

We now show how to efficiently count and report a certain type of red-blue intersections in the plane. This problem will prove useful in Section 4.5 for the Fréchet optimization problem.

Let $R = \{r_1, ..., r_m\}$ and $B = \{b_1, ..., b_n\}$ be sets of continuous functions such that $r_i, b_j : [\alpha, \beta] \to \mathbb{R}$ for $1 \le i \le m$ and $1 \le j \le n$. Assume every "red" function r_i is strictly *decreasing* and every "blue" function b_j is strictly *increasing*. Let $I(k)$ be the time to find the at most one intersection of the functions r_i and b_j,[2] and assume r_i, b_j can be evaluated at any position in $E(k)$ time.

Theorem 4.8. *The number of red-blue intersections between R and B in the slab $[\alpha, \beta] \times \mathbb{R}$ can be* counted *in $O(N(E(k) + \log N))$ total time, where $N = \max(m, n)$. These intersections can be*

[2]There is at most one intersection due to the monotonicities of the red and blue functions.



Index	L_α	$\Gamma(r_i(\alpha))$	L_β	$\Gamma(r_i(\beta))$
5	$r_3(\alpha)$	2	$b_2(\beta)$	
4	$b_2(\alpha)$		$r_2(\beta)$	1
3	$r_2(\alpha)$	1	$r_1(\beta)$	1
2	$r_1(\alpha)$	1	$r_3(\beta)$	1
1	$b_1(\alpha)$		$b_1(\beta)$	

Figure 4.6: r_3 lies above *two* blue functions at $x = \alpha$ but only lies above *one* blue function at $x = \beta$. Subtraction reveals that r_3 has one intersection in the slab $[\alpha, \beta] \times \mathbb{R}$.

reported *in* $O(N(E(k)+\log N)+K\cdot I(k))$ *total time, where* K *is the total number of intersections reported. After* $O(N(E(k)+\log N))$ *preprocessing time, a* random *red-blue intersection in* $[\alpha, \beta]\times \mathbb{R}$ *can be returned in* $O(\log N + I(k))$ *time, and the red function involved in the most red-blue intersections in* $[\alpha, \beta] \times \mathbb{R}$ *can be returned in* $O(1)$ *time. All operations require* $O(N)$ *space.*

Proof. Figure 4.6 illustrates the key idea. Suppose a red function r_3 lies *above* a blue function b_2 at $x = \alpha$. If it is also true that r_3 lies *below* b_2 at $x = \beta$, then r_3 and b_2 must intersect in the slab $[\alpha, \beta] \times \mathbb{R}$ due to the continuity and monotonicity of r_3 and b_2. All red-blue intersections in the slab $[\alpha, \beta] \times \mathbb{R}$ can be counted by taking *snapshots* of the functions at $x = \alpha$ and $x = \beta$. Let the α-snapshot L_α be the list of functions in the order they intersect the vertical line $x = \alpha$. Let L_β be the β-snapshot at $x = \beta$. These snapshots can be computed in $O(N(E(k) + \log N))$ time by computing and sorting red and blue values.[3]

Let $\Gamma(r_i(x))$ be the number of $b_j \in B$ such that $r_i(x) > b_j(x)$ at a given value of x. The number of red-blue intersections for each r_i in the slab $[\alpha, \beta] \times \mathbb{R}$ is simply $\Gamma(r_i(\alpha)) - \Gamma(r_i(\beta))$ due to the continuity and monotonicity of the red and blue functions (see Figure 4.6). All $\Gamma(r_i(\alpha))$ (resp. $\Gamma(r_i(\beta))$) values can be computed by a linear scan over L_α (resp. L_β). Intersection counting simply sums up the number of intersections over all red functions, and this process also reveals which red function is involved in the most red-blue intersections in the slab $[\alpha, \beta] \times \mathbb{R}$.

We now *report* intersections instead of *counting* them. Note that we avoid enumerating all pos-

[3]During the sorting process, if a red function has the same value as a blue function at $x = \alpha$, then the red function should be considered greater than the blue function. By contrast, if a red function has the same value as a blue function at $x = \beta$, then the red function should be considered less than the blue function. This ensures that intersections directly at $x = \alpha$ or $x = \beta$ are counted and reported properly.

sible intersections and instead use the idea illustrated by Figure 4.6 to only compute intersections that are actually reported. An intersection in the slab $[\alpha, \beta] \times \mathbb{R}$ occurs when $r_i(\alpha) \geq b_j(\alpha)$ and $r_i(\beta) \leq b_j(\beta)$. To test these conditions, *incrementally* construct a balanced binary search tree T. Let $I_\beta(r_i)$ denote the rank of r_i in the list L_β, and let $I_\beta(b_i)$ denote the rank of b_i in L_β. Begin with an empty tree T, and march through L_α in ascending order (i.e., bottom-to-top in Figure 4.6). During this traversal, process each b_j by inserting $I_\beta(b_j)$ into T in $O(\log N)$ time. Process each r_i by querying the tree T with $I_\beta(r_i)$. At query time for r_i, T contains only indices for b_j such that $r_i(\alpha) \geq b_j(\alpha)$. All b_j with indices in T greater than the query index must intersect r_i because both conditions $r_i(\alpha) \geq b_j(\alpha)$ and $r_i(\beta) \leq b_j(\beta)$ are satisfied.

To find a *random* red-blue intersection in the slab $[\alpha, \beta] \times \mathbb{R}$, count the number κ of red-blue intersections in $[\alpha, \beta] \times \mathbb{R}$, and pick a random integer ρ between 1 and κ. To find the red curve r_i that is involved in the ρ-th red-blue intersection, preprocess each r_i in L_α to store the total number λ of red-blue intersections for *all* r_j that lie below r_i in L_α and perform a binary search. Once r_i is identified, we must find the $(\rho - \lambda)$-th blue curve that intersects r_i in the slab $[\alpha, \beta] \times \mathbb{R}$. Recall that the reporting structure T allows finding all intersections involving r_i if queries are made at an appropriate time during an incremental construction. Sarnak and Tarjan [74] have shown how to build a *persistent* tree in $O(N \log N)$ time such that any previous version of a tree after x insertions is available in $O(\log N)$ time. Precompute T as a persistent tree. At query time, find the tree T' that defines T after $\Gamma(r_i(\alpha))$ insertions. Search T' to identify the subtree such that every node in the subtree represents a red-blue intersection involving r_i in $[\alpha, \beta] \times \mathbb{R}$. The $(\rho - \lambda)$-th node in this subtree is the desired intersection. Hence, a persistent [74] version of the reporting structure allows a random red-blue intersection in the slab $[\alpha, \beta] \times \mathbb{R}$ to be returned in $O(\log N + I(k))$ query time after $O(N(E(k) + \log N))$ preprocessing time. □

4.5 Fréchet Distance

Theorem 4.9. *After preprocessing a simple polygon P for shortest path queries in $O(k)$ time [55], the Fréchet decision problem (see Section 2.4) for polygonal curves A and B inside P can*

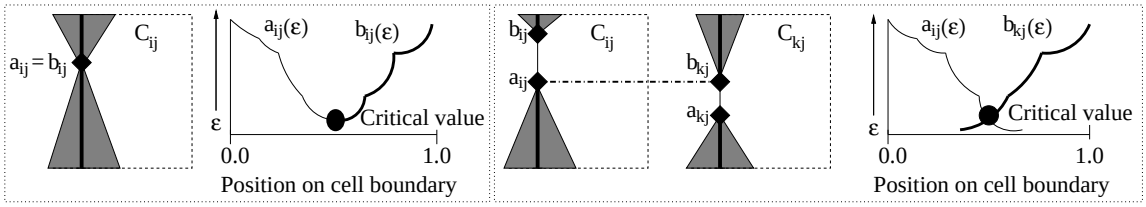


Figure 4.7: (a) Free space diagram and distance function for a type-B critical value of ε. (b) Free space diagram and distance function for a type-C critical value.

be solved for any $\varepsilon \geq 0$ *in* $O(N^2 \log k)$ *time and* $O(k + N)$ *space.*

Proof. Following the standard dynamic programming approach of [10], compute all cell boundaries in $O(N^2 \log k)$ time (cf. Lemma 4.5), and propagate reachability information through all cells in $O(N^2)$ time (cf. Lemma 4.7). $O(k)$ space is needed for the preprocessing structures of [55], and only $O(N)$ space is needed for dynamic programming if two rows of the free space diagram are stored at a time. □

For a given ε, a cell boundary segment of a cell C_{ij} has at most one free space interval (cf. Lemma 4.1). Denote the lower boundary of this interval as $a_{ij}(\varepsilon)$ and the upper boundary as $b_{ij}(\varepsilon)$ (see Figure 4.7). Let $\varepsilon^* = \delta_F(A, B)$ be the minimum value of ε such that the Fréchet decision problem returns true. Parametric search is a technique commonly used to find ε^* (see [5, 10, 29, 79] and Section 2.5). Using parametric search, ε^* is found by sorting all the functions $a_{ij}(\varepsilon), b_{ij}(\varepsilon)$ based on the unknown parameter ε^*. The comparisons performed during the sort guarantee that the result of the decision problem is known for all "critical values" [10] that could potentially define ε^*. Traditionally, such a sort operates on functions of constant complexity. The scenario is different for shortest paths because $a_{ij}(\varepsilon), b_{ij}(\varepsilon)$ have $O(k)$ complexity (cf. Lemma 4.1). A straightforward parametric search based on sorting $O(kN^2)$ constant complexity parabolic functions would require $O(kN^2 \log kN)$ time even when using Cole's [29] optimization.[4] We present a randomized algorithm with expected runtime $O(k + N^2 \log kN \log N)$.

[4] A variation of the general sorting problem called the "nuts and bolts" problem (see [63]) is tantalizingly close to an acceptable $O(N^2 \log N)$ sort but does not apply to our setting.

The seminal work of Alt and Godau [10] defines three types of critical values which are candidate values of ε for ε^*, and these critical values also apply to our setting inside a simple polygon. There are exactly two type-A critical values associated with distances between the starting points of A and B and the ending points of A and B. Type-B critical values occur $O(N^2)$ times when $a_{ij}(\varepsilon) = b_{ij}(\varepsilon)$. See Figure 4.7a. The $O(N^2)$ type-A and type-B critical values can be handled in $O(N^2 \log k \log N)$ total time by computing the critical values in $O(N^2 \log k)$ time,[5] sorting the values in $O(N^2 \log N)$ time, and finally performing a binary search using the decision problem. Resolving the type-A and type-B critical values as a first step defines a search interval for ε^* that simplifies the randomized algorithm for the type-C critical values.

Alt and Godau [10] show that type-C critical values occur when the position of $a_{ij}(\varepsilon)$ in a cell C_{ij} equals the position of $b_{kj}(\varepsilon)$ in cell C_{kj} in the free space diagram. See Figure 4.7b. As ε increases, Lemma 4.1 ensures that $a_{ij}(\varepsilon)$ is monotone decreasing on the cell boundary segment and $b_{ij}(\varepsilon)$ is monotone increasing (see Figure 4.7b). This means that $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ intersect at most once. Hence, there are $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ in row j and a total of $O(N^3)$ type-C critical values over all rows. There are also $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{ik}(\varepsilon)$ in *column* i and a total of $O(N^3)$ additional type-C critical values over all columns.

Lemma 4.10. *After preprocessing a simple polygon P for shortest path queries in $O(k)$ time [55], the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time.*

Proof. Recall that $a_{ij}(\varepsilon)$ is defined by the monotone decreasing component of a distance function $F_{p,\,\overline{cd}}$ and is composed of arcs that correspond to a subset of $\pi(d,p) \cup \pi(p,c)$. Similarly, $b_{kl}(\varepsilon)$ is defined by the monotone increasing component of $F_{p,\,\overline{cd}}$. Using the approach of Lemma 4.5, construct the balanced binary search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ in $O(\log k)$ time that are, respectively, associated with the shortest paths for $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$. Recall that these trees support binary searches because they store path edges at their nodes such that "the edges along the [shortest path], taken in order, are the same as the edges stored in the nodes, taken in symmetric order" [59]. We show

[5] Both of the type-A critical values can be computed in $O(\log k)$ time by computing shortest paths between the starting and ending points of A and B. Each type-B critical value occurs at the minimum value of $F_{p,\,\overline{cd}}$ and can be computed in $O(\log k)$ time by Lemma 4.5.

that a logarithmic search over $\mathcal{T}_a$ and $\mathcal{T}_b$ is sufficient to find the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ or report that no intersection exists.

Start at the roots of both trees and build the arcs α_a, α_b for the current nodes in $\mathcal{T}_a$, $\mathcal{T}_b$ in $O(1)$ time. If α_a and α_b intersect, then we are done. Otherwise, we need to move left or right in one of the trees such that no intersection is missed. The monotone decreasing nature of $a_{ij}(\varepsilon)$ ensures that all arcs left of α_a in the tree $\mathcal{T}_a$ must lie in the space defined by an infinite rectangular wedge r_{a1}, where r_{a1} is defined by the upper left quadrant with origin at the left endpoint of α_a. Similarly, r_{a2} is defined by the lower right quadrant with origin at the right endpoint of α_a, r_{b1} is defined by the lower left quadrant with origin at the left endpoint of α_b, and r_{b2} is defined by the upper right quadrant with origin at the right endpoint of α_b (see Figure 4.8). Let $\mathcal{A}$ be the "red" wedge-arc-wedge sequence defined by r_{a1}, α_a, and r_{a2}. Let $\mathcal{B}$ be the "blue" wedge-arc-wedge sequence defined by r_{b1}, α_b, and r_{b2}. The monotonicities of $\mathcal{A}$ and $\mathcal{B}$ ensure that at least one of the wedges will not have its boundary crossed by the other color. Hence, at least one of the wedges is guaranteed not to involve an intersection and can be discarded, so a binary search is sufficient to find an intersection if it exists.

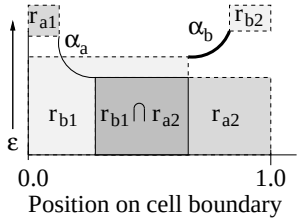


Figure 4.8: The intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ does not involve r_{a1}, α_b, and r_{b2}.

The runtime follows because each step of a binary search performs a constant amount of work on four wedges and two arcs. Since the binary search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ have $O(\log k)$ height, the total number of steps is $O(\log k)$. Hence, the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ can be found (or determined not to exist) in $O(\log k)$ time. □

The below randomized algorithm uses Theorem 4.8 to count type-C critical values for the $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ functions and solves the Fréchet optimization problem in $O(k + N^2 \log kN$

$\log N)$ expected time. This is faster than the standard parametric search approach which requires $O(kN^2 \log kN)$ time (see Section 2.5). The idea is to examine a *random* critical value to achieve a fast expected runtime. To improve the worst-case time, we maintain a pool of unresolved critical values in step (4). Although the a_{ij} and b_{kl} functions have $O(k)$ complexity, Corollary 4.6 ensures that these functions can be evaluated at any ε in $O(\log k)$ time, and Lemma 4.10 ensures that the (at most one) intersection of a_{ij} and b_{kl} can also be found in $O(\log k)$ time.

Algorithm 4.11. *Randomized Fréchet Distance*

1. Precompute and sort all type-A and type-B critical values in $O(N^2 \log kN)$ time (cf. Corollary 4.6 and Lemma 4.5). Run the decision problem $O(\log N)$ times to resolve these $O(N^2)$ values and shrink the search interval for ε^* down to $[\alpha, \beta]$ in $O(N^2 \log k \log N)$ time.

2. Count the number κ_j of type-C critical values for each row j in the slab $[\alpha, \beta] \times \mathbb{R}$ using Theorem 4.8. Let C_j be the resulting counting data structure for row j.

3. Pick a random intersection ϑ_j for each row using C_j (see Theorem 4.8).[6] Find the median Ψ of the $O(N)$ randomly selected ϑ_j in $O(N)$ time using a *weighted* median algorithm, where weights are based on the number of critical values κ_j for each row j.

4. Use C_j to find the curve $a_{Mj}(\varepsilon)$ in each row that has the most intersections (see Theorem 4.8). Add all intersections in $[\alpha, \beta] \times \mathbb{R}$ that involve $a_{Mj}(\varepsilon)$ to a global pool $\mathcal{P}$ of unresolved critical values and delete $a_{Mj}(\varepsilon)$ from any future consideration. Find the median Ξ of the values in $\mathcal{P}$ in $O(N^2)$ time using the standard median algorithm mentioned in [63].

5. Run the decision problem twice: once on Ξ and once on Ψ. This shrinks the search interval for ε^* and allows *at least* half the values in $\mathcal{P}$ to be discarded because the result of the decision problem is known for them. Repeat steps 2 through 5 until all *row*-based type-C critical values have been resolved.

[6] Picking a critical value at random is related to the distance selection problem [19] and is mentioned in [5], but to our knowledge this strategy has never been applied to the Fréchet distance.

6. Resolve all *column*-based type-C critical values in the same spirit as steps 2 through 5 and return the smallest critical value that satisfied the decision problem as the value of the Fréchet distance.

Theorem 4.12. *The Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P can be computed in $O(k+N^2 \log kN \log N)$ expected time and $O(k+N^3 \log kN)$ worst-case time, where N is the larger of the complexities of A and B and k is the complexity of P. $O(k + N^2)$ space is used.*

Proof. Preprocess P once for shortest path queries in $O(k)$ time [55]. In the expected case of Algorithm 4.11, each execution of the decision problem will eliminate a constant fraction of the remaining type-C critical values due to the proof of Quicksort's expected runtime and the weighted median approach for Ψ. Consequently, the expected number of iterations of Algorithm 4.11 is $O(\log N^3) = O(\log N)$.

In the worst-case, each of the $O(N)$ $a_{ij}(\varepsilon)$ in a row will be picked as $a_{Mj}(\varepsilon)$. Therefore, each row can require at most $O(N)$ iterations. Since *all* rows are processed in each iteration, Algorithm 4.11 requires at most $O(N)$ iterations for *row*-based critical values. By a similar argument, *column*-based critical values also require at most $O(N)$ iterations.

The size of the pool $\mathcal{P}$ is expressed by the inequality $S(x) \leq \frac{S(x-1)+O(N^2)}{2}$, where $S(0) = 0$ and x is the iteration number for the loop involving steps 2 through 5. Intuitively, each iteration adds $O(N^2)$ values to $\mathcal{P}$ and then at least half of the values in $\mathcal{P}$ are always resolved using the median Ξ. It is not difficult to show that $S(x) \in O(N^2)$ for any $x \geq 0$.

Each iteration of Algorithm 4.11 requires intersection counting and intersection calculations for $O(N)$ rows (or columns) at a cost of $O(N^2 \log kN)$ time. In addition, the global pool $\mathcal{P}$ has its median calculated in $O(N^2)$ time, and the decision problem is executed in $O(N^2 \log k)$ time. Consequently, the expected runtime is $O(k + N^2 \log kN \log N)$ and the worst-case runtime is $O(k+N^3 \log kN)$ including $O(k)$ preprocessing time [55] for shortest paths. The preprocessing structures use $O(k)$ space that must remain allocated throughout Algorithm 4.11, and the pool $\mathcal{P}$ uses $O(N^2)$ additional space. □

Although the traditional Fréchet distance (without obstacles) is normally computed in $O(N^2 \log N)$ time using parametric search (see [10]), parametric search is difficult to implement and involves enormous constant factors (see [5] and Section 2.5). Alternatively, Algorithm 4.11 provides a practical alternative to parametric search that can be used to compute the traditional Fréchet distance in $\mathbb{R}^v$. The main difference from Theorem 4.12 is that Euclidean distances can be computed in $O(1)$ time (instead of $O(\log k)$ time).

Corollary 4.13. *The traditional Fréchet distance (without obstacles) between two polygonal curves A and B in $\mathbb{R}^v$ can be computed in $O(N^2 \log^2 N)$ expected time, where N is the larger of the complexities of A and B. $O(N^2)$ space is required.*

4.6 Hausdorff Distance

The Hausdorff distance is a similarity metric that is commonly used to compare sets of points or sets of line segments (see Section 2.3). We compute the Hausdorff distance using shortest path distances in a simple polygon.

Theorem 4.14. *The Hausdorff distance $\delta_H(A, B)$ for point sets A, B inside a simple polygon P can be computed in $O((k + N) \log(k + N))$ time and $O(k + N)$ space, where N is the larger of the complexities of A and B and k is the complexity of P. If A and B are sets of line segments, $\delta_H(A, B)$ can be computed in $O((k + N) \log(k + N))$ time and space.*

Proof. The directed Hausdorff distance $\tilde{\delta}_H(A, B)$ is calculated in P by finding for each point $a \in A$ the shortest path distance to its nearest neighbor point in B and returning the maximum of these distances. To find nearest neighbors efficiently, precompute the Voronoi diagrams V_A, V_B for the point sets A and B inside the simple polygon P in $O((k + N) \log(k + N))$ time and $O(k + N)$ space using the algorithm of [72]. For each point $a \in A$, find its nearest neighbor $a' \in B$ in $O(\log(k+N))$ time and return the distance $d(a, a')$ via point location in V_B. Return the maximum of these distances as the value of $\tilde{\delta}_H(A, B)$. $\tilde{\delta}_H(B, A)$ can be computed in a similar manner.

For *line segment* sets A and B, the algorithm of Hershberger and Suri [61] can be used to

compute a Voronoi diagram in $O((k+N)\log(k+N))$ time and space. Candidate points in A for the directed Hausdorff distance $\tilde{\delta}_H(A,B)$ are defined by the $O(N)$ endpoints of the line segments in A and by the (at most two) extreme intersections of A with each of the $O(k+N)$ Voronoi edges in V_B. This follows because the distances on any Voronoi edge are bitonic, and a planesweep algorithm can be used to compute all candidate points in A [9]. Locating each of these candidate points in V_B via point location yields some maximum distance that equals $\tilde{\delta}_H(A,B)$. $\tilde{\delta}_H(B,A)$ can be computed in a similar manner. □

CHAPTER 5: SIMILARITY METRICS INSIDE A POLYGONAL DOMAIN

In this chapter, we develop algorithms to compute the Hausdorff distance and Fréchet distance inside a polygonal domain (a polygon with holes), where the edges of the polygonal domain are considered to be impassable obstacles. The Hausdorff distance and Fréchet distance are similarity metrics that are useful for comparing sets of objects (see Chapter 2). Most of the work in this chapter has been published as [34, 37] and was submitted to the journal *Computational Geometry: Theory and Applications* on January 30, 2009.

Sections 5.1 and 5.2 introduce shortest path structures in a polygonal domain. These structures are applied to the Fréchet distance in Section 5.3 and to a shortest path map in Section 5.4. Section 5.5 discusses the Hausdorff distance, and Section 5.6 investigates a homotopic variant of the Fréchet distance.

A shortest path between two line segments in a polygonal domain can have two forms. Let $s \in \overline{ab}$ and $t \in \overline{cd}$ be points on two line segments $\overline{ab}$ and $\overline{cd}$. Line of sight shortest paths $\pi(s,t) = s \circ t$ are represented by a structure that we call a *dynamic spotlight*. Shortest paths of the form $\pi(s,o_j) \circ t$ have their final turn at an obstacle vertex o_j and are represented by a structure that we call a *static spotlight*. Together, dynamic and static spotlights completely encode all shortest paths between $\overline{ab}$ and $\overline{cd}$.

5.1 Dynamic Spotlights

Line of sight shortest paths from $s \in \overline{ab}$ to $t \in \overline{cd}$ are represented by a structure that we call a *dynamic spotlight* (see Figure 5.1). Suppose that Δ_{s,o_i,o_j} is a triangle with apex $s \in \overline{ab}$, sides supported by $\overline{so_i}$ and $\overline{so_j}$, and base on $\overline{cd}$. Let $\mathcal{I}_{ab} \subset \overline{ab}$ be a maximal connected interval such that Δ_{s,o_i,o_j} contains no obstacles in its interior for any $s \in \mathcal{I}_{ab}$. A dynamic spotlight is defined as $\mathcal{L}_{\mathcal{D}}(\mathcal{I}_{ab}, o_i, o_j) = \{(s,t) \mid \pi(s,t) = s \circ t,\ s \in \mathcal{I}_{ab},\ t \in \Delta_{s,o_i,o_j} \cap \overline{cd}\}$. Note that the endpoints c and d are also candidates for o_i and o_j. As s varies over $\mathcal{I}_{ab}$, the maximal interval $\Delta_{s,o_i,o_j} \cap \overline{cd}$

that is directly visible from s changes continuously and defines $\mathcal{L}_\mathcal{D}$. The *free space* for $\mathcal{L}_\mathcal{D}$ is $\{(s,t) \in \mathcal{L}_\mathcal{D} \mid \|s-t\| \leq \varepsilon\}$. Both $\mathcal{L}_\mathcal{D}$ and its free space are contained in a cell C whose parameter space is defined by $\overline{ab}$ and $\overline{cd}$. Each dynamic spotlight can be associated with a unique visibility graph edge; thus, a cell C contains $O(k^2)$ non-empty dynamic spotlights. Note that each spotlight is interior-disjoint from all other spotlights in C because there is at most one line of sight path between any $s \in \overline{ab}$ and $t \in \overline{cd}$.

