On Efficient Shortest Path Computation on Terrain Surface: A Direction-Oriented Approach (Technical Report)

Victor Junqiu Wei[#], Raymond Chi-Wing Wong[#], Cheng Long[‡] David M. Mount[†], Hanan Samet[†]
[#]The Hong Kong University of Science and Technology, Hong Kong
[‡]Nanyang Technological University, Singapore
[†]University of Maryland, USA
[#]victorwei@ust.hk, [#]raywong@cse.ust.hk, [‡]c.long@ntu.edu.sg, [†]{mount,hjs}@umd.edu

Abstract—With the advance of the geo-positioning technology, the terrain surface data has become increasingly popular and has drawn much research attention from both academia and industry. Answering a shortest-path query for a given source and a given destination on a terrain surface is a fundamental problem and has many applications including Geographical Information System and 3D virtual games. We observe that all existing exact algorithms are only aware of the position of the source point and is unaware of the information of the destination point. Motivated by this, in this paper, we propose an efficient algorithm, namely *direction-oriented algorithm (DIO Algorithm)*, for answering shortest-path queries on a terrain surface. The algorithm properly guides the search along a direction towards the destination instead of blindly searching all possible directions from the source point. To this end, we convert the geodesic shortest path problem to a shortest obstacle-free Euclidean path problem in the 2D planar unfolding of the terrain surface. Based on this conversion, we derive for each part of the terrain surface a lower bound on the length of the shortest path from the source to the destination passing through the part with a novel method. The lower bounds provide useful information that can be used to decide the visiting order of the parts on the terrain surface and guides the search of finding the destination quickly. Our experiments verified that our algorithm runs faster than the state-of-the-art by more than one order of magnitude.

Index Terms—shortest path queries, terrain surfaces, spatial database, location-based services

1 Introduction

Due to the advance of the geo-positioning and computer technologies, terrain surfaces have emerged as an important data object and has attracted much attention from both academia and industry [1]–[8]. It has been used in many applications such as Microsoft's Bing Maps and Google Earth in industry. Terrain surface data is usually represented by a set of *faces*, each of which corresponds to a triangle. Each face (or triangle) has three line segments called *edges*, which are connected with each other at three *vertices*. An example of a piece of terrain surface data is shown in Figure 1, where we have 22 faces, 35 edges and 14 vertices.

The *geodesic distance* between two given locations (or points) on the surface of the terrain is the length of the *shortest* path/route from one point to the other traveling along the surface. For example, in Figure 1, s and t are two points on the terrain surface and the shortest path from point s to point t is shown as a sequence of dashed line segments and denoted by GP. In Figure 1, the Euclidean distance between point s and point t, denoted by EP, is the length of the straight line segment between these two points. Note that the geodesic distance is usually quite different from the Euclidean distance. According to [1], the ratio of the geodesic shortest distance and the Euclidean distance is up to 300% on the real terrain datasets in their study. Clearly, GP is much larger than EP in Figure 1.

Answering shortest-path queries on the terrain surface is a common building block in many algorithms and has a wide range of applications. Here, we list a few of them. (1) In Geographic Information System (GIS), hikers would like to find the shortest path on the terrain surface to design a proper hiking trail [9]. In addition, many vehicles (e.g., Google Map camera cars and military vehicles) have their route planning based on the shortest path query on the terrain surface [10], [11]. (2) In some online 3D virtual games such as INGRESS and PokemonGo, the shortest path on the terrain surface provides a proper route for the players to travel from one place to another in mountainous areas. (3) In military tactical analysis, computing the shortest path on the terrain surface is very important for guiding the movement of the troops and equipment [12]. (4) With the rise of the Metaverse, 3D modeling of buildings and infrastructures in urban areas and mountains, hills and valleys in the rural areas become more and more popular [13], [14]. The shortest path on the terrain surface [13], [14] provides a route/path for people to travel from one place to another in their own virtual world.

