BOURNEMOUTH UNIVERSITY

TRANSFER DOCUMENT

Efficient Geodesic Computation Using Spatial Conformity of Geodesic Paths

Author:
Yipeng QIN

Supervisor:
Dr. Hongchuan YU
Prof. Jianjun ZHANG

A Transfer Report to Doctor of Philosophy

National Centre for Computer Animation
School of Media

February 9, 2016

Abstract

The shortest path problem is a classic problem is graph theory. Its generalized version on 3D polyhedral surfaces is also known as the *Discrete Geodesic Problem*. This problem is usually studied theoretically from two areas, the differential geometry and the computational geometry. In addition, computing geodesic distances and geodesic paths are indispensable in many applications of computer graphics, including construction of Voronoi diagrams, remeshing, skeleton extraction, water wave animation, mesh segmentation, cloth simulation, point pattern analysis, etc. Therefore, designing an efficient algorithm for geodesic computations is important both theoretically and practically.

In this research, we focus on solving the *single source shortest path* (SSSP) problem on 3D polyhedral surfaces as most of the recent work did so. So far, two kinds of notable exact geodesic algorithms, i.e. the Mitchell-Mount-Papadimitriou (MMP) algorithm and the Chen-Han (CH) algorithm, have been proposed. Both of these two kinds of algorithm are based on individual window propagation framework that propagates one window across a face each time. Although these algorithms can compute exact geodesic distances, their computation costs are very high, which limits their applications for large-scale models and/or time critical applications.

To deal with this challenge and speed up the exact geodesic computations, we propose two novel strategies of redundancy reduction from the property of geodesic paths that they are not allowed to cross each other. Accompanying the two strategies, we proposed an edge-based propagation framework based on the spatial conformity of geodesic paths in the implementation which propagates all windows on one edge simultaneously. We evaluated our proposed algorithm on a wide range of large-scale real-world models and also compared their performance against existing algorithms. Experimental results show that our algorithm (1) reduces approximately 30% windows compared to MMP and 60% to ICH algorithms; (2) consumes space comparable to FWP-CH and ICH, which is considerably less than that of MMP and FWP-MMP algorithms; (3) runs 5-9 times faster than MMP and ICH algorithms and 2-4 times faster than FWP-MMP and FWP-CH algorithms, which is the fastest exact geodesic algorithm to date. This part has been finished in January, 2016.

This accomplished part of the research leads to a few future works as well, such as, improving the performance of current algorithm by adopting new redundancy reduction strategies, designing an approximate version of current algorithm which runs faster under bounded error, generalizing it to volume data and apply it to neuron tracing tasks. These future work will be carried out later before graduation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The shortest path problem is a classic problem in graph theory, which aims at finding a shortest path between two vertices of a graph. The most notable shortest path algorithm so far is the Dijkstra's algorithm [1] which solves the problem by dynamic programming. Since many real-world problems can be represented with graphs, shortest path algorithms are widely applied in various areas, for example, route planning and navigation, geographic information systems (GISs), computer games, sever selection. A detailed survey of graph-based shortest path algorithms can be found in [2]. However, demands of finding shortest paths are not limited to graphs.

In the 1970s and 1980s, applications in robotics and autonomous navigation motivated the study of optimal collision-free path planning algorithms in both 2-dimensional (2D) and 3-dimensional (3D) spaces [3] [4] [5] [6]. For the 2D-shortest path problem with polyhedral obstacles, a common method is to build a structure named *visibility graph* [3] [6] and compute shortest paths by searching it using Dijkstra's algorithm [1]. The overall time complexity of this method is $O(n^2 \log n)$. However, the 3D-shortest path problem with polyhedral obstacles is much more difficult, which is proved to be NP-hard by Canny and Reif [7] in 1987. Then, an important special case of the 3D-shortest path problem, i.e. find a shortest path between two points on a polyhedral surface S in 3D space, attracts research interests from computational geometry since it is easier to solve. This kind of shortest path is also known as the *geodesic path* and its length is called the *geodesic distance*.

There are three major kinds of *geodesic* problems classified by the number of sources and destinations: 1) finding the geodesic path between one source vertex and one destination vertex; 2) finding the geodesic paths from one source to all destination vertices, which is known as the *single source shortest path problem* (SSSP); 3) finding the geodesic paths between all pairs of vertices, which is known as the *all-pairs shortest path problem* (ASAP). Among the three problems, the SSSP problem is usually of more importance and attracts more research interests since it is usually the basis of solving the other two problems. Therefore, this research mainly focuses on solving the SSSP problem.

In general, the geodesic problems are studied in two areas: the differential geometry and the computational geometry. Therefore, a brief overview of finding geodesics from perspectives of the two areas is presented respectively as follows.

1.1.1 Geodesics in Differential Geometry

Kline [8] presented a review of the development of differential geometry as a chapter in his book in 1990. Originally, the theory of surfaces is founded by Euler and extended by Monge. Then, immense amount of work on this subject is made by Gauss, who started working on geodesy and map-making in 1816. In 1827, Gauss proposed his definitive paper titled "General Investigations of Curved Surfaces", in which Gauss solves the problem of finding geodesics on surfaces by calculus of variations. Note that the term *geodesic* is introduced by Liouville in 1850 from the word *geodesy*. Gauss also proposed the idea that a surface is a space in itself. This idea was generalized by Riemann and a new area in non-Euclidean geometry is opened up. Therefore, It can be noted that geodesics play an fundamental role in the theory of differential geometry.

Intuitively, a *geodesic* is a generalized notion of *straight lines* from planes to curved spaces, which is locally shortest. A curve is a geodesic only when its geodesic curvature is zero [9]. Therefore, geodesics on smooth surfaces can be computed by solving a ordinary differential equation (ODE) related to the geodesic curvature.

However, computers have a discrete nature that the data processed by computers are usually discretized in advance. Therefore, a smooth surface is usually discretized and represented by a polyhedral surface for computers to process in practice. As a result, since most points on polyhedral surfaces have zero curvature, concepts and methods from differential geometry cannot be applied directly and they should be discretized as well. For example, Polthier and Schmies [9] have proposed a discrete version of geodesic curvature and used Runge-Kutta method to solve the ODE associated with it. In addition, from observing the relations between geodesic distances and wave/heat, some other methods from differential geometry are proposed. For example, the Fast Marching Method (FMM) [10] based on solving the discrete Eikonal equation and the Heat Method [11] based on solving the Varadhan's formula and the discrete Poisson equation.

Note that the discretisation processes cannot avoid introducing errors and therefore algorithms using this approach cannot give an exact solution. Besides, although *globally shortest* is not a property that all geodesics satisfy in differential geometry, it is usually preferred since it is consistent with the demands for shortest paths and various real-world applications depend on it.

1.1.2 Geodesics in Computational Geometry

In computational geometry, surfaces are assumed to be polyhedral which are usually represented by triangle meshes. Since the geodesic problem in computational geometry is viewed as a special case of the 3D shortest path problem which is proven to be NP-hard [7], the first concern is that whether it is possible to be solved in polynomial time.

Pioneer work addressing this issue is the $O(n^3 \log n)$ algorithm proposed by Sharir and Schorr (SS) [6] that computes geodesics on convex polyhedra in 1986. Although their algorithm only operates on convex polyhedra, the observation that geodesic paths on 3D polyhedral surfaces can be solved in 2D planar space by a process named *planar unfolding* laid the foundation of many following works. Then, the SS algorithm is improved by Mount [5] to achieve a $O(n^2 \log n)$ time complexity and extended by O'Rourke et al. [4] to operate on possibly non-convex polyhedra in $O(n^5)$ time. Although O'Rourke et al.'s algorithm has a high time complexity of $O(n^5)$, their work demonstrates that the geodesic problem on 3D polyhedral surfaces can be solved in polynomial time. Then, the major concern of solving the geodesic problem moves to improving its asymptotic complexity.

In 1987, Mitchell et al. (MMP) [12] defined finding geodesics on polyhedral surfaces as the *Discrete Geodesic Problem* (DGP) and proposed an algorithm to solve it in $O(n^2 \log n)$ time and $O(n^2)$ space. In 1990, Chen and Han (CH) [13] proposed their algorithm to improve the time complexity and space complexity of solving DGP to $O(n^2)$ and $O(n)$ respectively, which are the best asymptotic complexities up-to-date (note that Kapoor [14] has proposed an $O(n \log^2 n)$ algorithm in 1999 but not widely accepted by the academia since the details of their algorithm is formidable [15]). Theoretically, this upper bound $O(n^2)$ is difficult to break. Practically, Surazhsky et al. [16] implemented Mitchell et al.'s algorithm [12] in 2005 and observed that the practical time complexity of the algorithm is $O(n^{1.5} \log n)$, which is much smaller than the worst-case analysis and runs much faster than the $O(n^2)$ algorithm proposed by Chen and Han [13]. Since then, the major concern moves to improve the practical performance of geodesic algorithms.

In 2009, Xin and Wang [17] observed that the CH algorithm has a lot of redundancy and improved it by employing a novel filtering rule and a priority queue that organizes events for processing. Although their method improved the time complexity of the CH algorithm to $O(n^2 \log n)$, the practical running time of their improved-CH (ICH) algorithm runs significantly faster than the CH algorithm. In 2013, Liu [18] observed the redundancy in the half-edge based structures in the implementation of MMP algorithm and improved it by using an edge-based structure instead. As a result, the running time of MMP algorithm is improved by approximately 44%. In 2014, Ying et al. [19] proposed an parallel implementation of the ICH algorithm that runs approximately an order of magnitude faster than the serial ICH algorithm. In 2015, observing that the priority queue consumes more than 70% time in MMP and ICH algorithms, Xu et al. [20] proposed to replace the priority queue with a bucket structure and achieves a speed-up factor of 3-10.

Apart from solving DGP exactly, many approximation algorithms are also proposed. These algorithms can be classified into two major classes. The first class of algorithms [21] [22] [23] [24] [25] only operates on convex polyhedra. Let S be a convex polyhedral surface, their main idea is to first approximate S by a larger convex surface S' which has much less vertices. Then, geodesic paths are calculated on S' and mapped from S' to S. The second class of algorithms [26] [27] [28] [29] [30] augment the graph of a polyhedral surface S by adding extra points and/or edges and solve the geodesic path problem by applying graph-based shortest path algorithms (e.g. Dijkstra's algorithm [1]) on the augmented graph.

The above mentioned studies all uses Euclidean distances as the cost metric on the surface. If using weighted cost metric, i.e. different weights are assigned to faces of the surfaces, the geodesic problems are quite different and addressed in some other works [31] [32] [33] [34] [35] [36] [37] [38] [39] [40].

1.2 Motivation

The motivation of this research comes from two sides. First, as above mentioned, the geodesic problem is an important problem in differential geometry and computational geometry. Therefore, analysis and improvement made to geodesic calculations will naturally push forward the research frontier of geodesic problems in the two areas. Second, geodesic distances and paths are widely used in computer graphics, such as, construction of Voronoi diagrams [41] [42], remeshing [42] [43], skeleton extraction [42] [44], water wave animation [45], mesh segmentation [46], cloth simulation [47] [48], point pattern analysis [42], etc. In some of these applications, the computation of geodesic distances and paths is the most time consuming part. For example, Liu et al. [42] proposed to construct iso-contours, bisectors and Voronoi diagrams based on an exact geodesic metric and use them to implement three applications, including remeshing, skeleton extraction and point pattern analysis. In all these three applications, the bottle-neck of their time complexities is the time consumed by performing the MMP algorithm to calculate geodesic distances, which is $O(n^2 \log n)$. Therefore, improving the performance of geodesic calculations on polyhedral surfaces will naturally improve the performance of these applications.

1.3 Contribution of Current Work

The two most notable exact algorithms in solving the SSSP geodesic problem are the MMP algorithm and the CH algorithm, along with their variants. The state-of-the-art implementations of both these two classes of algorithms adopt a structure named *window* to encode the geodesic distance information and calculate geodesic distance fields on polyhedral surfaces through propagating windows outwards from the source point. Besides, these implementations use a priority queue to organize the windows with respect to their distance to source vertex and always propagate the closest one each time. Therefore, we name this framework they follow as the *window-based* propagation framework.

In this research, we have proposed two novel strategies of redundancy reduction from the property of geodesic paths that they are not allowed to cross each other. Accompanying the two strategies, we have proposed a novel *edge-based* propagation framework to solve the SSSP geodesic problem. The major differences between this framework and the window-based propagation framework is that it introduces the spatial-conformity of geodesic paths to organize windows on edges. The spatial order of windows guarantees that paths in neighbouring windows are also next to each other on the polyhedral surface, which leads to three major advantages. First, geodesic crossing scenarios can be found and eliminated at early stages, which avoid useless window propagations. Second, spatial order of windows guarantees that geodesic crossing scenarios should first happen between successive windows, which simplifies the identifying operations (implemented by binary search in MMP-class algorithms) and reduces the cost of reducing useless windows. Third, since the propagation is no longer dependent on the distance of windows but requires the spatial conformity of windows on edges to be preserved, we propose to apply a distance priority queue of end vertices of edges to organize the propagation. Therefore, windows on the same edges are batched together for propagation in our algorithm, which dramatically decreases the time cost of priority queue maintenance.

In summary, we have proposed a new propagation framework based on edges to calculate geodesics on polyhedral surfaces and our algorithm outperforms all existing exact geodesic algorithms in the following aspects:

- With the novel windows reducing strategy proposed, useless window propagations are effectively reduced compared to existing window-based propagation methods.
- With the spatial order of windows on edges, windows can be trimmed in a sequential way, which eliminates the binary search part of MMP-class algorithms.
- The priority queue used in our algorithm is applied to vertices instead windows, which naturally and dramatically decreases the cost of maintaining the priority queue.

As shown by our experiments, our algorithm runs 5-9 times faster than MMP and ICH algorithms and 2-4 times faster than FWP-MMP and FWP-CH algorithms [20]. To our knowledge, it is the fastest exact geodesic algorithm to date.

1.4 Document Structure

The organization of the document is as follows:

- Chapter 1: Introduction. This chapter provides the background, the motivation and the contribution made in current research.
- Chapter 2: Literature Review. This chapter classifies and reviews related previous works in a systematic way.
- Chapter 3: Methodology and Prelimiaries. This chapter explains the methodology selected and defines some basic concepts used in the research.
- Chapter 4: Key Findings. This chapter introduces two key findings in this research, which are the basis of the proposed algorithm.
- Chapter 5: Proposed Algorithm. This chapter describes the proposed algorithm in details.
- Chapter 6: Conclusions and Future Work. This chapter concludes the progress up-to-date and discusses possible future work.

Chapter 2

Literature Review

This chapter reviews notable geodesic algorithms classified into two categories, algorithms from computational geometry and algorithms from differential geometry, which are reviewed in section 2.1 and section 2.2 respectively. The structure of this chapter is shown in Figure 2.1.

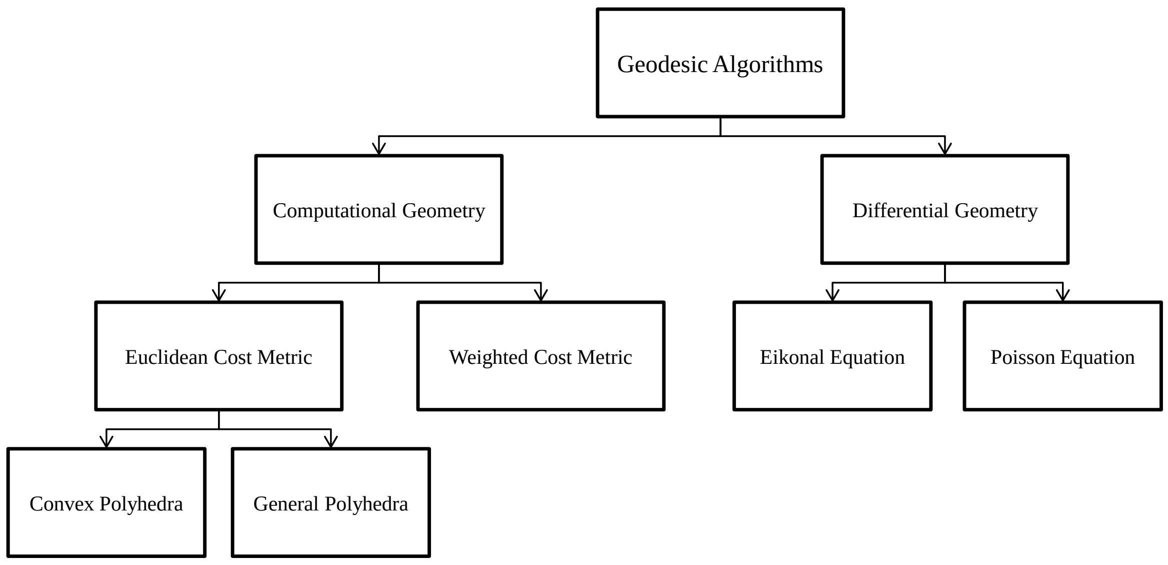


FIGURE 2.1: The Structure of Literature Review.

Without loss of generality, we denote the polyhedral surface on which the geodesics will be calculated as S, the source as s and the destination as t, the complexity of S as n (n denotes the number of vertices/edges/faces of S since they are approximately proportion to each other according to Euler's polyhedron formula).

2.1 Geodesic Algorithms From Computational Geometry

This section reviews geodesic algorithms based on computational geometry. In general, the cost metric of surface S is arbitrary, which results in different features of geodesic paths on S. In section 2.1.1, geodesic algorithms under an important special cost metric, the *Euclidean cost metric*, are reviewed. In section 2.1.2, geodesic algorithms under a more general cost metric, the *Weighted cost metric*, are reviewed.

2.1.1 Euclidean cost metric

Using Euclidean distances as a cost metric, each face f of S is located on a plane embedded in 3D space and the shortest path between two points on f is a straight line segment. Therefore, shortest paths on S are polylines denoted as $p = (A_1, A_2, A_3, ..., A_n)$, where $A_i(1 \leq i \leq n)$ are vertices of the polyline. It should be noted that despite the first vertex A_1 and the last vertex A_n, the inner vertices $A_i(2 \leq i \leq n-1)$ are located on: (1) edges of S for convex polyhedra [6]; (2) vertices or edges of S for general (non-convex) polyhedra [12].

We will first review geodesic algorithms designed for convex polyhedra and then those designed for general polyhedra.

2.1.1.1 Geodesic Algorithms on Convex Polyhedra

For a convex polyhedral surface S, all the vertices of S are sphere vertices, i.e. there doesn't exist a vertex with a more than 2π round angle on S.

Sharir and Schorr's (SS) Algorithm. In 1986, Sharir and Schorr [6] first proposed an efficient algorithm for geodesic computation on convex polyhedral surface S in $O(n^3 \log n)$ time. Their key idea is to subdivide S into $O(n)$ disjoint "peels" in the pre-processing stage and then unfold each "peel" in a common plane so that the geodesic computation can be operated in 2D planar space. The subdivision in SS algorithm relies on the *ridge* theory. A point p on S is defined as a *ridge* point if there exists more than one shortest paths from source point s to p. They proved that: (1) a shortest path on S cannot pass through vertices or ridge points of S; (2) ridge points form finite many straight line segments; (3) the set T of vertices and ridge points of S is closed, connected and contains no closed path. Thus, T is a tree with vertices of S as leaves and S can be subdivided into $O(n)$ peels with T as boundary. Algorithmically, the construction of T, namely *slice tree*, in SS algorithm follows the idea of Dijkstra's algorithm [1] that organise events in a priority queue and process them in a near-to-far manner. Denote a slice (peel) as $\sigma = (\theta_1, \theta_2)$ in a polar coordinate system with source point s as origin. In the initialisation process, slices adjacent to source point s are created and inserted into the priority queue. In the main loop, the slice σ on edge e closest to source point is extracted from the priority queue and trimmed with other slices on e. Then, the child slice of σ is inserted into the priority queue. The loop stops when the priority queue is empty. This construction procedure costs $O(n^3 \log n)$ time and $O(n^2)$ space to store the slice tree T. Then, S can be subdivided into $O(n)$ peels and the shortest path queries between any destination point t on S and source point s can be determined in $O(n)$ time. In general, the limitation of SS algorithm include: 1) only on convex polyhedra that ridge points form straight line segments, which makes it difficult to extend SS algorithm to general polyhedra; 2) SS algorithm solves the geodesic computation problem in a indirect way which requires a $O(n^3 \log n)$ pre-processing procedure that is time-consuming and difficult to implement. However, its idea of using *planar unfolding* to convert the complex 3D geodesic computation to 2D planar space laid the foundation of various follow-up works.