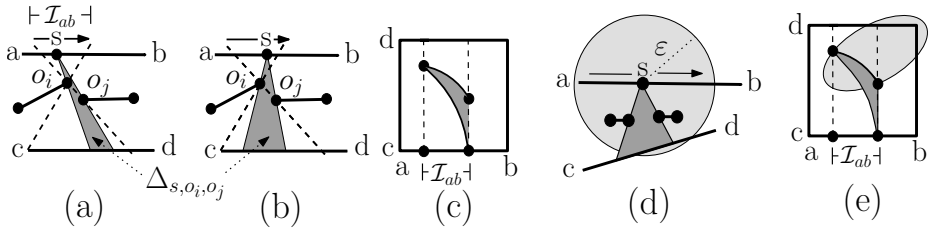


Figure 5.1: (a),(b) A dynamic spotlight $\mathcal{L}_\mathcal{D}(\mathcal{I}_{ab}, o_i, o_j)$ for different positions of $s \in \mathcal{I}_{ab}$. $\Delta_{s,o_i,o_j} \cap \overline{cd}$ changes continuously and defines (c) $\mathcal{L}_\mathcal{D}$ in a cell C. (d),(e) Free space for $\mathcal{L}_\mathcal{D}$ is the intersection of a lightly-shaded free space ellipse with the darkly-shaded dynamic spotlight $\mathcal{L}_\mathcal{D}$.

Lemma 5.1. *The free space for the $O(k^2)$ dynamic spotlights in a cell C has $O(k^2)$ complexity and can be computed in $O(k^2 \log k)$ time and $O(k^2)$ space.*

Proof. We compute all dynamic spotlights in C as follows. For every vertex o_j, perform an angular sweep around o_j to determine a partition of $[0, 2\pi]$ that encodes the angular ordering of vertices that are visible from o_j as well as intervals with line of sight to $\overline{ab}$ or to $\overline{cd}$. Overlaying this partition with a copy shifted by π yields a partition that encodes bidirectional line of sight information along a line through o_j. Each interval in this partition encodes line of sight to *both* $\overline{ab}$ and $\overline{cd}$ and defines a dynamic spotlight. The partition for each o_j has $O(k)$ complexity and can be computed in $O(k \log k)$ time. After computing the $O(k^2)$ dynamic spotlights in C, we intersect each spotlight with the free space ellipse [10] that corresponds to the Euclidean free space between $\overline{ab}$ and $\overline{cd}$ (see Figure 5.1e).[1] Since all spotlights are interior-disjoint, this computation yields the union of the free spaces for all dynamic spotlights in C. □

[1]The Euclidean free space $\{(s,t) \mid s \in \overline{ab},\ t \in \overline{cd},\ \|s-t\| \leq \varepsilon\}$ is an *ellipse* [10] in C because the disk described by $\|s-t\| \leq \varepsilon$ undergoes an affine transformation when mapped to the parameter space.

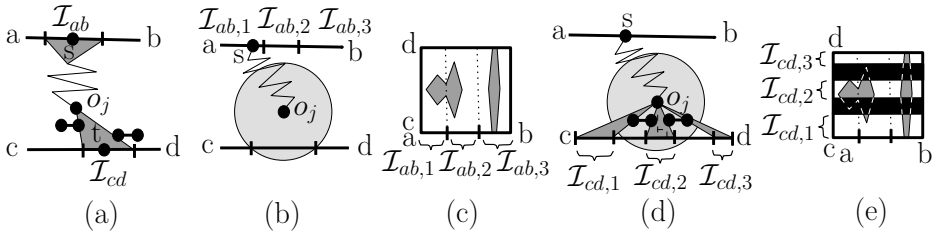


Figure 5.2: **(a)** A static spotlight represents paths of the form $\pi(s, o_j) \circ t$. **(b)** A circle centered at o_j with radius $\varepsilon - d(s, o_j)$ defines a candidate set of free space points in the **(c)** cell C. **(d)** Line of sight from o_j can be restricted to visible intervals $\mathcal{I}_{cd,1}, ..., \mathcal{I}_{cd,O(k)} \subset \overline{cd}$ by cutting out **(e)** darkly-shaded horizontal slabs.

5.2 Static Spotlights

Paths from $s \in \overline{ab}$ to $t \in \overline{cd}$ that have their final turn at an obstacle vertex o_j have the form $\pi(s, o_j) \circ t$ and are represented by a structure that we call a *static spotlight* (see Figure 5.2). Let $\mathcal{I}_{ab}$ be an interval on the partition of $\overline{ab}$ that is defined by SPM(o_j) $\cap\ \overline{ab}$. Note that $\pi(s, o_j)$ has the same combinatorial structure for all $s \in \mathcal{I}_{ab}$. Let $\mathcal{I}_{cd}$ be one of the $O(k)$ maximal connected intervals on $\overline{cd}$ with line of sight to o_j. A static spotlight is defined as $\mathcal{L}_{\mathcal{S}} = \{(s, t) \mid s \in \mathcal{I}_{ab},\ t \in \mathcal{I}_{cd}\}$. A cell C defined by line segments $\overline{ab}$ and $\overline{cd}$ contains $O(k^3)$ non-empty static spotlights because each obstacle vertex defines $O(k^2)$ possible $(\mathcal{I}_{ab}, \mathcal{I}_{cd})$ pairs. The *free space* for $\mathcal{L}_{\mathcal{S}}(\mathcal{I}_{ab}, o_j, \mathcal{I}_{cd})$ is $\{(s, t) \in \mathcal{L}_{\mathcal{S}} \mid d(s, o_j) + \|o_j - t\| \leq \varepsilon\}$ (see Figure 5.2).

Unlike dynamic spotlights, static spotlights represent shortest path *candidates* that can include suboptimal paths. This follows because a static spotlight for o_j forces a path from s to t to go through o_j even when $\pi(s, t)$ does not have its final turn at o_j. However, every shortest path $\pi(s, t)$ must be represented by at least one dynamic or static spotlight because shortest paths can only turn at obstacle vertices [14, 64].

Lemma 5.2. *The free space for the $O(k^3)$ static spotlights in a cell C has $O(k^3)$ complexity and can be computed in $O(k^3)$ expected time or $O(k^2 \lambda_6(k))$ deterministic time.*

Proof. For a fixed obstacle vertex o_j, SPM(o_j) $\cap\ \overline{ab}$ partitions $\overline{ab}$ into intervals $\mathcal{I}_{ab,1}$, ... , $\mathcal{I}_{ab,O(k)}$ [66]. These intervals induce $O(k)$ vertical slabs in the cell C. Within each slab, the free space inequality $d(s, o_j) + \|o_j - t\| \leq \varepsilon$ describes a *semi-algebraic set* of constant description complexity

(see Figure 5.2c and Chapter 2). The overlay of the vertical slabs for o_i and o_j has $O(k)$ complexity, and within each of the $O(k)$ overlayed slabs the semi-algebraic sets for o_i and o_j have $O(1)$ intersections. Thus, the union of semi-algebraic sets has $O(k \cdot k^2)$ complexity over all pairs of obstacle vertices.

We now enforce line of sight information from each o_j onto the maximal intervals $\mathcal{I}_{cd,1}$, ..., $\mathcal{I}_{cd,O(k)} \subset \overline{cd}$ that are visible from o_j (see Figures 5.2d and 5.2e). For each of the $O(k)$ intervals on $\overline{cd}$ that is *not* directly visible to o_j, we cut out a horizontal slab from the semi-algebraic set for o_j. Each of these $O(k)$ horizontal slabs intersects the semi-algebraic set for o_j $O(k)$ times, yielding $O(k^2)$ vertices per o_j. Thus, the free space has $O(k^3)$ complexity. It can be computed using an incremental algorithm [4] in $O(k^3)$ expected time or $O(k^2\lambda_6(k))$ deterministic time.[2] □

5.3 Fréchet Distance

This chapter computes the Fréchet distance with respect to Euclidean shortest paths for two polygonal curves A, B in a polygonal domain with k vertices. We use the arrangement of dynamic and static spotlights defined in Sections 5.1 and 5.2 to construct the free space inside C.

5.3.1 Decision Problem

Lemma 5.3. *Free space in a cell C has $O(k^4)$ complexity and can be constructed for any $\varepsilon \geq 0$ in $O(k^4)$ expected time or $O(k^3\lambda_6(k))$ deterministic time.*

Proof. The free space in a cell C is the union of the free spaces for the dynamic and static spotlights (see Lemmas 5.1 and 5.2). Each of the $O(k^2)$ dynamic spotlights can intersect the semi-algebraic set for o_j (see Figure 5.2c) $O(k)$ times, for a total of $O(k \cdot k^3)$ intersections of this type over all o_j. Each of the $O(k^2)$ dynamic spotlights can also intersect the $O(k^2)$ line of sight enforcing horizontal slabs (see Figure 5.2e). Hence, the arrangement of the dynamic and static spotlights has $O(k^4)$ complexity. The arrangement can be computed with an incremental algorithm [4]. □

[2]The deterministic construction is based on a near-linear function $\lambda_{s+2}(k)$ that is defined by the length of a Davenport-Schinzel sequence (see Chapter 2). Note that $s = 4$ is the number of intersections between two semi-algebraic sets that are defined by quadratic functions.

We now give an $\Omega(k^2)$ lower bound on the complexity of a cell C and an $\Omega(N^2k^2)$ lower bound on the complexity of the N^2 cells defining a free space diagram.

Lemma 5.4. *The free space diagram for the Fréchet distance between polygonal curves A and B in a polygonal domain with k vertices can have $\Omega(N^2k^2)$ complexity, where N is the complexity of A and B.*

Proof. Figure 5.3 illustrates a situation where $\Omega(k^2)$ points in a cell have the same shortest path distance, and all other distances in the cell are larger. Hence, for an appropriate choice of ε, the free space in this cell is defined by $\Omega(k^2)$ disjoint regions. Define the polygonal curves by their

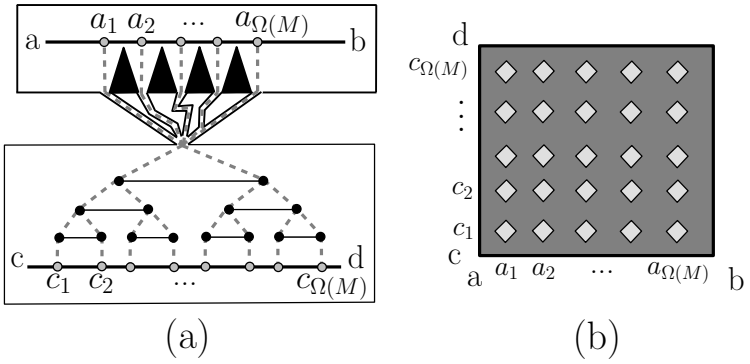


Figure 5.3: (a) Two line segments in a polygonal domain can define (b) a cell with $\Omega(k^2)$ complexity. Dashed lines depict shortest paths.

vertices as $A = \{a, b, a, b, ..., a, b\}$ and $B = \{c, d, c, d, ..., c, d\}$. Since the $\frac{N^2}{4}$ cells defined by $\overline{ab}$ and $\overline{cd}$ each have $\Omega(k^2)$ complexity, the free space diagram has $\Omega(N^2k^2)$ complexity. □

Theorem 5.5. *The Fréchet* decision problem *for polygonal curves A and B in a polygonal domain with k vertices can be answered for any constant $\varepsilon \geq 0$ in $O(N^2k^4 \log k)$ time, where N is the complexity of A and B. The* weak decision problem *can be answered in $O(N^2k^4)$ expected time or $O(N^2k^3\lambda_6(k))$ deterministic time. Both approaches use $O(Nk + k^4)$ space.*

Proof. The decision problem can be answered using dynamic programming [10] by propagating reachability information through each of the $O(N^2)$ cells with a plane sweep in $O(k^4 \log k)$ time. The weak decision problem can be answered by marking a set of connected free space components as reachable and checking if the upper-right corner of the free space diagram is reachable.

All shortest paths on a horizontal or vertical line segment in a cell can be described by a fixed-source shortest path map with $O(k)$ complexity [61, 66]. Thus, there are at most $O(k)$ *free space* and *reachable space* intervals on any cell boundary, and all cell boundaries on the two rows required for dynamic programming [10] can be stored in $O(Nk)$ space. An additional $O(k^4)$ storage is needed to propagate reachability through a single cell. □

5.3.2 Optimization Problem

Critical values [10] are candidate values for ε that are caused by a geometric configuration change of the free space. The smallest critical value ε^* that causes the decision problem to return true defines the exact value of the Fréchet distance. The standard approach [10] to find ε^* is to apply parametric search with Cole's [29] sorting trick (see Section 2.5). In the following theorem, we apply parametric search to a set of parameterized free space vertices.

Theorem 5.6. *The Fréchet distance for two polygonal curves A and B in a polygonal domain with k vertices can be computed in $O(N^2k^4 \log k \log Nk)$ time, where N is the complexity of A and B. The* weak *Fréchet distance can be computed in $O(N^2k^4 \log Nk)$ expected time or $O(N^2k^3\lambda_6(k) \log Nk)$ deterministic time. Both approaches use $O(Nk + k^4)$ space.*

Proof. In Lemmas 5.2 and 5.3, each of the $O(k^4)$ free space vertices was described combinatorially as the intersection of two semi-algebraic sets. Since the semi-algebraic sets depend on ε, the location of an intersection point in $\mathbb{R}^2$ can be described by a (possibly partially defined) algebraic curve $\rho_i(\varepsilon)$ that has constant degree and description complexity.

There are three types of critical values. Type-A critical values are values of ε such that some $\rho_i(\varepsilon)$ touches a corner of the free space diagram. Type-B critical values occur when two $\rho_i(\varepsilon)$ intersect or when free space becomes tangent to a cell boundary. Monotonicity-enforcing type-C critical values occur when a pair of intersection points lie on a horizontal/vertical line.[3]

Recall from Section 2.5 that parametric search with Cole's [29] sorting trick requires $O(n \log n + T_{\mathcal{D}(\varepsilon)} \log n)$ time, where $T_{\mathcal{D}(\varepsilon)}$ is the time to solve the decision problem and n is the number of

[3]Note that type-C critical values can be ignored for the weak Fréchet distance.

polynomial functions being sorted. In our case, we have $O(k^4)$ $\rho_i(\varepsilon)$ functions to sort for each of the $O(N^2)$ cells, so $n \in O(N^2k^4)$. Furthermore, $T_{\mathcal{D}(\varepsilon)} \in O(N^2k^4 \log k)$ by Theorem 5.5. Hence, the Fréchet distance can be computed in $O(N^2k^4 \log k \log Nk)$ time. The weak Fréchet distance can be computed similarly. The space requirement is identical to the decision problem. □

5.4 Two-Segment Shortest Path Map

Chiang and Mitchell [28] support queries between any two points in a polygonal domain with k vertices after $O(k^{11})$ preprocessing. Using our spotlight structures, we define a shortest path map SPM($\overline{ab}, \overline{cd}$) that supports optimal query time from any source point $s \in \overline{ab}$ to any destination point $t \in \overline{cd}$ after $O(k^4\lambda_6(k))$ preprocessing.

All dynamic spotlights encode $\pi(s,t) = s \circ t$ paths and contribute to SPM($\overline{ab}$, $\overline{cd}$). Unlike dynamic spotlights which are always interior-disjoint and consist entirely of shortest paths, static spotlights can overlap and encode suboptimal paths. The main task to construct SPM($\overline{ab}, \overline{cd}$) is to "clip" the static spotlights into an interior-disjoint set of optimal paths. Regions of overlap between all static spotlights defined by o_j and o_l can be resolved with a hyperbolic *bisector* $B_{jl}(s) = \{(s,t) \mid d(s,o_j)+\|o_j - t\| = d(s,o_l) + \|o_l - t\|\}$ that is commonly used in additively weighted Voronoi diagrams [17] (see Figure 5.4a). Hershberger and Suri [61] have shown that for a fixed s, $B_{jl}(s)$ intersects $\overline{cd}$ in at most two points.

Theorem 5.7. *SPM($\overline{ab}, \overline{cd}$) can be constructed in $O(k^5)$ expected time or $O(k^4\ \lambda_6(k))$ deterministic time and $O(k^5)$ space. After this preprocessing, queries $d(s,t)$, $\pi(s,t)$ for any $s \in \overline{ab}$ and $t \in \overline{cd}$ take $O(\log k + K)$ time, where K is the complexity of any returned path. SPM($\overline{ab}, \overline{cd}$) can have $\Omega(k^2)$ complexity.*

Proof. The goal is to partition the parameter space defined by $\overline{ab}$ and $\overline{cd}$ such that for all points (s,t) in a region the shortest path $\pi(s,t)$ has the same combinatorial structure. We obtain this partition by inserting a static spotlight bisector for each pair of obstacles into the $O(k^4)$ vertical slab arrangement of the dynamic and static spotlights (see Lemmas 5.2 and 5.3). We show next

that the arrangement with these bisectors has $O(k^5)$ complexity.

Consider the overlay of the $O(k)$ vertical slabs defining non-empty static spotlights for o_i and o_j. Inside each of these slabs, $B_{jl}(s) \cap \overline{cd}$ is a semi-algebraic set of constant complexity (see Figure 5.4). Since there are $O(k)$ vertical slabs per *pair* o_j, o_l, the bisectors define $O(k^3)$ constant-complexity semi-algebraic sets.

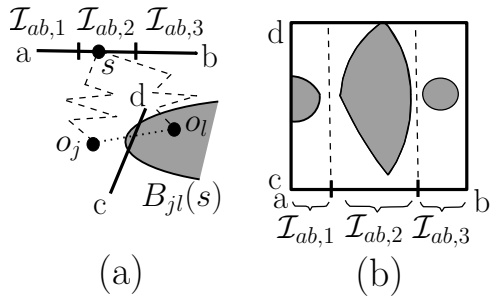


Figure 5.4: (a) For a fixed value of s, the bisector $B_{jl}(s)$ is a hyperbolic arc that intersects $\overline{cd}$ at most twice. (b) Inside a vertical slab in the overlay for o_j, o_l, $B_{jl}(s) \cap \overline{cd}$ is a semi-algebraic set with constant complexity. Darkly shaded points satisfy $d(s, o_l) + \|o_l - t\| \leq d(s, o_j) + \|o_j - t\|$ for all $t \in \overline{cd}$. Unshaded points satisfy $d(s, o_l) + \|o_l - t\| > d(s, o_j) + \|o_j - t\|$.

Each of these $O(k^3)$ semi-algebraic sets intersects: (1) each of the $O(k^2)$ dynamic spotlights $O(1)$ times (because the intersection of two constant complexity sets has constant complexity) and (2) each of the $O(k^2)$ line of sight enforcing horizontal slabs for the static spotlights $O(1)$ times. Hence, the arrangement has $O(k^5)$ complexity. An incremental algorithm (see [4]) can build both the arrangement and a point location structure in $O(k^5)$ expected time or $O(k^4 \lambda_6(k))$ deterministic time and $O(k^5)$ space. The $\Omega(k^2)$ lower bound follows easily from Lemma 5.4. □

5.5 Hausdorff Distance

The *Hausdorff distance* is a similarity metric used to compare two compact sets (see Section 2.3).

Theorem 5.8. *The Hausdorff distance can be computed using shortest paths in a polygonal domain in $O((k + N) \log(k + N))$ time and space between sets A and B, where A and B are either sets of points or sets of line segments. The complexity of the polygonal domain is k, and N is the complexity of A and B.*

Proof. The algorithm of [61] can be used to compute a Voronoi diagram in $O((k+N)\log(k+N))$ time and space for a set of points or line segments. For point sets, the Voronoi diagram can be used to compute the nearest neighbor distance $\min_{b \in B} d(a,b)$ for each $a \in A$ in $O(\log(k+N))$ time. For line segment sets, the Voronoi diagram can be used to identify and resolve $O(k+N)$ candidate values for the Hausdorff distance using a plane sweep algorithm that is described in [9]. Hence, the Hausdorff distance can be computed in $O((k+N)\log(k+N))$ time and space. See Theorem 4.14 for more details. □

5.6 Continuous Fréchet Distance

Consider the situation of a person walking a dog in a park. If the person and dog walk on opposite sides of a group of trees, then the leash will wrap around the trees. More formally, suppose the two polygonal curves A and B lie in a planar polygonal domain $\mathcal{D}$ [66] of complexity k. The leash is required to change continuously, i.e., it must stay inside $\mathcal{D}$ and may not pass through or jump over an obstacle. It may, however, cross itself. Let δ_C be the Fréchet distance for this scenario when the leash length is measured using a shortest path.

Due to the continuity of the leash's motion, the free space inside a free space cell is represented by an hourglass – just as it was for the Fréchet distance inside a simple polygon. Hence, free space in a cell is x-monotone, y-monotone, and connected (cf. Lemma 4.4), and reachability information can be propagated through a cell in constant time (cf. Lemma 4.7). We compute δ_C for a single initial leash that can be defined by computing a shortest path between the start points of the polygonal curves A and B in $O(k \log k)$ time [66]. This initial leash defines a *homotopy class* that represents the set of all paths that can be continuously deformed into each other without crossing any obstacles [60].

The main task in computing δ_C is to construct funnels for all cell boundaries. Once these funnels are known, the decision and optimization problems can be solved by the algorithms for the Fréchet distance inside a simple polygon (cf. Theorems 4.9 and 4.12). We use Hershberger and Snoeyink's homotopic shortest paths algorithm [60] to incrementally construct all funnels needed

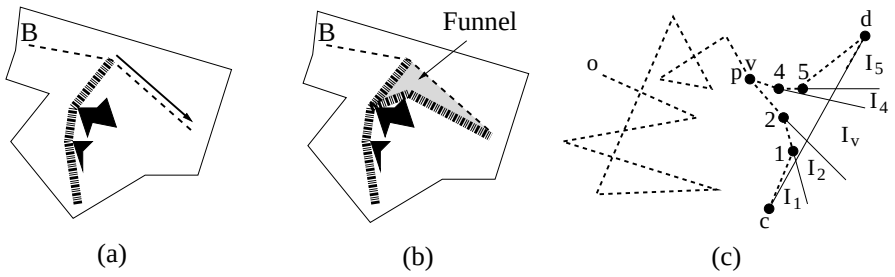


Figure 5.5: (a) A funnel for a δ_C-cell can be found by first extending an initial homotopic shortest path along one segment to create a path sketch and then (b) snapping this sketch into a homotopic shortest path. (c) Such a funnel $\mathcal{F}_{o,\,\overline{cd}}$ has $O(kN)$ complexity, but the distance function $F_{o,\,\overline{cd}}$ has only $O(k)$ complexity because $d(o,p)$ is a constant.

to compute δ_C. To use the homotopic algorithm, the polygonal domain $\mathcal{D}$ should be triangulated in $O(k \log k)$ time [66], and all obstacles should be replaced by their vertices.

Lemma 5.9. *After precomputing a homotopic shortest path for the bottom-left corner of a δ_C-cell $\mathcal{C}$, all four funnels representing the boundary segments of $\mathcal{C}$ and the homotopic shortest paths for the bottom-left corners of cells adjacent to $\mathcal{C}$ can be computed in $O(k)$ time.*

Proof. The funnels representing cell boundaries are constructed *incrementally*. The idea is to extend the initial homotopic shortest path into a homotopic "sketch" that describes how the shortest path should wind through the obstacles and then to "snap" this sketch into a homotopic shortest path (see Figures 5.5a and 5.5b).

Homotopic shortest paths have increased complexity over normal shortest paths because they can loop around obstacles. For example, if the person walks in a triangular path around all the obstacles, then the leash follows a homotopic shortest path that can have $O(k)$ complexity in a single cycle around the obstacles. By repeatedly winding around the obstacles $O(N)$ times a path achieves $O(kN)$ complexity.

To avoid spending $O(kN)$ time per cell, we extend a previous homotopic shortest path into a sketch by appending a single line segment to the previous path (see Figure 5.5a). Adding this single segment can unwind at most one loop over a subset of obstacles, so only the most recent $O(k)$ vertices of the sketch will need to be updated when the sketch is snapped into the true

homotopic shortest path. A turning angle is used to identify these $O(k)$ vertices by backtracking on the sketch until the angle is at least 2π different from the final angle.

Putting all this together, a funnel for a boundary segment of a free space cell can be computed in $O(k)$ time by starting with an initial shortest path L_I of $O(kN)$ complexity, constructing a homotopic "sketch" by appending a single segment to L_I, backtracking with a turning angle to find $O(k)$ vertices that are eligible to be changed, and finally "snapping" these $O(k)$ vertices to the true homotopic shortest path using Hershberger and Snoeyink's algorithm [60]. The result is a funnel that describes one cell boundary segment (see Figures 5.5a and 5.5b).

Extending and snapping L_I is sufficient to define funnels for all four boundary segments of a free space cell $\mathcal{C}$ that is defined by $\overline{ab}$ and $\overline{cd}$. For example, extending and snapping L_I along the current $\overline{ab} \subset A$ segment defines the funnel for the left cell boundary segment. Similarly, extending and snapping L_I along the $\overline{cd} \subset B$ segment defines the funnel for the bottom cell boundary segment. Extending and snapping L_I in this fashion also conveniently defines the initial homotopic shortest paths for cells that are adjacent to $\mathcal{C}$. □

Theorem 5.10. *The δ_C decision problem can be solved in $O(kN^2)$ time and $O(k+N)$ space.*

Proof. Each cell boundary segment is associated with a funnel $\mathcal{F}_{o,\,\overline{cd}}$ with $O(kN)$ complexity [50]. However, this high complexity is a result of looping over obstacles, and most of these points do not affect the funnel's distance function $F_{o,\,\overline{cd}}$. As illustrated in Figure 5.5c, $F_{o,\,\overline{cd}}$ has only $O(k)$ complexity because only vertices $\pi(d,p) \cup \pi(p,c)$ contribute arcs to $F_{o,\,\overline{cd}}$. To solve the decision problem, compute all cell boundary funnels in $O(kN^2)$ time using Lemma 5.9 and apply the approach of Theorem 4.9 (since all funnels once again have $O(k)$ complexity). □

Theorem 5.11. *The δ_C optimization problem can be solved in $O(kN^2 + N^2 \log kN \log N)$ expected time and $O(kN^2)$ space.*[4]

[4]If space is at a premium, the algorithm can also run with $O(k+N^2)$ space and $O(kN^2 \log N + N^2 \log kN \log N)$ expected time by recomputing the funnels each time the decision problem is computed (instead of precomputing the funnels). Note that $O(N^2)$ storage is required for the red-blue intersections algorithm (cf. Theorem 4.12).