While there are a large number of research studies for the shortest path query on terrain surfaces [5], [7], [8], [15]–[22], the methods proposed so far are still not efficient enough, which we elaborate on below. There are two branches of existing studies. The first branch targets exact algorithms. There are totally four exact algorithms [15]–[17], [22]. All of

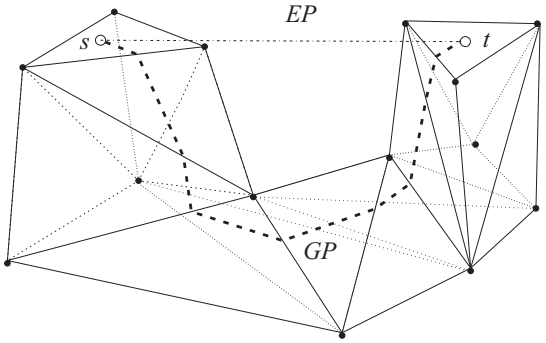

Fig. 1: An Example of Terrain Surface

these algorithms operate on-the-fly without pre-computing an index structure. Let N denote the total number of vertices defining the terrain surface. The time complexities of these four algorithms are $O(N^2 \log N)$, $O(N^2)$, $O(N^2 \log N)$ and $O(N^2 \log N)$, which are prohibitively expensive when N is large. For example, according to [8], the algorithm proposed in [16], which has the lowest time complexity among these four algorithms, took more than 300 seconds on a terrain with 200K vertices only.

The second branch targets approximation algorithms. Two representative on-the-fly approximation algorithms are [7], [8]. [7] computes the *shortest network path* on the terrain surface where the path returned passes through the edges of the terrain only. Therefore, the length of the path returned could be much larger than that of the shortest geodesic path. [8] finds a path on the terrain surface with an approximate ratio at most $(1 + \epsilon)$. The algorithm takes $O((N + N') \log(N + N'))$ time, where N' is the number of additional auxiliary points introduced to the terrain surface. But they are still not efficient enough to satisfy the real-time requirements of many applications. According to the experimental results in [8], the algorithm ran for more than 100 seconds on a terrain with 200K vertices even for a setting with a very loose error parameter $\epsilon = 0.25$. There are also three existing index-based approximation algorithms [20], [21], [23], which build a pre-computed indexing structure to accelerate the query processing. However, these two algorithms only provide approximate shortest paths and have huge time overheads and bulky space cost (of the indexing structures). Thus, they are infeasible to be applied to the cases where (1) the exact answer is required, or (2) the memory budget cannot afford to hold the bulky index. This motivates us to design an index-free exact shortest path algorithm in this paper.

We observe that all existing on-the-fly algorithms are only aware of the position of the source point and is unaware of the information of the destination point. As such, they explore all directions blindly which incurs a large execution time. Motivated by this, in this paper, we propose a destination-aware algorithm, namely *direction-oriented algorithm (DIO Algorithm)*, for computing the exact shortest geodesic path online. It does not incur additional space consumption with an indexing structure or any preprocessing overhead. Our algorithm first considers the terrain surface in the *planar unfolding* [24] by which we mean that the terrain surface is unfolded into a 2D plane. As such, this allows us to reformulate the shortest path problem as a obstacle-free shortest Euclidean path problem. We propose a new data structure, called *visibility tree*, which allows us to answer these queries. This is a rooted tree, whose root node corresponds to the start point s. Each non-root node in the tree corresponds to a vertex or an edge segment on the terrain surface. Our algorithm visits the nodes in a tree in a best-first fashion until the node o 'containing' t is visited. The geodesic path could be traced back by using the path from the root to o in the tree. A key component of our algorithm is to decide the priority of each node to be visited. In our algorithm, the priority is given to the edge segment or vertex o with a smaller estimated lower bound of the length of the shortest s-t path passing through o. The lower bound is derived based on the visibility information and the 2D geometry. As such, our algorithm is destination-oriented and the search is guided along a direction towards the position of t instead of searching all directions around s blindly. Our experiments verified that our method of estimating the lower bound is effective and our algorithm for the shortest path query outperforms the state-of-the-art by a notable margin.

Our contributions are threefold. First of all, we propose using a data structure called *visibility tree* and each node in the tree corresponds to a vertex or an edge segment on the terrain surface. We also develop a lower bound estimation method to calculate the lower bound of the length of the shortest s-t path passing through a given edge segment or a vertex. The lower bound serves as the priority of visiting each node in the visibility tree (i.e., correspondingly each edge segment or each vertex on the terrain surface). Our lower bound estimation method is lightweight and we theoretically prove the correctness of our lower bound estimation method. Second, based on this lower bound estimation method, we develop a novel exact shortest path computation algorithm, in which the lower bound estimation algorithm guides the search towards the destination quickly. Third, we conducted a thorough empirical study whose results demonstrate that our algorithm significantly outperforms all existing exact shortest path computation algorithms on the terrain surface by more than an order of magnitude.

The remainder of the paper is organized as follows. Section 2 reviews the related studies of our work. Section 3 formally presents our problem and introduces many notations to be used later. Section 4 presents our shortest path computation algorithm. Section 5 presents our empirical study. In Section 6, we discuss the extension of our algorithm to a variant of the terrain surface studied in this paper, namely *weighted terrain surfaces*. Finally, Section 7 concludes this paper.