Mount's Algorithm. From further investigating the ridge theory of SS algorithm, Mount [5] observed that the constructed peels are the Voronoi diagram of a set of points V containing the unfolded images of source point s. Since the points in V and the peels are in 1-1 correspondence, the size of V in a face is also bounded by $O(n)$. Based on this observation, Mount's algorithm avoids explicitly generating ridges but represents peels using a set of points. Therefore, the complex ridge constructions are replaced by simple set operations. In this way, Mount's algorithm improved the time complexity of pre-processing stage in the SS algorithm to $O(n^2 \log n)$. Besides, Mount observed that with the unfolded images of source, the shortest path query of a point in a face is equivalent to the 2D point location problem, which can be solved in $O(k + \log n)$ time with standard algorithms, where k is the number of faces passed through by the shortest path. Similarly, the answering time for distance queries can be reduced to $O(\log n)$. Addressing the $O(n^2)$ storage of the Voronoi diagram data structure, Mount shows that it can be reduced to $O(n \log n)$ with the same query time since the redundancy of storing $O(n)$ different but similar lists with $O(n)$ elements each can be reduced efficiently. However, this technique is applied after the construction of Voronoi diagram and thus the peak space cost of the algorithm is still $O(n^2)$.

In 1987, in order to reduce the space cost of building the Voronoi diagram, Mount [49] proposed to associate each edge e of S with a tree structure which encodes the geodesics crossing e in the leaves in order. By sharing common sub-trees, the overall space cost of Mount's algorithm is reduced to $O(n \log n)$.

Hershberger and Suri's Algorithm. For practical applications, Hershberger and Suri [21] proposed an algorithm with time complexity of $O(n)$ and worst-case bounded error of 2 to compute the approximate shortest path between two points on S. By approximating a shortest path tree, their algorithm can be extended to solve the SSSP problem under the same error bound in $O(n \log n)$ time. Inspired by the concept of *bounding box*, which is usually viewed as a first-order approximation of S and is independent to n (i.e. the complexity of S), they propose to use a structure named *wedge* to approximate the region where the shortest path passed through on S. Note the wedges always contain S here from convexity. In their algorithm, two kinds of wedges, the *2-plane wedge* and the *3-plane wedge*, are constructed under different conditions to reduce approximation error. Suppose faces f_s and f_t contain source s and destination t respectively. Let H_s and H_t be the planes defined by f_s and f_t. If the dihedral angle $\alpha(H_s, H_t) \geq \pi/3$, the wedge is constructed as a 2-plane wedge $W(H_s, H_t)$; otherwise, multiple 3-plane wedges will be constructed by combining *horizon plane* $H_i (1 \leq i \leq k, k \leq n)$ with H_s and H_t as $W(H_s, H_i, H_t)$. The horizon plane here satisfies $\alpha(H_s, H_i) = \alpha(H_i, H_t)$. With required wedges constructed, their algorithm first computes a shortest path on the wedge W and then maps the path onto S. They proved that the mapped path is no longer than the corresponding shortest path on the wedge.

Har-Peled et al.'s Algorithm. Although Hershberger and Suri's Algorithm is simple and achieves linear time complexity, its error bound of 2 is high. Based on their algorithm, Har-Peled et al. [22] proposed an $(1+\epsilon)$-approximation algorithm which constructs a path between two points on S at most $(1+\epsilon)$ times longer than the shortest one. The main idea of their method is: (1) construct a closer approximate polyhedral surface G to S with a grid lattice rather than the wedges used in [21]; (2) construct the shortest path on G; (3) map the shortest path from G to S without increasing its distance. In order to construct G, they first expands S to S' by a factor r calculated from ϵ and δ, where δ is an initial approximate distance between the two points bounded by factor 2 (see [21]). Here, S' contains S and G is constructed between the boundaries of S' and S that $S \subseteq G \subseteq S'$. Therefore, the approximated paths on G cannot be smaller than the shortest ones on S. However, the flexibility of error bound results in that the time complexity of their algorithm depends on both n and ϵ as $O(n \cdot \min\{1/\epsilon^{1.5}, \log n\} + 1/\epsilon^{4.5} log(1/\epsilon))$. In addition, their algorithm can be extended to solve the SSSP problem in $O(n/\epsilon^{4.5}(\log n + \log 1/\epsilon))$ time.

Agarwal et al. [23] improved Har-Peled et al.'s algorithm by constructing the approximated polyhedral surface Q between S and S' using the approximation scheme proposed by Dudley [50]. Their algorithm achieves a time complexity of $O(n \log(1/\epsilon) + 1/\epsilon^3)$ for path construction between two points on S and $O(n/\epsilon^3 + (n/\epsilon^{1.5}) \log n)$ for solving SSSP problem.

Har-Peled [24] further improved the algorithm by designing an alternative $O(n)$ approximation algorithm to [21] with a constant error bound factor of 8 and modified the corresponding part of Agarwal et al.'s algorithm [23]. Therefore, their algorithm pre-processes S in $O(n)$ time and achieves a time complexity of $O(\log n/\epsilon^{1.5} + 1/\epsilon^3)$ for path construction between two points on S and $O(n(1 + \log n/\epsilon^{1.5} + 1/\epsilon^3))$ for solving SSSP problem.

Agarwal et al.'s Algorithm. Agarwal et al. [25] proposed and implemented a $(1+\epsilon)$-approximation algorithm from graph-based approach. Their algorithm: (1) constructs a graph G in the vicinity of S in $O(n/\sqrt{\epsilon})$ time with $O(1/\epsilon^4)$ space, that makes the size of G independent of S; (2) a shortest path on the graph G is constructed and mapped onto S in $O(n/\epsilon)$ time; (3) a heuristic method is applied to improve the quality of the resulting path in the post-processing stage. Roughly speaking, their algorithm calculates geodesic from graph-based approach at global level and refines the path from geometric approach at local level.

Schreiber and Sharir's Algorithm. Schreiber and Sharir [51] presented an optimal-time exact algorithm to solve the SSSP geodesic algorithm based on Hershberger and Suri's algorithm [52] which calculates shortest paths on 2D planar space. Their algorithm computes an implicit representation of shortest paths on S which runs in $O(n \log n)$ time and requires $O(n \log n)$ space. Their algorithm generalize the *conforming subdivision* on 2D planes in [52] to *conforming surface subdivision* on 3D polyhedral surfaces using and oct-tree-like structure. Then, "transparent" edges are formed which allow propagating "wavefront" from the source s in discrete steps. After propagation, the shortest path queries can be reported in $O(\log n + k)$ time, where k is the number of faces passed through by the shortest path.

Schreiber [53] further extended this algorithm to three kinds of *realistic polyhedra* which can be non-convex: (1) *terrain* polyhedron, whose maximum facet slope is bounded by a fixed constant; (2) *uncrowded* polyhedron, that for any axis-parallel square with side length l and has a distance more than l from any vertex of S, intersects with at most $O(1)$ faces of S; (3) *self-conforming* polyhedron, that for each edge e of S, the number of faces within the region with distance less then $O(|e|)$ from e is bounded by a constant.

2.1.1.2 Geodesic Algorithms on General Polyhedra

For a general (non-convex) polyhedral surface S, the vertices of S may include saddle vertices, i.e. vertices with more than 2π round angles. As a result, geodesic paths on S are more complicated since they may pass through the saddle vertices.

O'Rourke et al.'s Algorithm. In 1985, O'Rourke et al. [4] first extended Sharir and Schorr's algorithm [6] which computes shortest paths on convex polyhedra to general polyhedra. The only requirement of their algorithm is that S must be orientable. In their algorithm, they treat the source and destination as vertices of S and extend the peels construction method in [6] to non-convex surfaces. The shortest path is then found between vertices of S. The main concern of their algorithm is to show that the problem can be solved in polynomial time. Therefore, their algorithm has a high time complexity as $O(n^5)$ and has not been implemented so far.

Mitchell et al.'s (MMP) Algorithm. In 1987, Mitchell et al. [12] first define the shortest path problem on general polyhedral surfaces as the *Discrete Geodesic Problem* and proposed a practical algorithm for solving SSSP problem in $O(n^2 \log n)$ time and $O(n^2)$ space. Their algorithm is based on the the *continuous Dijkstra* strategy of [5] and the *planar unfolding* idea of [6]. Similar to all algorithms using Dijkstra's paradigm, events in the algorithm is organized in a priority queue and the nearest one is processed each time. However, since shortest paths on non-convex polyhedra may pass through saddle vertices, the concept of planar unfolding is generalized to include these vertices as *pseudo-sources* and the faces containing the shortest path between each successive pair of *pseudo-sources* are unfolded to a plane as in [6]. For an edge e of S, there may be multiple unfolded planes from source s to e and each unfolded plane is associated with a set of paths from s named *interval*. In order to efficiently minimize redundancy among intervals on e, Mitchell et al. suggested to trim the intervals into disjoint parts and organized them in a ordered list according to their positions on e. After the algorithm terminates, each edge of S is subdivided into a list of end-to-end connected intervals and it can be observed that this subdivision forms the peels in [6]. They proved that the number of intervals on an edge is $O(n)$ and therefore the total number of intervals of S is $O(n^2)$, where n is the total number of edges of S. From that their algorithm applies a priority queue to organize the intervals and that a binary search process is introduce when locating an interval in the a list on an edge, processing each interval costs $O(\log n)$ time and the overall time complexity of MMP algorithm is $O(n^2 \log n)$. Similar to [5], the geodesic path queries can be reported in $O(k + \log n)$ time, where k is the number of faces passed through by the shortest path.

In 2005, Surazhsky et al. [16] implemented the MMP algorithm and tested its performance on various models. They used a data structure named *window* to represent the *interval* in [12] and implemented the trimming procedure between windows by solving a quadratic equation. Their experimental results show that MMP algorithm's window complexity is approximately $O(n^{1.5})$ and it runs in sub-quadratic time in practice, which is much faster than the worst case analysis. In addition, they modify MMP algorithm by introducing a merging procedure which merges two selected windows into one under bounded error and proposed a $(1 + \epsilon)$-approximation algorithm. Through experiments, they show that their approximation algorithm runs comparable to Fast Marching Method (FMM) [10] (will be reviewed later) in $O(n \log n)$ time in practice and produces much smaller error then FMM.

Since then, several incremental works are done to improve the performance of the MMP algorithm. In 2007, Liu et al. [54] observed that degenerate cases induced by floating point calculation occur frequently when applying the MMP algorithm to real world models. Therefore, they systematically analysed all the degenerated cases and handled them accordingly, which makes the MMP algorithm more robust. In 2013, Liu [18] observed that the implementations of MMP algorithm using half-edge structures [16] [54] induce redundancy of windows between each pair of opposite half-edges corresponding to an edge. Addressing this issue, Liu proposed to use one edge-based structure for each edge instead of two half-edges to reduce redundancy in the MMP algorithm. Experimental results show that this method reduces the running time of the half-edge based implementation of MMP algorithm by 44% and storage by 29% on average. In 2015, Xu et al. [20] observed that maintaining priority queue accounts for approximately 70% of running time for the MMP algorithm. Thanks to the fact that the correctness of MMP algorithm does not depend on the order of windows processed, they proposed the *Fast Wavefront Propagation* (FWP) framework which organize windows using a bucket structure and process them in a batched way. Experimental results show that the FWP-MMP algorithm is about 3-10 times faster than the MMP algorithm.

Chen and Han's (CH) Algorithms. In 1990, Chen and Han [13] proposed an algorithm to solve the discrete geodesic problem using a different approach than the continuous Dijkstra paradigm. Their algorithm is based on two observations: (1) *One angle one split*, that for any two windows propagated from an edge $\overline{BC}$ across a triangle ΔABC and cover opposite vertex A, only one of the two windows can have two children; (2) the *sequence tree T* constructed using *First-In-First-Out* (FIFO) queue can be viewed as a dual of the peels structure used in [6]. Chen and Han proved that T has only $O(n)$ leaf nodes for convex polyhedra and at most additional $O(n)$ leaf nodes from pseudo-sources for non-convex polyhedra. From the observation in [6] and [5] that a shortest path cannot pass more than N faces, where N is the total number of faces of S, the maximum level of T is bounded as $O(n)$. Therefore, the CH algorithm builds the sequence tree T in $O(n^2)$ time and $O(n)$ space, which are the best asymptotic complexities for exact geodesic calculation so far. After building T, the geodesic path queries can be reported in $O(k + \log n)$ time, where k is the number of faces passed through by the shortest path.

In 2000, Kaneva and O'Rourke [55] implemented the CH algorithm. Their experimental results verify the $O(n^2)$ time complexity and $O(n)$ space complexity in practice. However, Surazhsky et al. [16] reported that their $O(n^2 \log n)$ implementation of MMP algorithm is many times faster than Kaneva and O'Rourke's $O(n^2)$ implementation of CH algorithm.

In 2009, addressing this abnormal phenomenon, Xin and Wang [17] explored the problem and found that more than 99% of the windows generated in the CH algorithm are useless. Therefore, they proposed an efficient window filtering rule to filter out useless windows of CH algorithm. Besides, they suggested following the continuous Dijkstra paradigm and thus using priority queue to organize the windows as the MMP algorithm does. Although introducing the priority queue increases the time complexity of their improved-Chen and Han's (ICH) algorithm to $O(n^2 \log n)$, the practical running time of the ICH algorithm is considerably shorter than the CH algorithm. They also show that the space consumed by the ICH algorithm is much smaller than that of the MMP algorithm although it is difficult to prove that its space complexity is still $O(n)$. Experimental results show that their method greatly outperforms the CH algorithm and runs comparable as fast as the MMP algorithm while using considerably less space. In 2014, Ying et al. [19] proposed a parallel version of the ICH algorithm (PCH), which runs an order of magnitude faster than the serial ICH algorithm. In their algorithm, they replaced the sequential priority queue in ICH algorithm with a k-selection scheme to select k nearest windows to the source and propagate them in a parallel way. In 2015, Xu et al. [20] also applied the FWP framework to ICH algorithm and achieves a speed-up factor of 2-5. They observed that maintaining priority queue accounts for approximately 70% of running time for the ICH algorithm and proved that the correctness of the ICH algorithm does not depend on the order of windows processed as well.

Har-Peled's Algorithm. Har-Peled [26] generalized his algorithm in [24] which computes $(1+\epsilon)$-approximation geodesic paths to solve SSSP problem on convex polyhedra to general polyhedra. The main idea of the algorithm is to construct a map (i.e. a subdivision of S) of size $O((n/\epsilon)\log(1/\epsilon))$ such that the geodesic distance queries can be reported in $O(\log(n/\epsilon))$ time. The time complexity of constructing the map is $O(n^2\log n+(n/\epsilon)\log(1/\epsilon)\log(n/\epsilon))$ for general polyhedra and $O((n/\epsilon^3)\log 1/\epsilon+(n/\epsilon^{1.5})\log(1/\epsilon)\log n)$ for convex polyhedra.

Kapoor's Algorithm. Kapoor [14] proposed an algorithm to find the shortest paths between pairs of points on general polyhedra and claim that the algorithm runs in $O(n\log^2 n)$ time and consumes $O(n)$ space. His algorithm also uses the continuous Dijkstra paradigm and maintains the wave-front using a sequence of circular arcs. According to O'Rourke [15] and Surazhsky et al. [16], the details of the algorithm is formidable that it calls many other complicated computational geometry algorithms as subroutines. Therefore, this algorithm is not widely accepted by academia and may never been implemented.

Varadarajan and Agarwal's Algorithm. Varadarajan and Agarwal [27] proposed an algorithm to compute an approximate shortest path between two points on general polyhedra to answer the open problem raised by Agarwal's [23] that whether a sub-quadratic time approximation algorithm to solve such problem exists. As a result, two algorithms are proposed: 1) the first one computes a shortest path with error bound factor of $7(1+\epsilon)$ in $O(n^{5/3}\log^{5/3} n)$ time; 2) the second one computes a shortest path with error bound factor of $15(1+\epsilon)$ but is slightly faster that runs in $O(n^{8/5}\log^{8/5} n)$ time. The main idea of their algorithms is to partition S into $O(n/r)$ patches that each patch contains at most r faces, where r is a chosen parameter. Then, for each patch S_i, a set of points are added to its boundary and a graph G_i is constructed to approximate shortest paths between any pairs of points lying on the boundary of S. Finally, graphs of each patch are merged to form a global graph G. Note that the source and destination points must be vertices of G. The shortest path between source s and destination t is then constructed by performing Dijkstra's algorithm on G. However, since the algorithm is involved, its value is mainly theoretical and it has not been implemented so far.

Kanai and Suzuki's Algorithm. Kanai and Suzuki [28] proposed a new algorithm to compute the approximate shortest path between two points on general polyhedra. First, their method use Dijkstra's algorithm on the graph G_0 of S to give an initial path p_0 between the two points. Second, a region R_0 where the shortest path may exist is defined based on p_0. Third, R_0 is refined by adding extra points on its edges and forms a new graph G_1. Then, by replacing G_i by G_{i+1}, Dijkstra's algorithm is iteratively applied to narrow the region R_i and refine the graph G_i to get a closer approximated shortest path p_i on the mesh. The trade-off between accuracy and performance of their algorithm depends on the number of points added to each edge, which is user-defined. They implemented their algorithm and compared its performance to the CH algorithm. The experimental results show that their algorithm can run 100 to 1000 times faster than the CH algorithm when calculating shortest paths with 0.4% error.

Balasubramanian et al.'s Algorithm. Balasubramanian et al. [29] addressed the ASAP geodesic problem and proposed two algorithms to compute the all-pairs *minimal-geodesic* and *shortest-geodesic* distances/paths on polyhedral surfaces respectively. In contrast to the *shortest-geodesics* addressed in most literature, the *minimal-geodesics* defined by the authors are straight lines on the unfolded face sequences that do not pass any vertices except the start vertex and end vertex. Their first algorithm, the line-of-sight (LOS) algorithm, computes minimal-geodesic distances. Their second algorithm, the LOS-Floyd algorithm, computes shortest-geodesic distances by constructing a minimal-geodesic distance graph using the LOS algorithm. Both these two algorithms are based on planar unfolding operations similar to [6] [12] [13]. Although they observed the practical time complexities of their algorithms are no more than $O(n^3)$, the theoretical time complexities of them are exponential.

Ying et al.'s Algorithm. Ying et al. [30] proposed a structure named *Saddle Vertex Graph* (SVG) to encode the geodesic distance information on a polyhedral surface into a sparse graph. Then, each geodesic path on a polyhedral surface is mapped to a shortest path on a graph and can be computed efficiently via Dijkstra's algorithm [1] or other graph-based shortest path algorithms. The performance of their algorithm rely on the geometric features of the surface. The more complicated surface S (i.e. the more saddle vertices contained in S), the better performance their algorithm achieves. Thanks to the fact that real-world models are usually rich in details, their algorithm works well. In practice, the authors only implemented an approximate version of SVG since the exact SVG contains all geodesic information and constructing it is expected to be quite time-consuming. Their implementation uses the MMP/ICH algorithm [16] [17] as the building block to compute exact geodesic distances in a geodesic disk of user-defined size K around each vertex of S. The time complexity of constructing the approximate-SVG is then $O(nK^2 \log K)$. With it, the time complexity of solving SSSP problem on S is $O(Dn \log n)$, where D represents the complexity of the approximate-SVG and $D \ll n$. The experimental results show that if not counting the time of SVG construction, their implementation outperforms existing approximate geodesic methods in both time and accuracy.

2.1.2 Weighted cost metric

For some applications, pieces of the surface S are required to be weighted to model the difficulty of paths passing them. For example, walking on grass lands or mountains usually takes more time than walking on flat roads, which influences path planning tasks of computer games applications. Addressing these demands, the polyhedral surface S is weighted by associating a weight w_i to each of its faces f_i. Thus, for any sub-path that crosses f_i, its length is multiplied by w_i. The length of a path is then calculated by summing up the lengths of its sub-paths. Due to the arbitrary nature of the weighting scheme, it is more difficult to compute geodesic paths on weighted surfaces.

Mitchell and Papadimitriou's Algorithm. In 1991, Mitchell and Papadimitriou [31] defined finding a shortest path between two points on a weighted polygonal surface as the

Weighted Region Problem (WRP) and correspondingly its single-source sub-problem as *Single-Source Weighted Region Problem* (SSWRP). They also proposed an $(1+\epsilon)$-algorithm to solve the SSWRP problem based on the continuous Dijkstra paradigm [12]. The main idea of their algorithm is to apply Snell's Law of Refraction in optics theory to control the direction of paths crossing an edge of S. The reason that Snell's Law works is that the path light traverses is always the one with minimum propagation time (the *Fermat's Principle*). Therefore, the shortest paths from source point is viewed as a ray of light and will "bend" at edges according to Snell's Law to achieve local optimality. Their algorithm runs in $O(ES)$ time and consumes $O(E)$ space, where E is the number of "events" processed in their algorithm and is bounded by $O(n^4)$, S is the time cost of performing a numerical search procedure of finding an $(1+\epsilon)$-approximation path with a given edge sequence and is bounded by $O(n^4 \log \frac{nNW}{\epsilon w})$ (W are w are the maximum and minimum weight assigned to faces of S respectively). Therefore, the overall time complexity of their algorithm is $O(n^8 \log \frac{nNW}{\epsilon w})$ and the overall space complexity is $O(n^4)$. However, the authors expected that the practical performance of the algorithm may be much better.