Proof. If the funnels are precomputed in $O(kN^2)$ time and space, then Theorem 4.12 applies directly. □

CHAPTER 6: LINK DISTANCE PROBLEMS IN THE PLANE

This chapter presents link distance algorithms for Voronoi diagrams, shortest path maps, the Hausdorff distance, and the Fréchet distance in the plane with polygonal obstacles. Most of the work in this chapter has been published as [35, 37] and was submitted to the journal *Computational Geometry: Theory and Applications* on January 30, 2009.

A *link path* $\pi_L(s,t)$ is a polygonal path that connects two points s,t using the minimum number of obstacle-avoiding edges. The *length* $d_L(s,t)$ of a link path is the number of edges on the path $\pi_L(s,t)$, and this length is sometimes referred to as the *link distance* between the two points. Link paths have a wealth of applications including robotic motion, wireless communications, geographic information systems, VLSI, solid modeling, image processing, and even water pipe placement. These applications benefit from link paths because turns are costly while straight distances are inexpensive. See [64] for an excellent survey of link paths.

This chapter presents *tight* bounds for link distance Voronoi diagrams in the plane. We also develop shortest path maps that support link-based queries from any source point on a *line* to destination points in either a simple polygon or a polygonal domain (a polygon with holes). These structures are more versatile than the traditional shortest path map from a fixed source *point*, and we use them to compute the link-based Fréchet distance between two polygonal curves. In addition to developing exact algorithms for Voronoi diagrams, shortest path maps, the Hausdorff distance, and the Fréchet distance, we also present many approximation algorithms that are accurate to within 1, 2, or 3 links of an optimal solution.

The link-based Fréchet distance between polygonal curves inside a simple polygon interprets the boundary of the simple polygon as an obstacle. A *free space cell* is the parameter space that represents all link distances between a pair of line segments. We prove that a free space cell has the property that all link distances less than or equal to some threshold value ε define a *free space* that need not be convex but is always x-monotone, y-monotone, and connected. We use this property plus an *additive* $O(B)$ *approximation* for the Fréchet distance that runs in $O(\frac{kN}{B} + \frac{N^2}{B^2})$

time to compute the *exact* Fréchet distance in $O(kN + N^2)$ time, where N is the complexity of the polygonal curves and k is the complexity of the simple polygon. Note that the exact runtime is asymptotically competitive with the $O(N^2 \log N)$ runtime of the traditional Fréchet distance (without obstacles) [10]. For a polygonal domain, the monotonicity and connectedness properties no longer hold, so we use our shortest path map structure to represent a free space cell.

6.1 Voronoi Diagrams

Section 2.1 contains an overview of Voronoi diagrams. Let P be a simple polygon with vertices $o_1, ..., o_k$. Let $\mathcal{S} = s_1, ..., s_N$ be a set of point or line segment sites in P. Unlike traditional Euclidean Voronoi diagrams, each face in a link distance Voronoi diagram can be associated with multiple nearest sites. We define a *site-based* Voronoi diagram $\mathcal{V}_s(\mathcal{S})$ as a partition of P such that all points within one face have the same *set* of nearest neighbor sites with respect to link distance. We define a *distance-based* Voronoi diagram $\mathcal{V}_d(\mathcal{S})$ as a partition of P such that all points within a face have the same link distance to a nearest site. A distance-based Voronoi diagram is composed of interior-disjoint layers for $i = 0...k$ such that the i-th layer is $\mathfrak{L}_i = \{t \in P \mid i = \min_{s \in \mathcal{S}} d_L(s,t)\}$. Note that all points in a layer need not share the same set of nearest neighbor sites. Candidate edges for $\mathcal{V}_s(\mathcal{S})$ and $\mathcal{V}_d(\mathcal{S})$ can be determined by precomputing Suri's [77] shortest path map SPM(s_j) for each site $s_j \in \mathcal{S}$ in $O(Nk)$ total time and space. Let ξ be the set of $O(Nk)$ candidate edges that result from computing SPM(s_1),...,SPM(s_N). Related work by Gewali et al. [54] has considered the complexity of the arrangement of visibility polygons for a set of sites but did not explore this arrangement for all link distances from 0 to k.

Theorem 6.1. *$\mathcal{V}_s(\mathcal{S})$ has $\Theta(N^2k)$ complexity and can be constructed in $O(N^2k)$ expected time and $O(N^2k \log Nk)$ deterministic time. Link distance and path queries are supported in $O(\log Nk + K)$ time, where K is the complexity of any returned path.*

Proof. Consider the arrangement of all edges in ξ. $\mathcal{V}_s(\mathcal{S})$ can have $\Omega(N^2k)$ complexity because $\Omega(k)$ pairs of adjacent vertices $o_i,\ o_{i+1} \in P$ can each define an $\Omega(N^2)$ arrangement such that

$\Omega(N)$ edges pass through o_i and $\Omega(N)$ edges pass through o_{i+1} (see Figure 6.1a). $\mathcal{V}_s(\mathcal{S})$ has

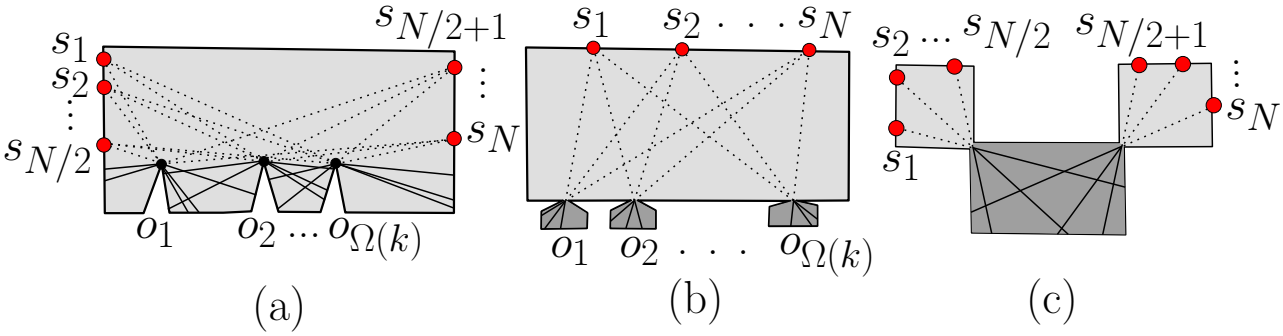


Figure 6.1: Edges in ξ are shown as solid lines. (a) $\mathcal{V}_s(\mathcal{S})$ has $\Omega(N^2k)$ complexity. $\mathcal{V}_d(\mathcal{S})$ can have (b) $\Omega(Nk)$ or (c) $\Omega(N^2)$ edges defining layer $\mathfrak{L}_1$. For both (b) and (c), layer $\mathfrak{L}_1$ consists of the lightly shaded area plus the solid visibility polygon edges, and layer $\mathfrak{L}_2$ consists of the remaining (disconnected) heavily shaded areas in P.

$O(N^2k)$ complexity because each of the $O(Nk)$ edges in ξ can intersect at most three faces in each of SPM(s_1),...,SPM(s_N) [14]. The arrangement for ξ can be constructed with an incremental algorithm [4] in $O(N^2k)$ expected time or in $O(N^2k \log Nk)$ deterministic time. Since some edges in this arrangement should not appear in $\mathcal{V}_s(\mathcal{S})$, a breadth-first postprocessing step should merge adjacent faces with link distance i that are separated by a suboptimal edge with link distance $j > i$. □

Compared to $\mathcal{V}_s(\mathcal{S})$, the *distance-based* Voronoi diagram $\mathcal{V}_d(\mathcal{S})$ has asymptotically superior complexity bounds because the points in a face of $\mathcal{V}_d(\mathcal{S})$ need not share the same set of nearest neighbor sites.

Theorem 6.2. *$\mathcal{V}_d(\mathcal{S})$ has $\Theta(N(N+k))$ complexity (for $k \geq 2$) and can be constructed in $O(N(N+k)\log N \log k)$ time. Link distance and path queries are supported in $O(\log Nk + K)$ time, where K is the complexity of any returned path.*

Proof. A distance-based Voronoi diagram $\mathcal{V}_d(\mathcal{S})$ has $\Omega(Nk)$ complexity because each of the N sites can contribute $\Omega(k)$ edges that contribute to a layer (see Figure 6.1b). $\mathcal{V}_d(\mathcal{S})$ has $\Omega(N^2)$ complexity because $\Omega(N)$ edges contributed by s_1,...,$s_{N/2}$ can each intersect $\Omega(N)$ edges contributed by $s_{N/2+1}$,...,s_N (see Figure 6.1c). We now show that $\mathcal{V}_d(\mathcal{S})$ has $O(N(N+k))$ complexity. Let

P_x^i be the set of points at link distance x from site s_i so that layer $\mathfrak{L}_x$ equals $\bigcup_{i=1}^{N} P_x^i$. Note that P_x^i cannot *enter* P_x^j in more than one connected interval because there are no vertices in the interior of a simple polygon (see Figure 6.2a). Furthermore, once the layers for a pair of sites have touched, they will always continue to be connected in the simple polygon. This means that there are $O(1)$ entrance vertices for each pair of sites. Now note that if P_x^i *exits* P_x^j in more than one interval, then a vertex of the simple polygon must separate these intervals (see Figure 6.2b). Thus, the number of exit vertices contributed by each site s_i is $O(N + k)$. Hence, the total number of exit vertices over all sites is $O(N(N + k))$, and $\mathcal{V}_d(\mathcal{S})$ has $O(N(N + k))$ complexity.

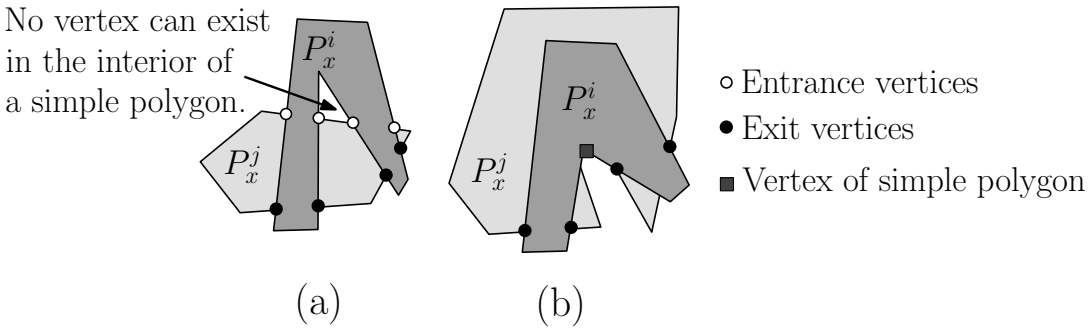


Figure 6.2: (a) At most two entrance vertices can exist inside a simple polygon for any pair of sites $s_i,\ s_j$. (b) Each pair of exit vertices can be charged to a vertex of the simple polygon.

Efrat and Har-Peled [53] show in their preliminaries how to construct layer $\mathfrak{L}_1$ using a divide-and-conquer approach. Their divide step recursively computes a layer for sites $s_1, ..., s_{N/2}$ and another layer for sites $s_{N/2+1}, ..., s_N$. At each of the $O(\log N)$ levels of the recursion, a plane sweep is used to merge layers together in $O(N(N + k) \log k)$ time. Using our complexity bounds, a straightforward extension of this approach can be used to sequentially compute all layers in $O(N(N + k) \log N \log k)$ total time and $O(N(N + k))$ space. The partition $\mathcal{V}_d(\mathcal{S})$ can then be triangulated and preprocessed for point location using standard techniques [4]. A nearest site to a query point t can be returned by locating the triangle containing t and backtracking to a nearest site. □

6.2 Shortest Path Maps

Section 2.2 contains an overview of shortest path maps. We develop link-based shortest path maps that support $O(\log k)$ time queries from any source point $s \in \overline{ab}$. Approximations accurate to within 1, 2, and 3 links are also explored.

Previous link-based work in a simple polygon with k vertices builds a shortest path map SPM(s) to support all queries from a fixed source s after $\Theta(k)$ preprocessing [77] and builds another shortest path map SPM($\mathbb{R}^2, \mathbb{R}^2$) to support queries between any two points after $\Theta(k^3)$ preprocessing [14, 51]. Our SPM($\overline{ab}$,$\mathbb{R}^2$) structure supports queries from any point $s \in \overline{ab}$ to any point $t \in \mathbb{R}^2$ after $O(k^2)$ preprocessing. Related work in a link-based polygonal domain with k vertices uses $\Theta(k^4)$ preprocessing to build SPM(s) [68]. Our SPM($\overline{ab}$,$\overline{cd}$) structure supports all link-based queries from any $s \in \overline{ab}$ to any $t \in \overline{cd}$.

We use the following terminology of [14]: the *combinatorial structure* of a shortest path map is a listing of the combinatorial structures of its edges. The combinatorial structure of a shortest path map edge $\mathcal{E}$ is a vertex-edge pair (v, e) such that $\mathcal{E}$ has one endpoint on an obstacle vertex v and has its other endpoint on an obstacle edge e. As the source point s varies along $\overline{ab}$, the position of $\mathcal{E}$'s endpoint on e is parameterized homographically by $g(s) = \frac{A+Bs}{C+Ds}$ for constants A, B, C, D (see Figure 6.3). We define an *edgelet* α as a maximal line segment such that the shortest path map for every source point $s \in \alpha$ has the same combinatorial structure.

Observe that all link distances between a pair of line segments $\overline{ab}$, $\overline{cd}$ must equal i, $i + 1$, or $i + 2$, where $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s, t)$. This is true because any link path between $s \in \overline{ab}$ and $t \in \overline{cd}$ can be extended by at most two extra links into a path that connects any $s' \in \overline{ab}$ and $t' \in \overline{cd}$. See Figure 6.3c.

Theorem 6.3. *A link-based shortest path map SPM($\overline{ab}$,$\mathbb{R}^2$) can be constructed in a simple polygon P with k vertices in $O(k^2)$ time and space. SPM($\overline{ab}$,$\mathbb{R}^2$) allows $d_L(s, t)$, $\pi_L(s, t)$ to be returned for any $s \in \overline{ab}$ and $t \in \mathbb{R}^2$ in $O(\log k + K)$ time, where K is the complexity of any returned path. These queries can also be answered to within* one *link of optimal after $O(k)$ preprocessing.*

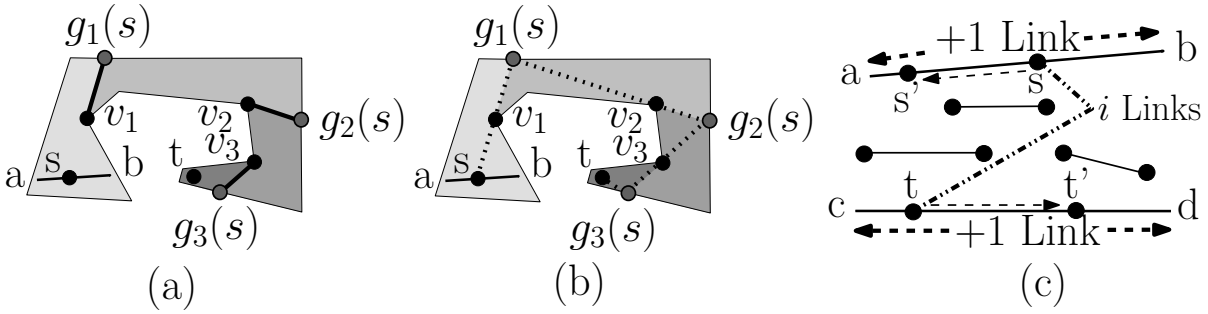


Figure 6.3: (a) Every shortest path map edge can be described as a vertex-edge pair. (b) All but the last link of $\pi_L(s,t)$ can be made to overlap a shortest path map edge. (c) All $d_L(s,t)$ for $s \in \overline{ab}$, $t \in \overline{cd}$ equal either i, $i+1$, or $i+2$, where $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$.

Proof. Partition $\overline{ab}$ into $O(k)$ edgelets by computing $\overline{ab} \cap$ SPM(o_j) [77] for each obstacle vertex $o_j \in P$. Each intersection induces at most three edgelets on $\overline{ab}$ by [14], and we construct a shortest path map SPM(α_i) [77] for each edgelet. A query $d_L(s,t)$ consists of a one-dimensional coordinate for s and a two-dimensional coordinate for $t \in P$. $s \in \overline{ab}$ lies in an edgelet α_i, and the edges of SPM(α_i) are parameterized by $s \in \alpha_i$. t is a point inside SPM(α_i). To speed up queries, we compute a non-parameterized triangulation of SPM(α_i) that permits evaluating at most two parameterized edges of SPM(α_i) at query time.

SPM(α_i) contains $O(k)$ parameterized edges that are represented as vertex-edge pairs (v,e). By [14], there are at most two parameterized shortest path map edges (v_1,e), $(v_2,e) \in$ SPM(α_i) that can intersect the interior of a fixed boundary edge $e \in P$. As s varies over α_i, the edge (v_1,e) touches a subsegment $\overline{\sigma\varsigma} \subset e$ such that the three points v_1, σ, and ς define a triangle Δ_1. Similarly, the parameterization for (v_2,e) defines a second triangle Δ_2. Triangulating $\Delta_1 \cup \Delta_2$ yields a constant number of triangles that can be associated with (v_1,e),(v_2,e). No other parameterized shortest path map edges can intersect these triangles for any $s \in \alpha_i$ because shortest path map edges in a simple polygon never cross each other.

A query $d_L(s,t)$ is handled in $O(\log k)$ time by identifying the edgelet α_i containing s, locating the triangle containing t in SPM(α_i) via point location, and evaluating the at most two parameterized edges associated with this triangle. $\pi_L(s,t)$ is calculated by following a chain of predecessors.

Approximate queries use a single shortest path map SPM($\overline{ab}$) [77] to return $\min_{s' \in \overline{ab}} d_L(s', t)$. This value always equals either $d_L(s,t)$ or $d_L(s,t) + 1$ (see Figure 6.3c). The approximate path is $s \circ \pi_L(s', t)$. □

Lemma 6.4. *In a polygonal domain, the intersection of a line segment $\overline{cd}$ with a link-based shortest path map SPM(s) of $\Theta(k^4)$ complexity [68] can be constructed without precomputing SPM(s) in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space.*

Proof. Mitchell, Rote, and Woeginger [68] represent all points with link distance j to a source by the union of $O(k^2)$ triangles. Since at most two distances define SPM(s) $\cap$ $\overline{cd}$ (see Figure 6.3c), SPM(s) $\cap$ $\overline{cd}$ is defined by the union of two sets of $O(k^2)$ triangles. Both the triangles and their intersection with $\overline{cd}$ can be computed in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space using the algorithm of [68]. □

Many problems in a polygonal domain with k vertices are difficult to compute. For example, Chiang and Mitchell [28] support Euclidean shortest path queries between any two points in a polygonal domain after $O(k^{11})$ preprocessing, and Mitchell, Rote, and Woeginger [68] construct a link-based shortest path map from a fixed source in $\Theta(k^4)$ time and space. Theorem 6.5 defines the first shortest path map in a polygonal domain that supports link distance queries from a *continuous* set of source points.

Theorem 6.5. *A link-based shortest path map SPM($\overline{ab}$,$\overline{cd}$) can be constructed in a polygonal domain in $O(k^7)$ space and either $O(k^7)$ expected time or $O(k^6\lambda_6(k))$ deterministic time. Using this structure, $d_L(s,t)$, $\pi_L(s,t)$ can be returned for any $s \in \overline{ab}$, $t \in \overline{cd}$ in $O(\log k + K)$ time, where K is the complexity of any returned path.* Approximate $O(\log k + K)$ *time queries are also supported for $s \in \overline{ab}$, $t \in \mathbb{R}^2$ to within* one *link of optimal after $O(k^4)$ preprocessing and to within* two *links of optimal after $O(k^{\frac{7}{3}}\log^{3.11} k)$ time and $O(k)$ space preprocessing.*

Proof. A set of edgelets on $\overline{ab}$ can be defined (as in the simple polygon case) by computing a shortest path map for each obstacle vertex and intersecting the edges in these structures with $\overline{ab}$.

Although each of the shortest path map structures for the $O(k)$ obstacle vertices has $\Theta(k^4)$ complexity [68, 78], Lemma 6.4 ensures that there are only $O(k \cdot k^2)$ edgelets on $\overline{ab}$ and that for each edgelet $\alpha_i \in \overline{ab}$, SPM($\alpha_i$) $\cap$ $\overline{cd}$ can be computed in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space.

We construct SPM($\overline{ab}$,$\overline{cd}$) as a partition of the parameter space defined by $\overline{ab}$ and $\overline{cd}$. For each edgelet α_i, a shortest path map edge $\mathcal{E} \in$ SPM(α_i) is described by a vertex-edge pair (v, e) and can be homographically parameterized by $s \in \alpha_i$ such that $\mathcal{E} \cap \overline{cd}$ defines a constant complexity algebraic curve. Constructing such a curve for each choice of v and e yields $O(k^2)$ curves whose arrangement can be constructed in $O(k^4)$ space and $O(k^4)$ expected time or $O(k^3\lambda_6(k))$ deterministic time [4]. Since there are $O(k^3)$ edgelets, SPM($\overline{ab}$,$\overline{cd}$) has $O(k^3 \cdot k^4)$ complexity.

The key idea for the approximate queries from $s \in \overline{ab}$ to $t \in \mathbb{R}^2$ is that $\min_{s' \in \overline{ab}} d_L(s', t)$ will always be within one link of $d_L(s, t)$ because s and s' are collinear. Thus, queries to within *one* link of optimal are supported by precomputing SPM($\overline{ab}$) in $O(k^4)$ time and space [68] and using this structure to return $\min_{s' \in \overline{ab}} d_L(s', t)$. Queries to within *two* links of optimal precompute an implicit version of SPM($\overline{ab}$) in $O(k^{\frac{7}{3}} \log^{3.11} k)$ time and $O(k)$ space [68] and use this structure to return the value of $\min_{s' \in \overline{ab}} d_L(s', t)$ to within one link of optimal. □

6.3 Hausdorff Distance

Section 2.3 contains an overview of the Hausdorff distance. The following observation will be useful to speed up Hausdorff and Fréchet distance problems. By connecting a set of input sites by a polygonal curve, point location need no longer be performed for every site. Instead, a single point location can be used to trace a polygonal curve through the plane, and this improves our runtimes by a log factor.

Observation 6.6. *Any set $\mathcal{S}$ of N points can be connected by a polygonal path with $\Theta(k+N)$ links that stays inside a simple polygon with k vertices. This path can be computed in $O(k + N \log k)$ time and $O(k + N)$ space.*

Proof. The optimal number of links for any polygonal path connecting $O(N)$ arbitrary points in

a simple polygon is $\Theta(k + N)$ because (1) a path connecting just two points in a simple polygon can have length $\Theta(k)$ and (2) a path connecting N points must in general have $\Theta(N)$ links. To construct the path, triangulate the simple polygon and preprocess it for point location in $O(k)$ time [25, 74]. For each point $p \in \mathcal{S}$, use point location to locate the triangle containing p, and associate p with one of the three vertices of its triangle. This takes $O(N \log k)$ total time. To construct a polygonal path that visits each $p \in \mathcal{S}$, visit the vertices of the simple polygon in clockwise order. At each vertex v, connect v to the previous vertex on the simple polygon and connect v to each $p \in \mathcal{S}$ that is associated with v. □

The *Hausdorff distance* is a similarity metric commonly used to compare two sets of objects (see Chapter 2). Although the Hausdorff distance is normally computed using a Voronoi diagram, our site-based and distance-based Voronoi diagrams have $\Theta(N^2k)$ and $\Theta(N(N + k))$ complexity and take $O(N^2k \log Nk)$ and $O(N(N + k) \log N \log k)$ time to compute in a simple polygon (see Theorems 6.1 and 6.2). These Voronoi diagrams encode more information than is necessary to compute the Hausdorff distance, and we achieve better bounds for our problems with shortest path map techniques.

Theorem 6.7. *The link-based Hausdorff distance between point sets can be computed in* $O(kN + N^2)$ *time and* $O(k + N)$ *space in a simple polygon and in* $O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2k)$ *time and* $O(k + N)$ *space in a polygonal domain.*[1]

Proof. For the simple polygon case, create a fixed-source shortest path map SPM(b) [77] for each point $b \in B$ and preprocess the point set A into an ordered polygonal path of length $O(k + N)$ using Observation 6.6. Tracing the polygonal path for A through SPM(b) for each $b \in B$ takes $O(N(k + N))$ total time [14] and reveals the nearest neighbor link distance from each $a \in A$ to any $b \in B$. The largest of these distances over all $a \in A$ is the Hausdorff distance. The Hausdorff distance can be computed similarly for a polygonal domain using the SPM(b) structure of [68]. □

[1]The runtime component $O(N^2k)$ can be reduced to $O(N^2 \log k)$ if one is willing to work with approximate distances that differ by at most one link from the true link distance [68].

Let $f_{\overline{ab}}(s) = \min_{t \in B} d_L(s,t)$ represent the nearest neighbor link distance from every $s \in \overline{ab}$ to B.