2 Related Work

2.1 Exact Algorithms for Geodesic Shortest Path Queries

To the best of our knowledge, all existing exact geodesic shortest path algorithms are on-the-fly algorithms [15]–[17], [22] where no pre-computed data structures are required.

The first algorithm in this category is the MMP algorithm [15]. The MMP algorithm visits all faces in the

descending order of their distance to the source point and its time complexity is $O(N^2 \log N)$, where N is the number of vertices on the terrain surface. Later on, the VS algorithm [22] further improved the MMP algorithm by introducing some pruning rules to filter out some irrelevant faces. As such, the query processing could be accelerated. The third algorithm, namely *CH* [16], cuts and unfolds the terrain surface into a 2D plane and the unfolded surface in the 2D plane is a star-shaped polygon which has the source point as its center. As such, the geodesic distance between any point on the terrain surface and the source point is equal to their Euclidean distance on this 2D polygon. Besides, their shortest path on this 2D polygon could be converted to their shortest geodesic path on the original terrain surface. The time complexity of the CH algorithm is $O(N^2)$. However, one drawback of CH is that it must process the whole terrain, which is more than necessary when the source and the destination are close to each other. Motivated by this, ICH [4] further improved CH algorithm. It incrementally establishes the 2D polygon in a BFS fashion and visits each part of the polygon in the ascending order of their distance to the source point. The empirical performance of ICH is highly boosted despite that its time complexity is $O(N^2 \log N)$.

Our algorithm also interprets the geodesic shortest path algorithm as a shortest Euclidean path problem through the 2D unfolding. But our key innovations consist of (1) a tree structure for encoding the shortest path information in the 2D unfolding (each node of which corresponds to a vertex or an edge segment on the terrain surface) and (2) the destination-aware priority estimation for each vertex or edge segment in the unfolding (where the priority is the length of the shortest path from s to t passing through this vertex or edge segment). It is worth mentioning that it is non-trivial to estimate this priority for the geometric objects (i.e., vertices and edge segments) with favorable theoretical guarantee since the complicated geometric properties of these objects together with the terrain surface imposes many challenges. As such, significant research effort and more advanced techniques are highly required for such a design which will be demonstrated in our algorithm and theoretical analysis.

In summary, all existing exact algorithms are only aware of the position of the source point and are *destination-unaware*. As such, they are not able to make direction-oriented search as what our proposed algorithm does and could only blindly search all directions of the source point. This renders them significantly slower than ours.

2.2 Approximation Algorithms for Geodesic Shortest Path Queries

All existing on-the-fly approximation algorithms [8], [25], [26] follow the same framework. Specifically, they all introduce some auxiliary points, namely *Steiner points*, on the terrain surface and also some auxiliary edges, namely *Steiner edges*, and obtain a so-called *Steiner graph* $\mathcal{G}$ based on the points and edges introduced. In the query phase, it creates an edge between the source point s (resp. the destination point t) and each Steiner point on the face that s (resp. t) lies on and inserts it into the graph $\mathcal{G}$ and performs Dijkstra's algorithm from s to t. The time complexity of each on-the-fly approximation algorithm is $O((N + N') \log(N + N'))$, where N' is the number of Steiner points introduced. Their differences lie on the method that they use for introducing the Steiner points and Steiner edges. The methods *Fixed Scheme* [25] and *K-Algo* [8] uniformly place several Steiner points on each edge and connect each pair of Steiner points on the same face with a Steiner edge. *Unfixed Scheme* [26] places several Steiner points on each bisector of each face and connects every pair of Steiner points on each two adjacent faces with a Steiner edge.

Later on, index-based algorithms [19], [21], [23], [27] for the geodesic shortest path computation were proposed to further accelerate the query processing. The first attempt in this category is a Single-Source All-Destination algorithm [27], where the source point must be known apriori and kept fixed in the query phase. But the application scenarios are limited by the fixed source point. SP-Oracle [19] builds an indexing structure for the shortest path query processing on the Steiner graph of Unfixed Scheme. Inspired by [28]–[30], SE-Oracle [21], [23] indexes the geodesic distances and paths between a set of pre-defined points-of-interest with a technique called *Well-Separated Pair Decomposition*. EAR-Oracle proposes a *highway network*-based indexing structure for the geodesic distance qeuries between arbitrary points. The index-based algorithms accelerate the query processing with a pre-computed indexing structure. But they also bring the overhead of preprocessing time and the additional storage consumption for the bulky indexing structure.