Mata and Mitchell's Algorithm. Mata and Mitchell [32] proposed an $(1+\epsilon)$-approximation algorithm based on constructing a sparse graph named *pathnet* and finding a shortest path between two points on a weighted polyhedral surface by searching the pathnet. Note that their ϵ is bounded by $O(\frac{W}{wk\theta})$, where W are w are the maximum and minimum weight assigned to faces of S respectively, k is a parameter set by users to balance the accuracy and costs of the algorithm, θ is the minimum interior face angles of S. In their algorithm, the pathnet is constructed by using k evenly-spaced directions to approximate the round angle of each vertex v (e.g. $[0, 2\pi]$ for planar vertices) of S and link v to other vertices or critical points on some edge of S according to the k cones. As a result, the size of the pathnet is $O(kn)$ and the construction time of the pathnet is bounded by $O(kn^3)$. The authors also implemented their algorithm and compared it with some other simple heuristic algorithms. The experimental results show that their algorithm outperforms those algorithms in both error and query time.

Lanthier et al.'s Algorithm. In 1997, Lanthier et al. [33] proposed an algorithm with three variants to compute approximate shortest paths on general polyhedra (a journal version of their work is published in 2001 [34]). The framework of their algorithm is to place extra points on edges of S and link them with extra edges to augment the graph associated with S in which the approximate shortest path is found by Dijkstra's algorithm. The differences among variations of their algorithm is the schemes used to place points on edges. In their paper, three such schemes are proposed: (1) *Fixed Scheme*. This scheme places m points evenly on each edge of S, where m is a positive integer. Then, for each face f_i of S, a graph whose edges connect every pair of points (or vertices) from different edges is computed. Using this scheme, an approximate shortest path with additive error bounded by $W\,|L|$ between two points on S can be computed in $O(n^5)$ time, where W is the maximum weight assigned to faces of S and L is the longest edge of S. (2) *Interval Scheme*. This scheme can be viewed as a improved version of the *Fixed Scheme*. Instead of placing a fixed number, m, of points on each edge, this scheme places points on edges with a fixed interval length, e.g. $|L|\,(m+1)$. As a result, the number of points added is considerably reduced in practice. However, the worst case analysis of this scheme remains the same as the *Fixed Scheme*. (3) *Spanner Scheme*. In order to improve the time complexity of *Fixed Scheme* and *Interval Scheme*, the authors introduces the notion of *spanner* and its computation method in [56] to reduce the number of connections between added points. As a result, the time complexity of the algorithm using *Spanner Scheme* is improved to $O(n^3 \log n)$. However, the approximation error is increased that the length of the approximate shortest path is bounded as $\beta p + W|L|$, $\beta > 1$, where p is the length of the exact shortest path, W is the maximum weight assigned to faces of S and L is the longest edge of S. The authors also implemented and tested their algorithm. The experimental results show that adding a smaller constant number of points (i.e. six) per edge is sufficient to obtain a near-optimal result and the practical running time complexity is reduced to $O(n \log n)$.

In 2003, Lanthier et al. [35] proposed a parallel implementation of their algorithm to speed up the shortest path calculation. Their implementation uses a general spatial indexing storage structure, named *multidimensional fixed partition* (MFP), to achieve load balancing and reduce processor idle time. They also tested their implementation in different environments including, PCs connected using fast ethernet, Beowulf cluster with gigabit interconnect and SunFire (a shared-memory architecture). The experimental results show that the speed-up ratio is acceptable.

Aleksandrov et al.'s Algorithm. In 1998, Aleksandrov et al. [36] proposed a $(1+\epsilon)$-approximation algorithm to compute a shortest path between two points on general polyhedra. The main idea of their algorithm is similar to [33] [34] that extra points are placed on edges of S to generate an augmented graph G and thus the approximate shortest path calculation is performed on G. In their algorithm, they place $m = O(\log_\delta |L|/r)$ points on each edge in a geometric progression manner, where L is the long edge of S, r equals ϵ times the minimum distance from any vertex of S to the boundary of the union of its incident faces, θ is the minimum of all the face angles of S and $\delta \geq 1 + \epsilon \sin\theta$. The time complexity of their algorithm is $O(mn \log mn + nm^2)$, where m is the total number of extra points added.

In 2000, Aleksandrov et al. [37] improved the performance of their algorithm by modifying the point placing scheme and using Snell's Law to prune the search when performing Dijkstra's algorithm. As a result, the time complexity of their new algorithm is reduced to $O(\frac{n}{\epsilon} \log \frac{1}{\epsilon}(\frac{1}{\sqrt{\epsilon}} + \log n))$ while the error bound is still $(1+\epsilon)$.

In 2005, Aleksandrov et al. [38] further improved their previous algorithm by introducing a new discretization scheme. As a result, their algorithm achieves $O(C(P)\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$ time complexity with the same error bound, where $C(P)$ is a constant derived from the geometric features and the weights of faces of S. In their algorithm, extra points are placed on bisectors rather than edges [36] [37] of each triangle of S in a geometric progression way. With this discretization scheme, they proved in Lemma 2.3 of [38] that: (1) for a bisector l of angle α at a vertex v, the number of extra points placed on it is bounded by $\frac{1.61}{\sin\alpha} \log_2 \frac{2|l|}{r(v)} \frac{1}{\sqrt{\epsilon}} \log_2 \frac{2}{\epsilon}$, where $r(v)$ is a parameter representing the weighted radius $r(v)$ for each face incident to v (see Lemma 2.1 in [38]); (2) the size of the constructed graph G is bounded by $C(P)\frac{n}{\sqrt{\epsilon}} \log_2 \frac{2}{\epsilon}$, where $C(P) < 4.83\Gamma \log_2 2L$, L is the maximum of the ratios $\frac{|l(v)|}{r(v)}$ among all vertices $v \in S$, Γ is the average of the reciprocals of the sinuses of angles in S.

In 2009, Aleksandrov et al. [39] presented two query algorithms addressing approximate SSSP problem and ASAP problem on weighted polyhedra respectively. For solving SSSP queries problem, they proposed a single-source query data structure named SSQ based on their previous algorithm [38] which allows answering distance queries in logarithmic time. Therefore, the SSQ data structure can be built in $O(\frac{n}{\sqrt{\epsilon}}\log\frac{n}{\epsilon}\log\frac{1}{\epsilon})$ time and consumes $O(\frac{n}{\sqrt{\epsilon}}\log\frac{1}{\epsilon})$ space as in [38]. For solving ASAP queries problem, they proposed to build an all-pairs query data structure named APQ based on SSQ and a graph separator algorithm which is a generalized version of Lipton and Tarjan's algorithm for partitioning planar graphs [57]. With the APQ structure, the approximate shortest distance queries between arbitrary pair of points on S can be answered in $O(q)$ time that $q \in (\frac{1}{\sqrt{\epsilon}}\log^2\frac{1}{\epsilon}, \frac{(g+1)^{2/3}n^{1/3}}{\sqrt{\epsilon}})$ is a user-defined parameter, where g is the genus of S. The APQ structure is constructed in $O(\frac{(g+1)n^2}{\epsilon^{3/2}q}\log\frac{n}{\epsilon}\log^4\frac{1}{\epsilon})$ time and $O(\frac{(g+1)n^2}{\epsilon^{3/2}q}\log^4\frac{1}{\epsilon})$ space.

Sun and Reif's Algorithm. In 2006, Sun and Reif [40] improved the traditional approach proposed by Lanthier et al. [33] and Aleksandrov et al. [36] [37] by their proposed algorithm named BUSHWHACK. Their algorithm still adopted the scheme used in the previous methods which represented the surface S with an augmented graph G. However, by replacing the geodesic cones used in previous methods with a geometric structure named *interval*, their algorithm computes the approximate shortest path on G more efficiently since the computation is performed only on a sparse sub-graph of G. As a result, their algorithm runs in $O(\frac{n}{\epsilon}(\log\frac{1}{\epsilon}+\log n)\log\frac{1}{\epsilon})$ time. Besides, they also improved the point placing scheme used in [37] and thus the constant in its time complexity is no longer dependent on $\frac{W}{w}$, where W and w are the maximum and minimum weight assigned to faces of S respectively. This feature differs it from all previous methods.

2.2 Geodesic Algorithms From Differential Geometry

This section reviews geodesic algorithms based on differential geometry. Since methods in differential geometry only operate on smooth surfaces, the key issue in these algorithms is how to discretize these methods. In general, most of these algorithms are fast and easy to implement. However, their performances depend heavily on the triangulation and thus they usually have unbounded error.

Two major classes of notable algorithms from differential geometry approaches are proposed up-to-date: 1) algorithms based on solving the discrete version of *Eikonal equation*, $|\nabla T|\,F = 1$; 2) algorithms based on solving the discrete version of *Poisson equation*, $\Delta d = \nabla \cdot X$. In the following sections, we will review the two classes of algorithms respectively.

2.2.1 Algorithms Based on Solving Discrete Eikonal Equation

The Eikonal equation, $|\nabla T|\,F = 1$, is a partial differential equation which shows that the gradient of arrival time is inversely proportional to the speed of the wavefront [58]. If setting the speed of wavefront $F \equiv 1$, then the arrive time of each point on the surface S is equivalent to the Euclidean geodesic distance of that point on S.

Sethian's Algorithm. In 1996, Sethian [58] proposed an algorithm for monotonically advancing fronts on regular grids named *Fast Marching Method* (FMM) that solves the Eikonal equation. The key idea of their algorithm is to advance the front in an "upwind" fashion to compute the arrival time T of each vertex of the grid, where "upwind" denotes the direction of front propagation. The computation of T for each vertex follows the discretized Eikonal Equation,

$$[\max(\max(D_{ij}^{-x}T,0)-\min(D_{ij}^{+x}T,0))^2+\max(\max(D_{ij}^{-y}T,0)-\min(D_{ij}^{+y}T,0))^2]=1/F_{ij}^2 \tag{2.1}$$

In the above equation, the gradient of T is discretized by finite difference method, for example, $D_{ij}^{+x}T=(T_{i+1,j}-T_{i,j})/(\Delta x)$. Geodesic distances of vertices of S can be calculated by setting $F_{ij}\equiv 1$. Similar to Dijkstra's algorithm [1], the front vertices are located in a narrow band and stored in the priority queue and each time the one with minimum T is processed. Therefore, the time complexity of their algorithm is bounded as $O(n\log n)$, where n is the number of vertices of the grid.

Yatziv et al. [59] proposed a variant implementation of FMM algorithm which uses an *untidy queue* (bucket structure) to replace the priority queue used in [58]. Therefore, the time complexity of their implementation is $O(n)$ and they proved that the error bound of their algorithm is of the same magnitude as the original FMM algorithm.

Bertelli et al. [60] generalized the FMM algorithm to solve ASAP problem on regular grids. Observing that more than 90% of the distance calculations are repeated and redundant, they proposed a method to reuse the previous calculated distances, which differs their algorithm from naively run FMM algorithm for n times. The experimental results show that their algorithm reduced the redundancy in distance calculation and improved the accuracy at the same time. However, the theoretical complexity of their algorithm remains $O(n^2\log n)$, which is the same as naively run FMM algorithm for n times.

Kimmel and Sethian's Algorithm. In 1998, Kimmel and Sethian [10] extended the Fast Marching Method [58] to solve SSSP problem on triangular meshes by developing a method to discretize Eikonal equation inside a triangle. In their algorithm, the front vertices are still stored in a priority queue. Each time a vertex with minimum T is removed from the priority queue and processed, the T values of all its neighbours are updated. For acute triangulations, it can be guaranteed that the T values of two vertices is available before updating the third vertex in a triangle. Therefore, the T value of any vertex can be calculated through solving the discrete Eikonal equation in a form of quadratic equation derived in a triangle (see section 4.1 in [10]). However, for general (obtuse) triangulations, there may be only one vertex's T value available in a triangle and therefore extra numerical support is required. In this case, for a vertex v to be updated, the authors propose to locally unfold adjacent triangles opposing the obtuse angle to a plane until another processed vertex p is found. With the unfolding process, a virtual edge is formed between v and p. Then, the T value of v can be updated by solving the discrete Eikonal equation. This numerical support yields that the accuracy of FMM method depends on the triangulation of the surface. The authors proved that the error bound of geodesic distances is $O(\frac{e_{max}}{\pi-\theta_{max}})$, where e_{max} is the length of the longest edge of the triangle mesh, θ_{max} is the largest angle of the triangle mesh. Therefore, the error of FMM algorithm on triangular meshes is unbounded and may yield poor results on bad triangulated meshes. However, it is still the most popular algorithm in practice since it is easy to implement. In addition, the authors also provide a method to construct geodesic paths based on gradient descent method after the geodesic distance calculation.

Since then, several works are done to improve or extend FMM algorithm.

In 2005, Martínez et al. [61] proposed an algorithm based on Polthier's straight geodesics theory [9] to refine the geodesic path generated by FMM algorithm. Their algorithm iteratively refined the path generated by FMM algorithm by using a discrete geodesic flow. The authors proved that the refined path will have a shorter length and their algorithm will finally converge to a local minimum.

In 2007, Xin and Wang [62] proposed an algorithm to improve FMM algorithm's performance on obtuse triangulation and iteratively refine the geodesic paths generated. Addressing the difficulty of dealing with obtuse triangulations in original FMM algorithm, their algorithm classified all edges into seven types to control the wavefront propagation more precisely. This modified FMM algorithm is then used to generate the initial geodesic path p_0. Let F_0 be the initial face sequence passed by the initial geodesic path p_0. In the $(i+1)th$ iteration, the algorithm first calculate the exact shortest path p_{i+1} within the face sequence F_i. If p_{i+1} is the locally optimal path in F_i, the algorithm terminates; otherwise, the algorithm optimizes F_i to generate F_{i+1} and begins the next iteration.

2.2.2 Algorithms Based on Solving Discrete Poisson Equation

Solving the Poisson Equation $\Delta d = \nabla \cdot X$ is equivalent to solving the minimization problem $\int_M |\nabla d - X|$ on triangular mesh M, where d is the distance field and X is a vector field constructed from the input mesh M representing the approximation of d's gradient ∇d. Therefore, the calculation of distance field d is then converted to the calculation of its gradient ∇d.

Xin et al.'s Algorithm. Xin et al. [63] proposed an algorithm to iteratively compute geodesics on broken meshes containing various kinds of defects, such as holes, gaps and short-cuts. Their algorithm takes the Euclidean distances from source vertex s to all other vertices as initial distance field d_0. In the $(i+1)th$ iteration, (1) the gradient field of d_i, ∇d_i, is calculated and normalized ($\|\nabla d_i\| = 1$ from Eikonal equation); (2) the next distance field d_{i+1} is constructed by solving Poisson equation with ∇d_i; (3) d_{i+1} is smoothed to remove the minimal points other than the source point s. Their experimental results show that the distances computed by their algorithm is not sensitive to defects of the mesh but smoother than the geodesic distances. Besides, the convergence of their algorithm is not guaranteed.

Crane et al.'s Algorithm. Crane et al. [11] proposed an algorithm to compute geodesics on triangular meshes using the well-known Varadhan's formula which describes the relationship between heat and geodesic distance between a pair of points x, y on a Riemannian manifold as,

$$d(x,y) = \lim_{t \to 0} \sqrt{-4t \log k_{t,x}(y)} \tag{2.2}$$

where $d(x,y)$ is the geodesic distance between x and y, t is the time, $k_{t,x}(y)$ is the heat kernel. Their algorithm separates the geodesic calculation into three major modules: (1) calculate the heat flow for a fixed time t over the mesh to approximate the gradient field ∇d of geodesic distance field d; (2) normalizes the gradient field ∇d ($\|\nabla d_i\| = 1$ from Eikonal equation); (3) solve the corresponding Poisson equation to recover the geodesic distance field d of the mesh. They also show that their algorithm can be implemented in an efficient way to run in near-linear time by pre-factoring the Laplacian matrix used. Similar to [63], the distances calculated by their method is a smoothed approximation of the geodesic distances. The experimental results show that their method runs faster than the FMM algorithm while produces similar error.

Chapter 3

Methodology and Preliminaries

3.1 Methodology

From the previous literature, it can be noted that there are generally two notable approaches in solving discrete geodesic problems: the computational geometry approach (e.g. the MMP algorithm and the CH algorithm) and the PDE approach (e.g. the Fast Marching Method and the Heat Method).

Since polyhedral surfaces can be viewed as the first order approximation of smooth surfaces, a discretisation process is required when applying PDE methods on solving discrete geodesic problem. However, this discretisation process inevitably introduces errors at some level and therefore cannot produce exact solutions. Therefore, the PDE approach is not adopted in this research. On the other hand, since every triangle patch of a polyhedral surface is on its associated flat plane, well-studied methods in computational geometry can be applied without any compromise. Therefore, the computational geometry approach is adopted in this research.

3.2 Geometric Preliminaries

From computational geometry's point of view, a triangulated surface S in R^3 is a polyhedron defined by its associated vertices, edges and faces. To characterise geodesics on S, some basic terminologies are introduced as follows [12].

3.2.1 Triangle Strip

A triangle strip T is defined as a sequence of faces $f_1, f_2, ..., f_{m+1}$ such that f_i and f_{i+1} are adjacent by sharing common edge e_i (see Figure 3.1). The associated sequence of edges $e_1, e_2, ..., e_m$ is then referred to as edge sequence E and the vertex s in face f_1 opposite to edge e_1 is denoted as the source vertex S of T. Unless specified, the triangle strips mentioned later are all simple that the faces within any triangle strip are unique.

Although triangle strips are much simpler structures than polyhedral surfaces, they are still difficult to be studied since they are three dimensional. Therefore, in order to simplify the studies of triangle strips and utilize the extensive knowledge in 2D computational geometry, a technique named planar unfolding is usually adopted.

A triangle strip $T = (f_1, f_2, ..., f_{m+1})$ is unfolded according to its associated edge sequence $E = (e_1, e_2, ..., e_m)$ in this way: rotate f_1 around e_1 until its plane coincides with that of e_2,

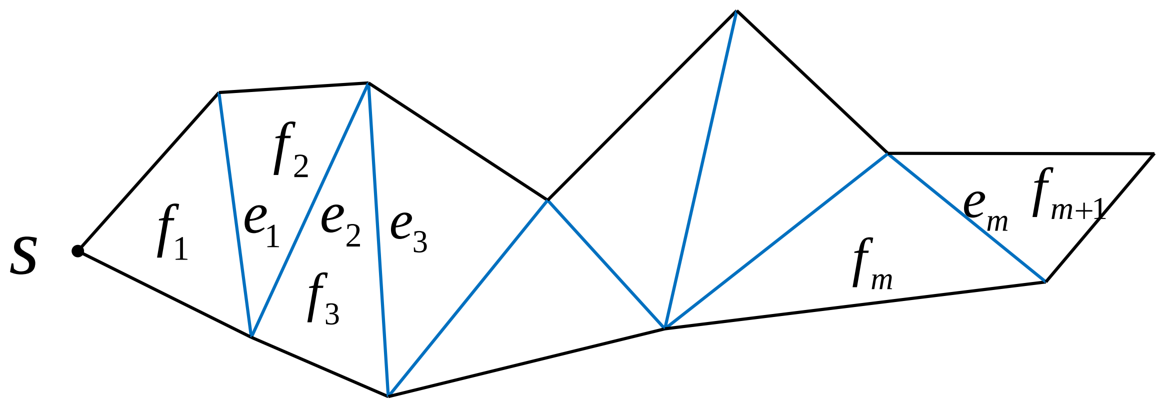


FIGURE 3.1: Illustration of Triangle Strip T and its associated edge sequence E. s denotes the source vertex of T. The blue line segments denotes the edges which form E.

rotate f_1 and f_2 around e_2 until their plane coincides with that of f_3, continue in this way until all faces $f_1, f_2, ..., f_m$ lie in the plane of f_{m+1}. In this way, a triangle strip is mapped from 3D to 2D where shortest path problem is well studied: the shortest path from a point to another one on a plane is a straight line segment connecting them.

With planar unfolding, Mitchell et al. [12] characterise geodesics and optimal paths on polyhedral surfaces as follows:

Lemma 3.1. *The general form of a locally shortest path is a path that goes through an alternating sequence of vertices and edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and both the angles of the path passing through a vertex are greater than or equal to π. A globally shortest path is a geodesic, and it has the additional property that no edge can appear in more than one edge sequence and each edge sequence must be simple.*

The Lemma 3.1 mentioned two special cases of geodesic paths when passing through vertices. The first case is planar vertex, where both the angles of the path passing the vertex equals π; the second case is saddle vertex, where at least one of the angles of the path passing the vertex is larger than π. In geodesic studies, the second case attracts more attention since it produces "invisible" regions on the mesh from source vertex. Geodesic paths of all the points in the "invisible" regions pass through the corresponding saddle vertices. Therefore, saddle vertices are often referred to as pseudo-sources in geodesic studies.