Theorem 6.8. *The link-based Hausdorff distance between line segment sets A and B in a simple polygon can be calculated in $O(kN + N^2 \log N)$ time and $\min(O(kN), O(k + N^2))$ space.*[2]

Proof. Using Observation 6.6, compute a connected polygonal curve of length $O(k + N)$ that visits every line segment in A. Trace this polygonal curve through SPM($\overline{cd}$) [77] for every $\overline{cd} \subset B$ in $O(N(k + N))$ total time [14]. $f_{\overline{ab}}$ equals the lower envelope of the resulting $O(N)$ piecewise constant functions involving $\overline{ab} \subset A$ and can be computed via plane sweep. The Hausdorff distance is the maximum value of any $f_{\overline{ab}}$ over all $\overline{ab} \subset A$. The space bound follows by either storing all shortest path maps at once or storing all piecewise constant functions at once. □

Theorem 6.9. *The link-based Hausdorff distance $\delta_H(A,B)$ between line segment sets A and B in a polygonal domain can be calculated* exactly *in $O(N^2k^2(\alpha(k) \log^2 k + \log Nk))$ time and $O(Nk^2)$ space. Approximations to within 1, 2, and 3 links of optimal can be computed in $O(N^2k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space, $O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2k)$ time and $O(k + N)$ space, and $O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2 \log k)$ time and $O(k + N)$ space.*

Proof. $f_{\overline{ab}}$ is the lower envelope of $O(N)$ functions that can be constructed by iteratively applying Lemma 6.4 to compute SPM($\overline{cd}$) $\cap$ $\overline{ab}$. The lower envelope for $f_{\overline{ab}}$ involves $O(Nk^2)$ piecewise constant functions and can be computed via plane sweep. Only one lower envelope is maintained at a time, so $O(Nk^2)$ space is sufficient.

We approximate SPM($\overline{cd}$) $\cap$ $\overline{ab}$ as follows. Let $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$. All link distances $d_L(s,t)$ for $s \in \overline{ab}$, $t \in \overline{cd}$ equal i, $i + 1$, or $i + 2$ (see Figure 6.3c). By Lemma 6.4, the value of $i + 1$ can be calculated for each of the $O(N^2)$ pairs of line segments in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space. An approximation to within two links of $\delta_H(A,B)$ can use any distance between $\overline{ab}$ and $\overline{cd}$ to approximate i (because this distance must equal i, $i + 1$, or $i + 2$). The approximation to

[2] Surprisingly, this runtime is a log factor slower than the corresponding Fréchet distance runtime in Theorem 6.12. Although the free space diagram can be used to compute the Hausdorff distance by projecting free space onto an axis, this approach does not improve the runtime for the Hausdorff distance.

within three links of $\delta_H(A, B)$ computes a distance between $\overline{ab}$ and $\overline{cd}$ *approximately* (to within one link of optimal [68]). □

6.4 Fréchet Distance in a Simple Polygon

Section 2.4 contains an overview of the Fréchet distance. This chapter shows how to compute the link-based Fréchet distance inside a simple polygon.

Lemma 6.10. *Although the free space in a link-based cell $\mathcal{C}$ for the Fréchet distance in a simple polygon need not be convex, it is x-monotone, y-monotone, and connected.*

Proof. Figures 6.4a and 6.4b illustrate a link distance scenario where the free space in $\mathcal{C}$ is not convex. However, the free space is x-monotone and y-monotone as shown next. The distance function $f_{s,\,\overline{cd}}$ for any horizontal or vertical line segment in $\mathcal{C}$ is defined by the link distance from a fixed point s to each point on a line segment $\overline{cd}$. Let $i = \min_{t \in \overline{cd}} d_L(s, t)$. It has been shown in [14] that $f_{s,\,\overline{cd}}$ can be represented by at most three intervals on $\overline{cd}$ with respective lengths $i + 1$, i, and $i + 1$. This implies that $f_{s,\,\overline{cd}}$ is decreasing-increasing bitonic and has constant complexity. This bitonicity plus the fact that free space consists of all distances less than or equal to ε ensures that the free space in $\mathcal{C}$ is x-monotone and y-monotone.

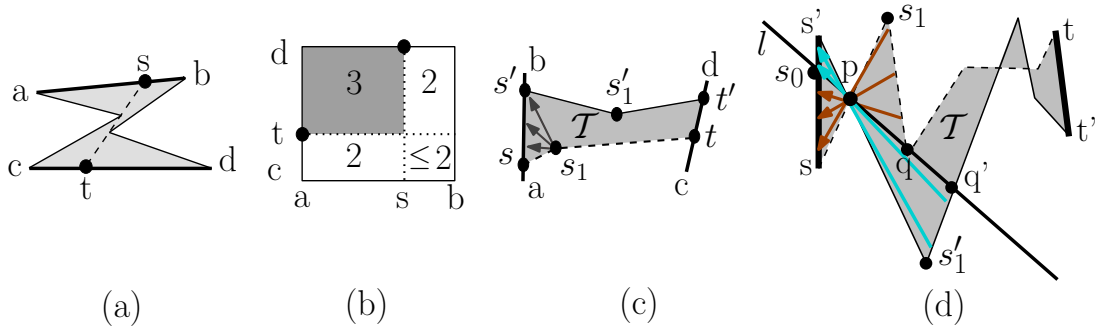


Figure 6.4: (a) Two line segments $\overline{ab}$ and $\overline{cd}$ in a simple polygon can produce (b) a cell with a *non-convex* free space. Link distances in the cell are labeled. Notice that the white free space in the cell is non-convex because the points (a, t) and (s, d) are in the white free space but there is no line of sight through the free space between these points. (c),(d) Free space for a cell in a simple polygon is connected.

To see that the free space in $\mathcal{C}$ is *connected*, pick two arbitrary free space points (s,t) and (s',t') such that $d_L(s,t),\ d_L(s',t') \leq \varepsilon$. We now show that a path through the free space always exists that connects these two points. Let $\pi_L(s,t)$ have the form $s, s_1, ..., t$ and $\pi_L(s',t')$ have the form $s', s'_1, ..., t'$ (see Figures 6.4c and 6.4d). Let $\mathcal{T}$ be the shaded region bounded by $\overline{s's}$, $\pi_L(s,t)$, $\overline{tt'}$, and $\pi_L(t',s')$. Although all link paths from $\rho \in \overline{ss'}$ to $\varrho \in \overline{tt'}$ need not lie entirely in $\mathcal{T}$, we shall see that it is possible to slide (s,t) through free space in $\mathcal{T}$ until it reaches (s',t').

If s_1 can see every point on $\overline{ss'}$, then (s,t) can be slid through free space in $\mathcal{T}$ to (s',t) (see Figure 6.4c). Otherwise, let $p = \overline{ss_1} \cap \overline{s's'_1}$ (see Figure 6.4d). Since a line l through p can always connect every point on $\overline{ss'}$ to some point on $s_1, ..., t, t', ..., s'_1$, it is always safe to slide (s,t) through free space in $\mathcal{T}$ to (s',t). We can now slide (s',t) through free space to (s',t') because both of these points lie on a vertical line in the cell, and a vertical line has a decreasing-increasing bitonic distance function. Hence, the free space in $\mathcal{C}$ is connected. □

Theorem 6.11. *After precomputing all free space on cell boundaries in $O(kN + N^2)$ time and $O(k + N^2)$ space, the* decision problem *for the link-based Fréchet distance between polygonal curves A and B in a simple polygon can be computed in $O(N^2)$ time and space.*

Proof. All free space on cell boundaries can be computed in $O(kN + N^2)$ time and $O(k + N^2)$ space by computing $O(k)$ SPM(s) structures [77] and tracing each line segment $\overline{cd} \subset B$ and $\overline{ab} \subset A$ through at most three regions [14] of SPM(s).

Although not necessarily convex, the free space in each cell is x-monotone, y-monotone, and connected by Lemma 6.10. Once the free space on the cell boundaries is known, the decision problem can be answered by propagating reachability information in constant time per cell via dynamic programming [10]. □

Theorem 6.12. *The link-based Fréchet distance in a simple polygon can be computed* exactly *in $O(kN + N^2)$ time and $O(k + N^2)$ space. It can also be computed to within $\pm\mathcal{B}$ links of optimal in $O(\frac{kN}{\mathcal{B}} + \frac{N^2}{\mathcal{B}^2})$ time and $O(k + \frac{N^2}{\mathcal{B}^2})$ space (for $\mathcal{B} \geq 2$).*

Proof. We use an *approximation* algorithm to narrow the search space to $O(1)$ candidate values

for the Fréchet distance. We then call the exact decision problem $O(1)$ times to compute the *exact* Fréchet distance.

To approximate the Fréchet distance to within $\pm\mathcal{B}$ links of optimal, let a *bundle* be a connected group of $O(\mathcal{B}^2)$ cells. Consider first a bundle containing only *one* cell defined by $\overline{ab},\ \overline{cd}$ such that $i = \min_{s\in\overline{ab},\ t\in\overline{cd}} d_L(s,t)$. Since all $d_L(s,t)$ for $s \in \overline{ab}$, $t \in \overline{cd}$ equal either i, $i+1$, or $i+2$ (see Figure 6.3c), any distance $\mathcal{D}$ in this bundle is within two links of all distances in the bundle. Now suppose the bundle size is increased from one to five by adding to the bundle the four cells adjacent to the original cell at its left, top, right, and bottom edges (ignore diagonally adjacent cells). $\mathcal{D}$ is accurate to within three links of all distances in this new bundle. By repeatedly applying this idea, a representative distance can be made accurate to within $\mathcal{B} \geq 2$ links of all distances in a diamond-shaped bundle with $O(\mathcal{B}^2)$ cells (see Figure 6.5a).[3]

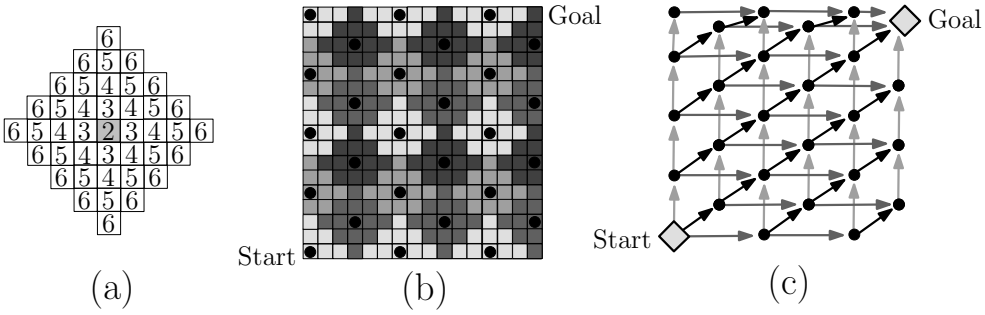


Figure 6.5: (a) A bundle of cells. Each cell is marked with its maximum error from a representative distance in the shaded node. (b) $O(\frac{N^2}{\mathcal{B}^2})$ bundles cover the free space diagram and define (c) a directed acyclic graph.

If each bundle contains $O(\mathcal{B}^2)$ cells, then $O(\frac{N^2}{\mathcal{B}^2})$ bundles are sufficient to cover the free space diagram. We organize these bundles into $O(\frac{N}{\mathcal{B}})$ columns and represent each bundle by a *node* and a representative distance (see Figure 6.5b). Although it would be easy to compute $O(\frac{N^2}{\mathcal{B}^2})$ representative distances in $O(\frac{kN}{\mathcal{B}} + \frac{N^2}{\mathcal{B}}\log k)$ time using fixed-source shortest path maps [77], we can improve this to $O(\frac{kN}{\mathcal{B}} + \frac{N^2}{\mathcal{B}})$ time as follows.

By projecting all bundle nodes onto the vertical axis of the free space diagram (see Figure 6.5b), we obtain a set of $O(\frac{N}{\mathcal{B}})$ points on the polygonal curve B. Using Observation 6.6, a polygonal path

[3] A rectangular bundle of cells can also be used but yields a slightly poorer approximation.

of length $O(k+\frac{N}{\mathcal{B}})$ can be created that visits all of these points. Such a polygonal path can be traced through a shortest path map [14] for every column of bundle nodes in $O(\frac{kN}{\mathcal{B}} + \frac{N^2}{\mathcal{B}})$ total time.

After defining nodes for the bundles and constructing edges between adjacent nodes, we have a directed acyclic graph with $O(\frac{N^2}{\mathcal{B}^2})$ nodes and edges that represents all monotone paths through the bundles (see Figure 6.5c). A breadth first search on this graph yields an approximation for the Fréchet distance to within $\pm\mathcal{B}$ links of optimal. The *exact* Fréchet distance can be computed by applying the approximation algorithm with a constant bundle size and executing the *exact* decision problem $O(1)$ times. □

The bundles technique can also be used to approximate the traditional Euclidean Fréchet distance (without obstacles) between polygonal curves A and B in $\mathbb{R}^d$. If $l_{\max}$ is the length of the longest line segment in $A \cup B$, then the result is accurate to within $\pm\mathcal{B} \cdot l_{\max}$ of the optimal distance after $O(\frac{N^2}{\mathcal{B}^2})$ time. Surprisingly, the bundles technique allows the link-based Fréchet optimization problem to be computed in the same asymptotic time and space as the decision problem. The only other known scenarios where it is possible to shave off the logarithmic Fréchet optimization factor are for the weak or discrete Fréchet distance (see Section 2.4). In these cases, the free space diagram can be treated as a planar graph [10], and any linear-time shortest path algorithm (e.g., [58]) can be applied to this planar graph to solve the optimization problem.

6.5 Fréchet Distance in a Polygonal Domain

The Fréchet distance is more difficult to compute in a polygonal domain than in a simple polygon because the free space inside a cell can be disconnected. We use the shortest path map SPM($\overline{ab}, \overline{cd}$) from Theorem 6.5 to compute the Fréchet distance.

Lemma 6.13. *The free space diagram for the link-based Fréchet distance in a polygonal domain can have* $\Omega(N^2k^4)$ *complexity.*

Proof. Figure 6.6 illustrates a single cell with $\Omega(k^4)$ complexity. In Figure 6.6a, $\overline{ab}$ and $\overline{cd}$ are enclosed in rectangles with tiny openings, and $\Omega(k^2)$ line of sight edges can be drawn between

pairs of these openings. Let the intersections of these $\Omega(k^2)$ edges with $\overline{ab}$ be $a_1, ..., a_{\Omega(k^2)}$ and the intersections with $\overline{cd}$ be $c_1, ..., c_{\Omega(k^2)}$. The points $a_1, ..., a_{\Omega(k^2)}$ and $c_1, ..., c_{\Omega(k^2)}$ define a grid of $\Omega(k^2)$ line segments in the cell such that all distances on these line segments are at most two, and all other distances are three. By choosing $\varepsilon = 2$, free space is the union of $\Omega(k^2)$ horizontal and vertical lines and has $\Omega(k^4)$ total complexity (see Figure 6.6b). Define the polygonal curves by

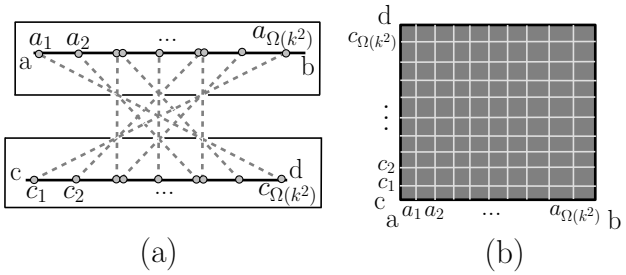


Figure 6.6: (a) Two line segments in a polygonal domain can define (b) a cell with $\Omega(k^4)$ complexity.

their endpoints as $A = \{a, b, a, b, ..., a, b\}$ and $B = \{c, d, c, d, ..., c, d\}$. Since the $\frac{N^2}{4}$ cells defined by $\overline{ab}$ and $\overline{cd}$ each have $\Omega(k^4)$ complexity, the free space diagram has $\Omega(N^2k^4)$ complexity. □

Theorem 6.14. *The link-based Fréchet distance $\delta_F(A, B)$ between polygonal curves A and B in a polygonal domain can be calculated in $O(N^2k^7 \log kN)$ time and $O(Nk^2 + k^4)$ space. Approximations are, respectively, available to within 1, 2, and 3 links of $\delta_F(A, B)$ in $O(N^2k^2\alpha(k)\log^2 k)$ time and $O(k^2+N^2)$ space, $O(Nk^{\frac{7}{3}}\log^{3.11} k+N^2k)$ time and $O(k+N^2)$ space, and $O(Nk^{\frac{7}{3}}\log^{3.11} k+ N^2 \log k)$ time and $O(k + N^2)$ space.*

Proof. The exact decision problem is computed by representing each of the $O(N^2)$ free space cells by the SPM($\overline{ab}$,$\overline{cd}$) structure of Theorem 6.5 and combining dynamic programming [10] with a plane sweep to propagate reachability information in $O(N^2k^7 \log kN)$ time. $O(Nk^2 + k^4)$ space is sufficient to store one edgelet of a cell at a time plus the cell boundaries on the two rows required for dynamic programming [10].

Any of the below approximation algorithms can be applied to quickly determine the Fréchet distance to within $O(1)$ links of the true value. The exact decision problem can then be executed $O(1)$ times in $O(N^2k^7 \log kN)$ time.

We now approximate the Fréchet distance by computing a representative link distance for each cell. Let $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$, and recall that all link distances in a cell are either i, $i+1$, or $i+2$ (see Figure 6.3c). For a given query point t, SPM($\overline{ab}$) returns $\min_{s \in \overline{ab}} d_L(s,t)$ in $O(\log k)$ time (see [68]). Hence, the *shortest* distance defined by SPM($\overline{ab}$)$\cap\overline{cd}$ equals $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$. This value can be computed in $O(k^2 \alpha(k) \log^2 k)$ total time and $O(k^2)$ space by Lemma 6.4. Repeating this process for each of the $O(N^2)$ cells takes $O(N^2 k^2 \alpha(k) \log^2 k)$ total time and $O(k^2 + N^2)$ space. Once the value of i is known for each cell, it is trivial to obtain the value $i+1$ for each cell. Since all link distances in a cell are either i, $i+1$, or $i+2$, the value of $i+1$ is within one link of all link distances in a cell. The Fréchet distance can now be computed to within one link of optimal in $O(N^2)$ time by constructing a directed acyclic graph and performing a breadth first search as in Theorem 6.12.

The approximation to within two links of optimal constructs a shortest path map SPM($\overline{ab}$) for each of the N line segments in A in $O(Nk^{\frac{7}{3}} \log^{3.11} k)$ total time and $O(Nk)$ space [68]. These shortest path maps can be used to compute one exact link distance for each of the $O(N^2)$ cells in $O(N^2 k)$ total time [68]. For any given cell, this representative distance will equal either i, $i+1$, or $i+2$ and is consequently within two links of all link distances in the cell. Alternatively, these same N shortest path maps can be used to compute a representative value that is within three links of all link distances in a cell in $O(N^2 \log k)$ time. In either case, the approximate Fréchet distance can now be computed in $O(N^2)$ time by constructing a directed acyclic graph and performing a breadth first search as in Theorem 6.12. □

CHAPTER 7: SHORTEST PATH PROBLEMS ON A POLYHEDRAL SURFACE

This chapter contains all of our Euclidean shortest path results on a polyhedral surface. Most of the work in this chapter has been published as [38, 40, 39, 41] and was submitted to the journal *Algorithmica* on April 1, 2009. It also won the *WADS 2009 Best Paper Award*. Sections 7.1 and 7.2 describe algorithms that compute shortest path edge sequences and the diameter of a convex polyhedral surface. These results improve algorithms of Agarwal et al. [1] by a linear factor. Sections 7.3, 7.4, and 7.5 contain algorithms to compute the Fréchet distance, several types of shortest path maps, and the Hausdorff distance.

Throughout this chapter, M is the total complexity of a problem space that contains a polyhedral surface and auxiliary objects on the surface such as points, line segments, and polygonal curves. A shortest path on a polyhedral surface between points s and t is denoted by $\pi(s,t)$, and $d(s,t)$ signifies the Euclidean length of $\pi(s,t)$. The notations $\pi_L(s,t)$ and $d_L(s,t)$ represent analogous concepts for link distance. A convex polyhedral surface is denoted by $\mathcal{P}$, and a non-convex polyhedral surface is signified by $\mathcal{P}_N$. The extremely slowly growing inverse Ackermann function is represented by $\alpha(M)$ (see Chapter 2). The line segment with endpoints a and b is denoted by $\overline{ab}$.

7.1 Shortest Path Edge Sequences

This chapter contains two algorithms to compute the $\Theta(M^4)$ shortest path edge sequences on a convex polyhedral surface $\mathcal{P}$. The first algorithm computes a superset of the shortest path edge sequences that contains $O(M^5)$ edge sequences, and the second algorithm computes the exact set of $\Theta(M^4)$ shortest path edge sequences. Both of these algorithms improve results of Agarwal et al. [1] by a linear factor.

Let $v_1, ..., v_M$ be the vertices of $\mathcal{P}$, and let $\Pi = \{\pi(s,v_1), ..., \pi(s,v_M)\}$ be an angularly ordered

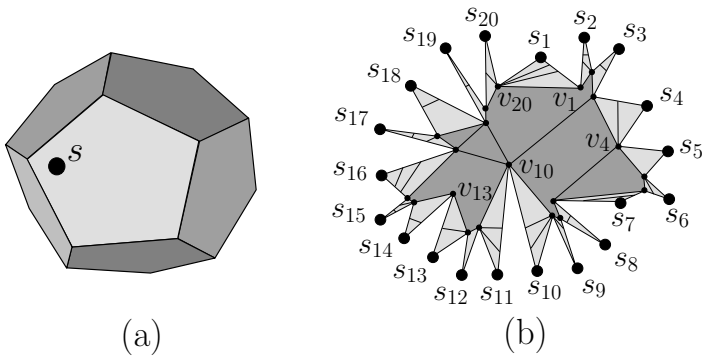


Figure 7.1: (a) A convex polyhedral surface $\mathcal{P}$. (b) The star unfolding of $\mathcal{P}$ is created by cutting along shortest paths from the source point s to every vertex $v_1, ..., v_{20}$ of $\mathcal{P}$. The source point s has an *image* s_i in the star unfolding for each vertex v_i. The heavily-shaded *core* of $\mathcal{P}$ is a simple polygon defined by the closed polygonal *equator* through the points $v_1, ..., v_{20}, v_1$ [1, 27].

set of non-crossing shortest paths from a source point $s \in \mathcal{P}$ to each vertex $v_j \in \mathcal{P}$.[1] The *star unfolding* $\mathcal{S}$ is a simple polygon [16] defined by cutting $\mathcal{P}$ along each of the shortest paths in Π and unfolding the resulting shape into the plane. Since the source point s touches all of the M cuts, $s \in \mathcal{P}$ maps to M image points $s_1, ..., s_M$ on the (two-dimensional) boundary of the unfolded simple polygon $\mathcal{S}$ (see Figure 7.1).

The *equator* [27] in the star unfolding is the closed polygonal curve through the points v_1, ..., v_M, v_1. The region inside the equator contains no source image and is called the *core* [45].[2] The regions outside the core each contain a source image and are collectively referred to as the *anti-core* [45]. A *core edge* is the image of an edge of $\mathcal{P}$ that was not cut during the unfolding process. Each of the $O(M)$ core edges has both of its endpoints at vertices and is entirely contained in the core. An *anti-core edge* is the image of a connected portion of an edge of $\mathcal{P}$ that was defined by cuts during the unfolding process. Each of the $\Theta(M^2)$ anti-core edges either has both of its endpoints on a cut or has one endpoint on a cut and the other endpoint at a vertex. The interior of an anti-core edge is entirely contained in one anti-core region (see Figure 7.1b). The dual graph of the star unfolding is a tree,[3] and this tree defines a unique edge sequence that can be used to

[1] If multiple shortest paths exist from s to v_j, then any of these shortest paths can be used to represent $\pi(s, v_j)$.

[2] The core has also been referred to as the *kernel* or the *antarctic* in [1, 27]. Note that neither the star unfolding nor its core are necessarily star-shaped [1].

[3] In [1], this dual graph is referred to as the *pasting tree*.

connect any two points in the star unfolding.

The star unfolding of s can be used to compute a shortest path $\pi(s,t)$ for points $s,t \in \mathcal{P}$ as follows. If an image of t lies in the anti-core region containing s_i, then the line segment in the star unfolding from s_i to the image of t is an optimal shortest path [1]. By contrast, if an image of t lies in the core, then a nearest source image can be determined with Voronoi diagram techniques, and the line segment in the star unfolding from this nearest source image to the image of t is an optimal shortest path [1, 27]. This means that it is easier to determine a shortest path when t maps to an anti-core region than when t maps to the core.

Agarwal et al. [1] partition the M edges of the convex polyhedral surface $\mathcal{P}$ into $O(M^3)$ line segment *edgelets*. All source points on an edgelet can be associated with the same combinatorial star unfolding, and all source points in the interior of an edgelet have a unique shortest path to each vertex of $\mathcal{P}$ [1]. These edgelets are constructed in $O(M^3 \log M)$ time by computing a shortest path between each pair of vertices on $\mathcal{P}$ and intersecting these $O(M^2)$ shortest paths with each of the M edges of $\mathcal{P}$ [1]. Agarwal et al. [1] compute a star unfolding for each edgelet and use these structures to construct an $O(M^6)$ superset of the $\Theta(M^4)$ shortest path edge sequences for $\mathcal{P}$ in $O(M^6)$ time and space [1]. In addition, Agarwal et al. [1] show how to compute the exact set of $\Theta(M^4)$ shortest path edge sequences in $O(M^6 2^{\alpha(M)} \log M)$ time.