Furthermore, the approximation algorithms could only find approximate results which prevent their usage in the applications where the exact distances are highly required.

2.3 Other Related Studies

We review some other related studies [2]–[8], [31]–[34] on the terrain surface in this section. Specifically, [2], [31] study the kNN query processing problem and propose to use a simplified low-resolution terrain surface constructed from the original data for pruning some irrelevant regions for better efficiency. [3] proposes a Voronoi diagram-based method for kNN queries and [4] studies the problem of monitoring the kNN in a dynamic setting. [6] studies reverse k nearest neighbors queries. [5] studies the problem of finding the shortest geodesic path satisfying a slope constraint. It is worth mentioning that although the algorithm in [5] could be applied to the geodesic shortest path finding without this constraint, there is no guarantee that it can return the exact geodesic shortest path [5], [21], [23] and its performance is inferior to that proposed in [4]. Besides, a plethora of research effort [32]–[34] has been put on the problem of energy-efficient path planning for autonomous unmanned vehicles (AUV). But in their problem setting, the AUV can only pass through several pre-defined links connecting two pre-selected nodes on the terrain surface, and the pre-selected nodes and pre-defined links comprise a terrain graph. The weight of each link whose physical meaning is the energy consumption of traversing the link is calculated by using the geometric properties of the terrain. The energy-efficient path in their papers is the shortest path on the

T, V, E, F	Terrain, Vertices, Edges and Faces.
s, t	Two arbitrary points on the terrain.
$\Pi_g(s,t)$	The geodesic shortest path from s to t.
$\Pi_g(s,t\|e)(resp.\Pi_g(s,t\|v))$	The geodesic shortest path from s to t passing through an edge segment e (resp. a vertex v).
$d_g(s,t)$	The geodesic distance from s to t.
$\mathcal{P}(s,p)$	The shortest path from s to p on 2D unfolding.
o	A node on the Visibility Tree.
r_o	The parent of o on the Visibility Tree.
$d(o)$	The associated distance of o.
$c(o)$	The corresponding vertex or edge segment of o.
$\overline{s}_o$	The light point of o and $c(o)$.

TABLE 1: Notations

terrain graph. Thus, their techniques do not apply to our problem since it is very unlikely that the geodesic shortest path only passes the pre-defined links. Given a polygon on a 2D plane, the Euclidean shortest path problem aims to find the shortest path between a given source point and a given destination point inside the polygon where the path is only allowed to pass through the interior of the 2D polygon. A lot of research effort [35]–[38] has been devoted into this research problem. The shortest path finding problem on terrain surfaces studied in this paper is a generic version of the obstacle-free shortest path finding problem on the 2D Euclidean space. In particular, the latter (the Euclidean shortest path finding) is a special case of the former problem (i.e., the shortest path finding problem on terrain surfaces) in the sense that the interior of a given polygon on the 2D Euclidean space in the latter problem is a special type of the terrain surface where all faces lie on the same plane. In our problem, besides the shortest path finding in the interior of a 2D polygon, we also need to find the optimal unfolding of the 3D terrain surface where the shortest distance from s to t in this unfolding is equal to their geodesic distance on the original terrain surface. It is a challenging task to find the optimal unfolding since there can be quite a number of distinct ways of unfolding of the terrain surface. To this end, we develop a novel data structure called *visibility tree* which encodes both the unfolding information that is dynamically maintained in the procedure of our direction-aware search and the shortest path information on the partially unfolded terrain surface. Note that all algorithms in this section have a different problem setting from our problem.

3 Problem Definition

Consider a terrain surface T. Let V be the set of all vertices on T, and E be the set of all edges on T. For example, in Figure 1, each solid point is a vertex and each solid line segment is an edge. The size of a terrain surface T, denoted by N, is defined to be the total number of vertices (That is $N = |V|$). Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v.

Consider two points s and t on the terrain surface T. A path, denoted by $\pi_g(s,t)$, from s to t on the terrain surface consists of a sequence $\mathcal{S}$ of line segments. Each line segment l in $\mathcal{S}$ lies on a face of the terrain surface and each pair of adjacent line segments share one endpoint. The length of a given line segment l is denoted by $|l|$. The length of the path $\pi_g(s,t)$ is the sum of the lengths of the line segments in $\mathcal{S}$ (i.e., $\sum_{l\in\mathcal{S}} |l|$). We denote the Euclidean distance between s and t by $d(s,t)$ (i.e., $d(s,t) = \sqrt{(x_s-x_t)^2+(y_s-y_t)^2+(z_s-z_t)^2}$). Based on the concepts above, we have the definition of *geodesic shortest path*.