3.2.2 Window Structure

The process of unfolding triangle strips are usually implemented using a data structure named "window" which was first proposed by Mitchell et al. [12]. As Figure 3.2 shows, a window structure is usually represented by a six tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$, in which b_0, b_1 represent the position of the window on the edge e and $b_0, b_1 \in [0, \|e\|]$; d_0, d_1 are the distances from the last pseudo-source vertex p to b_0, b_1 respectively which defines the relative position of p to b_0, b_1; σ is the distance from source vertex S to p; τ records the side of the edge from which the window propagated. Thus, the discrete geodesic problem can be solved through propagating windows across the surface.

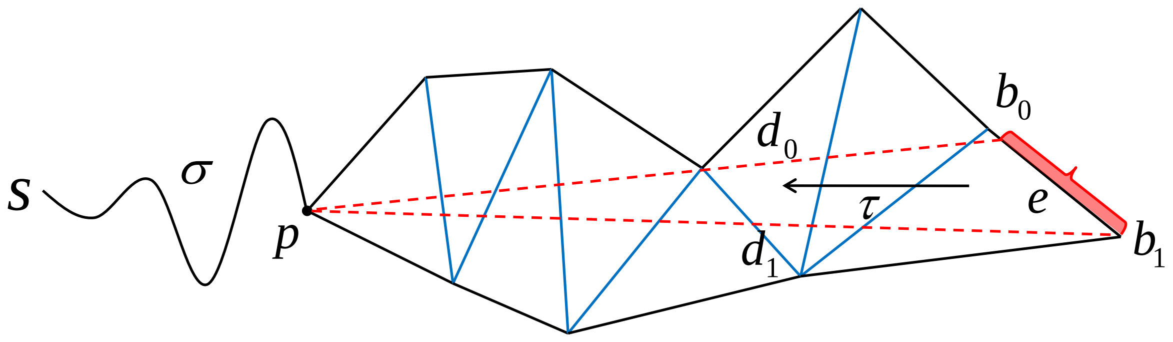


FIGURE 3.2: Illustration of "Window" Structure.

3.2.3 Branching Point

We will introduce a new concept, *branching point*, in this section, which is illustrated in Figure 3.3. Let the source be S. There exist two geodesic paths respectively to the end vertices of the edge $\overline{BC}$. This forms a virtual triangle covering a patch of the surface, ΔSBC. Within this virtual triangle, all the vertices are viewed as the branching points. For each branching point, there exists a triangle strip from the S or some pseudosource S_i, where the geodesic path lies. Consider two successive branching points, which may be not connected by an edge. One branching points shares its triangle strip with another one, that is, their individual geodesic paths share the same source or pseudosource. Extending these two geodesic paths by unfolding the triangle strip, i.e. extending the pencil formed by the geodesic paths, intersects with an edge and forms a window on it. Denote the number of windows on an edge as m, and the number of the covered branching points as n. In general, the windows' number is computed as, $m = n - 1$. However, consider the scenarios of saddle vertex shown in Figure 3.3 (b). It can be noted that each saddle vertex will result in one new window on the edge. Therefore the windows' number is estimated as, $m < 2n$, supposing that all the branching points are regarded as saddle vertices.

Moreover, for arbitrary pair of two successive branching points, each pair may have their individual pseudosources. This may result in window intersection on an edge. It can be noted that the pencils may intersect within a face. The order of the resulting windows on an edge may be different from the spatial order of the branching points.

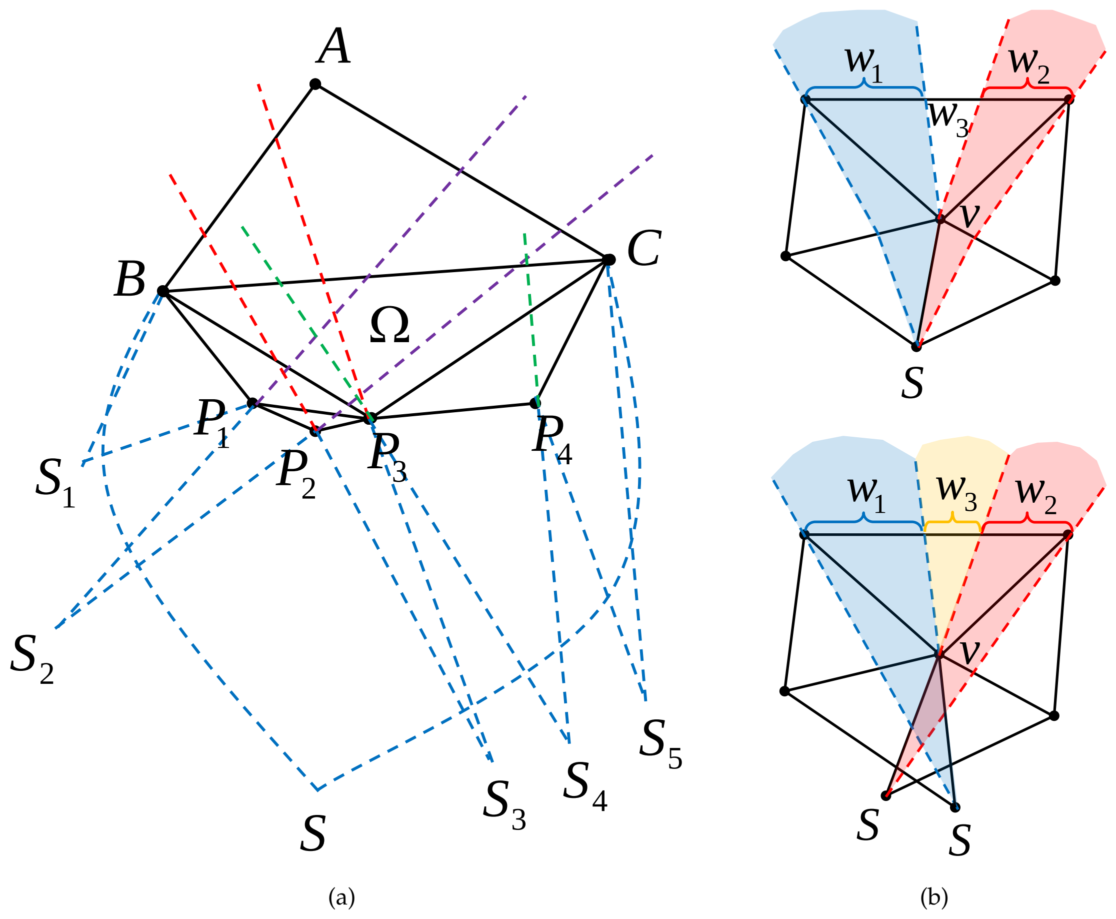


FIGURE 3.3: Illustration of Branching Points. (a) The virtual triangle ΔSBC covers a patch, which contains vertices, $P_1 ... P_4$, i.e. branching points. Assume that the triangles within the region of $(BP_1 ... P_4C)$ are unfolded into a common plane Ω. The shortest paths of two successive branching points can go through the same triangle strip back to the pseudosources, $S_1, ..., S_5$, respectively, and form their individual pencils which intersect with edge $\overline{BC}$. All the resulting pencils share their common plane Ω. The intervals are called as the windows. (b) Due to the saddle vertex v, there is an invisible gap w_3 on the edge (upper). Unfolding the triangles into a plane results in two different "images" of S, i.e. pseudosources (bottom). The saddle vertex v causes one new window w_3 on the edge.

Chapter 4

Key Findings

During this research, two major key findings are concluded as the theoretical basis for the proposed algorithm in the next chapter.

4.1 Windows Cleaning Principle

The property of shortest paths that they are not allowed to cross each other is first proposed by Mitchell et al. in the Lemma 4.3 of their paper [12]. This property reveals that if a path emanating from source vertex cannot provide a shorter distance at a point on a polyhedral surface, its emanation should be stopped. Both the two most notable classes of algorithms, the MMP-class algorithms and the CH-class algorithms, utilize this property on edges of the polyhedral surface to terminate useless window propagations. However, since geodesic paths are not sets of discrete points on edges but continuous line segments across the whole surface, this property can be used more efficiently by involving points in faces. This distinguishes it from MMP-class and CH-class algorithms which only utilize it on edges. In this way, the redundancy of geodesic computation will be further reduced and the performance will be improved.

4.2 Spatial Conformity of Geodesic paths

Both MMP-class algorithms and practical CH-class algorithms (i.e. ICH algorithm and its follow-up variants) adopt the continuous Dijkstra paradigm which propagates shortest distance information outwards in a near-to-far way. Therefore, there is always some kind of "priority queue" in the implementations of these algorithms which organises the propagation of individual windows according to distances. However, this simple mimic have neglected the differences of application scope between classic Dijkstra algorithm and geodesic algorithms, which introduces redundancy to the computation and worsens the performance. Classic Dijkstra algorithm is originally designed for shortest paths problem on graphs which are discrete and contain only edges and vertices. Therefore, the connection of its vertices on graphs are based on 1D edges and the distance updating processes only happen at vertices. However, geodesic algorithms are performed on polyhedral surfaces which are continuous and additionally contain faces. Therefore, the connection of points on surfaces is based on 2D faces. This reveals that when geodesic paths emanate from source point across the surface, the distance updating process may happen on points in faces besides edges and vertices. Recall the finding in previous section, this illustrates that the earliest termination of useless window propagations may also happen in faces. Thus, redundancy may be created if failing to perform this kind of termination. For example, MMP-class algorithms and practical CH-class algorithms always select the window with minimum distance to propagate, which usually results in a random unfolding propagation pattern on the wavefront, i.e. successive propagation processes are usually far away from each other on the polyhedral surface. Therefore, it is difficult for them to terminate useless window propagations in faces, which yields redundancy. In order to minimise the redundancy, our new algorithm utilizes the spatial conformity of geodesic paths and encodes the adjacency information of windows and their corresponding triangle strips throughout the execution. In this way, redundancy reduction can be performed in faces and a better performance can be achieved.

Chapter 5

Proposed Algorithm

5.1 Redundancy Reduction Strategies

5.1.1 Strategy I: Windows Trimming

Figure 3.3 (a) shows the scenarios of a set of windows on an edge. It can be noted that there are two cases, disjoint windows and overlapping windows. An important observation is that the pencils associated with the windows may intersect or even cross each other. The crossing pencils may lead to geodesic intersection, which can be avoided by the Lemma 6.2 of [12] as illustrated in Figure 5.1.

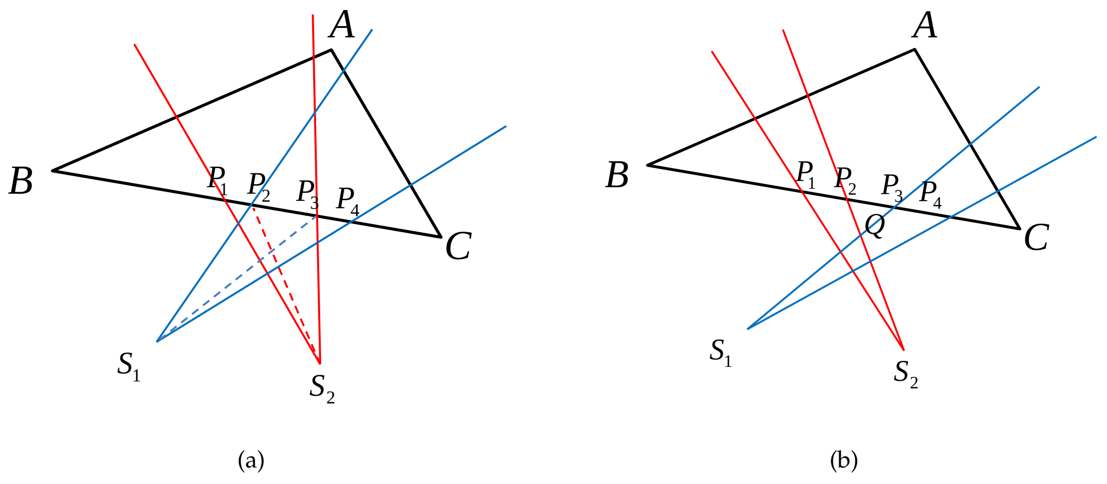


FIGURE 5.1: Illustration of Window Trimming. (a) two pencils from the pseudosources, S_1 and S_2, intersect. The associated windows $\overline{P_2P_4}$ and $\overline{P_1P_3}$ intersect with each other on edge $\overline{BC}$. (b) two pencils from the pseudosources, S_1 and S_2, cross. The associated windows $\overline{P_1P_2}$ and $\overline{P_3P_4}$ are independent from each other on edge $\overline{BC}$.

In Figure 5.1 (a), consider two windows $\overline{P_2P_4}$and $\overline{P_1P_3}$. It can be noted that when P_2 is located within the $\overline{P_1P_3}$, $P_2 \in \overline{P_1P_3}$, P_2 may have two paths connecting to the pseudosources S_1 and S_2 respectively. Let the geodesic distances from the source to the pseudosources, S_1 and S_2, be σ_1 and σ_2. According to Lemma 6.2 of [12], if the length $\sigma_1 + \overline{S_1P_2}$ is shorter than the length $\sigma_2 + \overline{S_2P_2}$, any path within the $\overline{P_1P_2}$ derived from the S_2 should not be defined as a geodesic. Therefore, the window $\overline{P_1P_3}$ is trimmed to the $\overline{P_2P_3}$. Otherwise, the window $\overline{P_2P_4}$ will be deleted. The same operation can be applied to the interior $\overline{P_3}$ as well.

In Figure 5.1 (b), consider two crossing pencils associated with the windows $\overline{P_1P_2}$ and $\overline{P_3P_4}$ respectively. Unfolding these two pencils into a common plane, it can be noted that the crossing point Q may have two paths connecting to the pseudosources S_1 and S_2 respectively. According to Lemma 6.2 of [12], if the length $\sigma_1 + \overline{S_1Q}$ is shorter than the length $\sigma_2 + \overline{S_2Q}$, any path within the window $\overline{P_1P_2}$ derived from the S_1 should not be defined as a geodesic. Therefore, it is deleted. Otherwise, the window $\overline{P_3P_4}$ will be deleted. These are summarized as,

Proposition 5.1. *Each window on an edge may be subdivided and associated with a set of pencils back to the same pseudosource. From the edge to different pseudosources, if any two pencils cross each other, one of the associated windows will be deleted.*

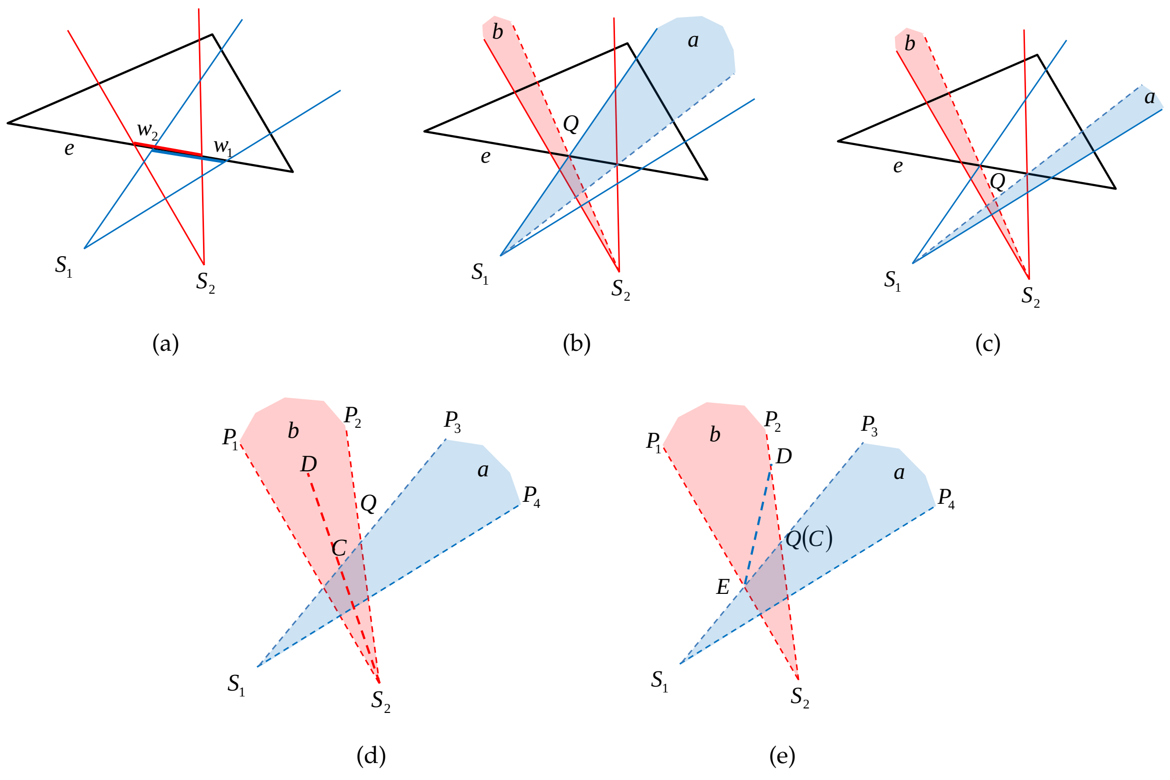


FIGURE 5.2: Illustration of Proof of Proposition 5.1.

Proof. Assume there are two windows w_1 and w_2 on an edge e, emanating from pseudosources S_1 and S_2 respectively (see Figure 5.2 (a)). Subdivide w_1 into a set of pencils and denote a as one of them. Similarly, denote b as one of the subdivided pencils of w_2. If a and b cross, there must be a crossing point Q. Q is either located on edge e (see Figure 5.2 (b)) or the unfolded common plane from windows w_1 and w_2 to their pseudosources (see Figure 5.2 (c)). Let the geodesic distances from the source to the pseudosources, S_1 and S_2, be σ_1 and σ_2.

Assume the distance from w_1 to point Q, $\sigma_1 + \overline{S_1Q}$, is smaller or equal to the distance from w_2 to point Q, $\sigma_2 + \overline{S_2Q}$ (i.e. $\sigma_1 + \overline{S_1Q} \leq \sigma_2 + \overline{S_2Q}$). Denote a and b as $\vee P_3S_1P_4$ and $\vee P_1S_2P_2$ respectively (see Figure 5.2 (d)). Assume geodesic path $\overline{S_2D}$ in pencil b that intersects the boundary $\overline{S_1P_3}$ of pencil a at point C. Since geodesic path is also the shortest path, the distance $\sigma_2 + \overline{S_2D}$ shouldn't be larger than the polyline $\sigma_1 + \overline{S_1C} + \overline{CD}$. As these two paths share the same portion from point C to D, the distance $\sigma_2 + \overline{S_2C}$ shouldn't be larger than $\sigma_1 + \overline{S_1C}$ accordingly. Thus, adding the same line segment from C to Q, the length of polyline $\sigma_2 + \overline{S_2C} + \overline{CQ}$ shouldn't be larger than $\sigma_1 + \overline{S_1Q}$. Furthermore, from triangle inequality, the polyline $\overline{S_2C} + \overline{CQ}$ shouldn't be smaller than $\overline{S_2Q}$. Therefore, it can be inferred that the distance from w_1 to point Q, $\sigma_1 + \overline{S_1Q}$, is larger or equal to the distance from w_2 to point Q, $\sigma_2 + \overline{S_2Q}$ (i.e. $\sigma_1 + \overline{S_1Q} \geq \sigma_2 + \overline{S_2Q}$). This contradicts the assumption $\sigma_1 + \overline{S_1Q} \leq \sigma_2 + \overline{S_2Q}$ unless the case degenerates and satisfies two conditions that the distance $\sigma_1 + \overline{S_1Q}$ equals $\sigma_2 + \overline{S_2Q}$ and the intersecting point C coincides with Q (see Figure 5.2 (e)). Denote E as the intersecting point of pencil boundaries $\overline{S_1P_3}$ and $\overline{S_2P_1}$. In this degenerated case, $\sigma_2 + \overline{S_2D}$ equals $\sigma_1 + \overline{S_1Q} + \overline{QD}$ and is larger than $\sigma_1 + \overline{S_1E} + \overline{ED}$ ($\overline{EQ} + \overline{QD} > \overline{ED}$ from triangle inequality). Therefore, $\sigma_2 + \overline{S_2D}$ is not the shortest distance to point D and $\overline{S_2D}$ is not a geodesic. In summary, any path within the pencil b that crosses $\overline{S_1Q}$ should not be defined as a geodesic and pencil b should be deleted.

A similar proof holds if $\sigma_1 + \overline{S_1Q} > \sigma_2 + \overline{S_2Q}$. $\square$

Remark: Trimming windows is time consuming in general; the "filtering rule" in the ICH algorithm does not take into account the "trimming" issue, which results in lots of redundant windows. The MMP algorithm uses its own "trimming" rule to disjoint the overlapping windows which spends much time. However, our Strategy I does not require disjointing the overlapping windows. Instead, Strategy I tries to find the balance between the number of redundant windows and the trimming time.

Moreover, it can be observed that the order of the trimmed windows on an edge is consistent with that of the branching points.