Although we have defined the star unfolding only for a *convex* polyhedral surface $\mathcal{P}$, the concept generalizes to a *non-convex* polyhedral surface $\mathcal{P}_N$ because the star unfolding can still be defined by an angularly ordered set of non-crossing shortest path cuts from the source to every vertex [27, 67]. In addition, there are still $O(M^3)$ edgelets on $\mathcal{P}_N$ because a shortest path between each pair of vertices can intersect each edge at most once.

We show how to *maintain* a combinatorial star unfolding in $O(M^4)$ total time and space as a source point varies continuously over all $O(M^3)$ edgelets on a possibly non-convex polyhedral surface $\mathcal{P}_N$. Our approach takes advantage of small combinatorial changes between adjacent edgelets and achieves a linear factor improvement over the approach of computing a separate star unfolding for each edgelet [1].

Theorem 7.1. *A star unfolding can be maintained as a source point s varies continuously over all M edges of a (possibly non-convex) polyhedral surface $\mathcal{P}_N$ in $O(M^4)$ time and space.*

Proof. Let $\Pi = \{\pi(s, v_1), ..., \pi(s, v_M)\}$ be an angularly ordered set of non-crossing shortest path edge sequences from a source point $s \in \mathcal{P}$ to each vertex $v_j \in \mathcal{P}$. Since a star unfolding is defined combinatorially by Π, we first maintain Π as s varies continuously over all M edges of $\mathcal{P}_N$. To do this, we compute a star unfolding for each vertex $v_j \in \mathcal{P}_N$ in $O(M^3)$ total time and space [27], and we compute $O(M^3)$ edgelets in $O(M^3 \log M)$ time and space [1]. As s varies continuously over all edges, a shortest path $\pi(s, v_j) \in \Pi$ can only change combinatorially when s touches one of the $O(M^3)$ edgelet endpoints. Each edgelet endpoint is defined by an edge of a shortest path map SPM(v_j) and consequently identifies which shortest path $\pi(s, v_j)$ changes at that endpoint. The new path $\pi(s, v_j)$ can be looked up using the precomputed star unfolding for v_j and substituted for the old path in Π in $O(M)$ time. Thus, Π can be maintained over all edgelets in $O(M \cdot M^3)$ total time.

Each change to Π requires removing and adding a constant number of $O(M)$ complexity anti-core regions from the star unfolding $\mathcal{S}$ and possibly updating all $O(M)$ core edges in $\mathcal{S}$. As s varies continuously in the interior of an edgelet, each source image is parameterized along a line segment in the star unfolding, and the remaining vertices in the star unfolding are fixed [1]. See Figure 7.3a. Thus, $O(M \cdot M^3)$ time and space is sufficient to maintain $\mathcal{S}$ over all edgelets. □

The below lemma computes an implicit superset of the shortest path edge sequences on $\mathcal{P}$ in $O(M^5)$ time and space. Note that we do not attempt to compute shortest path edge sequences on a non-convex polyhedral surface $\mathcal{P}_N$ because Mount [69] has shown that there can be exponentially many shortest path edge sequences on $\mathcal{P}_N$.

Theorem 7.2. *An implicit superset of the $\Theta(M^4)$ shortest path edge sequences for a convex polyhedral surface $\mathcal{P}$ with M vertices can be computed in $O(M^5)$ time and space.*

Proof. Each edgelet defines a star unfolding with source images $s_1, ..., s_M$. For each s_i, use the relevant star unfolding's dual graph tree to construct an edge sequence from s_i to each of the

$O(M)$ anti-core edges in the anti-core region containing s_i and to each of the $O(M)$ core edges. This yields $O(M^2)$ edge sequences per edgelet, and $O(M^5)$ edge sequences over all edgelets. The result is the desired superset because only core edges have shortest path edge sequences to multiple sites, and this approach considers all possibilities. $O(M^5)$ storage is sufficient to store a dual graph tree for each edgelet, and these trees collectively encode an implicit representation of the desired superset of shortest path edge sequences [1]. □

The *exact* set of shortest path edge sequences for each combinatorial star unfolding can be determined with a kinetic Voronoi diagram that allows its defining point sites to move. In our case, the moving sites are the source images $s_1, ..., s_M$, and each source image is parameterized along a line segment as a source point varies continuously over an edgelet [1]. This behavior ensures that each *pair* of moving source images defines $O(M)$ Voronoi events [6], and Albers et al. [6] show how to maintain a kinetic Voronoi diagram in $O(\log M)$ time per event with a priority queue.

Theorem 7.3. *A kinetic Voronoi diagram of source images $s_1, ..., s_M$ can be maintained in $O(M^4 \log M)$ time and $O(M^4)$ space as a source point varies continuously over* one *edge e of a convex polyhedral surface $\mathcal{P}$.*

Proof. A kinetic Voronoi diagram for the *first* edgelet on e defines $O(M^2 \cdot M)$ events [6] due to the linear motion of $O(M^2)$ *pairs* of source images in the star unfolding. Each of the $O(M^2)$ subsequent edgelets on e can be handled by by removing and adding two source image sites. All other sites continue to be parameterized along the same line segments as in the previous edgelet. Thus, each of these $O(M^2)$ edgelets contributes $M-1$ new *pairs* of sites and $O(M \cdot M)$ new events to the priority queue. Handling each event in $O(\log M)$ time and $O(1)$ space as in [6] yields the stated runtime. □

We construct the exact set of shortest path edge sequences as follows. For the moment, fix an edgelet α and a core vertex $v_i \in \mathcal{S}$ such that v_i touches the anti-core region that contains the source image s_i. Maintaining a kinetic Voronoi diagram over all points $s \in \alpha$ yields a two-dimensional parameterized Voronoi cell φ_i for the source image s_i. Due to the properties of Voronoi diagrams,

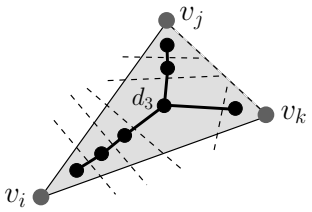


Figure 7.2: The subtree D_Δ has at most one degree three vertex d_3. Core edges are dashed.

the unique edge sequence in the star unfolding from s_i to a core edge e represents a shortest path if and only if e intersects φ_i for some $s \in \alpha$. This follows because s_i must be a nearest source image to some point on e in order to define a shortest path to e.

One way to decide whether a core edge e ever intersects a parameterized kinetic Voronoi cell φ_i is to represent each vertex of φ_i as an algebraic curve and to exhaustively test whether each algebraic curve intersects e. However, this approach ultimately requires testing whether each algebraic curve intersects each of the $O(M)$ (fixed) core edges. To improve this technique from $O(M)$ time per algebraic curve to $O(\log M)$ time per algebraic curve, Agarwal et al. [1] combine triangulation and upper envelope techniques to ensure that each algebraic curve defines at most two maximal shortest path edge sequences. A modified version of their approach is described below.

Agarwal et al. [1] triangulate the region of the core that is directly visible to core vertex v_i such that each triangle Δ has apex v_i. The dual graph D of the (fixed) core for the edgelet α is a *tree* [1] that defines candidate edge sequences. Let the portion of D inside a fixed triangle Δ be the subtree D_Δ. By [1], the maximum degree of any vertex in D_Δ is three, and there is at most one degree three vertex. This follows for two reasons (see Figure 7.2). First, a triangle Δ cannot contain a core vertex in its interior, so each core edge is a *chord* of Δ. Second, a degree three vertex must have three core edges defining its face, and this happens at most once.

Agarwal et al. [1] compute each subtree D_Δ in $O(M)$ time. In the following lemma, we improve this process to $O(\log M)$ time.

Lemma 7.4. *A subtree D_Δ can be computed in $O(\log M)$ time.*

Proof. Assume Δ has vertices v_i, v_j, v_k (see Figure 7.2). Since D_Δ has at most one degree three

vertex and the maximum degree of any vertex in D_Δ is three, D_Δ consists of one path from the face containing v_i to the face containing v_j and a second path from the face containing v_i to the face containing v_k (see Figure 7.2). Point location in the core can identify the endpoints of these two paths in the tree D in $O(\log M)$ time, and these paths define a representation of D_Δ. □

After computing the subtree D_Δ for each triangle Δ, Agarwal et al. [1] use polar coordinates centered at core vertex v_i to compute an upper envelope μ of the algebraic curves defining the kinetic Voronoi cell φ_i. This upper envelope is then refined into a set of curve segments such that each curve segment is contained in some triangle Δ. For each curve segment, a binary search is performed on the two paths in D_Δ. The deepest edge on each of these two paths that is intersected by a curve segment defines a *maximal* shortest path edge sequence. Repeating this technique for all core vertices defined by all edgelets yields $\Theta(M^3)$ *maximal* shortest path edge sequences. The set of all prefixes of these maximal sequences defines all $\Theta(M^4)$ shortest path edge sequences of $\mathcal{P}$ [1].

Theorem 7.5. *The exact set of $\Theta(M^4)$ shortest path edge sequences for a convex polyhedral surface $\mathcal{P}$ with M vertices can be explicitly constructed in $O(M^5 2^{\alpha(M)} \log M)$ time. This set can be implicitly stored in $O(M^4)$ space or explicitly stored in $O(M^5)$ space.*

Proof. Let n_i be the total number of parameterized Voronoi vertices over all edgelets, and let t_Δ be the time to process each triangle Δ. There are $O(M^5)$ possible triangles Δ because each of the $O(M^3)$ edgelets defines $O(M)$ core vertices, and each of these vertices defines $O(M)$ triangles. The technique of Agarwal et al. [1] requires $O(n_i 2^{\alpha(M)} \log M + M^5 t_\Delta)$ time. Since they assume $n_i \in O(M^6)$ and $t_\Delta \in O(M)$, this yields $O(M^6 2^{\alpha(M)} \log M)$ time.

We improve this runtime as follows. By Theorem 7.3, $n_i \in O(M^5)$ over all $O(M)$ edges of $\mathcal{P}$. By Lemma 7.4, $t_\Delta \in O(\log M)$ time. Thus, we achieve $O(M^5 2^{\alpha(M)} \log M)$ total time. The implicit space bound follows by storing the kinetic Voronoi diagram for only one edge at a time and storing each of the $\Theta(M^3)$ *maximal* shortest path edge sequences [75] in $O(M)$ space. □

7.2 Diameter

The *diameter* of a convex polyhedral surface is the largest shortest path distance between any pair of points on the surface. O'Rourke and Schevon [70] originally gave an algorithm to compute the diameter in $O(M^{14} \log M)$ time. Subsequently, Agarwal et al. [1] showed how to compute the diameter in $O(M^8 \log M)$ time. The approach of Agarwal et al. [1] computes shortest paths between all pairs of vertices, and these shortest paths induce an arrangement of $O(M^4)$ *ridge-free regions* on the surface. Each ridge-free region can be associated with one combinatorial star unfolding that defines a set of source images in the unfolded plane. Each of these source images can be linearly parameterized according to the position of a source point s in a (two-dimensional) ridge-free region. Using these linear parameterizations, Agarwal et al. [1] represent a kinetic Voronoi diagram of the source images as a lower envelope in $\mathbb{R}^9$. The upper bound theorem for convex polyhedra ensures that this kinetic Voronoi diagram has $O(M^4)$ complexity [1].

The below approach computes the diameter a linear factor faster than Agarwal et al. [1]. Instead of representing a kinetic Voronoi diagram of parameterized source images as a high-dimensional lower envelope, we maintain a kinetic Voronoi diagram over a set of collinear and co-circular Voronoi events that are defined by a set of continuously moving sites. The idea of maintaining a kinetic Voronoi diagram for a set of continuously moving points is due to Albers et al. [6]. They show that a kinetic Voronoi diagram can be maintained in $O(\log M)$ time per event.

Theorem 7.6. *The diameter of a convex polyhedral surface $\mathcal{P}$ with M vertices can be computed in $O(M^7 \log M)$ time and $O(M^4)$ space.*

Proof. To compute the diameter, we maintain a kinetic Voronoi diagram of the source images and return the largest distance ever attained between a source image site and any of its Voronoi vertices. We begin by picking an initial ridge-free region r. As mentioned above, the upper bound theorem for convex polyhedra ensures that the parameterized source images in the star unfolding for r define $O(M^4)$ Voronoi events. We now process the remaining ridge free regions in depth-first order so that the current ridge-free region r_c is always adjacent to a previously processed region

r_p. Due to the definition of ridge-free regions, the star unfolding for r_c can always be obtained from the star unfolding for r_p by removing and inserting two source image sites. This implies that the kinetic Voronoi diagram for r_c involves only $O(M^3)$ Voronoi events that were not present in r_p. This follows because each of these $O(M^3)$ events must involve at least one of the two new source image sites. These bounds imply that there are a total of $O(M^7)$ Voronoi events over all $O(M^4)$ ridge-free regions, and each of these events can be handled in $O(\log M)$ time by [6]. Each parameterized Voronoi vertex v can now be associated with a function $f(v)$ that represents the distance from v to its defining source image. The diameter is the largest distance defined by any of these functions. □

7.3 Fréchet Distance

This chapter contains algorithms to compute the Fréchet distance on both convex and non-convex polyhedral surfaces. Let $\delta_C(A, B)$ (resp. $\delta_N(A, B)$) denote the Fréchet distance between polygonal curves A and B on a convex (resp. non-convex) polyhedral surface. We define M as the number of vertices in a triangulated version of the surface such that every edge of A and B appears as a set of edges in the triangulation.

Maheshwari and Yi [65] have previously shown how to compute $\delta_C(A, B)$ in $O(M^7 \log M)$ time by enumerating all edge sequences. However, their approach relies on [62] whose key claim "has yet to be convincingly established" [1]. By contrast, we use the star unfolding from Section 7.1 to compute $\delta_C(A, B)$ in $O(M^6 \log^2 M)$ time and $O(M^2)$ space. We build a *free space diagram* to measure the distance $d(s, t)$ between all pairs of points $s \in A$ and $t \in B$. Each *cell* in our free space diagram is the parameter space defined by an edgelet $\alpha \in A$ and either a core edge or an anti-core edge in the combinatorial star unfolding for α. A cell is always interior-disjoint from all other cells.

To compute $\delta_C(A, B)$, we determine for a given constant $\varepsilon \geq 0$ all points $\{(s, t) \mid s \in A,\ t \in B,\ d(s, t) \leq \varepsilon\}$ that define the *free space* [10]. The star unfolding $\mathcal{S}$ maps a fixed source point $s \in A$ to a set $s_1, ..., s_M$ of source image points in $\mathcal{S}$ and maps the polygonal curve B to a set

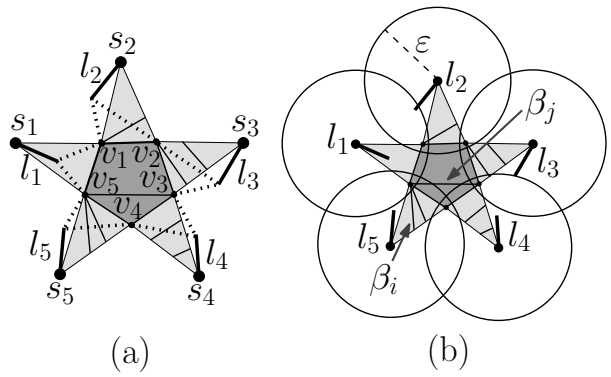


Figure 7.3: **(a)** As a source point s varies continuously along an edgelet α, the darkly-shaded core of $\mathcal{S}$ is fixed while the images of s vary along line segments $l_1, ..., l_5$ [1]. Although the lightly-shaded triangles in the anti-core vary continuously with respect to $s \in \alpha$, their movement does not change the combinatorial structure of $\mathcal{S}$. **(b)** Inside the core, there are $O(M)$ images of the edges of $\mathcal{P}$ (e.g., β_j) that touch two vertices of $\mathcal{P}$. Inside the anti-core, there are $O(M^2)$ images of the edges of $\mathcal{P}$ (e.g., β_i) that were cut during the unfolding process. Free space for edges in the anti-core is completely described by a single disk (e.g., the disk centered on l_5 is always closest to β_i). Free space for edges in the core (e.g., β_j) is defined by the union of $O(M)$ disks.

$\beta_1, ..., \beta_{O(M^2)}$ of core and anti-core edges in $\mathcal{S}$. Since s maps to multiple images in $\mathcal{S}$, free space is defined by the union of a set of disks $d_1, ..., d_M$, where each disk d_i has radius ε and is centered at s_i (see Figure 7.3). This follows by [16, 24] because all L_2 distances in the star unfolding for a *convex* polyhedral surface are at least as large as the shortest path between those two points (even when the L_2 path does not stay inside the boundary of the star unfolding). As the source point s varies continuously over an edgelet $\alpha \in A$, the core is fixed and each s_i is parameterized along a line segment l_i in the star unfolding [1]. This is illustrated in Figure 7.3a. The below $\delta_C(A, B)$ *decision problem* decides whether the Fréchet distance between polygonal curves A and B on a convex polyhedral surface is at most some given constant $\varepsilon \geq 0$.

Theorem 7.7. *The $\delta_C(A, B)$ decision problem can be solved in $O(M^6 \log M)$ time and $O(M^2)$ space.*

Proof. Partition the polygonal curve A into $O(M^3)$ edgelets and maintain a star unfolding for these edgelets in $O(M^4)$ total time by Theorem 7.1. Let β be an *anti-core* edge that lies in the anti-core region containing the source image s_i. The anti-core edge β changes length inside its

triangular anti-core region as s varies continuously over an edgelet $\alpha \in A$ (see Figure 7.3a). In the free space diagram, this motion defines a constant complexity algebraic cell C that represents the parameter space for the edgelet $\alpha \in A$ and the anti-core edge $\beta \in B$. *Free space* in the cell C is defined by the intersection of C with the ellipse $E_{l_i,\beta} = \{(s,t) \mid s \in l_i,\ t \in \beta,\ \|s-t\| \leq \varepsilon\}$ [10]. This follows because the anti-core edge β is always at least as close to the source image s_i as to any other source image s_j for $j \neq i$ [24]. Since the free space defined by each anti-core edge has constant complexity and is interior-disjoint from all other cells, the $O(M^5)$ anti-core edges contribute $O(M^5)$ total complexity to the free space diagram.

The $O(M)$ *core edges* have fixed positions as s varies continuously over an edgelet α, and the free space for any core edge γ is defined by the union of the M ellipses $E_{l_1,\gamma}, ..., E_{l_M,\gamma}$ (see Figure 7.3b). The union of these ellipses has $O(M^2)$ complexity and can be computed for each core edge in $O(M^2 2^{\alpha(M)})$ time [4]. Thus, the free space defined by all $O(M)$ core edges has $O(M^3)$ total complexity per edgelet and $O(M^6)$ complexity over all edgelets. Reachability information can be propagated through the free space diagram via plane sweep in $O(M^6 \log M)$ time. By storing one star unfolding, one cell, and one vertical line segment of the free space diagram at a time, $O(M^2)$ space is sufficient.[4] The decision problem returns true if and only if the upper right corner of the free space diagram is reachable. □

For a *non-convex* polyhedral surface, the star unfolding can overlap itself [27], and shortest paths can turn at vertices in the star unfolding [67]. This behavior implies that vertices can have multiple images in the star unfolding, and this can cause shortest paths in the star unfolding to be disconnected. For example, the shortest path $\pi(s,p_1)$ in Figure 7.4 is defined by the line segments $\overline{s_1 b_2}$, $\overline{b_2 b_3}$, $\overline{b_3'' b_4}$, $\overline{b_4 b_5}$, and $\overline{b_5' p_1}$. Rather than computing potentially disconnected paths directly, Chen and Han [27] assign a weight $d(s,v_j)$ to each star unfolding vertex v_j and build an additively weighted Voronoi diagram. Their technique returns $\pi(s,p_1)$ as a minimum length path of the form

[4]One vertical line segment of the free space diagram is defined by all distances between a fixed point $s \in A$ and the polygonal curve B. The free space defined by s and the polygonal curve B has $O(M^2)$ complexity because each of the $O(M^2)$ anti-core edges is intersected with a single fixed ellipse and each of the $O(M)$ core edges is intersected with $O(M)$ fixed ellipses.

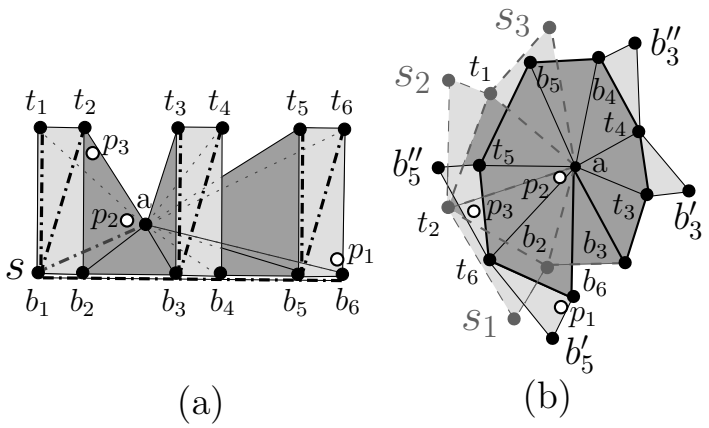


Figure 7.4: (a) A non-convex polyhedral surface can define (b) a star unfolding with an unfolded core that overlaps itself. Vertices on the boundary of the core are shown in the sequence a, b_6, t_6, t_5, b_5, b_4, t_4, t_3, b_3, b_2, t_2, t_1. The core begins to overlap itself in this ordering after vertex b_3, and these edges are drawn dashed. The three unfolded images of s are labeled s_1, s_2, and s_3.

$\pi(s, v_j) \circ \overline{v_j p_1}$, where $\circ$ denotes concatenation and v_j and p_1 both lie in a common face of the star unfolding.

Although it was previously known [27] that anti-core regions for a non-convex polyhedral surface could overlap other anti-core regions, it was not previously known whether the core could overlap itself. The below lemma affirms that the core can indeed overlap itself.

Lemma 7.8. *The core of the star unfolding for a non-convex polyhedral surface can overlap itself.*

Proof. Figure 7.4 illustrates a non-convex polyhedral surface whose star unfolding has an overlapping core. The three-dimensional surface consists of three rectangular faces and eleven triangular faces. The three rectangular faces all lie in a common plane and are defined by the set of vertices $V = \{b_1, ..., b_6, t_1, ..., t_6\}$. The eleven triangular faces are all defined by a common apex vertex a and two vertices from V. The eleven angles incident to a are $\angle b_1 a t_1$, $\angle t_1 a t_2$, $\angle t_2 a b_2$, $\angle b_2 a b_3$, $\angle b_3 a t_3$, $\angle t_3 a t_4$, $\angle t_4 a b_4$, $\angle b_4 a b_5$, $\angle b_5 a t_5$, $\angle t_5 a t_6$, and $\angle t_6 a b_6$. The sum of these angles can be greater than 2π by making the three rectangular faces sufficiently tall (because this causes the six angles $\angle b_i a t_i$ for $i = 1, ..., 6$ to exceed $\frac{\pi}{3}$). Shortest paths from the source point $s = b_1$ to each vertex are shown as thick dash-dotted line segments. Note that the shortest path $\pi(s, a)$ is the line segment from s to a that overlaps the edge $\overline{b_1 a}$. Cutting along these shortest paths and unfolding

into the plane yields the star unfolding for this surface. The shortest paths $\pi(s, t_4)$ and $\pi(s, t_6)$ turn at a vertex because all vertices in V lie in the same plane, and the $\frac{\pi}{2}$ angles defining the rectangular faces prevent shortcuts through the interiors of "alley" faces such as b_2ab_3.

The rectangular faces and the hidden face b_1at_1 in Figure 7.4a map to anti-core regions in the star unfolding because each of these faces can be associated with one maximal shortest path edge sequence. We now justify why the remaining heavily-shaded points in Figure 7.4a map to the core in the star unfolding. Consider the points p_2, p_3 in the interior of the face t_2ab_2 in Figure 7.4a. The shortest path $\pi(s, p_2)$ travels from s to a to p_2 while $\pi(s, p_3)$ travels from s to edge $\overline{t_2b_2}$ to p_3. Since multiple shortest path edge sequences exist to interior points of the face t_2ab_2, this face must lie in the core. The remaining heavily-shaded faces lie in the core by analogous reasoning. Figure 7.4b illustrates that faces in the core such as t_6ab_6 and t_2ab_2 partially overlap when the core is unfolded into the plane. This follows because the sum of the eleven angles incident to a is greater than 2π. Thus, the core of the star unfolding for a non-convex polyhedral surface can overlap itself. □

Even though the star unfolding of a non-convex polyhedral surface can overlap itself, the star unfolding can still be defined by a tree of angularly ordered shortest path cuts from the source to every vertex [27, 67]. Furthermore, a core can still be defined by a polygonal equator with $O(M)$ complexity that connects adjacent leaves in the tree.

Since a shortest path can only turn at vertices in the star unfolding [67], an anti-core region now has an *hourglass* shape [55, 57] (cf. Section 4.1). This follows because an anti-core region is bounded by a (possibly parameterized) vertex, a line segment, and two shortest paths in the unfolded plane. Such hourglasses encode all shortest paths from a source to any point in its anti-core region. The free space defined by an hourglass is a connected, semi-algebraic shape with $O(M)$ complexity (cf. Section 4.1). The below $\delta_N(A, B)$ *decision problem* decides whether the Fréchet distance between polygonal curves A and B on a non-convex polyhedral surface is at most some given constant $\varepsilon \geq 0$.

Theorem 7.9. *The $\delta_N(A, B)$ decision problem can be solved in $O(M^7 \log M)$ time and $O(M^3)$*

space.