Definition 1 (Geodesic Shortest Path). The *geodesic shortest path* between s and t, denoted by $\Pi_g(s,t)$, is defined to be the path with the shortest length between the two points on T.

We denote the Euclidean distance between s and t by $d(s,t)$ (i.e., $d(s,t) = \sqrt{(x_s-x_t)^2+(y_s-y_t)^2+(z_s-z_t)^2}$). Then, we further have a definition called *geodesic distance*.

Definition 2 (Geodesic Distance). The *geodesic distance* between s and t, denoted by $d_g(s,t)$, is defined to be the length of $\Pi_g(s,t)$.

Besides, in this paper, by 'point', we refer to an arbitrary point on the terrain surface, which may or may not be a vertex of the terrain surface. For example, in Figure 1, s and t are two points on the terrain surface but neither of them is a vertex of the terrain surface.

Consider the example in Figure 1. The geodesic shortest path between two points s and t is denoted by GP. The geodesic distance between s and t is equal to the sum of the lengths of all line segments on GP.

Now, we are ready to formally define the problem.

Problem 1. (Shortest Path on Terrain Surface) Given a terrain surface T and two points s and t on T, find the shortest path $\Pi_g(s,t)$ from s to t on the terrain surface T.

4 DIO Algorithm

In this section, we present our proposed algorithm, namely *direction-oriented algorithm (DIO Algorithm)*, for the shortest geodesic path query. We present that our problem is equivalent to a visibility problem in a 2D Euclidean space in Section 4.1 and then demonstrate the proposed *Visibility Tree* in Section 4.2. Next, we present two key components in the *Visibility Tree* construction, namely lower bound estimation and children propagation, in Section 4.3 and Section 4.4 respectively. Then, Section 4.6 presents theoretical analysis of the correctness and the time complexity of our algorithm.