5.1.2 Strategy II: One Angle Two Sides

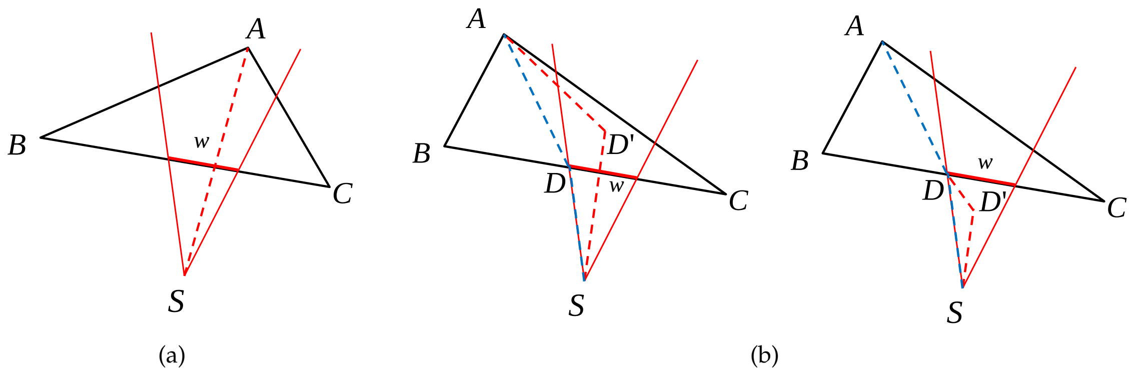


FIGURE 5.3: Illustration of the Shortest Path from a Window to its Opposite Vertex. (a) There exists a straight line within the pencil from the unfolded source S to the opposite vertex A. The pencil intersects with edge $\overline{BC}$ at the window w. (b) When the opposite A is invisible within the pencil, the polyline going through the endpoint of window, D, is shorter than others, i.e. $\|\overline{ADS}\| \leq \|\overline{AD'S}\|$.

Consider a pencil of lines emanating from the unfolded source or pseudosource vertex, S, which intersects the edge $\overline{BC}$ at the window, w (see Figure 5.3). An important observation is that the shortest path from the window w to the opposite vertex A is either a straightest line or a polyline passing through an endpoint of the window w. There are only two scenarios here and are illustrated well in Figure 5.3.

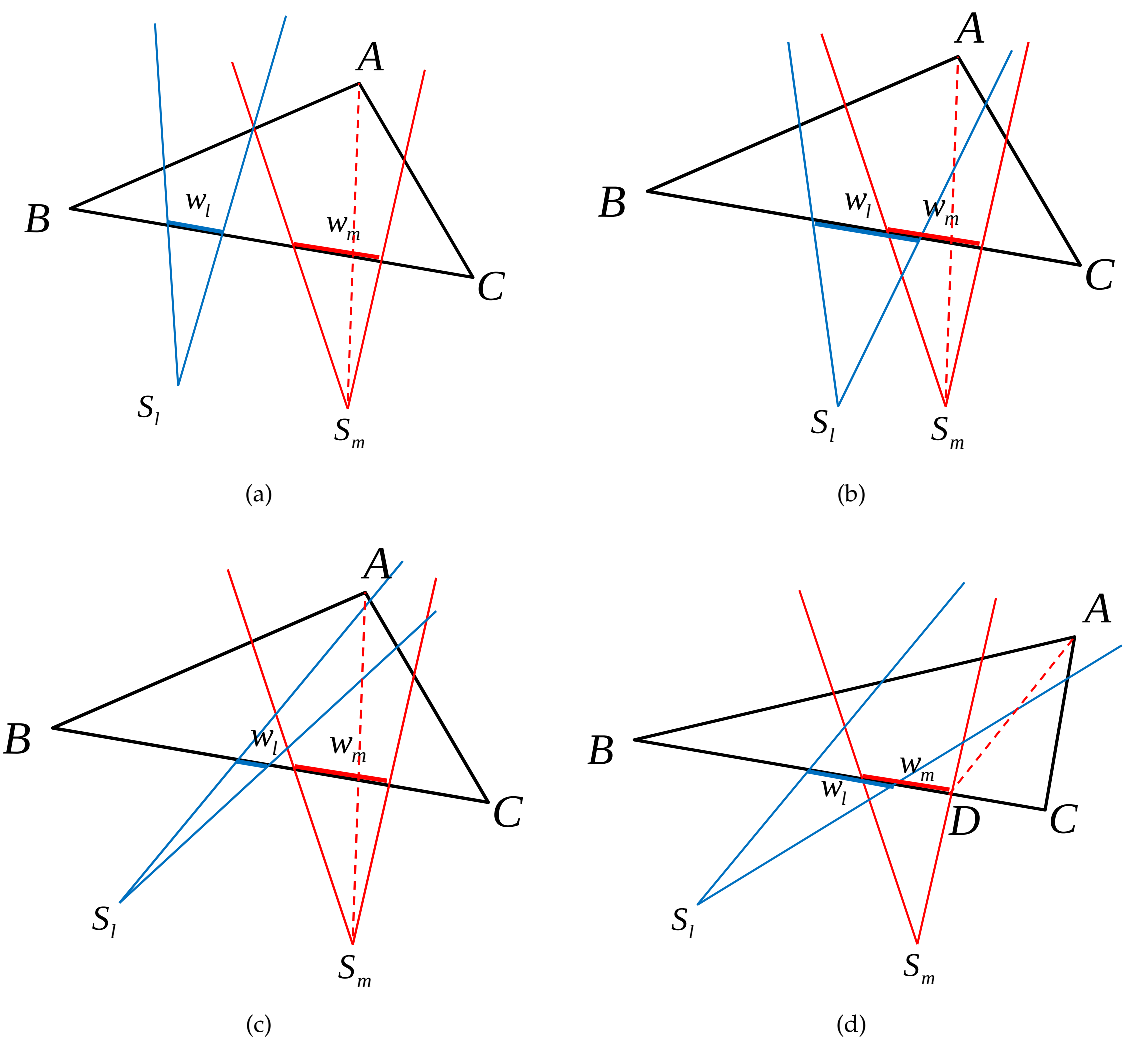


FIGURE 5.4: Illustration of One Angle Two Sides. The shortest path $\overline{S_m A}$ classifies the pencil S_l to the left side. The pencil S_m is the mid-pencil that contains the shortest path $\overline{S_m A}$. (a) The window w_l may be propagated to the edge $\overline{AB}$. The window w_m may be propagated to the edge $\overline{AB}$ and $\overline{AC}$. (b) Propagating the w_l to edge $\overline{AC}$ is occluded by the $\overline{S_m A}$. (c) Propagating the w_l to edge $\overline{AC}$ is occluded by the $\overline{S_m A}$. (d) The shortest path may be a polyline on an unfolding plane, $\overline{ADS_m}$. However, propagating the w_l to the edge $\overline{AC}$ is still occluded by the segment $\overline{AD}$.

Let d_w be the distance of the shortest path from the window w to the opposite A, and suppose that there is a set of windows $L = \{w_1, w_2, ..., w_n\}$ on the edge $\overline{BC}$. The minimum distance to the opposite A can be found, $d_L = \min(d_{w_1}, d_{w_2}, ..., d_{w_n})$. The resulting shortest path divides the branching points into two parts, left side and right side. The pencils associated with the pairs of successive branching points can be categorized into three groups accordingly, left side, right side and mid-pencil, that is, the shortest path is from the mid-pencil. The windows can be therefore classified into three parts accordingly. Note that the order of the windows on an edge may be different from that of the branching points below the edge in general. However, after the Strategy I here, the order of the trimmed windows is consistent with that of the branching points. According to the Lemma 4.3 of [12], the pencils associated with the windows cannot cross the shortest path. These are illustrated in Figure 5.4 and summarized as,

Proposition 5.2. *For a given triangle, suppose that there is a set of windows on an edge and the shortest path from the edge to the opposite vertex is given. The windows can be classified into three parts and the pencils associated with the windows cannot cross the shortest path.*

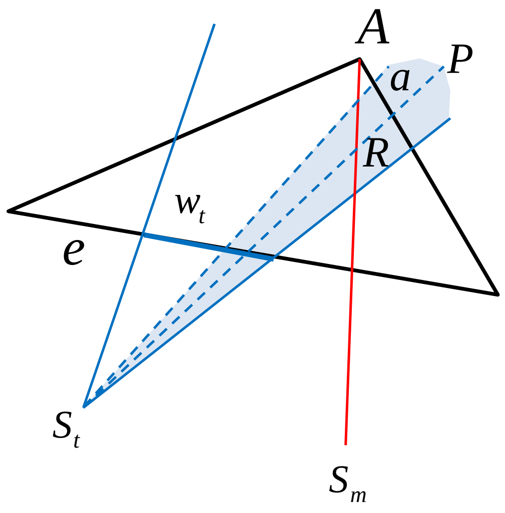


FIGURE 5.5: Illustration of Proof of Proposition 5.2.

Proof. As Figure 5.5 shows, suppose that there is a set of windows on an edge e. Assume that $\overline{S_m A}$ is the shortest path from them to the opposite vertex A. w_t is a window on e with S_t as its pseudosource. a is a pencil associated with w_t that crosses $\overline{S_m A}$. Let the geodesic distances from the source to the pseudosources, S_t and S_m, be σ_t and σ_m.

Assume there is a geodesic path $\overline{S_t P}$ in pencil a that intersect $\overline{S_m A}$ at point R. Since geodesic path is also the shortest path, the distance from pseudosource S_t to point P, $\sigma_t + \overline{S_t P}$, should be smaller than the distance from pseudosource S_m to point P, $\sigma_m + \overline{S_m R} + \overline{RP}$ (the equal case is excluded from that $\sigma_m + \overline{S_m R} + \overline{RP}$ is not a straight line segment in a triangle strip that its distance should not be geodesic). Since the two paths share the same portion from point R to P, the distance $\sigma_t + \overline{S_t R}$ should be smaller than $\sigma_m + \overline{S_m R}$ accordingly. Thus, adding the same line segment from R to A, the length of polyline $\sigma_t + \overline{S_t R} + \overline{RA}$ should be smaller than $\sigma_m + \overline{S_m A}$ (i.e. $\sigma_t + \overline{S_t R} + \overline{RA} < \sigma_m + \overline{S_m A}$. Since $\sigma_t + \overline{S_t R} + \overline{RA}$ mustn't be shorter than the shortest distance from w_t to A, it contradicts the assumption that $\overline{S_m A}$ is the shortest path from the windows to the opposite vertex A. Therefore, $\overline{S_t P}$ should not be defined as a geodesic and all the pencils associated with w_t cannot cross the shortest path $\overline{S_m A}$. □

5.2 Edge-based Wavefront Propagation

For an approximately isotropic metric, the previous algorithms usually employ a priority queue to sort the windows on a wavefront using the current estimation of the distance from the source.

This ensures that each window is visited only once by propagating the windows on the wavefront over a model. However, due to too much windows on the wavefront, it is desired to speed up sorting windows [20] and reduce the number of the windows on the wavefront as much as possible. Our idea is to exploit edges to speed up window propagation. Unlike the previous algorithms, our method defines a collection of edges as wavefront, to which all the windows are propagated, and then propagates a set of windows on an edge instead of one window each time (see Figure 5.6 (a)). All the windows lie in a wavefront. Roughly speaking, window propagation is to propagate the edges as a wavefront, which is also called the edge-based propagation framework.

This framework includes, accessing windows on an edge and accessing edges of a wavefront. To the former, thanks to branching points, we can access windows in terms of the spatial order of branching points. To the latter, we sort the end vertices of the edges according to their current distance and store them in the priority queue. The implementation of edge-based propagation is simple as follows,

1) Choose the vertex with the smallest distance from the priority queue. The 1-ring neighbours that are not visited are involve into the wavefront, while this selected vertex is removed from the wavefront and the priority queue. The discarded edges from the wavefront are stored in a FIFO queue;

2) Pop an edge from FIFO queue and propagate the windows on the discarded edge to the edges of the updated wavefront, and update the distance of vertices;

3) Update the priority queue by sorting the vertices in the wavefront and then go to step 1.

Herein, the key point is the step 2. Due to the change of the local wavefront (i.e. some edges are discarded from the wavefront while some others are involved), we need to propagate the windows on the discarded edges to the updated wavefront. Windows may be propagated to both the lately involved edges and the existing edges in the wavefront, that is, the existing windows may be updated. We still employ the Proposition 5.1 here to make the order of windows consistent with that of the branching points, so that we can access the windows on edges. Then, the Proposition 5.2 is employed to control the window propagation here as well. Intuitively, the Proposition 5.2 states that the shortest path divides the windows on an edge into three parts, left side, right side and middle, and the windows on the left side are propagated to the edge on the left while the windows on the right side are propagated to the edge on the right.

In Figure 5.6 (b), when the shortest path (red dashed line) within the triangle $\triangle P_1P_2V$ is from the edge $\overline{P_1V}$ to the opposite vertex P_2, it divides the windows on the edge $\overline{P_1V}$ into three parts. The windows on the left side are propagated to the edge $\overline{P_1P_2}$, that is, updating the windows on the $\overline{P_1P_2}$. The windows on the right side are propagated to the edge $\overline{P_2V}$. Then the windows on the $\overline{P_1V}$ are going on propagation in the same manner until arriving the edge $\overline{P_4V}$. At that time, the shortest path within the triangle $\triangle VP_4P_5$ is from the edge $\overline{P_5V}$ to the opposite vertex P_4 instead of from the edge $\overline{P_4V}$ as shown in Figure 5.6 (c). As a result, it is impossible to propagate the windows on the $\overline{P_4V}$ to the edge $\overline{P_4P_5}$ according to Lemma 4.3 of [12]. However, the windows are probably propagated to the edge $\overline{P_5V}$, that is, there may be the shorter path from the triangle strip (..., $\triangle P_1P_2V, \triangle P_2P_3V, \triangle P_3P_4V, \triangle P_4P_5V$) to the interior vertex Q. This means that the region bounded by the wavefront is not a monotonic

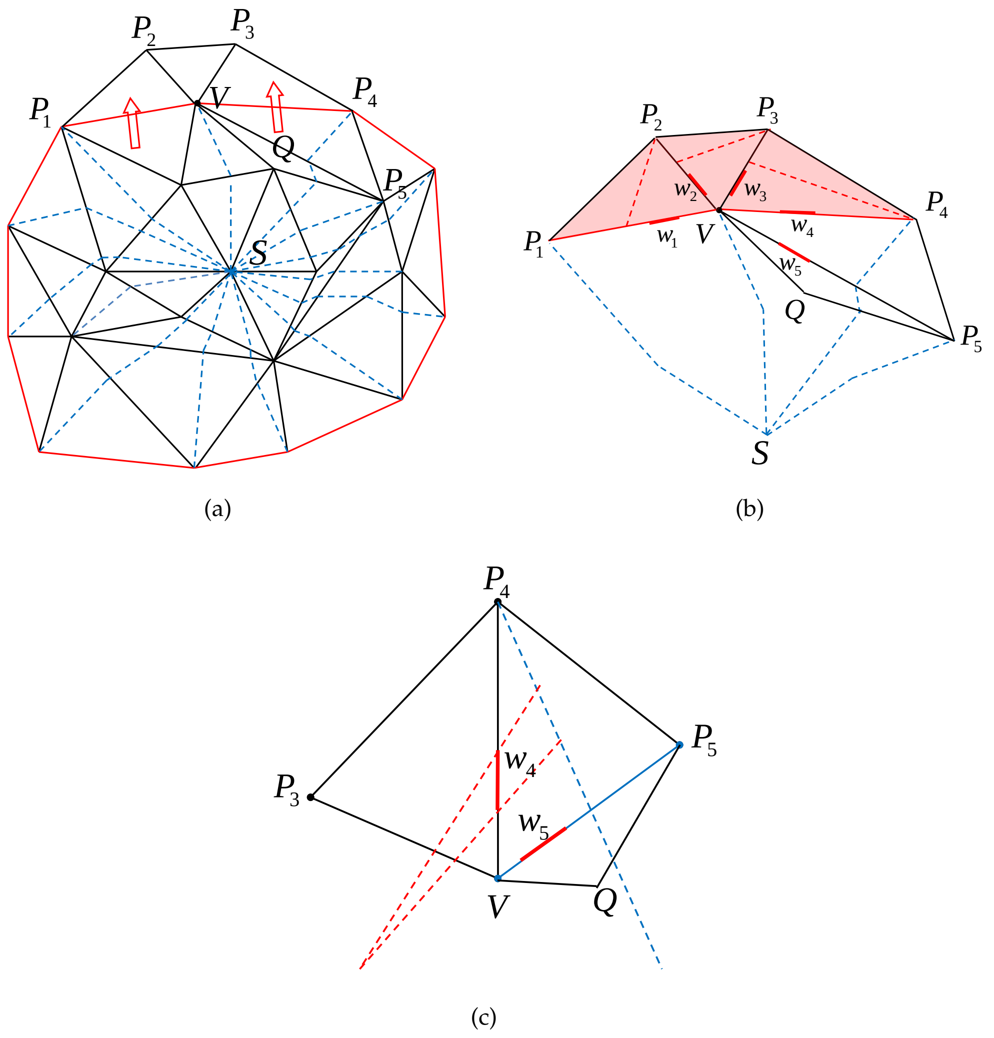


FIGURE 5.6: Illustration of Wavefront Propagation. (a) The dashed lines denote the shortest paths. The wavefront is depicted in red. Each edge on the wavefront may have the individual virtual triangle, which is bounded by the shortest paths from the source to two end vertices of the edge. Wavefront propagation is fulfilled by involving the neighbouring edges adjacent to vertex V; (b) the windows on edge $\overline{P_1V}$ is divided by the shortest path (red dashed line) into two parts. Windows w_1 is propagated to edge $\overline{P_2V}$. Windows w_2 is propagated to edge $\overline{P_3V}$. Until arriving the edge $\overline{P_4V}$, windows w_4 cannot be propagated to edge $\overline{P_4P_5}$, but probably to edge $\overline{P_5V}$; c) the shortest path to vertex P_4 occludes the propagation of windows w_4 to edge $\overline{P_4P_5}$. It is possible to propagate windows to w_5.

distance field. Our experimental results suggest that such propagations take place in the 1-ring neighbourhood of the wavefront. This is due to the fact that the constructed wavefront is only an approximate isoline (see Figure 5.7). Thus, the monotonicity cannot be guaranteed at the surrounding of the wavefront.

The distinct advantages include that, (1) the priority queue is applied to vertices rather than windows, which evidently decreases the cost of sorting; (2) the wavefront consists of edges instead of windows, which guarantees the quality of the wavefront (i.e. smooth and with small variance).

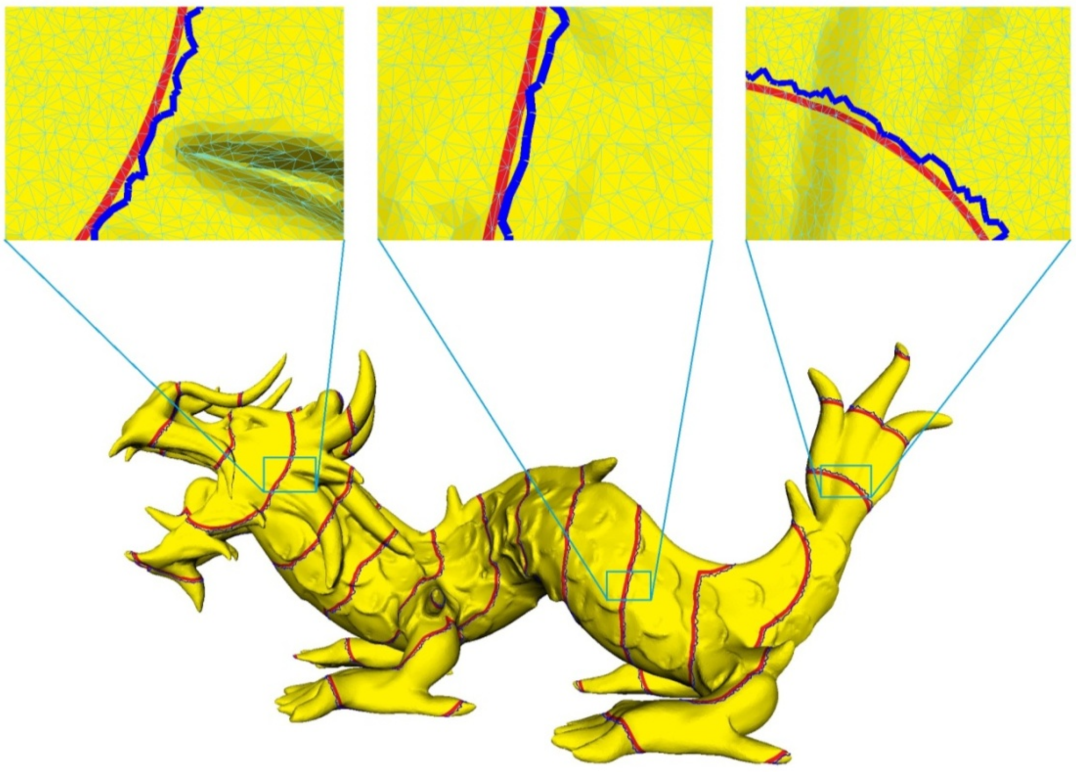


FIGURE 5.7: Wavefront Shape during Algorithm Execution. Red line denotes the isoline of the algorithm and blue line denotes the wavefront.