Proof. Partition the polygonal curve A into $O(M^3)$ edgelets such that all points on an edgelet can be associated with the same combinatorial star unfolding. Maintain a star unfolding for these edgelets in $O(M^4)$ total time by Theorem 7.1. Let C be the parameter space for an edgelet and either a core edge or an anti-core edge. Free space for an *anti-core edge* is the intersection of C with the $O(M)$ complexity free space for *one* hourglass. Free space for a *core edge* is the intersection of C with the union of the free spaces for $O(M)$ hourglasses. This union has $O(M^3)$ complexity because the free space for any pair of hourglasses has $O(M)$ complexity. Since each core edge is a chord of the core, the dual graph of the core is a tree. Consequently, the $O(M)$ hourglasses for a core edge γ can be defined by iteratively extending an hourglass from every vertex in the star unfolding [27] through the dual graph of the core to γ.

In total, the free space for each edgelet has $O(M^4)$ complexity because it involves $O(M^2)$ anti-core edges and $O(M)$ core edges, and this free space can be computed in $O(M^4 \log M)$ time [4]. Thus, each of the $O(M^3)$ edgelets contributes $O(M^4)$ complexity to the free space diagram. A plane sweep can be used to answer the decision problem in $O(M^7 \log M)$ time. By storing one star unfolding, one cell, and one vertical line segment of the free space diagram at a time, $O(M^3)$ space is sufficient. □

Critical values are candidate values for the Fréchet distance (see Sections 2.4 and 2.5). In the following theorem, we apply parametric search to a set of parameterized free space vertices.

Theorem 7.10. *The Fréchet distance can be computed on a convex polyhedral surface in* $O(M^6 \log^2 M)$ *time and* $O(M^2)$ *space and on a non-convex polyhedral surface in* $O(M^7 \log^2 M)$ *time and* $O(M^3)$ *space, where* M *is the total complexity of the surface and the polygonal curves* A, B*. Furthermore, the free space diagram for a non-convex polyhedral surface can have* $\Omega(M^4)$ *complexity.*

Proof. Represent each of the $O(M^6)$ (resp. $O(M^7)$) free space vertices from Theorems 7.7 and 7.9 as a (possibly partially defined) algebraic curve $\rho_i(\varepsilon)$ that has constant degree and description

complexity.

Given the $O(M^6 \log M)$ (resp. $O(M^7 \log M)$) decision problem runtimes from Theorems 7.7 and 7.9, parametric search with Cole's [29] sorting trick can be applied to the $\rho_i(\varepsilon)$ functions to compute the Fréchet optimization problem in $O(M^6 \log^2 M)$ (resp. $O(M^7 \log^2 M)$) time. The space requirements are identical to the decision problems. The free space diagram for a non-convex polyhedral surface can have $\Omega(M^4)$ complexity by Lemma 5.4. □

7.4 Shortest Path Maps

This chapter presents shortest path maps on convex and non-convex polyhedral surfaces. These structures support queries from any point on an arbitrary source line segment $\overline{ab}$ that lies on the surface. Throughout this chapter, M denotes the complexity of either a convex or non-convex polyhedral surface, and K is the complexity of any returned path.

Theorem 7.11. *A shortest path map SPM($\overline{ab}$,$\mathcal{P}$) can be built for a* convex *polyhedral surface $\mathcal{P}$ in $O(M^4 \log M)$ time and $O(M^4)$ space. For all points $s \in \overline{ab} \subset \mathcal{P}$ and $t \in \mathcal{P}$, SPM($\overline{ab}$,$\mathcal{P}$) can return $d(s,t)$ in $O(\log^2 M)$ time and $\pi(s,t)$ in $O(\log^2 M + K)$ time.*[5]

Proof. The line segment $\overline{ab}$ can be partitioned into $O(M^2)$ edgelets, and a kinetic Voronoi diagram can be maintained for these edgelets in $O(M^4 \log M)$ total time and $O(M^4)$ space by Theorem 7.3. Point location queries in this kinetic Voronoi diagram take $O(\log^2 M)$ time by [46]. □

Our next theorem uses the star unfolding [1] and the hourglass structure of [55] to encode all shortest paths between two line segments. Such an hourglass defines a semi-algebraic free space that has $O(M)$ complexity (cf. Section 4.1).

Theorem 7.12. *A shortest path map SPM($\overline{ab}$,$\mathcal{P}_N$) can be built for a* non-convex *polyhedral surface $\mathcal{P}_N$ in $O(M^{9+\epsilon})$ time and $O(M^9)$ space for any constant $\epsilon > 0$. For all points $s \in \overline{ab} \subset \mathcal{P}_N$ and $t \in \mathcal{P}_N$, SPM($\overline{ab}$,$\mathcal{P}_N$) can return $d(s,t)$ in $O(\log M)$ time and $\pi(s,t)$ in $O(\log M + K)$ time.*

[5] $O(\log M)$ time queries are also possible by [46] but at the cost of essentially squaring both the time and space preprocessing bounds.

Proof. Let α represent one of the $O(M^2)$ edgelets on $\overline{ab}$ (see Section 7.1). A shortest path between a point $s \in \alpha$ and any fixed point in the *anti-core* can be resolved using one hourglass (cf. Section 7.3). Distance and shortest path queries in this hourglass can be resolved in the desired query times by [57, 55]. By contrast, all shortest paths between $s \in \alpha$ and a fixed point in a face of the *core* are defined by $O(M)$ hourglasses (see Section 7.3). To support logarithmic query time for all points in a fixed face of the core, we can form $O(M^3)$ constant complexity distance functions from these hourglasses and compute their lower envelope and a vertical decomposition structure in $O(M^{6+\epsilon})$ time and $O(M^6)$ space, for any constant $\epsilon > 0$ [4]. Repeating this procedure for all $O(M)$ faces in the core yields $O(M^{7+\epsilon})$ time per edgelet and $O(M^{9+\epsilon})$ time over all $O(M^2)$ edgelets. □

7.5 Hausdorff Distance

The Hausdorff distance is a similarity metric that was defined in Chapter 2.

Theorem 7.13. *The Hausdorff distance $\delta_H(A, B)$ can be computed on a* convex *polyhedral surface in $O(M \log M)$ time and space between two sets of points. $\delta_H(A, B)$ can also be computed on a possibly non-convex polyhedral surface in $O(M^2 \log M)$ time and $O(M^2)$ space between two sets of points or between two sets of line segments. The variable M represents the complexity of the surface and the two sets A and B.*

Proof. For a *convex* polyhedral surface, the algorithm of [76] can be used to compute an *implicit* representation of a Voronoi diagram for the point set B in $O(M \log M)$ time and space. For the remaining problems, the continuous Dijkstra technique of [67] can be used to generate an explicit Voronoi diagram on a possibly non-convex polyhedral surface in $O(M^2 \log M)$ time and $O(M^2)$ space.

Once a Voronoi diagram has been constructed, the Hausdorff distance can be computed for point sets by finding the nearest neighbor distance $\min_{b \in B} d(a, b)$ for each $a \in A$ in $O(\log M)$ time. For line segment sets, an explicit Voronoi diagram can be used to resolve candidate values for the Hausdorff distance using an $O(M^2 \log M)$ time plane sweep algorithm described in [9]. □

CHAPTER 8: LINK DISTANCE PROBLEMS ON A POLYHEDRAL SURFACE

This chapter presents algorithms for many link distance problems on a polyhedral surface. Most of the work in this chapter has been published as [38, 40, 39, 41] and was submitted to the journal *Algorithmica* on April 1, 2009.

A *link path* $\pi_L(s,t)$ is a polygonal path that connects two points s,t using the minimum number of obstacle-avoiding edges (see Chapter 2). Surprisingly, we appear to be the first to consider link distance problems on a polyhedral surface. We define $\mathcal{C}$ as a special type of polyhedral surface such that every face of $\mathcal{C}$ is convex and no two adjacent faces are coplanar. We define $\mathcal{N}$ as a special type of polyhedral surface such that no two adjacent faces are coplanar. $\mathcal{N}$ can be created from any polyhedral surface by triangulating the surface and merging adjacent triangles that are coplanar. The essential property of both $\mathcal{C}$ and $\mathcal{N}$ is that a path must turn whenever it enters a new face. The difference between $\mathcal{C}$ and $\mathcal{N}$ is that each face of $\mathcal{C}$ is convex, whereas a planar face of $\mathcal{N}$ need not be convex and can contain holes.

Section 8.1 shows how to compute shortest path maps, Voronoi diagrams, the diameter, and the Hausdorff distance for $\mathcal{C}$. Sections 8.2 and 8.3 extend these algorithms to $\mathcal{N}$. Section 8.4 presents an algorithm to compute the Fréchet distance for either $\mathcal{C}$ or $\mathcal{N}$.

Throughout this chapter, we assume that s,t are points and $\overline{ab}, \overline{cd}$ are line segments on either $\mathcal{C}$ or $\mathcal{N}$. The link distance between s and t is denoted by $d_L(s,t)$, and $\pi_L(s,t)$ denotes a link path between s and t. We define M as the complexity of $\mathcal{C}$ and $\mathcal{N}$.

8.1 Link Distance Problems on $\mathcal{C}$

This chapter describes algorithms to compute link-based shortest path maps on $\mathcal{C}$ and $\mathcal{N}$, and these structures are used to compute Voronoi diagrams, the diameter, and the Hausdorff distance. Unless otherwise stated, all of the shortest path maps and Voronoi diagrams in this chapter support link-based queries in $O(\log M + K)$ time, where K is the complexity of any returned path.

Since each face of $\mathcal{C}$ is convex, optimal link-based paths only turn in $\mathcal{C}$ when they enter a new face. Hence, a breadth-first search is sufficient to calculate the link distance from each face to its nearest site. Such a breadth-first search can be applied to compute a shortest path map SPM($\overline{ab}$, $\mathcal{C}$) and a link-based Voronoi diagram of a set of interior-disjoint sites on $\mathcal{C}$ in linear time. The link-based Voronoi diagram can also be used [9] to compute the Hausdorff distance on $\mathcal{C}$. The link diameter of $\mathcal{C}$ is the largest link distance between any pair of points on $\mathcal{C}$ and can be computed in quadratic time by running a breadth-first search from each face. The below Theorems summarize these results.

Theorem 8.1. *A link-based shortest path map SPM($\overline{ab}$, $\mathcal{C}$) can be constructed on $\mathcal{C}$ in $\Theta(M)$ time and space. This structure supports $d_L(s,t)$, $\pi_L(s,t)$ queries from any $s \in \overline{ab}$ to any $t \in \mathcal{C}$.*

Theorem 8.2. *A link-based Voronoi diagram for M point or line segment sites can be computed on $\mathcal{C}$ in $\Theta(M)$ time and space as long as the line segment sites intersect $O(M)$ times.*[1]

Theorem 8.3. *The link-based Hausdorff distance of M point or line segment sites can be computed on $\mathcal{C}$ in $O(M \log M)$ time and $O(M)$ space as long as the line segment sites intersect $O(M)$ times.*

Theorem 8.4. *The link diameter of $\mathcal{C}$ can be computed in $O(M^2)$ time and $O(M)$ space.*

8.2 Shortest Path Maps and Diameter on $\mathcal{N}$

Each face of $\mathcal{N}$ is a *polygonal domain* (a two-dimensional polygon with polygonal holes). Mitchell, Rote, and Woeginger [68] compute a shortest path map in a polygonal domain by iteratively constructing the set of points at link distance $1, 2, ..., M$ from the source. This approach can be modified to work on the surface $\mathcal{N}$ by propagating link distance edges into adjacent faces at each step. An explicit arrangement of these edges requires $\Theta(M^4)$ [68, 78] time and space, and such an arrangement can be used to compute shortest path maps SPM(s, $\mathcal{N}$) and SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) that support logarithmic time queries. Alternatively, a partial arrangement [68] of shortest path map

[1] This runtime requires that each site has previously been associated with the face that contains it. When this is not the case, either point location techniques [43] or a brute force approach may be used to associate each site with the face that contains it.

edges that supports linear time queries can be constructed in $O(M^{\frac{7}{3}} \log^{3.11} M)$ time and $O(M)$ space. Calculating a linear number of these structures is sufficient to compute all pairwise distances needed for the Hausdorff distance of points on $\mathcal{N}$. The Hausdorff distance of line segments on $\mathcal{N}$ can be computed by intersecting $O(M^2)$ shortest path maps with a line segment in $O(M^2 \cdot M^2 \alpha(M) \log^2 M)$ total time and applying lower envelope and plane sweep techniques to piecewise constant functions. The link-based Voronoi diagram on $\mathcal{N}$ can be constructed by computing a quadratic number of shortest path map edges for each site, computing an arrangement of these $O(M^3)$ line segments, and using a postprocessing scheme to remove suboptimal edges.

By combining Mitchell, Rote, and Woeginger's [68] technique with an edgelet-based approach (cf. Section 7.1), we can additionally compute a link-based shortest path map SPM($\overline{ab}, \overline{cd}$) and the link diameter of $\mathcal{N}$. The shortest path map SPM($\overline{ab}, \overline{cd}$) is constructed by partitioning $\overline{ab}$ into $O(M^3)$ edgelets and computing an arrangement of $O(M^2)$ parameterized shortest path map edges for each edgelet in $O(M^6 \lambda_6(M))$ total time. To calculate the link diameter, we compute a partial arrangement of shortest path map edges for $O(M^4)$ edgelets in $O(M^4 \cdot M^{\frac{7}{3}} \log^{3.11} M)$ total time.

Theorem 8.5. *Link-based shortest path maps SPM(s, $\mathcal{N}$) and SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) can be constructed on $\mathcal{N}$ in $\Theta(M^4)$ time and space. SPM(s, $\mathcal{N}$) supports queries from a fixed source point s to any point $t \in \mathcal{N}$. SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) supports queries $\min_{s \in \overline{ab}} d_L(s,t)$ and $\min_{s \in \overline{ab}} \pi_L(s,t)$ from a fixed line segment $\overline{ab} \subset \mathcal{N}$ to any point $t \in \mathcal{N}$.* Implicit *representations of SPM(s, $\mathcal{N}$) and SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) can be built in $O(M^{\frac{7}{3}} \log^{3.11} M)$ time and $O(M)$ space to support exact queries in $O(M)$ time and queries accurate to within one link in $O(\log M + K)$ time.*

Proof. Each face of $\mathcal{N}$ is a *polygonal domain* (a two-dimensional polygon with polygonal holes). Mitchell, Rote, and Woeginger [68] compute a shortest path map in a polygonal domain by iteratively constructing the set of points at link distance $1, 2, ..., M$ from the source in $\Theta(M^4)$ [78] time and space. This approach can be modified to work on the surface $\mathcal{N}$ by propagating link paths into adjacent faces at each step. The technique of [68] also supports exact $O(M)$ time queries and approximate queries to within one link of optimal in $O(\log M)$ time. □

Notice that SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) only supports $\min_{s \in \overline{ab}} d_L(s,t)$ queries. As an alternative, we can also construct a shortest path map SPM($\overline{ab}, \overline{cd}$) that supports all possible queries between any $s \in \overline{ab} \subset \mathcal{N}$ and $t \in \overline{cd} \subset \mathcal{N}$. This lemma will be useful for constructing SPM($\overline{ab}, \overline{cd}$) because it efficiently computes the intersection of a line segment with our $\Theta(M^4)$ complexity shortest path maps from Theorem 8.5.

Lemma 8.6. *The intersection of a line segment $\overline{cd}$ with the $\Theta(M^4)$ complexity shortest path maps SPM(s, $\mathcal{N}$), SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) can be constructed in $O(M^2 \alpha(M) \log^2 M)$ time and $O(M^2)$ space (without precomputing either shortest path map).*

Proof. Recall from Section 6.2 that all link distances between a pair of line segments $\overline{ab}$, $\overline{cd}$ in a polygonal domain must equal i, $i + 1$, or $i + 2$, where $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$. Observe that this property continues to hold for $\overline{ab}$, $\overline{cd}$ on $\mathcal{C}$ or $\mathcal{N}$ because any link path between $s \in \overline{ab}$ and $t \in \overline{cd}$ can be extended by at most two extra links into a path that connects any $s' \in \overline{ab}$ and $t' \in \overline{cd}$.

Suri and O'Rourke [78] have demonstrated that all points with link distance i, $i+1$, or $i+2$ from $\overline{ab}$ can be represented by the union of $O(M^2)$ triangles. These triangles intersect $\overline{cd}$ in $O(M^2)$ intervals that can be constructed in $O(M^2 \alpha(M) \log^2 M)$ time and $O(M^2)$ space by modifying the algorithm of [68] so that it propagates link paths into adjacent faces of the surface $\mathcal{N}$ at each step. □

Recall from Section 6.2 that the combinatorial type of a (link-based) shortest path map is a listing of the combinatorial types of its edges. The combinatorial type of a shortest path map *edge* $\mathcal{E}$ is a vertex-edge pair (v, e) such that $\mathcal{E}$ has one endpoint at a vertex v and has its other endpoint on an edge e. As the source point s varies continuously along $\overline{ab}$, the position of $\mathcal{E}$'s endpoint on e is parameterized homographically by $g(s) = \frac{A+Bs}{C+Ds}$ for constants A, B, C, D. An *edgelet* α is a line segment such that the shortest path map for every source point $s \in \alpha$ has the same combinatorial structure [14].

Theorem 8.7. *The link-based shortest path map SPM($\overline{ab}, \overline{cd}$) can have $\Omega(M^4)$ complexity and can be constructed on $\mathcal{N}$ in $O(M^7)$ space and either $O(M^7)$ expected time or $O(M^6 \lambda_6(M))$*

deterministic time. The SPM($\overline{ab}, \overline{cd}$) structure supports $d_L(s,t)$, $\pi_L(s,t)$ queries for any $s \in \overline{ab} \subset \mathcal{N}$ and $t \in \overline{cd} \subset N$ in $O(\log M + K)$ time, where K is the complexity of any returned path.

Proof. SPM($\overline{ab}, \overline{cd}$) is a partition of the parameter space defined by $\overline{ab}$ and $\overline{cd}$ into maximal regions such that for all points (s,t) in a region the shortest path $\pi_L(s,t)$ has the same combinatorial structure. Edgelet endpoints are defined by positions $s \in \overline{ab}$ where there are at least two distinct shortest paths $\pi_L(s,v)$ to some vertex $v \in \mathcal{N}$ [14]. A total of $O(M^3)$ edgelet endpoints can be defined by computing $\overline{ab} \cap$ SPM($v, \mathcal{N}$) for every vertex $v \in \mathcal{N}$ in $O(M \cdot M^2 \alpha(M) \log^2 M)$ total time and $O(M^3)$ space (see Lemma 8.6).

For each edgelet $\alpha \in \overline{ab}$, choose a point $s \in \alpha$, and construct SPM(s, $\mathcal{N}$). A shortest path map edge $\mathcal{E} \in$ SPM(s, $\mathcal{N}$) is a vertex-edge pair (v,e) that is homographically parameterized by $s \in \alpha$ such that $\mathcal{E} \cap \overline{cd}$ defines a constant complexity algebraic curve in the parameter space for $\overline{ab}$ and $\overline{cd}$. Constructing such a curve for each choice of v and e yields $O(M^2)$ curves whose arrangement can be constructed in $O(M^4)$ space and $O(M^4)$ expected time or $O(M^3 \lambda_6(M))$ deterministic time [4]. Since there are $O(M^3)$ edgelets, SPM($\overline{ab}$,$\overline{cd}$) has $O(M^3 \cdot M^4)$ complexity. SPM($\overline{ab}, \overline{cd}$) can have $\Omega(M^4)$ complexity by Lemma 6.13. □

We now show how to compute the largest link distance between any pair of points on $\mathcal{N}$.

Theorem 8.8. *The link diameter of $\mathcal{N}$ can be computed in $O(M^{\frac{19}{3}} \log^{3.11} M)$ time and $O(M^3)$ space.*

Proof. Partition each of the $O(M)$ edges of $\mathcal{N}$ into $O(M^3)$ edgelets with Lemma 8.6. Choose a point s in each of these $O(M^4)$ edgelets. For each of these points, construct the $O(M^2)$ possible pairs of vertices and edges of $\mathcal{N}$ that define the edges of SPM(s, $\mathcal{N}$). This can be done (without computing the arrangement of these edges) in $O(M^{\frac{7}{3}} \log^{3.11} M)$ time per edgelet by the techniques of [68] and Theorem 8.5. The largest link distance defined by any of these edges is the link diameter for $\mathcal{N}$. □

8.3 Voronoi Diagrams and Hausdorff Distance on $\mathcal{N}$

The next theorem discusses a link-based Voronoi diagram for $\mathcal{N}$.

Theorem 8.9. *A link-based Voronoi diagram for M point or line segment sites can be computed on $\mathcal{N}$ in $O(M^6)$ time and space and can have $\Omega(M^4)$ complexity.*

Proof. For each point site s on $\mathcal{N}$, construct the $O(M^2)$ possible edges that define SPM(s, $\mathcal{N}$) using the techniques of [68] and Theorem 8.5. Similarly, construct the $O(M^2)$ possible edges that define SPM($\min_{s \in \overline{ab}}$, $\mathcal{N}$) for each line segment site $\overline{ab}$ on $\mathcal{N}$. Since there are M sites, this procedure defines $O(M^3)$ edges on $\mathcal{N}$. The arrangement of these edges can be constructed with an output-sensitive algorithm [26] in $O(M^6)$ time and space. Since some edges in this arrangement should not appear in the final Voronoi diagram, a breadth-first postprocessing step can be used to merge adjacent faces with link distance i that are separated by a suboptimal edge with link distance $j > i$. The $\Omega(M^4)$ lower bound follows from the lower bound for SPM(s, $\mathcal{N}$). □

Theorem 8.10. *The link-based Hausdorff distance can be computed between two sets of M points on $\mathcal{N}$ in $O(M^{\frac{10}{3}} \log^{3.11} M)$ time and $O(M^2)$ space and between two sets of M line segments on $\mathcal{N}$ in $O(M^4 \alpha(M) \log^2 M)$ time and $O(M^3)$ space. In addition, the Hausdorff distance can be approximated to within two links of optimal for line segments on $\mathcal{N}$ in $O(M^{\frac{10}{3}} \log^{3.11} M)$ time and $O(M^2)$ space.*

Proof. For point sets on $\mathcal{N}$, Theorem 8.5 can be used to construct an implicit shortest path map for every point. These shortest path maps are sufficient to return the Hausdorff distance.

Line segment sets A and B on $\mathcal{N}$ can be handled as follows. Intersecting a fixed line segment $\overline{ab} \subset A$ with SPM($\min_{t \in \overline{cd}}$, $\mathcal{N}$) for every $\overline{cd} \subset B$ produces a set of piecewise-constant distance functions. The lower envelope of these functions defines for every point $s \in \overline{ab}$ the nearest neighbor distance $\min_{t \in B} d_L(s,t)$. Such a lower envelope can be computed for each $\overline{ab} \subset A$ in $O(M^3 \alpha(M) \log^2 M)$ time and $O(M^3)$ space by combining Lemma 8.6 with a plane sweep technique. Repeating this step for each $\overline{ab} \subset A$ and returning the largest distance on any lower envelope

yields the Hausdorff distance $\delta_H(A, B)$ in $O(M^4\alpha(M)\log^2 M)$ time. Recall from the proof of Lemma 8.6 that all link distances between a pair of line segments equal i, $i+1$, or $i+2$, where $i = \min_{s\in\overline{ab},\ t\in\overline{cd}} d_L(s,t)$. This implies that the Hausdorff distance can be approximated to within two links of optimal for line segment sets on $\mathcal{N}$ by computing the Hausdorff distance of the *endpoints* of the line segments. □

8.4 Fréchet Distance

This chapter contains link-based algorithms to compute the Fréchet distance between polygonal curves on the polyhedral surfaces $\mathcal{C}$ and $\mathcal{N}$. These surfaces were defined at the beginning of Chapter 8. The below theorem shows how to compute the link-based Fréchet distance on $\mathcal{C}$ a logarithmic factor faster than the traditional Fréchet distance in the plane [10].

Theorem 8.11. *The link-based Fréchet distance between two polygonal curves A and B on $\mathcal{C}$ can be computed in $O(M^2)$ time and space, and the free space diagram can have $\Omega(M^2)$ complexity. The total complexity of A, B, and $\mathcal{C}$ is M.*

Proof. Although the free space in a cell defined by $\overline{ab} \subset A$ and $\overline{cd} \subset B$ can be disconnected, reachability information can always be propagated through a cell solely based on the link distances defining the cell boundary. In the simplest case, $\overline{ab}$ and $\overline{cd}$ are in the same convex face of $\mathcal{C}$, and all link distances in the cell are either zero (at any intersection of $\overline{ab}$ and $\overline{cd}$) or one. When $\overline{ab}$ and $\overline{cd}$ are in separate faces, all points in the *interior* of a cell have the same link distance; furthermore, all link distances on the *interior* of a cell's boundary edge are the same (see Figure 8.1). This behavior ensures that reachability information can always be propagated through a cell in constant time solely based on the distances defining the cell boundary.

The distances defining all cell boundaries in the free space diagram can be computed in $\Theta(M^2)$ time using our shortest path map structures on $\mathcal{C}$. The Fréchet distance can then be computed by building a planar graph on the free space diagram and applying a linear-time shortest path algorithm (e.g., [58]). □

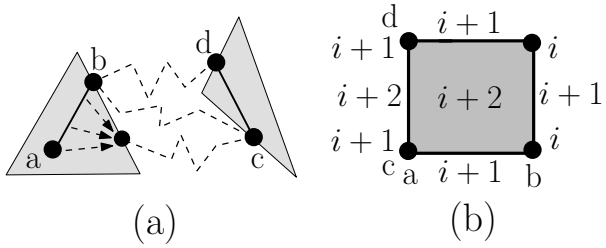


Figure 8.1: (a) When $\overline{ab}$ and $\overline{cd}$ are in separate faces of $\mathcal{C}$, all distances in the interior of a cell defined by $\overline{ab}$ and $\overline{cd}$ have the same value. (b) Free space in a cell can be disconnected by choosing $\varepsilon = i$ so that all link distances less than or equal to i define the free space.