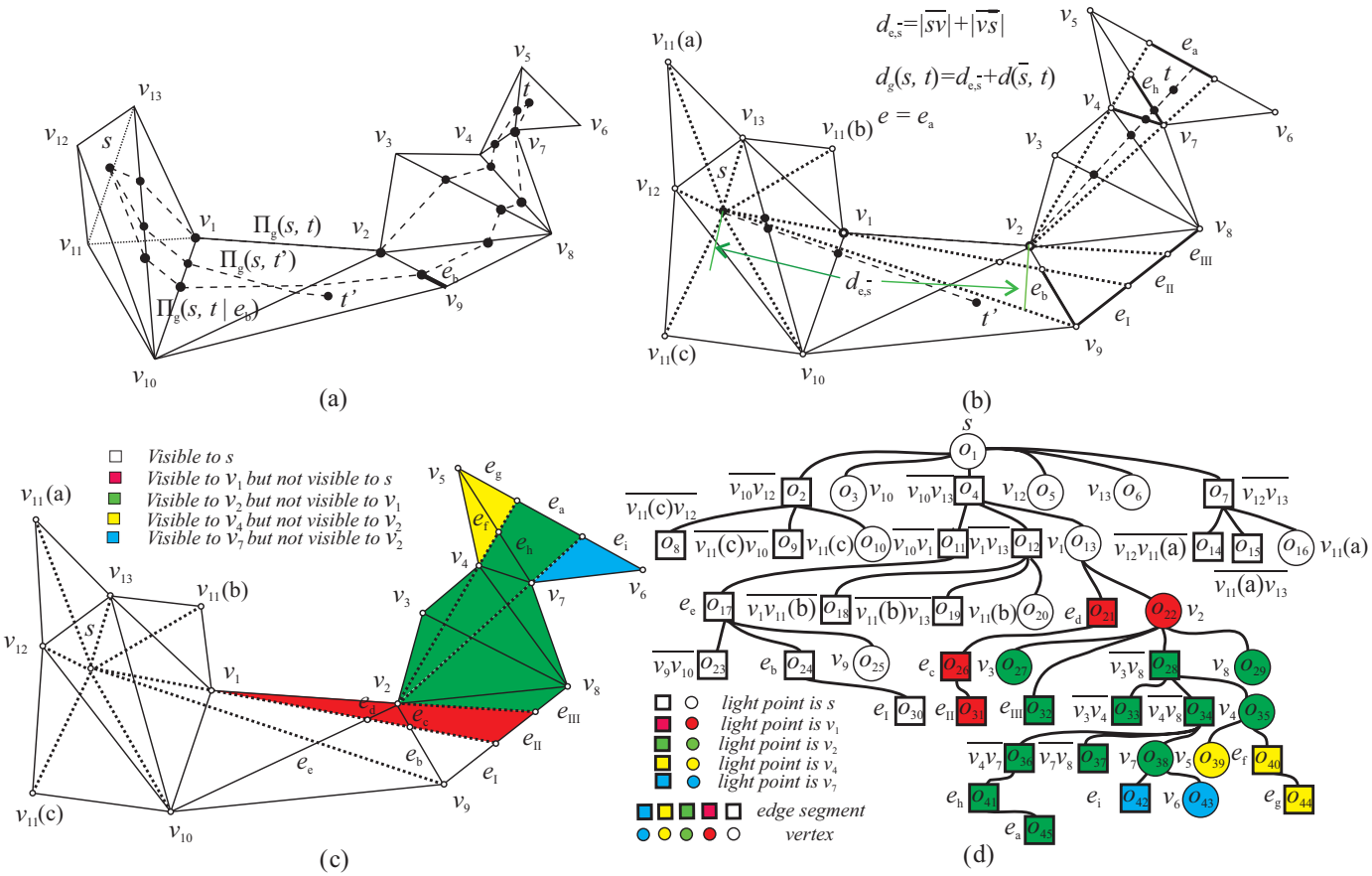

Fig. 2: An Illustration of DIO Algorithm

Vertex	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
Corresponding Node	o_{13}	o_{22}	o_{27}	o_{35}	o_{39}	o_{43}	o_{38}	o_{29}	o_{25}	o_3	o_{20}	o_5	o_6

TABLE 2: Vertex Table

4.1 Shortest Geodesic Path as A Visibility Problem

We demonstrate in the section that our geodesic shortest path problem is equivalent to a visibility problem in 2D Euclidean space. Consider the example shown in Figure 2. Figure 2(a) shows a terrain surface, the shortest geodesic path from s to t and the shortest geodesic path from s to t'. Figure 2(b) shows a planar unfolding [24] of the terrain surface in which each face is unfolded into the same plane. Note that in this figure, the original vertex v_{11} is split into three vertices in the unfolding and we denote them by $v_{11}(a)$, $v_{11}(b)$ and $v_{11}(c)$. Thus, as could be observed from Figure 2(b), all faces unfolded form a polygon and the we call this unfolded terrain surface the *planar unfolding* of T. If we consider the space outside the polygon as obstacles, we define the *shortest Euclidean path* from s to any point p, denoted by $\mathcal{P}(s,p)$, as the shortest path from s to p without any collision with the obstacles. In Figure 2(b), the shortest Euclidean path $\mathcal{P}(s,t')$ from s to t' is the line segment $\overline{st'}$ and the shortest Euclidean path $\mathcal{P}(s,t)$ from s to t consists of three line segments ($\overline{sv_1}$, $\overline{v_1v_2}$ and $\overline{v_2t}$) marked in a dashed line. Then, we proceed to show a lemma which connects the shortest geodesic path and the shortest Euclidean path in the unfolding.

Lemma 1. The shortest geodesic path $\Pi_g(s,p)$ from s to any point p on the terrain surface T coincides with the shortest Euclidean path $\mathcal{P}(s,p)$ in the unfolding of T.

Proof: We prove this lemma by contradiction. Assume that $\Pi_g(s,p)$ does not coincides with $\mathcal{P}(s,p)$ in the unfolding of T. Then, let $\mathcal{P}'(s,t)$ denote the path which coincides with $\Pi_g(s,p)$ in the unfolding. $\mathcal{P}'(s,t)$ has the same length as $\Pi_g(s,p)$ but $\mathcal{P}'(s,t)$ has a larger length than $\mathcal{P}(s,p)$ since $\mathcal{P}(s,p)$ is the shortest Euclidean path from s to t in the unfolding. Thus, the length of $\Pi_g(s,p)$ is larger than that of $\mathcal{P}(s,p)$. Consider the path $\Pi'_g(s,p)$ on the terrain surface which coincide with $\mathcal{P}(s,p)$ in the unfolding. $\Pi'_g(s,p)$ has the same length as $\mathcal{P}(s,p)$ but $\Pi'_g(s,p)$ has a larger length than $\Pi_g(s,p)$ since $\Pi_g(s,p)$ is the shortest geodesic path from s to t. Thus, we obtain that the length of $\Pi_g(s,p)$ is smaller than that of $\mathcal{P}(s,p)$. Contradiction! □