5.3 Complexity Analysis

5.3.1 Space Complexity

During propagation, windows on the wavefront need to be stored and updated gradually, which occupies the most of memory. Thus, space complexity analysis will focus on the number of windows. Recall the branching point introduced at the beginning of this section. We define the virtual triangle to crop a mesh, in which the covered vertices are viewed as branching points. The windows on an edge are associated with the pairs of the successive branching points within some virtual triangle. Moreover, for the wavefront, the edges correspond to their individual virtual triangles as shown in Figure 5.6 (a). These virtual triangles are independent of each other, i.e. the mesh is partitioned by these virtual triangles. Let the number of branching points within each virtual triangle be $k_i, i = 1, 2, ...$ respectively. Let the number of edges on the wavefront be m and n for vertices on a mesh. The number of the windows on the wavefront, n_w, is estimated as,

$$n_w < 2\sum_{i=0}^{m} k_i \leq 2n$$

Thus, it can be concluded that the space costs $O(n)$.

5.3.2 Time Complexity

As above-mentioned, the number of windows on the wavefront should be less than the vertex number $2n$. It is known that there is one triangle strip from the source for one window. We need to estimate the maximum of the triangle numbers for triangle strips here. Consider window propagation. The wavefront is initialized with one edge adjacent to the source for one window. The windows' number is variable during propagating windows on the wavefront outwards. This can be described by B-tree structure. The maximum of the triangle numbers is indeed the height of the B-tree.

The running time consists of two parts, propagating windows over a mesh and maintaining the priority queue. The former is estimated as $O(n \log_b n)$, where b denotes how many "son-windows" each "parent-window" yields. It is too difficult to accurately estimate the parameter b in practice, since it depends on models. To estimate the parameter b, we perform our implementation on several models. We estimate that the b should be less than 2, since most of windows will not be split except a small number of windows covering the opposite vertices. From Table 5.1, we can note that the parameter b is ranged from 1.015 to 1.029, i.e. $b \in$[1.015,1.029].

TABLE 5.1: Time Complexity Analysis. Performing the proposed algorithm on different models shows how the parameter b changes. The term $\log_b n$ indicates the height of B-tree, i.e. the maximum of triangle numbers for triangle strips.

Model	n **(vertex number)**	$\log_b n$	b
knot	28,800	407	1.0255
horse	48,484	374	1.0293
bunny	72,020	418	1.0271
hand	88,357	603	1.0191
rocker arm	241,056	857	1.0146
lucy	262,909	466	1.0271
asian dragon	833,873	580	1.0238
cervino terrain	1,574,913	936	1.0154

The latter is estimated as $O(n \log n)$. Therefore, the overhead of running time is estimated as $O(n \log_b n + n \log n)$.

5.4 Implementation

The priority queue is initialized with the 1-ring neighbour vertices of the source vertex. The geodesic distance of these neighbours are equal to the corresponding edge lengths. The wavefront is the collection of the edges adjacent to the source. Each edge is allocated a window. Windows on an edge can be accessed by a linked-list associated with this edge. Propagating a window still employs the tuple structure used in MMP-class algorithms.

The proposed Exact Algorithm is summarized as,

Algorithm 1 Edge-based Exact Geodesic Computation

```
Input: Mesh: $M$; Source: $S$
Output: Geodesic distance from $S$ to all vertices on $M$
1: procedure INITIALIZATION
2:     Priority queue, $Q$, and Wavefront, $W$;
3:     Let FIFO queue (edges) empty;
4:     Let a set of linked-lists $L$ (windows), associated with edges, contain their individual
   windows.
5: end procedure
6: while $Q$ is not empty do
7:     Pop a vertex $v$ from $Q$;
8:     Update $W$ and $Q$;
9:     Add the discarded edges into FIFO queue;
10:    while FIFO queue is not empty do
11:        Pop an edge $E$ from FIFO;
12:        procedure WINDOWS TRIMMING($E$)
13:            Identify crossing/overlapping windows pairwise;
14:            Delete the crossing parts of the windows;
15:        end procedure
16:        procedure WINDOWS PROPAGATION($E$)
17:            Find shortest paths from windows on the edge;
18:            Classify windows into left side, right side and middle;
19:            Propagate windows to left or right edges accordingly;
20:        end procedure
21:        Update FIFO Queue;
22:    end while
23: end while
```

5.5 Experimental Results and Analysis

We test the exact geodesic algorithms on a HP Z420 Workstation with Intel(R) Xeon(R) CPU E5-1650 0 @ 3.20GHz and 32.0GB memory. MMP-class and CH-class algorithms usually need to remove tiny windows by setting threshold of windows' width. Our algorithm does not generate tiny windows, since it is not required to generate disjoint windows on edges. For a fair comparison, we follow [20] and set the threshold $\epsilon = 10^{-6}$ for MMP algorithm, CH algorithm and their variants throughout all the experiments.

5.5.1 General Performance

As shown in Table 5.2, our algorithm outperforms the state-of-art exact algorithms, particularly useful for large models. For instance, our algorithm 1) runs 5-9 times faster than MMP and ICH algorithms and 2-4 times faster than FWP-MMP and FWP-CH algorithms; 2) consumes space comparable to ICH and FWP-CH algorithms, which is considerably less than that of MMP and FWP-MMP algorithms; 3) reduces approximately 30% windows compared to MMP/FWP-MMP algorithms and 60% to ICH/FWP-CH algorithms. We further address the above issues in terms of Table 5.2 as follows.

1) Effectiveness of redundancy reduction. "Window Propagation" refers to the number of all the propagated windows. "Valid Windows" refer to the number of the windows after redundancy reduction. For MMP-class algorithms, the "trimmed" windows need to be re-inserted into the priority queue by re-sorting. Intuitively, this is equivalent to insert a new window into the priority queue. Thus, the re-inserted windows are regarded as new windows here. According to MMP algorithm [12], the distribution of the optimal windows should cover the whole mesh without any overlap or gap. Thus, we perform MMP algorithm on models and take "Window Limit" in Table 5.2 as "the number of windows left" when MMP algorithm terminates. "Peak Wavefront" refers to the maximum numbers of the windows on the wavefront. The first observation is that "Windows Propagation" and "Valid Windows" of MMP-class algorithms are close to each other as well as our algorithm, while for CH-class algorithms, "Valid Windows" is nearly half of "Windows Propagation". Moreover, our algorithm is the most closest to the "Window Limit". This means that our algorithm generates the least useless windows. Additionally, due to lack of "trimming windows" in the CH-class algorithms, this inevitably causes lots of redundant windows (see ICH and FWP-CH in Table 5.2). These algorithms have to spend much time on propagating redundant windows and then filtering out them, which can be observed by comparing FWP-CH with FWP-MMP and ours.

2) Space Complexity. MMP-class algorithms store all the windows, which consumes more space than CH-class algorithms. Our algorithm only stores the windows on the wavefront as well as CH-class algorithms. However, comparing "Peak Wavefront", it can be noted that MMP-class algorithms have smaller numbers. A guess is that since MMP-class algorithms require windows disjointed on edges, this probably tends to the optimal distribution of windows on edges. It can also be noted that "Peak Wavefront" of our algorithm is still close to that of MMP-class algorithms. The important issue is that MMP-class algorithms require storing all the windows, while our algorithm and CH-class algorithms only store the windows on

TABLE 5.2: Performance Comparison of State-of-Art Exact Shortest Path Algorithms on Models

Model	**Performance**	**Algorithms**					**window limit**
		MMP	**FWP-MMP**	**ICH**	**FWP-CH**	**New Algorithm**	
knot V: 28,800 F: 57,600	Time(s)	1.888	0.749	1.716	1.108	0.444	1,801,493
	Space (MB)	144.12	144.13	0.45	0.45	0.37	
	Window Propagations	2,556,815	2,544,030	4,158,338	4,148,550	1,989,300	
	Valid Windows	2,979,446	2,521,478	2,167,236	2,161,063	1,972,566	
	Peak Wavefront	4,378	4,290	5,656	5,629	4,595	
horse V: 48,484 F: 96,964	Time(s)	3.696	1.354	3.839	2.245	0.747	2,893,843
	Space (MB)	231.51	231.51	1.80	1.78	1.40	
	Window Propagations	4,404,114	4,384,441	7,944,632	7,963,743	3,304,586	
	Valid Windows	4,841,219	4,175,795	4,151,326	4,155,743	3,283,055	
	Peak Wavefront	16,578	16,461	22,563	22,296	17,529	
bunny V: 72,020 F: 144,036	Time(s)	5.576	1.983	6.091	3.426	1.058	4,255,625
	Space (MB)	340.47	340.45	1.70	1.70	1.19	
	Window Propagations	6,485,160	6,450,962	12,305,827	12,327,991	4,751,268	
	Valid Windows	7,104,880	6,105,087	6,428,025	6,431,133	4,744,906	
	Peak Wavefront	13,988	13,826	21,171	21,263	14,925	
hand V: 88,357 F: 176,176	Time(s)	14.829	4.596	12.106	6.673	2.455	9,166,680
	Space (MB)	733.34	733.56	3.03	3.06	2.78	
	Window Propagations	13,848,267	13,940,341	21,915,576	22,052,365	12,040,473	
	Valid Windows	15,183,300	13,135,194	11,323,464	11,394,155	11,714,677	
	Peak Wavefront	33,177	33,780	37,813	38,238	34,636	
rocker arm V: 241,056 F: 482,112	Time(s)	36.481	12.449	38.781	19.818	5.184	22,506,685
	Space (MB)	1800.54	1801.36	5.29	5.42	3.72	
	Window Propagations	33,989,628	36,468,136	68,553,961	70,548,022	25,896,785	
	Valid Windows	38,633,005	36,020,803	35,214,886	36,282,108	25,804,671	
	Peak Wavefront	41,946	44,836	66,169	67,760	46,514	
lucy V: 262,909 F: 525,814	Time(s)	14.383	5.756	16.287	10.013	2.837	11,198,737
	Space (MB)	895.90	893.27	2.01	2.06	1.36	
	Window Propagations	16,938,945	17,360,336	31,659,467	32,018,581	12,923,556	
	Valid Windows	18,656,576	16,687,100	16,692,900	16,873,845	12,905,847	
	Peak Wavefront	17,323	17,654	25,109	25,800	17,034	
asian dragon V: 833,873 F: 1,667,742	Time(s)	Out of memory	Out of memory	96.021	51.34	13.521	N/A
	Space (MB)			6.32	6.35	5.12	
	Window Propagations			142,953,293	143,440,954	62,092,856	
	Valid Windows			74,477,499	74,682,895	62,047,736	
	Peak Wavefront			78,998	79,351	63,948	
cervino terrain V: 1,574,913 F: 3,145,728	Time(s)	Out of memory	Out of memory	292.293	153.6	33.507	N/A
	Space (MB)			11.88	11.95	10.90	
	Window Propagations			393,115,221	396,125,296	164,223,488	
	Valid Windows			207,015,046	208,293,176	166,536,343	
	Peak Wavefront			148,569	149,433	136,190	

the wavefront. This is very useful for large models. Moreover, we further check the change of the windows on the wavefront for two models. From Figure 5.8, we see that the windows' numbers of FWP-MMP are much closed to those of our algorithm, and most of the time, the windows' number of our algorithm is less than that of FWP-MMP. The wavefront preserves the valid windows that will be propagated out at the next iteration. Our algorithm can preserve less valid windows for propagation all the time.

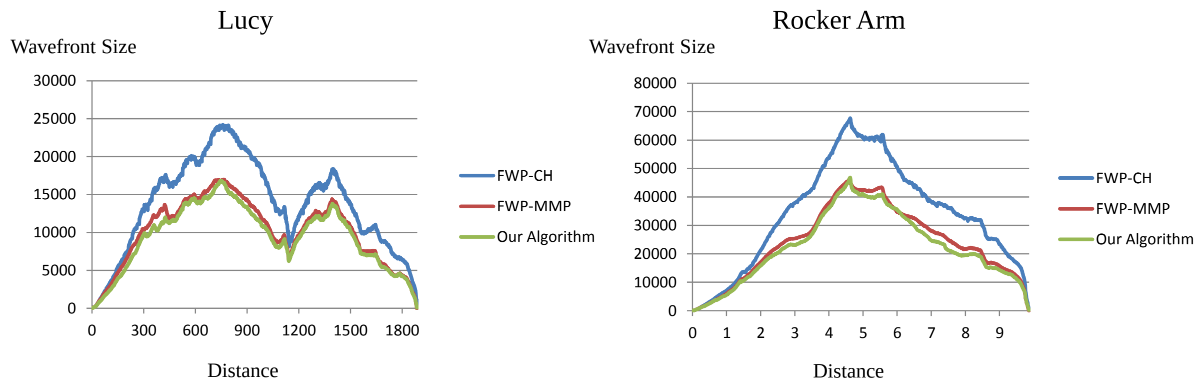


FIGURE 5.8: Comparison of Windows on Wavefronts. We sample the windows' number with different geodesic distance levels from the source to wavefronts.

3) Time Complexity. From Table 5.2, it can be noted that our algorithm is evidently faster than the others. To have an insight into the performance of algorithms, we firstly summarize the state-of-art exact algorithms in Table 5.3, and then compare every module of algorithms.

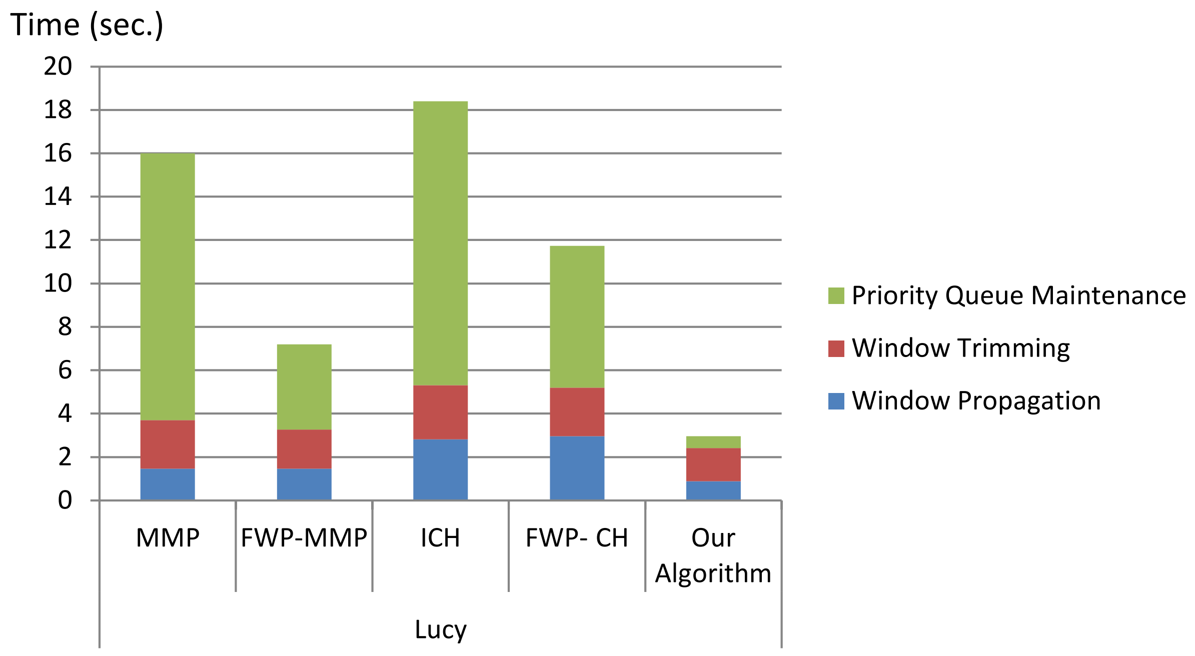
TABLE 5.3: Comparison of Dijkstra-like Exact Geodesic Algorithms.

Method		**Data Structure**	**Time Complexity**	**Space Complexity**	**Propagation Organization**	**Spatial order available?**
CH	CH	FIFO queue	$O(n^2)$	$\theta(n)$	Window	No
	ICH	Priority queue	$O(n^2 logn)$	$O(n^2)$	Window	No
	FWP-CH	Bucket & FIFO queue	$O(n^2)$	$O(n^2)$	Window	No
MMP	MMP	Priority queue	$O(n^2 logn)$	$O(n^2)$	Window	No
	FWP-MMP	Bucket & FIFO queue	$O(n^2 logn)$	$O(n^2)$	Window	No
Our Algorithm		List & Priority queue	$O(nlog_b n$ $+ nlogn$	$O(n)$	Edge	Yes (branching points' order)

MMP-class algorithms may be divided into three modules, including window propagation, window trimming (binary search and partition) and priority queue maintenance (traditional priority queue / bucket structure & FIFO queues). Similarly CH-class algorithms may be divided into three modules, including window propagation, window trimming("one angle one split" and window filtering rule) and priority queue maintenance (traditional priority queue / bucket structure & FIFO queues). Our algorithm can also be divided into three modules, including window propagation, window trimming (Strategies I and II) and priority queue maintenance (i.e. sorting vertices rather than windows).

We performed the five algorithms (i.e. MMP, FWP-MMP, ICH, FWP-CH, ours) on two models, and recorded the running times of the three modules respectively as shown in Figure 5.9.

It can be observed that our algorithm distinctly compresses the running time of every module. This is because (1) our Strategies I and II can effectively reduce redundancy (see the time of "Window Propagation"); (2) the cost of our Strategies I and II is too low (see the time of "Window Trimming"); (3) applying the vertex-based priority queue dramatically speed up the implementation of algorithm (see the time of "Priority Queue maintenance").



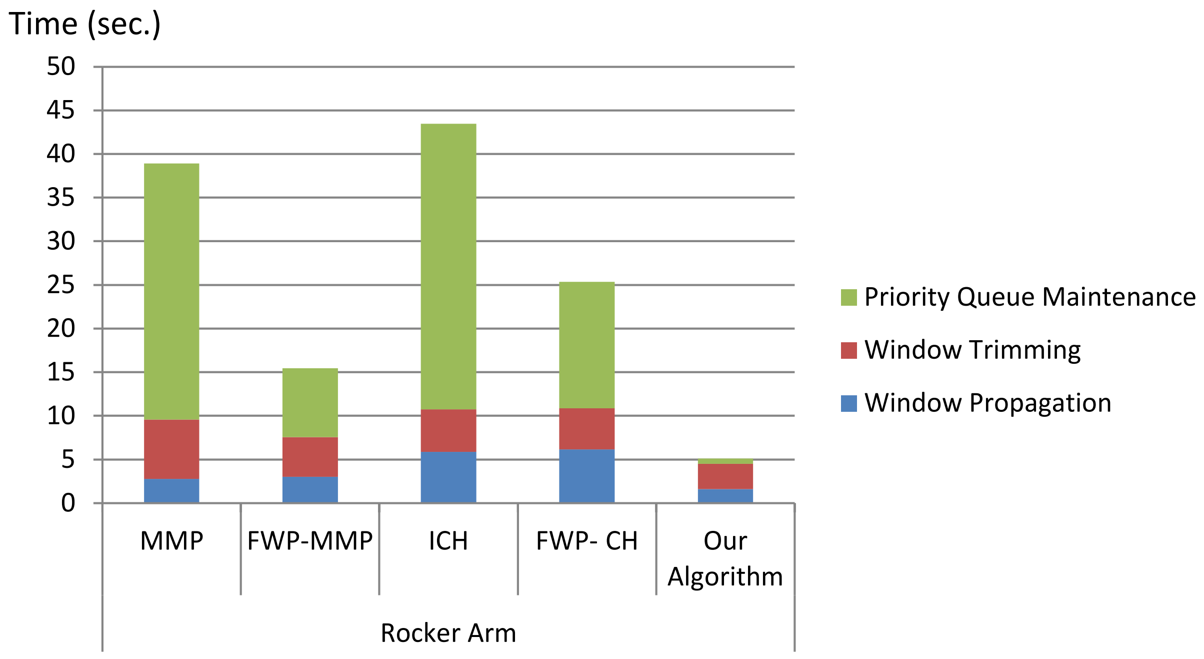


FIGURE 5.9: Statistics of Running Time.

5.5.2 Wavefront Quality

The quality of wavefront influences the monotonicity of distance field. If wavefront is of poor quality, it will cause many useless windows and result in convergence slow. Unlike the existing algorithms, our algorithm applies the priority queue to vertices instead of windows, which yields the wavefront consisting of a set of edges. Thus, the resulting wavefront is smoother. Figure A.1-A.5 in Appendix A shows the comparison of the wavefronts created by MMP, ICH, FWP-MMP, FWP-CH, ours. For a quantitative comparison, we compute the distance from the source vertex for windows and show the standard deviation (SD) of all windows in Figure 5.10. It can be noted that MMP and ICH have smaller SD. This is because MMP and ICH strictly apply the priority queue to windows. Our algorithm has still comparable result. The important issue is that the priority queue is applied to vertices in our algorithm, which dramatically speeds up the implementation. All the experiments suggest that our algorithm balances the wavefront quality and the convergence speed.