We now use our shortest path map SPM($\overline{ab}, \overline{cd}$) to compute the link-based Fréchet distance on $\mathcal{N}$. The Fréchet *decision problem* [10] decides whether the Fréchet distance $\delta_F(A, B) \leq \varepsilon$ for some non-negative constant ε.

Theorem 8.12. *The link-based Fréchet distance between two polygonal curves A and B on $\mathcal{N}$ can be computed* exactly *in $O(M^9 \log M)$ time and $O(M^4)$ space, and the free space diagram can have $\Omega(M^6)$ complexity. The Fréchet distance can also be approximated to within one link of optimal in $O(M^4 \alpha(M) \log^2 M)$ time or to within two links of optimal in $O(M^{\frac{10}{3}} \log^{3.11} M)$ time. Both approximations use $O(M^2)$ space.*

Proof. The *exact* Fréchet decision problem is computed by representing each of the M^2 cells defined by $\overline{ab} \subset A$ and $\overline{cd} \subset B$ by our SPM($\overline{ab}$,$\overline{cd}$) structure and combining dynamic programming [10] with a plane sweep that propagates reachability information through each cell. Alternatively, our shortest path maps on $\mathcal{N}$ can be used to approximate all link distances in any cell to within one or two links of optimal. The approximate Fréchet distance can then be computed by constructing a directed acyclic graph on the approximate free space diagram and performing a breadth first search. After executing either of these approximation algorithms, the Fréchet distance is known to within $O(1)$ links of its true value, and the exact decision problem can be executed $O(1)$ times to return the exact Fréchet distance. The free space diagram can have $\Omega(M^6)$ complexity because each of the $O(M^2)$ cells can have $\Omega(M^4)$ complexity due to the lower bound for SPM($\overline{ab}$,$\overline{cd}$). □

CHAPTER 9: VISITING A SEQUENCE OF POINTS WITH A BEVEL-TIP NEEDLE

This chapter shows how to guide a bevel-tip needle through a sequence of treatment points in the plane. Most of the work in this chapter has been presented at the 2009 *Fall Workshop on Computational Geometry* [42] and has been submitted to the *9th Latin American Theoretical Informatics Symposium*. This chapter is the primary work of Atlas F. Cook IV and his doctoral advisor Carola Wenk. Steven Bitner, Yam K. Cheung, Ovidiu Daescu, and Anastasia Kurdia discussed some of the initial ideas of this work with Atlas F. Cook IV when he used his 2009 Presidential Dissertation Fellowship funds to visit the University of Texas at Dallas. We would also like to thank Carlos Esquivel, Sotirios Stathakis, and the *Cancer Therapy and Research Center at the University of Texas Health Science Center at San Antonio* for a thorough demonstration of needle steering hardware and clinical procedures.

Many surgical procedures could benefit from guiding a bevel-tip needle through a sequence of treatment points in a patient. For example, brachytherapy procedures permanently implant radioactive seeds that irradiate the surrounding cancerous tissue, and biopsy procedures extract tissue samples to test for cancer [13].

A bevel-tip needle has an asymmetric tip that cuts through tissue along a circular arc. By rotating the needle during its insertion into soft tissue, the needle can be steered along a sequence of circular arcs so that it reaches a treatment point while avoiding bones and vital organs. Although unpredictable deflections can occur due to tissue heterogeneity, these deflections are typically accounted for by periodically taking snapshots of the needle during surgery [13].

Although current procedures typically create a new puncture for each individual target, our work permits multiple targets to be visited with a single puncture. Since each reorientation of the needle inherently complicates the path for the physician, we minimize the number of arcs that are required for the needle to visit a sequence of treatment points.

We always assume that the needle travels on circular arcs with a fixed radius r. Note that this

radius can be controlled to some extent by varying the stiffness of the needle. Although deflections can in practice lead to deviations from an ideal path, the needle position can be periodically sampled during a procedure to account for these deflections.

We consider problems in problem spaces without obstacles and show how to compute several data structures that can quickly return a needle path. Section 9.1 shows that given one start position for the needle in $\mathbb{R}^d$, a shortest path map can quickly return a needle path to any target point in $\mathbb{R}^d$. Additionally, for multiple candidate needle start positions in $\mathbb{R}^d$, a Voronoi diagram can efficiently identify an optimal needle start position to use to reach any given destination in $\mathbb{R}^d$. Sections 9.2 and 9.3 describe how to compute a needle path that visits an (ordered) sequence of treatment points in the plane.

To our knowledge, no previous algorithm exists to compute needle paths that visit multiple treatment points. Related work on this topic computes Euclidean shortest paths that tour a sequence of polygons [47]. Other previous work on needle paths is usually based on techniques such as probabilistic roadmaps.

9.1 Needle Steering in $\mathbb{R}^d$

A bevel-tip needle has an asymmetric tip that cuts through soft tissue along a circular arc. Each rotation of the needle during its insertion into tissue causes the needle to start moving along a new circular arc that is tangent to the previous circular arc [12]. A *needle arc* is a directed circular arc that lies on a *needle circle* with radius r. A *needle vector* is a vector that is tangent to the needle's current position on a needle arc. Figure 9.1a illustrates that a *needle path* $\pi_N(s,t)$ is a connected sequence of needle arcs that connects two points s and t while satisfying the following properties. First, consecutive needle arcs of $\pi_N(s,t)$ lie on tangent needle circles. Second, consecutive needle arcs always have opposite orientations (e.g., if the current arc travels clockwise, then the next arc must travel counter-clockwise). Third, the *length* $d_N(s,t)$ of a needle path is optimal in that it equals the smallest possible number of needle arcs that can be used to connect s and t.

A shortest path map can quickly return a needle path $\pi_N(s,t)$ from some fixed source point

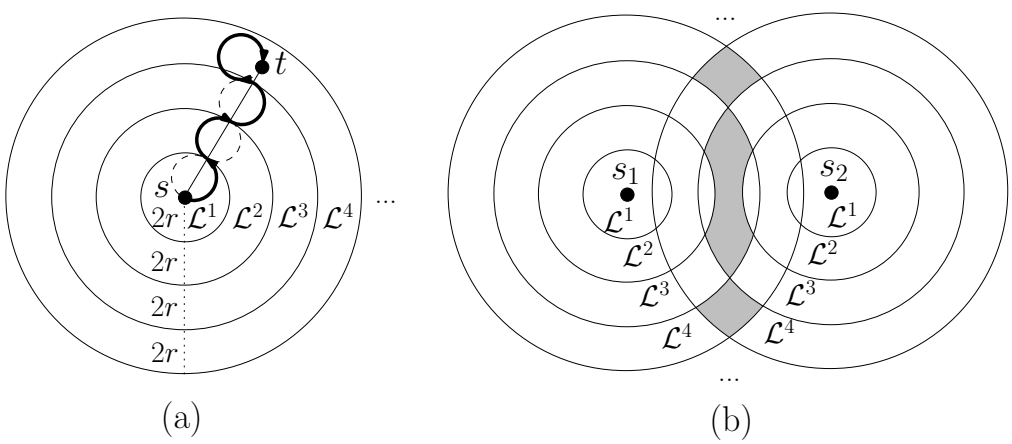


Figure 9.1: (a) The shortest path map $\text{SPM}_N(s)$ is an arrangement of concentric hyperspheres. A needle path $\pi_N(s,t)$ with four needle arcs is shown. (b) The shaded bisector of $\text{SPM}_N(s_1)$ and $\text{SPM}_N(s_2)$ is a superset of the Euclidean bisector for s_1, s_2.

$s \in \mathbb{R}^d$ to any target point $t \in \mathbb{R}^d$. Assuming that the initial needle vector can be chosen arbitrarily, all points within Euclidean distance $2r$ of s can be reached with one needle arc. More generally, for any integer $j \geq 1$ all points $\{t \in \mathbb{R}^d \mid (j-1) \cdot 2r < \|s-t\| \leq j \cdot 2r\}$ can be reached by a needle path that is composed of j needle arcs (see Figure 9.1a). Thus, the length of a needle path from s to t is $d_N(s,t) = \lceil \frac{\|s-t\|}{2r} \rceil$. The needle arcs of $\pi_N(s,t)$ can be constructed by stacking a sequence of needle circles with centers on the line segment $\overline{st}$. Note that the final circle on $\pi_N(s,t)$ should be tangent to t and may not have its center on $\overline{st}$. We can now define a shortest path map $\text{SPM}_N(s)$ as a partition of $\mathbb{R}^d$ into an interior-disjoint set of layers $\mathcal{L}^1, ..., \mathcal{L}^M$ such that all points $t \in \mathcal{L}^j$ have $d_N(s,t) = j$. Such a shortest path map is an arrangement of concentric hyperspheres that are centered at s and have radii $j \cdot 2r$ for all integers $j \geq 1$ (see Figure 9.1a).

Given multiple candidate start positions $\{s_1, ..., s_n\}$ for the needle, we may wish to quickly determine a nearest starting point to a given query point $t \in \mathbb{R}^d$. Such a task can be efficiently accomplished using a traditional Euclidean Voronoi diagram [44]. The *bisector* for any two points s_i, s_j is the set of all points $\{t \in \mathbb{R}^d \mid d_N(s_i,t) = d_N(s_j,t)\}$ (see Figure 9.1b). This bisector is always a superset of the Euclidean bisector of s_i, s_j because being equidistant with respect to Euclidean distance implies being equidistant with respect to the length of a needle path.

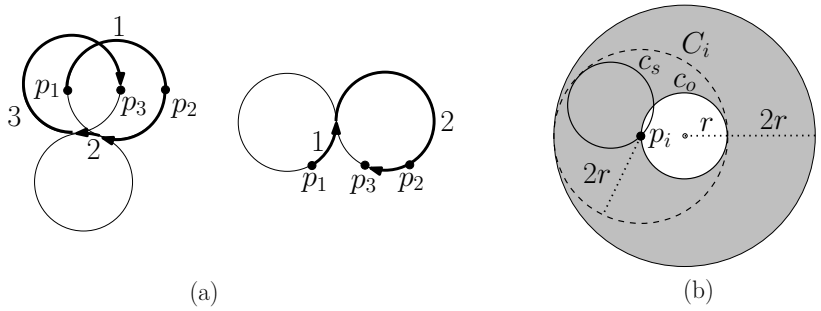


Figure 9.2: (a) Optimal substructure need not hold for a needle path that visits an arbitrary sequence of points. (b) Optimal substructure does hold for moderately-separated points.

9.2 Visiting a Sequence of Well-Separated Points in the Plane

This section shows how to compute a needle path $\pi_N(p_1, ..., p_n)$ in the plane that starts at a point p_1, visits a sequence of points $p_2, ..., p_{n-1}$, and ends at a point p_n. A needle path is said to *visit* a sequence of points when these points appear in order along the path. Let $d_N(p_1, ..., p_n)$ be the number of needle arcs on $\pi_N(p_1, ..., p_n)$. Assume for now that the needle vector is allowed to be arbitrary when the needle visits each point p_i. We call a sequence of points $p_1, \ldots, p_n$ *moderately-separated* whenever $||p_i - p_{i-1}|| \geq 2r$ for all $2 \leq i \leq n$ and *well-separated* whenever $||p_i - p_{i-1}|| \geq 4r$ for all $2 \leq i \leq n$.

Lemma 9.1. *Optimal substructure need not hold for a needle path that visits an arbitrary sequence of points. Optimal substructure does hold for moderately-separated points.*

Proof. Figure 9.2a illustrates a needle path $\pi_N(p_1, p_2, p_3)$ that does not have optimal substructure. Although a locally optimal path from p_1 to p_2 is possible with one needle arc, all possible extensions of such a path can require a total of three needle arcs to reach p_3. By contrast, a globally optimal path can reach p_3 with just two needle arcs if a locally suboptimal path with two needle arcs is used to connect p_1 to p_2.

To see that optimal substructure does hold for moderately-separated points, consider two paths to p_i. Let c_o be a needle circle that intersects p_i and lies on an (optimal) needle path $\pi_N(p_1, ..., p_i)$ with length $d_N(p_1, ..., p_i) = j$. Let c_s be a needle circle that intersects p_i and lies on a suboptimal

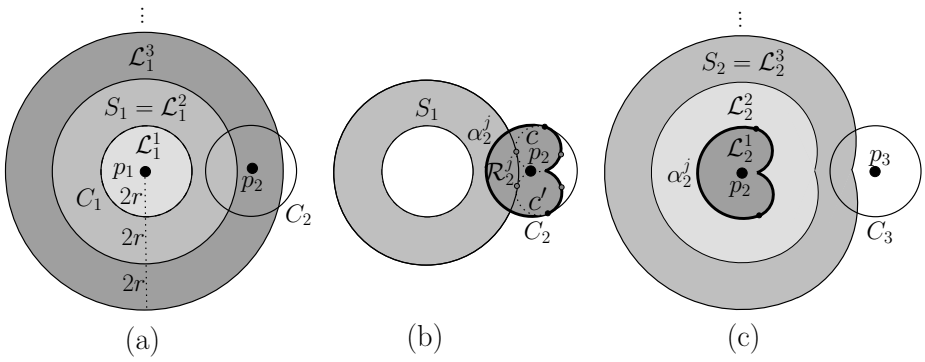


Figure 9.3: (a) The sequence of layers $\mathcal{L}_1^1, ..., \mathcal{L}_1^M$ for p_1 can always be described by the shortest path map $\text{SPM}_N(p_1)$. We refer to S_1 as the first layer in this sequence that intersects C_2. (b,c) Layer $\mathcal{L}_2^1$ is the set of all needle circles that intersect both p_2 and S_1. Layer S_2 is the first layer in the sequence $\mathcal{L}_2^1, ..., \mathcal{L}_2^M$ that intersects C_3.

path with length at least $j + 1$. Figure 9.2b illustrates that the shaded set of all points reachable from c_o with one additional needle arc will always contain all points on the suboptimal circle c_s that lie outside the disk d_o that is bounded by c_o. This implies that for any point $p_{i+1} \notin d_o$, a path $\pi_N(p_1, ..., p_{i+1})$ that contains c_o can always be at least as short as any path $\pi'_N(p_1, ..., p_{i+1})$ that contains c_s. Since moderately-separated points guarantee that $p_{i+1} \notin d_o$, optimal substructure holds for moderately-separated points. □

For each point p_i, we can now define a partition of the plane such that each point $t \in \mathbb{R}^2$ has an associated distance $d_N(p_1, ..., p_i, t)$. Such a partition can be described by a sequence of pairwise disjoint layers $\mathcal{L}_i^1, ..., \mathcal{L}_i^M$ such that all points $t \in \mathcal{L}_i^j$ have associated distance $d_N(p_1, ..., p_i) + j - 1$ (see Figure 9.3). Note that all needle circles composing $\mathcal{L}_i^1$ must necessarily intersect p_i, so the boundary of $\mathcal{L}_i^1$ is a connected sequence of needle arcs.

Although the number of layers needed to partition the plane can be unbounded, the optimal substructure of moderately-separated points ensures that it is always possible to efficiently construct any layer $\mathcal{L}_i^j$ directly from $\mathcal{L}_i^1$ because all needle paths $\pi_N(p_1, ..., p_i, t)$ must pass through $\mathcal{L}_i^1$. In particular, any layer $\mathcal{L}_i^{j \geq 2}$ is the set of all points $\{t \in \mathbb{R}^d \mid (j-2) \cdot 2r < \min_{s \in \mathcal{L}_i^1} \|s - t\| \leq (j-1) \cdot 2r\}$. The boundary separating $\mathcal{L}_i^{j \geq 2}$ and $\mathcal{L}_i^{j+1}$ can be computed by additively *enlarging* the radius of arcs on the boundary of $\mathcal{L}_i^1$ by $(j-1) \cdot 2r$ while keeping the center of each arc fixed.

We now show how to compute $\mathcal{L}_i^1$ from $\mathcal{L}_{i-1}^1$ for a sequence of moderately-separated points. Let C_i be the circle with radius $2r$ that is centered at the point p_i. Notice that each needle circle in $\mathcal{L}_i^1$ must intersect both p_i and some unique point of C_i. For our purposes, the first layer in the sequence $\mathcal{L}_{i-1}^1, ..., \mathcal{L}_{i-1}^M$ that intersects C_i will be of such importance that we will give this layer the special name S_{i-1} (see Figures 9.3b and 9.3c).

Optimal substructure implies that $\mathcal{L}_i^1$ is the union of all needle circles that intersect p_i and are tangent to a circle in S_{i-1}. For each arc on the boundary of S_{i-1}, compute the at most two needle circles that are tangent to this arc and intersect p_i. Order the resulting needle circles into a sequence by the polar coordinates of their centers with respect to p_i. Let c and c' be two adjacent needle circles in this sequence such that if c is continuously rotated counter-clockwise about p_i to c', then all of the needle circles during this rotation will intersect S_{i-1} (see Figure 9.3b). Notice that the union of all needle circles in the rotation from c to c' can be compactly described by a constant complexity region $\mathcal{R}_i^j \subseteq \mathcal{L}_i^1$. We call the arc $\alpha_i^j \subseteq C_i$ that intersects S_{i-1} and lies between c and c' a *generating arc* because α_i^j encodes all of the information necessary to generate the region $\mathcal{R}_i^j$.

Each region $\mathcal{R}_i^j$ is always bounded by its associated generating arc $\alpha_i^j \subseteq C_i$ and arcs of c and c' (see Figures 9.4a and 9.4b). Whenever a generating arc α_i^j covers less than half of C_i, the points in the interior of $c \cap c'$ are contained in layer $\mathcal{L}_i^2$ (see Figure 9.4c). Optimal substructure ensures that all layers $\mathcal{L}_i^{j \geq 3}$ will be connected.

Notice that a layer $\mathcal{L}_i^1$ can have multiple associated generating arcs. Consequently, $\mathcal{L}_i^1$ can be defined as the union of a sequence of regions $\mathcal{R}_i^1, ..., \mathcal{R}_i^k$ (see Figure 9.4d). In the remainder of this section, we will avoid computing an explicit union of these regions and instead work directly with the individual generating arcs and regions that compose a layer.

Lemma 9.2. *A sequence of points $p_1, ..., p_n$ that are* not *well-separated can define a layer $\mathcal{L}_n^1$ with $\Theta(2^n)$ generating arcs.*

Proof. The lower bound can be realized by placing $p_1, ..., p_n$ on a line such that $2r < ||p_i - p_{i-1}|| < 4r$ for all $2 \leq i \leq n$. As illustrated in Figure 9.5a, layer $\mathcal{L}_1^1$ is a disk. Place p_2 to the right of p_1 so that layer $\mathcal{L}_2^1$ is defined by one generating arc $\alpha_2^1 \subset C_2$ that represents all needle circles that intersect p_2

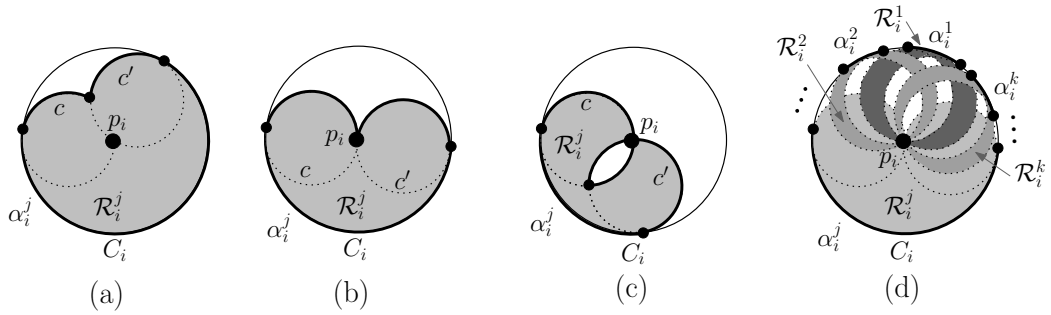


Figure 9.4: (a,b) Region $\mathcal{R}_i^j \subseteq \mathcal{L}_i^1$ is bounded by $\alpha_i^j \subseteq C_i$ and two circular arcs that may or may not touch p_i. (c) Whenever a generating arc α_i^j covers less than half of C_i, the points in the interior of $c \cap c'$ are contained in layer $\mathcal{L}_i^2$. (d) In general, layer $\mathcal{L}_i^1$ is the union of a sequence of regions $\mathcal{R}_i^1, ..., \mathcal{R}_i^k$. These regions can be computed from the associated sequence of generating arcs $\alpha_i^1, ..., \alpha_i^k \subseteq C_i$.

and are tangent to some needle circle in $\mathcal{L}_1^1$. Place p_3 to the right of p_2 such that only two discrete needle circles intersect p_3 and are tangent to some needle circle in $\mathcal{L}_2^1$. Each of these needle circles defines a (degenerate) generating arc, and the union of these two needle circles is $\mathcal{L}_3^1$. Place p_4 to the left of p_3 such that four discrete needle circles intersect p_4 and are tangent to some needle circle in $\mathcal{L}_3^1$. The union of these four needle circles is $\mathcal{L}_4^1$ (see Figure 9.5b). Place p_5 to the right of p_4 such that eight discrete needle circles define $\mathcal{L}_5^1$ (see Figure 9.5c). Continuing in this fashion for every $i \geq 3$, we can place p_{2i} to the left of p_{2i-1} and p_{2i+1} to the right of p_{2i} such that layer $\mathcal{L}_n^1$ is composed of 2^{n-2} circles. Thus, $\mathcal{L}_n^1$ can have $\Omega(2^n)$ generating arcs.

We now show that layer $\mathcal{L}_n^1$ has $O(2^n)$ generating arcs. Assume that $\mathcal{L}_{i-1}^1$ is defined by k generating arcs. By definition, each region associated with these generating arcs is bounded by an arc on C_{i-1} and at most two additional arcs that can lie on unique circles (see Figure 9.4a). This implies that the arcs bounding any fixed layer $\mathcal{L}_{i-1}^j$ lie on at most $2k + 1$ unique circles. Now consider the circle C_i. Since any pair of circles intersect at most twice, C_i intersects each of the $2k+1$ unique circles for S_{i-1} at most twice. Since each generating arc of $\mathcal{L}_i^1$ can always be charged to two unique intersections of C_i with these circles, $\mathcal{L}_i^1$ is defined by at most $2k+1$ generating arcs. An inductive argument now ensures that $\mathcal{L}_n^1$ has $O(2^n)$ generating arcs. □

Exponential behavior occurred above because C_i was able to pass through $\mathcal{L}_{i-1}^1$ many times.

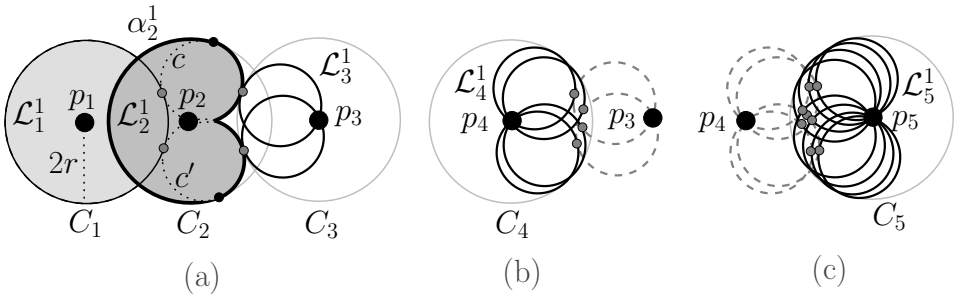


Figure 9.5: When points are not well-separated, layer $\mathcal{L}_n^1$ can have $\Theta(2^n)$ generating arcs. Part (a) shows the disk defining $\mathcal{L}_1^1$, the generating arc α_2^1 defining $\mathcal{L}_2^1$, and the two circles defining $\mathcal{L}_3^1$. Parts (b) and (c), respectively, illustrate the four circles defining $\mathcal{L}_4^1$ and the eight circles defining $\mathcal{L}_5^1$.

However, if we assume that $\|p_i - p_{i-1}\| \geq 4r$, then C_i intersects $\mathcal{L}_{i-1}^1$ in at most one point. This property is used in the following lemma to show that when the points $p_1, ..., p_n$ are well-separated, each layer $\mathcal{L}_i^1$ is defined by $O(i)$ generating arcs.

Lemma 9.3. *Given both a sequence of well-separated points $p_1, ..., p_n$ and the sequence of generating arcs defining layer $\mathcal{L}_{i-1}^1$, there exist $O(i)$ generating arcs for any layer $\mathcal{L}_i^1, ..., \mathcal{L}_i^M$. These $O(i)$ generating arcs can be constructed in $O(i)$ total time.*

Proof. Assume that $\mathcal{L}_{i-1}^1$ is defined by a sequence of k sorted, pairwise disjoint generating arcs $\alpha_{i-1}^1, ..., \alpha_{i-1}^k \subseteq C_{i-1}$. Recall that S_{i-1} denotes the first layer in the sequence $\mathcal{L}_{i-1}^1, ..., \mathcal{L}_{i-1}^M$ that intersects C_i. If $S_{i-1} = \mathcal{L}_{i-1}^1$, then the well-separated property ensures that at most one circle defines $\mathcal{L}_{i-1}^1$. Otherwise, S_{i-1} equals some layer $\mathcal{L}_{i-1}^{j \geq 2}$, and the well-separated property ensures that C_i only intersects the connected component of S_{i-1} that lies outside the disk bounded by C_{i-1}. Such a connected component has two closed boundaries, and we let $\mathcal{B}$ be the boundary that is farthest from p_{i-1}. Observe that $\mathcal{B}$ is a closed sequence of arcs (see Figure 9.6). Every third arc of $\mathcal{B}$ is an additively enlarged version of a generating arc such that all additively enlarged generating arcs lie on one fixed circle. Adjacent additively enlarged generating arcs in $\mathcal{B}$ are always connected by two arcs that intersect in one concave vertex. Hence, there are at most k concave vertices on $\mathcal{B}$.