By this observation, the shortest geodesic path finding problem could be converted to the shortest Euclidean path finding problem in the planar unfolding. Given a point p in a planar unfolding of a terrain surface, we call another point p' in the planar unfolding *visible to* p if the line segment $\overline{pp'}$ is within the planar unfolding. In our algorithm, the planar unfolding is decomposed into several disjoint regions and each region R has an associated point p which is either the point s or a vertex of the unfolded terrain. Given a

region R whose associated point is p, we call R is a *traceable region* if (1) any point p' in the region is visible to p, and (2) the shortest Euclidean path from s to any point p' in the region passes through the point p. For the ease of the presentation, we call this associated point p *the light point of the region* R since for each traceable region (which our algorithm considers only), each point inside is visible to p. Similarly, each vertex v is also associated with a light point p. We call v is a *traceable vertex* if (1) it is visible to the light point p, and (2) the shortest Euclidean path from s to v passes through the light point p.

Example 1 (Traceable Regions, Light Points and Traceable Vertices). In Figure 2(c), there are five disjoint regions. The first one is the white region and s is the light point of the region. Any point p_w in the white region is visible to s and as such, the shortest Euclidean path from s to p_w is the line segment $\overline{sp_w}$. The second one is the red region. v_1 is the only vertex on the boundary of the white region which is adjacent to the red region and it is the light point of this region. Any point p_r in the red region is visible to v_1 but not visible to s and the shortest Euclidean path from s to p_r is (s, v_1, p_r). The third region is the green region. v_2 is the only vertex on the boundary of the red region which is adjacent to the green region and it is the light point of the green region. Any point p_g in the green region is visible to v_2 but not visible to v_1 and the shortest Euclidean path from s to p_g is (s, v_1, v_2, p_g). The last two regions are the yellow and blue regions. v_4 (resp. v_7) is the light point of the yellow (resp. blue) region and any point p_y (resp., p_b) in the yellow (resp., blue) region is visible to v_4 (resp., v_7) but not visible to v_2. As such, the shortest Euclidean path from s to p_y (resp., p_b) is (s, v_1, v_2, v_4, p_y) (resp., (s, v_1, v_2, v_7, p_b)). Thus, we obtain that each of the five regions is a traceable region and v_1 and v_2 are both traceable vertices. Besides, v_1 and v_2 are also the light points of the red region and the green region, respectively. □

Consider an arbitrary destination point t in a traceable region. The idea of our algorithm is to find the shortest Euclidean path from s to t in the unfolding of T. To this end, we find the light point p of the region R that t lies on and then, we could repeatedly find the light point p' of p and so on so forth until we find s. Finally, the sequence of the light points found together with t forms the shortest Euclidean path. Consider the point t in Figure 2(b). It lies on the green region whose light point is v_2. The light point of v_2 is v_1 and the light point of v_1 is s. Finally, we obtain that the shortest Euclidean path from s to t is (s, v_1, v_2, t).

It is worth mentioning that our algorithm does not explicitly create the unfolding, light points and regions and the information is established and maintained through a tree-like structure called *Visibility Tree* to be shown in the next section.

4.2 Visibility Tree & Back-Tracing Algorithm

This section presents a tree-like structure, namely *Visibility Tree*, which encodes the visibility information in the planar unfolding. We also present a *Back-Tracing Algorithm* which utilizes the visibility tree to find the shortest Euclidean path in the unfolding and also the geodesic shortest path on the original terrain surface.

We first introduce a key concept called *edge segment* as follows. We call an edge *a heterogeneous edge* if the edge has overlap with more than one region and we call it *homogeneous edge* otherwise. As could be observed from Figure 2(c), the edges $\overline{v_2v_{10}}$, $\overline{v_2v_9}$ and $\overline{v_8v_9}$ are all heterogeneous edges and the edges $\overline{v_{12}v_{13}}$, $\overline{v_9v_{10}}$ and $\overline{v_6v_7}$ are all homogeneous edges. Each homogeneous edge is naturally considered as an edge segment in our algorithm. Each heterogeneous edge e is decomposed into several disjoint edge segments, each of which is the overlap between e and one region in our algorithm. Consider the heterogeneous edge $\overline{v_8v_9}$. It is decomposed into three disjoint edge segments e_I, e_{II} and e_{III}, where e_I is the overlap between $\overline{v_8v_9}$ and the white region, e_{II} is the overlap between $\overline{v_8v_9}$ and the red region and e_{III} is the overlap between $\overline{v_8v_9}$ and the green region. In this figure, we use the hollow point to indicate each endpoint of each edge segment.