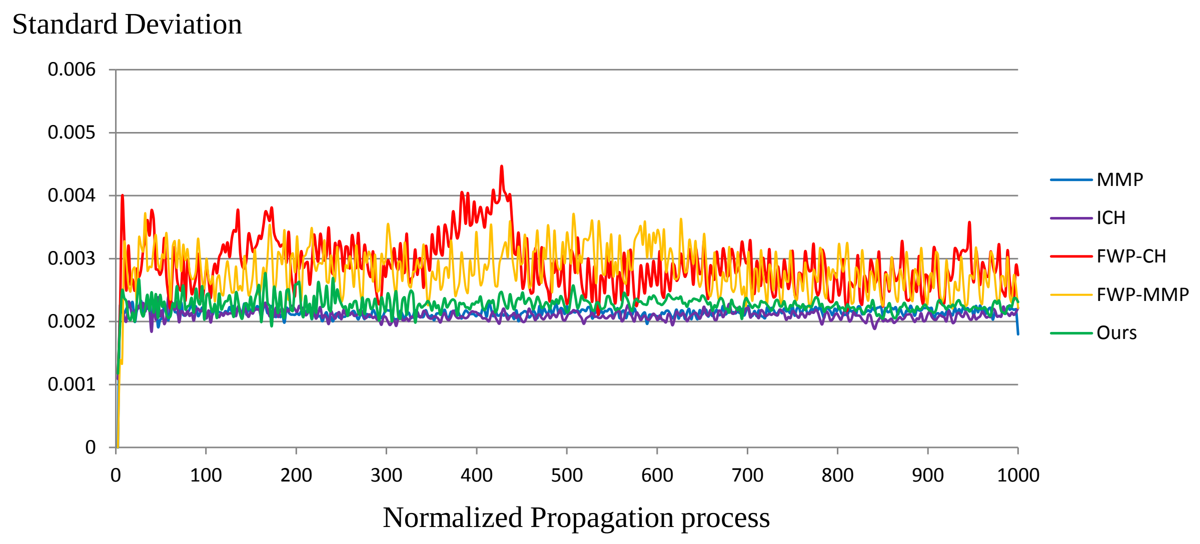


FIGURE 5.10: Wavefront Quality of Various Exact Geodesic Algorithms. We follow the method used in FWP algorithms [20] that calculates the standard deviation of windows' distances on the wavefront to measure the wavefront quality.

5.6 Comparison

This section compares our algorithm against the two most notable classes of exact geodesic algorithms, the MMP-class algorithms and the CH-class algorithms.

5.6.1 Comparison with MMP-class Algorithms

- Reducing useless window propagation. MMP-class algorithms firstly propagate windows to edges without any pre-filtering, and then identify overlapping windows by binary searching. After that the windows are trimmed by partitioning windows. This means that lots of useless windows are likely propagated. Moreover, lots of time is spent on binary searching and partitioning windows as well. In our algorithm, lots of useless windows can be identified by simple classification (the Strategy II) and filtered out before propagation.

- Computational effectiveness. In our algorithm, since the spatial order of the branching points is available, identifying the crossing/overlapping parts of the windows (see the Strategy I) does not require extra binary search, that is, identifying can be fulfilled in $O(1)$. Besides, trimming windows does not require generating disjoint windows on edges, which unloads a big computational burden.

- Low cost of priority queue maintenance. MMP-class algorithms are to propagate the windows as a wavefront and spend lots of time on sorting windows in the priority queue. Our algorithm employs the collection of edges to delineate the wavefront and sort vertices in the priority queue instead of windows. As a result, windows are batched up in terms of edges, and propagated outwards in a batch processing way. This evidently speeds up the algorithm.

5.6.2 Comparison with CH-class Algorithms

- Reducing useless window propagation. CH algorithm employs the "one angle one split" rule to evaluate window propagation. ICH algorithm [17] further introduces the window filtering rule and claims that more than 99% useless windows created by the CH algorithm can be filtered out. Except that, they do not take into account trimming windows. In our algorithm, the strategy II is more general and covers "one angle one split" rule, that is, CH algorithm only deals with the "middle" windows. Additionally, ICH algorithm cannot identify useless windows before propagation, as well as MMP-class algorithm. The combination of the Strategies I and II in our algorithm is more effective to reduce redundant windows.

- Low cost of priority queue maintenance. CH-class algorithms encounter the same issue as MMP-class algorithms.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Geodesic computation on surfaces is indispensable in many research and industrial areas. So far, we have proposed a new method in solving the SSSP discrete geodesic problem based on the spatial conformity of geodesic paths, which differs it from the two notable classes of exact geodesic algorithms, i.e. the Mitchell-Mount-Papadimitriou (MMP) algorithm, the Chen-Han (CH) algorithm and their variants. Both of these two classes of algorithm are based on individual window propagation framework that propagates one window across a face each time, which is time-consuming.

Generally speaking, there are three central modules for these windows-based exact geodesic algorithms which make contribution to its computational cost: window propagation, window trimming (including filtering) and priority queue maintenance. Incremental researches have been done to reduce the costs of each module. Among which, a notable work is the recent proposed FWP-MMP and FWP-CH algorithms which aim at reducing the time costs of priority queue maintenance. However, it is very challenging to decrease the computational costs for all of the three parts simultaneously using the existing individual window propagation framework. To deal with this challenge, we propose two novel strategies of redundancy reduction from the property of geodesic paths that they are not allowed to cross each other. Accompanying the two strategies, we proposed an edge-based propagation framework based on the spatial conformity of geodesic paths in the implementation which propagates all windows on one edge simultaneously.

The distinct advantages of our algorithm against existing geodesic algorithms include, (1) with the novel windows reducing strategies proposed, useless window propagations are effectively reduced compared to existing window-based propagation methods; (2) using the spatial conformity of windows on edges, windows can be trimmed in a sequential way, which eliminates the binary search part of MMP-class algorithms and achieves similar results; (3) the priority queue used in our algorithm is applied to vertices instead windows, which naturally and dramatically decreases the cost of maintaining the priority queue.

As shown by our experiments, our algorithm outperforms all existing exact geodesic algorithms that it runs 5-9 times faster than MMP and ICH algorithms and 2-4 times faster than FWP-MMP and FWP-CH algorithms. It is believed that our algorithm may be a new choice for many applications involving geodesic computation in computer graphics. The calculated geodesic distance fields on multiple models are illustrated by isolines in Figure 6.4.

6.2 Future Work

Although the proposed algorithm is already the fastest exact geodesic algorithm to date, there is still further work to do, including improving its performance, modifying it to obtain an approximate version and use it in other applications to achieve better performance.

6.2.1 Improving Current Algorithm

One possible future work is to further improve the performance of current algorithm. We observed that there is still redundancy in the proposed algorithm since many useless windows which cover no vertices are propagated across the surface (see Figure 6.1). It can be observed that the window w_1 (blue) on edge e covers vertex P and thus its propagation is solid and without redundancy. However, for the window w_2 (green) on e, its propagations are all redundant until reaches vertex Q. For the window w_3 (red) on e, all its propagations in the triangle strip shown is redundant. One idea of reducing this kind of redundancy is to delay the propagation of such windows and only propagate those which may produce the geodesic path to a vertex on the graph. For example, in Figure 6.1, when propagating windows on edge e, we may only propagate w_1 and keep w_2 and w_3 on edge e until the overall propagation reaches vertex Q. Then, w_2 is propagated directly to vertex Q across several faces with one operation. In this way, the performance of the algorithm is expected to have a further improvement.

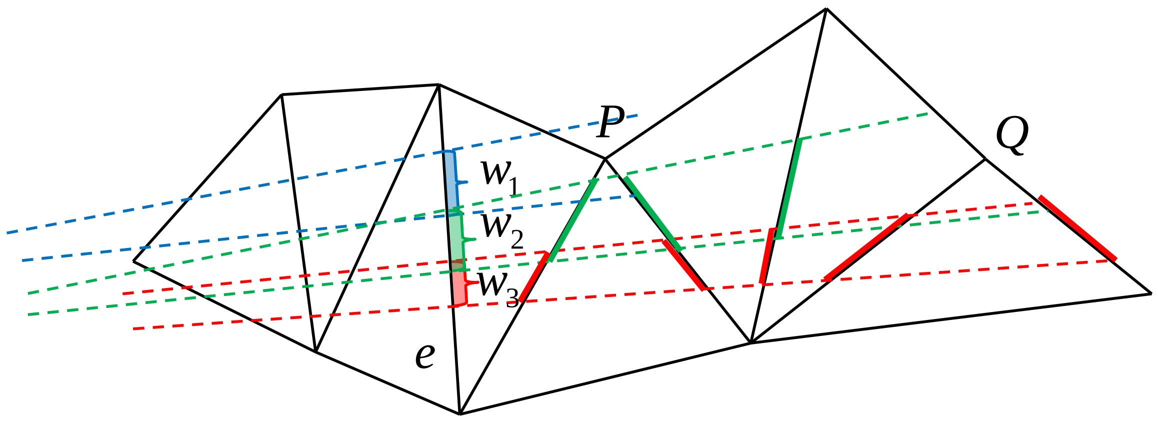


FIGURE 6.1: Illustration of Remaining Redundancy in Current Algorithm. The red and green line segments represents the redundant windows generated during propagation.

6.2.2 Approximate Version of Current Algorithm

Another possible future work is to modify the current algorithm to obtain an approximate version which consumes less time and space under bounded error. In 2005, Surazhsky et al. [16] proposed an approximate version of the MMP algorithm based on an operation named *window merging* (see Figure 6.2 (a)). This operation use one new window to approximately represent two existing windows under bounded error and therefore reduces the time cost of the MMP algorithm. Note that their algorithm is from the window based propagation framework and thus the operation can only happen between two windows. With our edge-based propagation frameworks, the windows are all organized on edges and therefore a natural idea is to merge the most windows on an edge in one operation (see Figure 6.2 (b)). Besides, the priority queue used can be replace by a bucket structure like Xu et al. [20] do. Therefore, it is expected that the practical time complexity of the algorithm should be $O(n)$ with acceptable error.

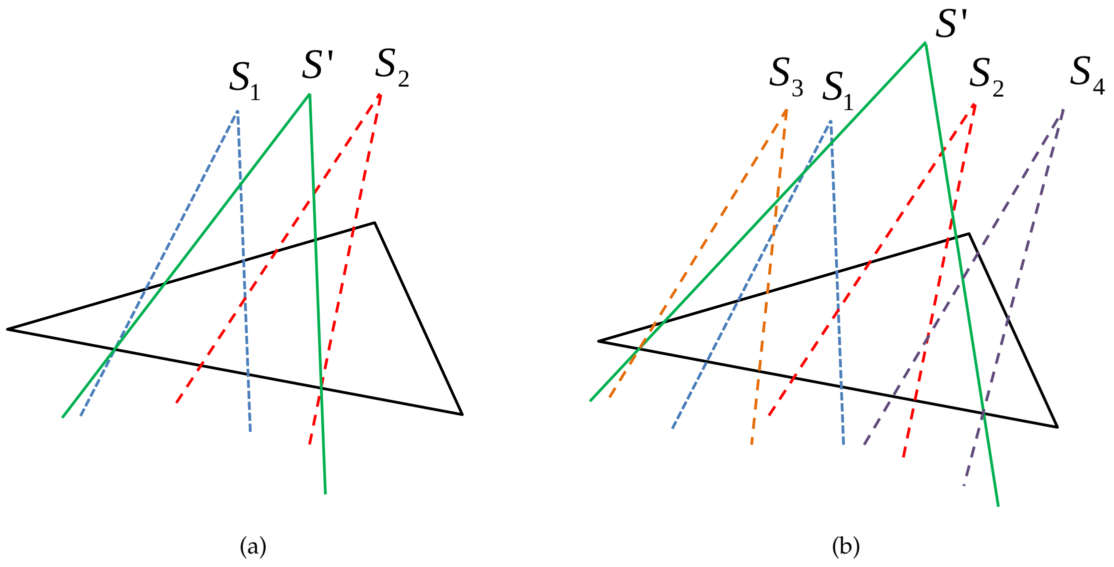


FIGURE 6.2: Illustration of Window Merging Operation in the MMP Algorithm and Our Future Work. (a) window merging in the MMP algorithm; (b) window merging in our future work.

6.2.3 Application in Neuron Tracing

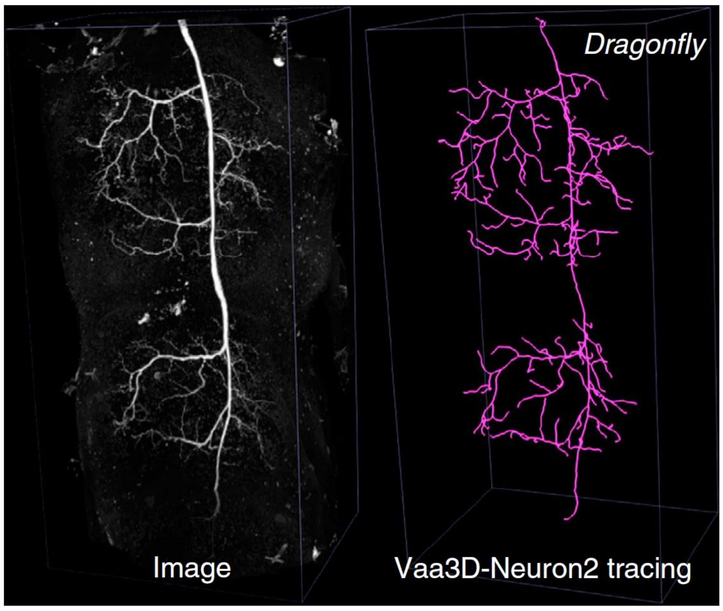


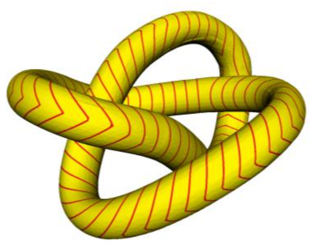
FIGURE 6.3: Illustration of Neuron Tracing application. Adapted from the Figure 8(c) in [64].

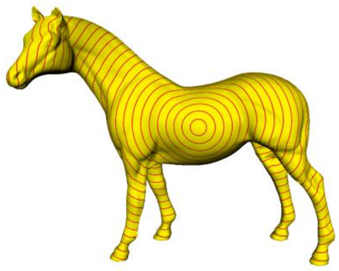
The current algorithm can also be applied to real-world applications. We found that one application of our algorithm is the neuron-tracing tasks in brain science (see Figure 6.3). The neuron tracing task aims at extracting the topology of a neuron from 3D volume data, which is similar to the skeleton extracting task in computer graphics. Liu et al. [42] has shown that the computation of geodesics plays an fundamental role in some skeleton extracting algorithms and it can be inferred that the idea can also be applied to neuron tracing tasks. Since one of the challenging issue in neuron tracing is that its scale of data is terabyte. One possible work here is to use a simpler polyhedron to approximate the neuron data and calculate geodesics on the polyhedron instead. Some other issues that may occur include the influences of noises, the complexity of neuron morphology, etc.

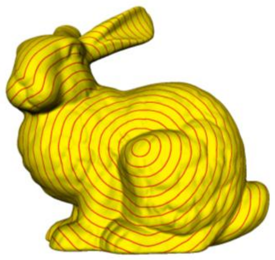
6.2.4 PhD Research Plan

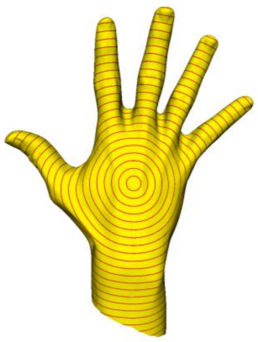
TABLE 6.1: Schedule of Future Research Plan.

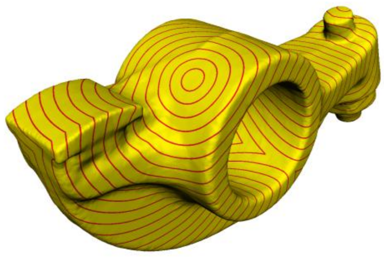
Year	**Research Activity**	**Target Completion Date**
2016	Design and implement the approximate version of this algorithm. Do experiments accordingly.	01/03/2016
	Generalize the geodesic algorithm on volume Data and try to make a prototype.	01/04/2016
	Pay a visit to Allen Institute for Brain Science to learn the Vaa3D tool.	15/05/2016
	Try to adapt the geodesic algorithm in neuron tracing application.	15/07/2016
	Write final Thesis for PhD graduation.	01/10/2016

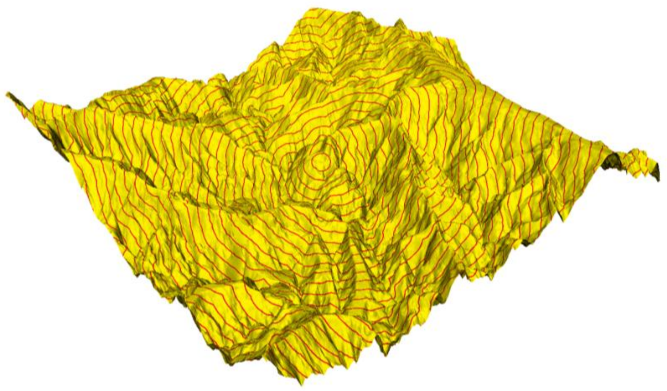

(a) $knot$

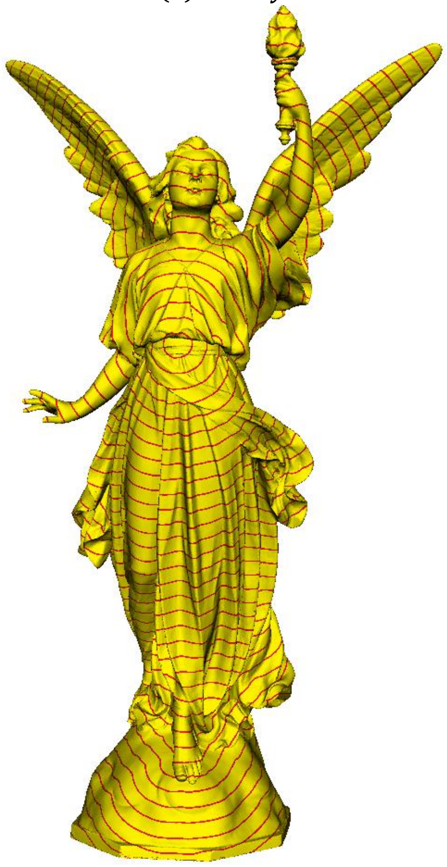

(b) $horse$


(c) $bunny$


(d) $hand$


(e) $rocker\ arm$


(f) $cervino\ terrain$


(g) $lucy$

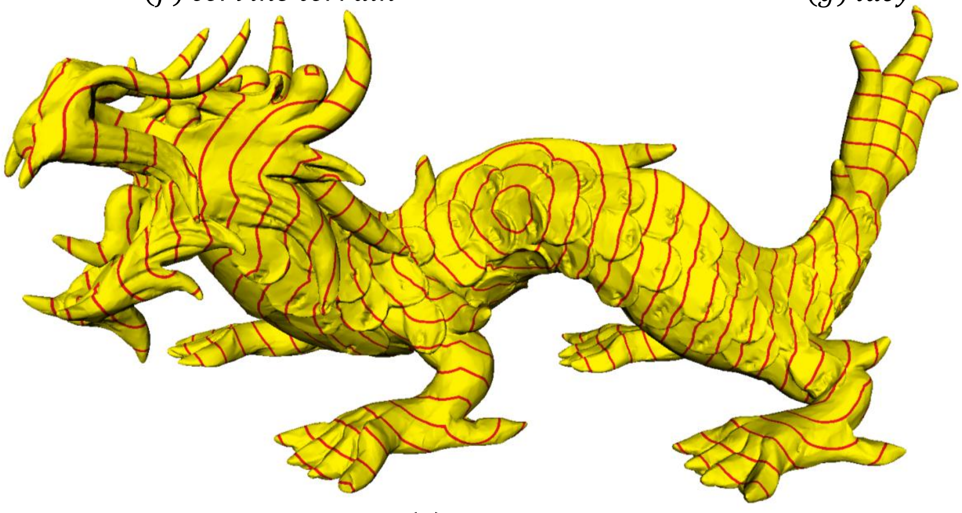

(h) $asian\ dragon$

FIGURE 6.4: Isolines of Several Models.