We now bound the number of generating arcs that can define $\mathcal{L}_i^1$ by counting the number of times that C_i can intersect $\mathcal{B}$. There are two types of intersections to consider. First, C_i can

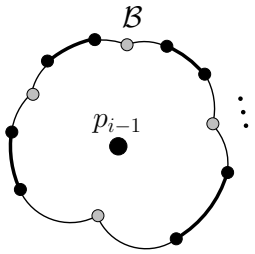


Figure 9.6: An arbitrary circle C_i intersects the boundary $\mathcal{B} \subseteq S_{i-1}$ at most twice for each of the gray concave vertices on $\mathcal{B}$ and at most twice over all points on the thick additively enlarged generating arcs.

intersect $\mathcal{B}$ at most twice over all additively enlarged generating arcs because these arcs all lie on a single circle. Second, C_i can intersect $\mathcal{B}$ at most twice for each of the k concave vertices on $\mathcal{B}$. Since a generating arc is always defined by *two* unique intersections, the at most $2k+2$ intersections of C_i with $\mathcal{B}$ ensure that $\mathcal{L}_i^1$ is defined by at most $k+1$ generating arcs. An inductive argument can now easily show that $O(i)$ generating arcs define $\mathcal{L}_i^1$. These generating arcs can be constructed in sorted order on C_i in $O(i)$ total time by traversing the boundary $\mathcal{B}$ in sorted order. □

We can now compute a needle path $\pi_N(p_1, ..., p_n)$ that passes through a sequence of well-separated points.

Theorem 9.4. *Given a sequence $p_1, ..., p_n$ of well-separated points, a needle path $\pi_N(p_1, ..., p_n)$ can be computed in $O(n^2 + K)$ time and space, where K is the complexity of the returned path.*

Proof. The generating arcs for $\mathcal{L}_i^1$ and S_i can be constructed from the generating arcs of $\mathcal{L}_{i-1}^1$ in $O(i)$ time by Lemma 9.3. Hence, the generating arcs for $S_1, ..., S_{n-1}$ and $\mathcal{L}_n^1$ can be constructed in $O(n^2)$ total time.

To compute $\pi_N(p_1, ..., p_n)$, pick any circle $c_n \subseteq \mathcal{L}_n^1$. This circle intersects some point $q \in S_{n-1}$. To connect $q \in S_{n-1}$ to p_{n-1}, draw a line segment from p_{n-1} to q and find the last intersection point q' of this line segment with the boundary of $\mathcal{L}_{n-1}^1$. Draw the circle c_{n-1} that touches p_{n-1} and q'. To connect the two circles c_{n-1} and c_n, identify the closest pair of points $r \in c_{n-1}$, $r' \in c_n$ (with respect to Euclidean distance) and create a chain of tangent needle circles along the line segment from r to r'. Note that the needle circle touching r' may need to be rotated slightly about the previous

circle in the chain to ensure that this circle is tangent to c_n. We have now found an optimal path from p_n to p_{n-1}, and this process can be iteratively repeated from c_{n-1} to construct the remainder of $\pi_N(p_1, ..., p_n)$. □

Note that if we require the needle to have a specific needle vector when it visits each p_i, then each layer $\mathcal{L}_i^1$ consists of the at most two needle circles that are tangent to this vector. For this scenario, our layers technique can return a needle path in only $O(n+K)$ time and space, where K is the complexity of the returned path.

9.3 Visiting a Sequence of Arbitrary Points in the Plane

This section describes a fixed-parameter tractable algorithm that computes a needle path for a sequence of points $p_1, ..., p_{m+n}$ such that any m pairs of consecutive points are positioned arbitrarily and the remaining pairs of consecutive points are well-separated. Note that optimal substructure can fail for this scenario as depicted by Figure 9.2a. However, an optimal path can still be returned by accounting for locally suboptimal needle circles that touch multiple consecutive points in the sequence.

Lemma 9.5. *Given a sequence of points $p_1, ..., p_{m+n}$, the generating arcs for layer $\mathcal{L}_i^1$ can be constructed in time proportional to the number of generating arcs defining $\mathcal{L}_{i-1}^1$ and $\mathcal{L}_{i-2}^1$.*

Proof. If $||p_i - p_{i-1}|| \geq 2r$, then optimal substructure holds by Lemma 9.1, and we can compute the generating arcs for $\mathcal{L}_i^1$ from the generating arcs of $\mathcal{L}_{i-1}^1$ as in the previous section. Now assume that $||p_i - p_{i-1}|| < 2r$, and let $\gamma = d_N(p_1, ..., p_{i-1})$. Our first step is to determine the value of $d_N(p_1, ..., p_i)$.

If $p_i \in \mathcal{L}_{i-1}^1$, then we can immediately return $d_N(p_1, ..., p_i) = \gamma$. If there exists any fixed needle circle $c \in \mathcal{L}_{i-1}^1$ that does not surround p_i, then the precondition $||p_i - p_{i-1}|| < 2r$ ensures that $d_N(p_1, ..., p_i) = \gamma + 1$. The only other way to obtain $d_N(p_1, ..., p_i) = \gamma + 1$ is for a locally suboptimal path with total length $\gamma + 1$ to end in a needle circle that touches both p_i and p_{i-1}. Such paths can be accounted for by determining whether the layer $\mathcal{L}_{i-2}^j$ with associated distance $\gamma + 1$ contains a needle circle that touches both p_i and p_{i-1}. Otherwise, the precondition $||p_i - p_{i-1}|| < 2r$ ensures

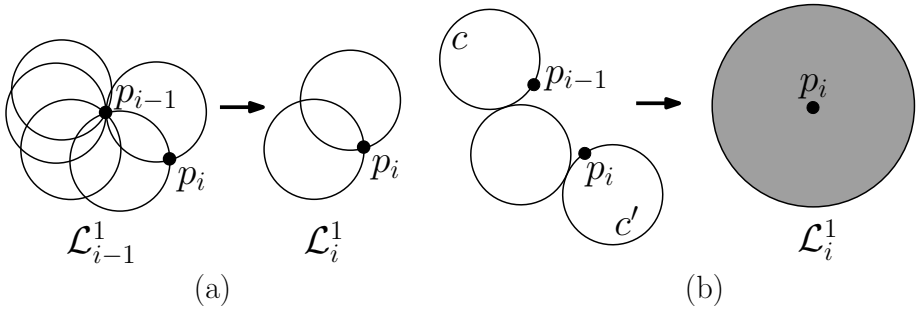


Figure 9.7: (a) If $d_N(p_1, ..., p_i) = \gamma$, then there are at most two circles that define $\mathcal{L}^1_i$. (b) If $\|p_i - p_{i-1}\| < 2r$ and $d_N(p_1, ..., p_i) = \gamma + 2$, then at most one extra circle is needed to connect an arbitrary circle $c \in \mathcal{L}^1_{i-1}$ to an arbitrary circle c' that touches p_i.

that $d_N(p_1, ..., p_i) = \gamma + 2$. This means that we can compute $d_N(p_1, ..., p_i)$ in time proportional to the number of generating arcs defining $\mathcal{L}^1_{i-1}$ and $\mathcal{L}^1_{i-2}$.

We now use the value of $d_N(p_1, ..., p_i)$ to compute $\mathcal{L}^1_i$. If $d_N(p_1, ..., p_i) = \gamma$, then $\mathcal{L}^1_i$ is composed of the at most two circles that touch both p_i and p_{i-1} and are contained in $\mathcal{L}^1_{i-1}$ (see Figure 9.7a).

If $d_N(p_1, ..., p_i) = \gamma + 1$, then we can partially compute $\mathcal{L}^1_i$ as in Lemma 9.3 using locally optimal paths from $\mathcal{L}^1_{i-1}$. In addition, we must also account for locally suboptimal paths by testing whether the layer $\mathcal{L}^j_{i-2}$ with associated distance $\gamma + 1$ contains either of the at most two needle circles that touch both p_i and p_{i-1}.

If $d_N(p_1, ..., p_i) = \gamma + 2$, then $\mathcal{L}^1_i$ is the disk centered at p_i with radius $2r$. This follows from $\|p_i - p_{i-1}\| < 2r$ because it will always be possible to connect any needle circle $c \in \mathcal{L}^1_{i-1}$ to an arbitrary needle circle c' that touches p_i by choosing one needle circle that it tangent to both c and c' (see Figure 9.7b). Hence, all of the operations needed to construct the generating arcs for $\mathcal{L}^1_i$ take time proportional to the number of generating arcs defining $\mathcal{L}^1_{i-1}$ and $\mathcal{L}^1_{i-2}$. □

Theorem 9.6. *A needle path $\pi_N(p_1, ..., p_{m+n})$ can be computed in $O(2^m n + n^2 + K)$ time and space, where K is the complexity of the returned path.*

Proof. Given a sequence of m arbitrary points, layer $\mathcal{L}^1_m$ can have $\Theta(2^m)$ generating arcs by Lemma 9.2. Lemma 9.3 ensures that each of the well-separated pairs of consecutive points adds

only a constant number of additional generating arcs to the current layer. Consequently, the worst-case time to compute the generating arcs for $\mathcal{L}_1^1, ..., \mathcal{L}_{m+n}^1$ is on the order of $\Sigma_{i=1}^{n}(2^m + i) \in O(2^m n + n^2)$. Once these generating arcs have been computed, it is a simple matter to construct $\pi_N(p_1, ..., p_{m+n})$ in the same manner as Theorem 9.4. □

CHAPTER 10: CONCLUSION AND FUTURE WORK

Our work combines similarity metrics with Euclidean shortest paths and link paths. The motivation for this work is that these types of paths are natural distance measures for complex environments such as simple polygons, polygonal domains (i.e., polygons with polygonal holes), and polyhedral surfaces.

In a simple polygon, we give the first algorithms to compute the Fréchet distance and Hausdorff distance with respect to both Euclidean shortest paths and link paths. In our Fréchet distance algorithms, we prove that the free space in a cell is x-monotone, y-monotone, and connected. Combining these properties with an extension of the shortest path algorithms of [55, 59] leads efficient algorithms for the Fréchet decision problem. We also present a randomized algorithm based on red-blue intersections that solves the Fréchet optimization problem in lieu of the standard parametric search approach (see Section 2.5 for details on parametric search). In addition, we develop link-based Voronoi diagrams and shortest path maps.

In a polygonal domain (a polygon with polygonal holes), we develop dynamic and static spotlight shortest path structures that describe all possible Euclidean shortest paths between a pair of line segments $\overline{ab}$ and $\overline{cd}$. These spotlights are used to develop algorithms to compute the Fréchet distance and a shortest path map SPM($\overline{ab}, \overline{cd}$). SPM($\overline{ab}, \overline{cd}$) supports output-sensitive shortest path queries from any source point $s \in \overline{ab}$ to any destination point $t \in \overline{cd}$. Although it is possible to use SPM($\overline{ab}, \overline{cd}$) to construct a free space cell for the Fréchet distance, SPM($\overline{ab}, \overline{cd}$) contains additional information that is not required by the Fréchet distance. Consequently, it speeds up our algorithm by a linear factor to compute the Fréchet distance directly using the dynamic and static spotlights.

We also consider link distance problems in a polygonal domain. We develop a shortest path map SPM($\overline{ab}, \mathbb{R}^2$) that can answer queries from any source point $s \in \overline{ab}$ to any target point $t \in \mathbb{R}^2$. This technique is more general than the traditional shortest path map (which can only answer queries from a single fixed point), and we apply this structure to compute the link-based Fréchet distance in a polygonal domain. We also explore many link-based variants of the Hausdorff and

Fréchet distance in the presence of obstacles and develop approximation algorithms that are accurate to within 1, 2, or 3 links of optimal.

It would be interesting to generalize our link-based shortest path maps in a polygonal domain from SPM($\overline{ab}$,$\overline{cd}$) to SPM($\overline{ab}$,$\mathbb{R}^2$) (see Section 6.2). This extension would permit queries from any point $s \in \overline{ab}$ to any point $t \in \mathbb{R}^2$. We already support this scenario in a simple polygon by building a special triangulation such that at most two parameterized shortest path map edges need to be evaluated to resolve a query. Although this triangulation approach does not immediately yield logarithmic query time in a polygonal domain, we do believe that a careful ordering of the parameterized edges in the shortest path map should permit logarithmic queries. Such a paradigm might also permit the lifting of our *Euclidean* SPM($\overline{ab}$,$\overline{cd}$) structure in Section 5.4 to SPM($\overline{ab}$,$\mathbb{R}^2$).

Since shortest paths are NP-Hard to compute in $\mathbb{R}^3$ [22], we study shortest path problems on polyhedral surfaces in $\mathbb{R}^3$. Our results include algorithms to compute edge sequences, Voronoi diagrams, shortest path maps, the Fréchet distance, and the diameter for a polyhedral surface. Despite efforts by Chandru et al. [24] to improve edge sequence algorithms, runtimes had not improved since 1997. Our work speeds up the edge sequence and diameter approaches of Agarwal et al. [1] by a linear factor and introduces many new shortest path and link distance algorithms that apply to both convex and non-convex polyhedral surfaces. Future work could attempt to construct the $\Theta(M^4)$ shortest path edge sequences on a convex polyhedral surface in $o(M^5)$ time.

We also describe two algorithms to guide a bevel-tip needle through a sequence of target points in the plane. Such paths are potentially useful for biopsy and brachytherapy procedures because they reduce the number of times that a physician is required to insert and withdraw a needle during a medical procedure. We are currently extending our technique to needle paths that visit multiple edges instead of multiple points. It would be interesting to develop an algorithm that could be incrementally updated in real-time to account for unpredictable deflections during a procedure.

BIBLIOGRAPHY

[1] P. K. Agarwal, B. Aronov, J. O'Rourke, and C. A. Schevon. Star unfolding of a polytope with applications. *SIAM Journal on Computing*, 26(6):1689–1713, 1997.

[2] P. K. Agarwal, T. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides. Curvature-constrained shortest paths in a convex polygon. *14th Symposium on Computational Geometry (SoCG)*, pages 392–401, 1998.

[3] P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, 2000.

[4] P. K. Agarwal and M. Sharir. *Davenport–Schinzel Sequences and Their Geometric Applications*, pages 1–47. Handbook of Computational Geometry, Elsevier, 2000.

[5] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. volume 17, pages 292–318, Duluth, MN, USA, 1994. Academic Press, Inc.

[6] G. Albers, J. S. B. Mitchell, L. J. Guibas, and T. Roos. Voronoi diagrams of moving points. *Journal of Computational Geometry & Applications*, 8:365–380, 1998.

[7] L. Aleksandrov, H. Djidjev, G. Huo, A. Maheshwari, D. Nussbaum, and J.-R. Sack. Approximate shortest path queries on weighted polyhedral surfaces. *31st Mathematical Foundations of Computer Science (MFCS)*, 4162:98–109, 2006.

[8] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM*, 52(1):925–53, 2005.

[9] H. Alt, P. Braß, M. Godau, C. Knauer, and C. Wenk. Computing the Hausdorff distance of geometric patterns and shapes. *Discrete and Computational Geometry - The Goodman-Pollack-Festschrift*, pages 65–76, 2003.

[10] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Journal of Computational Geometry & Applications*, 5:75–91, 1995.

[11] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[12] R. Alterovitz, M. Branicky, and K. Goldberg. Motion planning under uncertainty for image-guided medical needle steering. *International Journal of Robotics Research*, 27(1361), 2008.

[13] R. Alterovitz, K. Goldberg, and A. Okamura. Planning for steerable bevel-tip needle insertion through 2d soft tissue with obstacles. *IEEE International Conference on Robotics and Automation*, pages 1640–1645, 2005.

[14] E. M. Arkin, J. S. B. Mitchell, and S. Suri. Optimal link path queries in a simple polygon. *3rd Symposium on Discrete Algorithms (SODA)*, pages 269–279, 1992.

[15] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distance for curves, revisited. *14th Annual European Symposium on Algorithms (ESA)*, pages 52–63, 2006.

[16] B. Aronov and J. O'Rourke. Nonoverlap of the star unfolding. *Discrete and Computational Geometry*, 8(1):219–250, 1992.

[17] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[18] S. Bereg and D. Kirkpatrick. Curvature-bounded traversals of narrow corridors. *21st Symposium on Computational Geometry (SoCG)*, pages 278–287, 2005.

[19] S. Bespamyatnikh and M. Segal. Selecting distances in arrangements of hyperplanes spanned by points. volume 2, pages 333–345, September 2004.

[20] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? *23rd European Workshop on Computational Geometry (EuroCG)*, pages 170–173, 2007. Graz, Austria.

[21] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons in polynomial time. *22nd Symposium on Computational Geometry (SoCG)*, pages 80–87, 2006.

[22] J. Canny and J.H. Reif. New lower bound techniques for robot motion planning problems. *28th IEEE Foundations of Computer Science (FOCS)*, pages 49–60, 1987.

[23] E. W. Chambers, É. Colin de Verdiére, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Walking your dog in the woods in polynomial time. *24th Symposium on Computational Geometry (SoCG)*, pages 101–109, 2008.

[24] V. Chandru, R. Hariharan, and N. M. Krishnakumar. Short-cuts on star, source and planar unfoldings. *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 174–185, 2004.

[25] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

[26] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, 1992.

[27] J. Chen and Y. Han. Shortest paths on a polyhedron. *Journal of Computational Geometry & Applications*, 6(2):127–144, 1996.

[28] Y. Chiang and J. S. B. Mitchell. Two-point Euclidean shortest path queries in the plane. *10th Symposium on Discrete Algorithms (SODA)*, pages 215–224, 1999.

[29] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, 1987.

[30] A. F. Cook IV, J. Sherette, and C. Wenk. Computing the Fréchet distance between polyhedral surfaces with acyclic dual graphs. *19th Fall Workshop on Computational Geometry*, 2009.

[31] A. F. Cook IV and C. Wenk. Geodesic Fréchet and Hausdorff distance inside a simple polygon. Technical Report CS-TR-2007-004, University of Texas at San Antonio, 2007.

[32] A. F. Cook IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *17th Fall Workshop on Computational Geometry*, 2007.

[33] A. F. Cook IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *25th Symposium on Theoretical Aspects of Computer Science (STACS)*, 2008.

[34] A. F. Cook IV and C. Wenk. Geodesic Fréchet distance with polygonal obstacles. Technical Report CS-TR-2008-010, University of Texas at San Antonio, 2008.

[35] A. F. Cook IV and C. Wenk. Min-link shortest path maps and Fréchet distance. Technical Report CS-TR-2008-011, University of Texas at San Antonio, 2008.

[36] A. F. Cook IV and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms (TALG)*, 2009.

[37] A. F. Cook IV and C. Wenk. Link distance and shortest path problems in the plane. *5th Algorithmic Aspects in Information and Management (AAIM)*, pages 140–151, 2009.

[38] A. F. Cook IV and C. Wenk. Shortest path problems on a polyhedral surface. Technical Report CS-TR-2009-001, University of Texas at San Antonio, 2009.

[39] A. F. Cook IV and C. Wenk. Shortest path problems on a polyhedral surface. *Dagstuhl Seminar Proceedings 09111: Computational Geometry*, 2009.

[40] A. F. Cook IV and C. Wenk. Shortest path problems on a polyhedral surface. *25th European Workshop on Computational Geometry (EuroCG)*, 2009.

[41] A. F. Cook IV and C. Wenk. Shortest path problems on a polyhedral surface. *Algorithms and Data Structures Symposium*, pages 156–167, 2009.

[42] A. F. Cook IV and C. Wenk. Visiting points with a bevel-tip needle. *19th Fall Workshop on Computational Geometry*, 2009.

[43] M. de Berg and O. Schwarzkopf. Cuttings and applications. *Journal of Computational Geometry & Applications*, 5(4):343–355, 1995.

[44] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2nd edition, 2000.

[45] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, NY, USA, 2007.

[46] O. Devillers, M. Golin, K. Kedem, and S. Schirra. Queries on Voronoi diagrams of moving points. *Computational Geometry: Theory & Applications*, 6(5):315–327, 1996.

[47] M. Dror, A. Efrat, A. Lubiw, and J.S.B. Mitchell. Touring a sequence of polygons. *35th ACM Symposium on Theory of Computing (STOC)*, pages 473–482, 2003.

[48] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, July 1957.

[49] V. Duindam, J. Xu, R. Alterovitz, S. Sastry, and K. Goldberg. 3d motion planning algorithms for steerable needles using inverse kinematics. *Eighth International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2008.

[50] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. *Foundations of Computer Science*, 17(5):1143–1164, 2006.

[51] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali. Sweeping simple polygons with a chain of guards. *11th Symposium on Discrete Algorithms (SODA)*, pages 927–936, 2000.

[52] A. Efrat, L. J. Guibas, S. Har-Peled, J. S. B. Mitchell, and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete and Computational Geometry*, 28(4):535–569, 2002.

[53] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Information Processing Letters*, 100(6):238–245, 2006.

[54] L. Gewali, A. Meng, J. S. B. Mitchell, and S. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *4th Symposium on Computational Geometry (SoCG)*, pages 266–278, 1988.

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

[56] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. *2nd Symposium on Computational Geometry (SoCG)*, pages 1–13, 1986.

[57] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[58] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[59] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.

[60] J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class (extended abstract). In *Workshop on Algorithms & Data Structures (WADS)*, pages 331–342, 1991.

[61] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

[62] Y.-H. Hwang, R.-C. Chang, and H.-Y. Tu. Finding all shortest path edge sequences on a convex polyhedron. *Workshop on Algorithms & Data Structures (WADS)*, 1989.

[63] J. Komlós, Y. Ma, and E. Szemerédi. Matching nuts and bolts in O(n log n) time. *7th Symposium on Discrete Algorithms (SODA)*, pages 232–241, 1996.

[64] A. Maheshwari, J.-R. Sack, and H. N. Djidjev. Link distance problems. *Handbook of Computational Geometry*, 1999.

[65] A. Maheshwari and J. Yi. On computing Fréchet distance of two paths on a convex polyhedron. *21st European Workshop on Computational Geometry (EuroCG)*, 2005.

[66] J. S. B. Mitchell. Geometric shortest paths and network optimization. *Handbook of Computational Geometry*, 1998.

[67] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987.

[68] J. S. B. Mitchell, G. Rote, and G. J. Woeginger. Minimum-link paths among obstacles in the plane. *6th Symposium on Computational Geometry (SoCG)*, pages 63–72, 1990.

[69] D. M. Mount. The number of shortest paths on the surface of a polyhedron. *SIAM Journal on Computing*, 19(4):593–611, 1990.

[70] J. O'Rourke and C. Schevon. Computing the geodesic diameter of a 3-polytope. *5th Symposium on Computational Geometry (SoCG)*, pages 370–379, 1989.

[71] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–310, 1994.

[72] E. Papadopoulou and D. T. Lee. A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains. *Algorithmica*, 20(4):319–352, 1998.

[73] G. Rote. Computing the Fréchet distance between piecewise smooth curves. Technical Report ECG-TR-241108-01, May 2005.

[74] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[75] C. Schevon and J. O'Rourke. The number of maximal edge sequences on a convex polytope. *26th Allerton Conference on Communication, Control, and Computing*, pages 49–57, 1988.

[76] Y. Schreiber and M. Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discrete & Computational Geometry*, 39(1-3):500–579, 2008.

[77] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision and Graphical Image Processing (CVGIP)*, 35(1):99–110, July 1986.

[78] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. *2nd Symposium on Computational Geometry (SoCG)*, pages 14–23, 1986.

[79] R. van Oostrum and R. C. Veltkamp. Parametric search made practical. *18th Symposium on Computational Geometry (SoCG)*, pages 1–9, 2002.

[80] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. *18th Conference on Scientific and Statistical Database Management (SSDBM)*, pages 379–388, 2006.

[81] J. Xu, V. Duindam, R. Alterovitz, and K. Goldberg. Motion planning for steerable needles in 3d environments with obstacles using rapidly-exploring random trees and backchaining. *IEEE Conference on Automation Science and Engineering (CASE)*, 2008.

VITA

Atlas F. Cook IV received his undergraduate training in Computer Science with a minor in Mathematics from the University of Texas-Pan American. He received his Bachelor of Science Summa Cum Laude in the summer of 2005 with 136 hours and a 4.0 out of 4.0 GPA. Upon graduating, Atlas immediately began the Computer Science Ph.D. program at the University of Texas at San Antonio in the Fall 2005 semester. He chose Carola Wenk to be his advisor and quickly discovered that she is the best advisor in the galaxy. During his first four semesters as a Ph.D. student, Atlas was a teaching assistant. During subsequent semesters, Atlas was a research assistant.

Atlas enjoys working with obstacle-avoiding similarity metrics and needle paths and was awarded the UTSA Presidential Dissertation Fellowship in March 2009. His most recent paper was awarded the *WADS 2009 Best Paper Award*. Atlas enjoys choral singing, playing the piano, lifting weights, and jogging. His intent is to become a professor and to inspire the next generation to "boldly go where no one has gone before."