Appendix A

Wavefront Shapes of State-of-the-art Algorithms

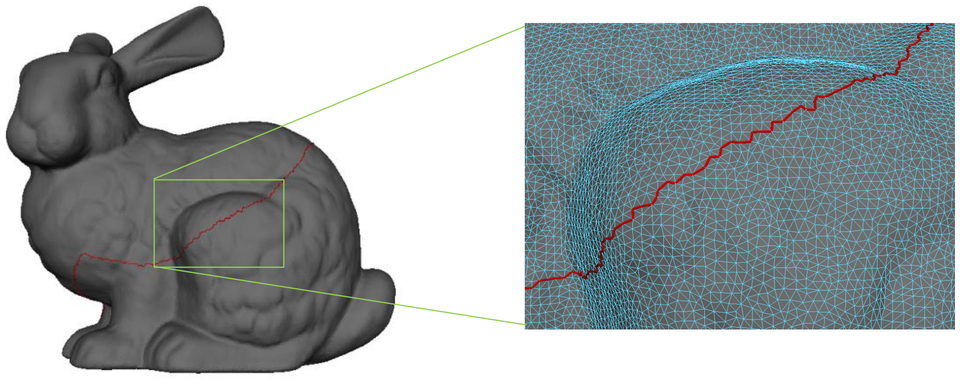


FIGURE A.1: Wavefront Shape of Our Algorithm.

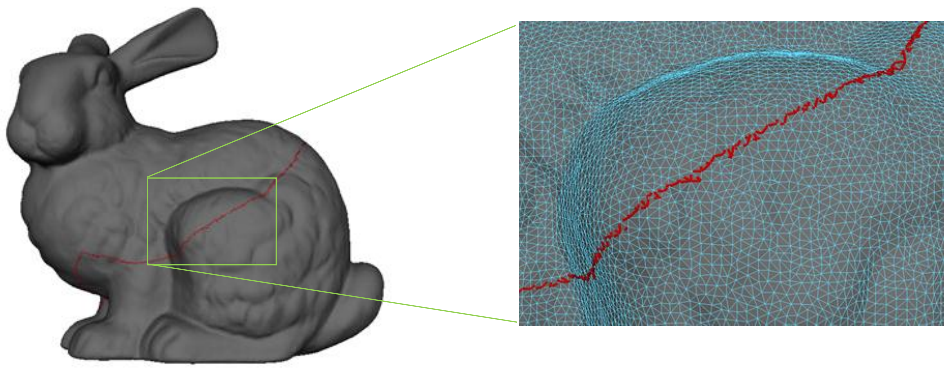


FIGURE A.2: Wavefront Shape of MMP Algorithm.

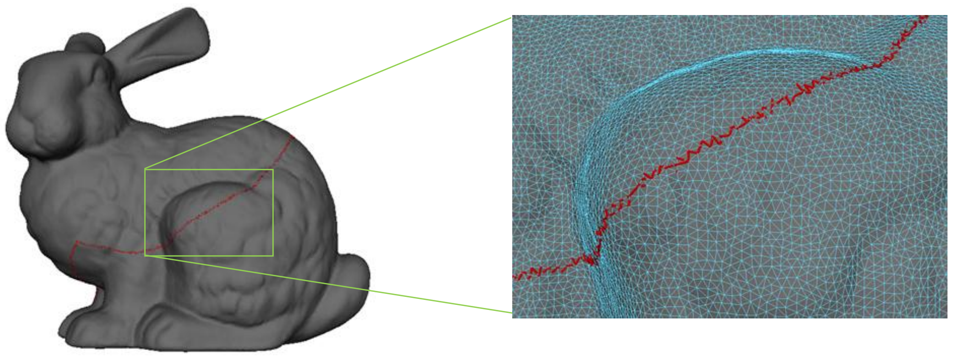


FIGURE A.3: Wavefront Shape of FWP-MMP Algorithm.

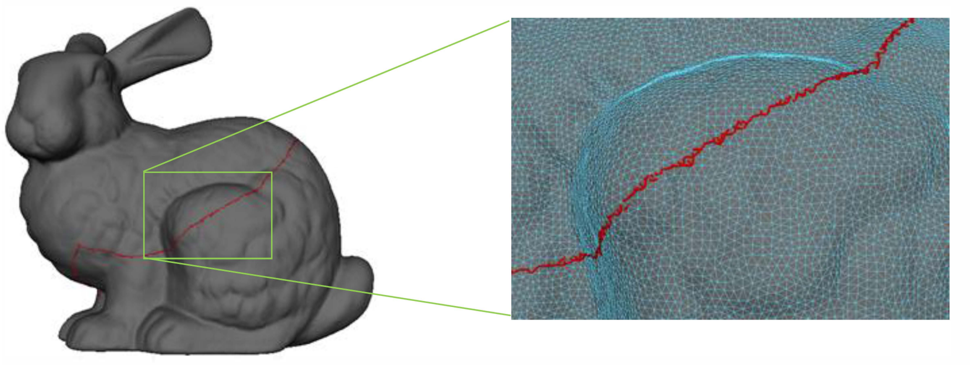


FIGURE A.4: Wavefront Shape of ICH Algorithm.

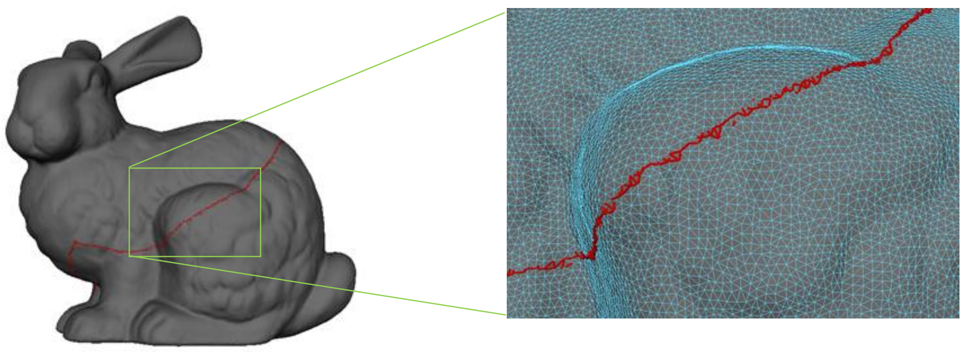


FIGURE A.5: Wavefront Shape of FWP-CH Algorithm.

Bibliography

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: http://dx.doi.org/10.1007/BF01386390 1, 3, 7, 13, 17

[2] C. Sommer, "Shortest-path queries in static networks," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 45:1–45:31, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2530531 1

[3] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979. [Online]. Available: http://doi.acm.org/10.1145/359156.359164 1

[4] J. O'Rourke, S. Suri, and H. Booth, *STACS 85: 2nd Annual Symposium on Theoretical Aspects of Computer Science Saarbrücken, January 3–5, 1985*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, ch. Shortest paths on polyhedral surfaces, pp. 243–254. [Online]. Available: http://dx.doi.org/10.1007/BFb0024013 1, 3, 10

[5] D. M. Mount, "On finding shortest paths on convex polyhedra." DTIC Document, Tech. Rep., 1985. 1, 3, 7, 10, 11

[6] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 193–215, 1986. [Online]. Available: http://dx.doi.org/10.1137/0215014 1, 2, 7, 10, 11, 13

[7] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," in *Foundations of Computer Science, 1987., 28th Annual Symposium on*, Oct 1987, pp. 49–60. 1, 2

[8] M. Kline, *Mathematical thought from ancient to modern times*. Oxford University Press, 1990, vol. 3. 2

[9] K. Polthier and M. Schmies, *Mathematical Visualization: Algorithms, Applications and Numerics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ch. Straightest Geodesics on Polyhedral Surfaces, pp. 135–150. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-03567-2_11 2, 18

[10] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of the National Academy of Sciences*, vol. 95, no. 15, pp. 8431–8435, 1998. 2, 10, 17

[11] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in heat: A new approach to computing distance based on heat flow," *ACM Trans. Graph.*, vol. 32, no. 5, pp. 152:1–152:11, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2516971.2516977 2, 18

[12] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987. [Online]. Available: http://dx.doi.org/10.1137/0216045 3, 7, 10, 13, 14, 20, 21, 24, 26, 27, 30, 31, 36

[13] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, ser. SCG '90. New York, NY, USA: ACM, 1990, pp. 360–369. [Online]. Available: http://doi.acm.org/10.1145/98524.98601 3, 11, 13

[14] S. Kapoor, "Efficient computation of geodesic shortest paths," in *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '99. New York, NY, USA: ACM, 1999, pp. 770–779. [Online]. Available: http://doi.acm.org/10.1145/301250.301449 3, 12

[15] J. O'Rourke, "Computational geometry column 35," *SIGACT News*, vol. 30, no. 2, pp. 31–32, Jun. 1999. [Online]. Available: http://doi.acm.org/10.1145/568547.568559 3, 12

[16] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 553–560, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073228 3, 10, 11, 12, 13, 43

[17] S.-Q. Xin and G.-J. Wang, "Improving chen and han's algorithm on the discrete geodesic problem," *ACM Trans. Graph.*, vol. 28, no. 4, pp. 104:1–104:8, Sep. 2009. [Online]. Available: http://doi.acm.org/10.1145/1559755.1559761 3, 11, 13, 41

[18] Y.-J. Liu, "Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures," *Computer-Aided Design*, vol. 45, no. 3, pp. 695 – 704, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448512002758 3, 11

[19] X. Ying, S.-Q. Xin, and Y. He, "Parallel chen-han (pch) algorithm for discrete geodesics," *ACM Trans. Graph.*, vol. 33, no. 1, pp. 9:1–9:11, Feb. 2014. [Online]. Available: http://doi.acm.org/10.1145/2534161 3, 11

[20] C. Xu, T. Wang, Y.-J. Liu, L. Liu, and Y. He, "Fast wavefront propagation (fwp) for computing exact geodesic distances on meshes," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 21, no. 7, pp. 822–834, July 2015. 3, 5, 11, 12, 31, 36, 40, 44

[21] J. Hershberger and S. Suri, "Practical methods for approximating shortest paths on a convex polytope in {R3}," *Computational Geometry*, vol. 10, no. 1, pp. 31 – 46, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772197000047 3, 8, 9

[22] S. Har-Peled, M. Sharir, and K. R. Varadarajan, "Approximating shortest paths on a convex polytope in three dimensions," in *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, ser. SCG '96. New York, NY, USA: ACM, 1996, pp. 329–338. [Online]. Available: http://doi.acm.org/10.1145/237218.237402 3, 8

[23] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan, "Approximating shortest paths on a convex polytope in three dimensions," *J. ACM*, vol. 44, no. 4, pp. 567–584, Jul. 1997. [Online]. Available: http://doi.acm.org/10.1145/263867.263869 3, 9, 12

[24] S. Har-Peled, "Approximate shortest paths and geodesic diameter on a convex polytope in three dimensions," *Discrete & Computational Geometry*, vol. 21, no. 2, pp. 217–231, 1999. [Online]. Available: http://dx.doi.org/10.1007/PL00009417 3, 9, 12

[25] K. P. Agarwal, S. Har-Peled, and M. Karia, "Computing approximate shortest paths on convex polytopes," *Algorithmica*, vol. 33, no. 2, pp. 227–242, 2002. [Online]. Available: http://dx.doi.org/10.1007/s00453-001-0111-x 3, 9

[26] S. Har-Peled, "Constructing approximate shortest path maps in three dimensions," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1182–1197, 1999. [Online]. Available: http://dx.doi.org/10.1137/S0097539797325223 3, 12

[27] K. R. Varadarajan and P. K. Agarwal, "Approximating shortest paths on a nonconvex polyhedron," *SIAM Journal on Computing*, vol. 30, no. 4, pp. 1321–1340, 2000. [Online]. Available: http://dx.doi.org/10.1137/S0097539799352759 3, 12

[28] T. Kanai and H. Suzuki, "Approximate shortest path on a polyhedral surface and its applications," *Computer-Aided Design*, vol. 33, no. 11, pp. 801 – 811, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448501000975 3, 12

[29] M. Balasubramanian, J. Polimeni, and E. Schwartz, "Exact geodesics and shortest paths on polyhedral surfaces," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 6, pp. 1006–1016, June 2009. 3, 13

[30] X. Ying, X. Wang, and Y. He, "Saddle vertex graph (svg): A novel solution to the discrete geodesic problem," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 170:1–170:12, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508363.2508379 3, 13

[31] J. S. B. Mitchell and C. H. Papadimitriou, "The weighted region problem: Finding shortest paths through a weighted planar subdivision," *J. ACM*, vol. 38, no. 1, pp. 18–73, Jan. 1991. [Online]. Available: http://doi.acm.org/10.1145/102782.102784 3, 13

[32] C. S. Mata and J. S. B. Mitchell, "A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract)," in *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, ser. SCG '97. New York, NY, USA: ACM, 1997, pp. 264–273. [Online]. Available: http://doi.acm.org/10.1145/262839.262983 3, 14

[33] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating weighted shortest paths on polyhedral surfaces," in *Proceedings of the Thirteenth Annual Symposium on Computational Geometry*, ser. SCG '97. New York, NY, USA: ACM, 1997, pp. 274–283. [Online]. Available: http://doi.acm.org/10.1145/262839.262984 3, 14, 15, 16

[34] M. Lanthier, A. Maheshwari, and J. R. Sack, “Approximating shortest paths on weighted polyhedral surfaces,” *Algorithmica*, vol. 30, no. 4, pp. 527–562, 2001. [Online]. Available: http://dx.doi.org/10.1007/s00453-001-0027-5 3, 14, 15

[35] M. Lanthier, D. Nussbaum, and J.-R. Sack, “Parallel implementation of geometric shortest path algorithms,” *Parallel Computing*, vol. 29, no. 10, pp. 1445 – 1479, 2003, high Performance Computing with geographical data. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819103001121 3, 15

[36] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J. R. Sack, *Algorithm Theory — SWAT'98: 6th Scandinavian Workshop on Algorithm Theory Stockholm, Sweden, July 8–10, 1998 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ch. An ϵ — Approximation algorithm for weighted shortest paths on polyhedral surfaces, pp. 11–22. [Online]. Available: http://dx.doi.org/10.1007/BFb0054351 3, 15, 16

[37] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, “Approximation algorithms for geometric shortest path problems,” in *Proceedings of the Thirty-second Annual ACM Symposium on Theory of Computing*, ser. STOC '00. New York, NY, USA: ACM, 2000, pp. 286–295. [Online]. Available: http://doi.acm.org/10.1145/335305.335339 3, 15, 16

[38] ——, “Determining approximate shortest paths on weighted polyhedral surfaces,” *J. ACM*, vol. 52, no. 1, pp. 25–53, Jan. 2005. [Online]. Available: http://doi.acm.org/10.1145/1044731.1044733 3, 15, 16

[39] L. Aleksandrov, H. N. Djidjev, H. Guo, A. Maheshwari, D. Nussbaum, and J.-R. Sack, “Algorithms for approximate shortest path queries on weighted polyhedral surfaces,” *Discrete & Computational Geometry*, vol. 44, no. 4, pp. 762–801, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00454-009-9204-0 3, 15

[40] Z. Sun and J. H. Reif, “On finding approximate optimal paths in weighted regions,” *Journal of Algorithms*, vol. 58, no. 1, pp. 1 – 32, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677404001191 3, 16

[41] C. Xu, Y.-J. Liu, Q. Sun, J. Li, and Y. He, “Polyline-sourced geodesic voronoi diagrams on triangle meshes,” *Computer Graphics Forum*, vol. 33, no. 7, pp. 161–170, 2014. [Online]. Available: http://dx.doi.org/10.1111/cgf.12484 4

[42] Y. jin Liu, Z. Chen, and K. Tang, “Construction of iso-contours, bisectors, and voronoi diagrams on triangulated surfaces,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 8, pp. 1502–1517, Aug 2011. 4, 45

[43] G. Peyré and L. D. Cohen, “Geodesic remeshing using front propagation,” *International Journal of Computer Vision*, vol. 69, no. 1, pp. 145–156, 2006. [Online]. Available: http://dx.doi.org/10.1007/s11263-006-6859-3 4

[44] Y. Liu, “Semi-continuity of skeletons in two-manifold and discrete voronoi approximation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 9, pp. 1938–1944, Sept 2015. 4

[45] S. Liu, X. Jin, C. C. L. Wang, and J. X. Chen, “Water wave animation on mesh surfaces,” *Computing in Science and Engg.*, vol. 8, no. 5, pp. 81–87, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2006.83 4

[46] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, and R. R. Martin, “Feature sensitive mesh segmentation,” in *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, ser. SPM ’06. New York, NY, USA: ACM, 2006, pp. 17–25. [Online]. Available: http://doi.acm.org/10.1145/1128888.1128891 4

[47] N. Zink and A. Hardy, “Cloth simulation and collision detection using geometry images,” in *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, ser. AFRIGRAPH ’07. New York, NY, USA: ACM, 2007, pp. 187–195. [Online]. Available: http://doi.acm.org/10.1145/1294685.1294716 4

[48] T.-Y. Kim, N. Chentanez, and M. Müller-Fischer, “Long range attachments - a method to simulate inextensible clothing in computer games,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’12. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2012, pp. 305–310. [Online]. Available: http://dl.acm.org/citation.cfm?id=2422356.2422399 4

[49] D. M. Mount, “Storing the subdivision of a polyhedral surface,” *Discrete & Computational Geometry*, vol. 2, no. 2, pp. 153–174, 1987. [Online]. Available: http://dx.doi.org/10.1007/BF02187877 8

[50] R. Dudley, “Metric entropy of some classes of sets with differentiable boundaries,” *Journal of Approximation Theory*, vol. 10, no. 3, pp. 227 – 236, 1974. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0021904574901208 9

[51] Y. Schreiber and M. Sharir, “An optimal-time algorithm for shortest paths on a convex polytope in three dimensions,” *Discrete & Computational Geometry*, vol. 39, no. 1, pp. 500–579, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00454-007-9031-0 9

[52] J. Hershberger and S. Suri, “An optimal algorithm for euclidean shortest paths in the plane,” *SIAM Journal on Computing*, vol. 28, no. 6, pp. 2215–2256, 1999. [Online]. Available: http://dx.doi.org/10.1137/S0097539795289604 9

[53] Y. Schreiber, “An optimal-time algorithm for shortest paths on realistic polyhedra,” *Discrete & Computational Geometry*, vol. 43, no. 1, pp. 21–53, 2009. [Online]. Available: http://dx.doi.org/10.1007/s00454-009-9136-8 9

[54] Y.-J. Liu, Q.-Y. Zhou, and S.-M. Hu, “Handling degenerate cases in exact geodesic computation on triangle meshes,” *The Visual Computer*, vol. 23, no. 9, pp. 661–668, 2007. [Online]. Available: http://dx.doi.org/10.1007/s00371-007-0136-5 10, 11

[55] B. Kaneva and J. O’Rourke, “An implementation of chen & han’s shortest paths algorithm,” in *Proceedings of the 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, Canada, August 16-19, 2000*, 2000. [Online]. Available: http://www.cccg.ca/proceedings/2000/8.ps.gz 11

[56] K. Clarkson, "Approximation algorithms for shortest path motion planning," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: ACM, 1987, pp. 56–65. [Online]. Available: http://doi.acm.org/10.1145/28395.28402 15

[57] R. J. Lipton and R. E. Tarjan, "A separator theorem for planar graphs," *SIAM Journal on Applied Mathematics*, vol. 36, no. 2, pp. 177–189, 1979. [Online]. Available: http://dx.doi.org/10.1137/0136016 16

[58] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996. [Online]. Available: http://www.pnas.org/content/93/4/1591.abstract 16, 17

[59] L. Yatziv, A. Bartesaghi, and G. Sapiro, "O(n) implementation of the fast marching algorithm," *Journal of Computational Physics*, vol. 212, no. 2, pp. 393 – 399, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999105003736 17

[60] L. Bertelli, B. Sumengen, and B. Manjunath, "Redundancy in all pairs fast marching method," in *Image Processing, 2006 IEEE International Conference on*, Oct 2006, pp. 3033–3036. 17

[61] D. Martínez, L. Velho, and P. C. Carvalho, "Computing geodesics on triangular meshes," *Computers & Graphics*, vol. 29, no. 5, pp. 667 – 675, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849305001299 18

[62] S.-Q. Xin and G.-J. Wang, "Efficiently determining a locally exact shortest path on polyhedral surfaces," *Computer-Aided Design*, vol. 39, no. 12, pp. 1081 – 1090, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010448507001959 18

[63] S.-Q. Xin, D. T. P. Quynh, X. Ying, and Y. He, "A global algorithm to compute defect-tolerant geodesic distance," in *SIGGRAPH Asia 2012 Technical Briefs*, ser. SA '12. New York, NY, USA: ACM, 2012, pp. 23:1–23:4. [Online]. Available: http://doi.acm.org/10.1145/2407746.2407769 18, 19

[64] H. Peng, J. Tang, H. Xiao, A. Bria, J. Zhou, V. Butler, Z. Zhou, P. T. Gonzalez-Bellido, S. W. Oh, J. Chen *et al.*, "Virtual finger boosts three-dimensional imaging and microsurgery as well as terabyte volume image visualization and analysis," *Nature communications*, vol. 5, 2014. 44