Next, we are ready to present the *Visibility Tree* which indexes all vertices and edge segments. The visibility information and the shortest Euclidean path information is naturally encoded in this tree structure. In this tree structure, the root corresponds to the source point s and any other node corresponds to either a vertex or an edge segment in the planar unfolding. Consider a node o in the Visibility Tree, we denote the corresponding vertex or edge segment of o in the unfolding by $c(o)$. We highlight three components of a node o in the tree as follows. (1) a light point $\overline{s}_o$ of o, which is the light point of $c(o)$ if $c(o)$ is a vertex and is the light point of the region containing $c(o)$ otherwise, (2) an associated distance d_o (which is equal to the length of the shortest Euclidean path from s to $\overline{s}_o$), and (3) its parent node, denoted by r_o, of o. Let $A(o)$ denote the set containing all ancestors of o in the tree. Each node o in the Visibility Tree satisfies three properties: (1) *visible property:* Any point p on $c(o)$ is visible to $\overline{s}_o$, (2) *traceable property*: each vertex or edge segment which has intersection with the path from s to any point on $c(o)$ (no matter $c(o)$ is a vertex or an edge segment) must belong to the set $\{c(o')|o' \in A(o)\}$, and (3) *co-located property*: if o is not the root node, there must be a face containing both $c(o)$ and $c(o_p)$, where o_p is the parent of o.

Consider a node o which corresponds to a vertex in the unfolding, we denote the corresponding vertex of o on the original terrain by $v(o)$. In addition to the tree structure, we store a hash table called the *vertex table*. The table stores a unique node called *corresponding node* for each vertex on the original terrain. We call a node o the *corresponding node* of a vertex v if $v(o)$ is v and the value of $D(o) = d_o + d(\overline{s}_o, c(o))$ is the smallest among all nodes whose corresponding vertex on the original terrain is v (i.e., $\arg\min_{D(o')}\{o'|v(o') = v\}$).

Example 2 (Visibility Tree). Consider the tree shown in Figure 2(d) which is the tree structure of the vertices and edge segments in Figure 2(c). In this figure, we use a square to denote that the node corresponds to an edge segment and use a circle to denote that the node corresponds to a vertex. There are totally 45 nodes in the tree, namely o_1, o_2, ..., and o_{45}. We put the corresponding vertex or edge segment of each node in the unfolding next to the node. In Figure 2(d), the light

point of each white node is s. Similarly, the light points of nodes in other colors are shown in the figure.

Consider the node o_{45} (whose corresponding edge segment is e_a). Its light point is v_2 and its associated distance $d_{o_{45}}$ is equal to $|\overline{sv_1}| + |\overline{v_1v_2}|$. Its parent $r_{o_{45}}$ is o_{41}. Let p denote a point on e_a. We proceed to show that o_{45} satisfies the three properties mentioned above. Since p is visible to v_2 in Figure 2(c), o satisfies the visible property. $A(o_{45})$ consists of $o_1, o_4, o_{13}, o_{22}, o_{28}, o_{34}, o_{36}, o_{41}$ and their corresponding vertices or edge segments are s, $\overline{v_{10}v_{13}}$, v_1, v_2, $\overline{v_3v_8}$, $\overline{v_4v_8}$, $\overline{v_4v_7}$, e_h, respectively. It could be verified through Figure 2(b) that the shortest Euclidean path from s to p intersects with each of them. Thus, o_{45} satisfies the traceable property. o_{41} (whose corresponding edge segment is e_h) is the parent of o_{45} in the tree, e_h lies on the same face as e_a and also intersects with the path $\mathcal{P}(s, p)$. As such, we obtain that o_{45} satisfies the co-located property.

Table 2 shows the vertex table which contains the corresponding node of each vertex. Consider the vertex v_{11}. There are three nodes in the tree whose corresponding vertex is v_{11} on the original terrain and they are node o_{10}, node o_{16} and node o_{20} (i.e., $v(o_{10}) = v(o_{16}) = v(o_{20}) = v_{11}$). Note that the light point of all three points is s and the associated distance of the three nodes is 0. The corresponding vertices of o_{10}, node o_{16} and node o_{20} in the unfolding are $v_{11}(c)$, $v_{11}(a)$ and $v_{11}(b)$ (i.e., $c(o_{10}) = v_{11}(c)$, $c(o_{16}) = v_{11}(a)$ and $c(o_{20}) = v_{11}(b)$), in which $v_{11}(b)$ has the smallest Euclidean distance to s. Thus, we obtain that the corresponding node of v_{11} is node o_{20}. Similarly, the corresponding node of any other vertex is stored in this table. □

Note that it is possible that an edge e is on the boundary of two regions. In this case, any point on e is visible to the light points of both regions and in our algorithm, e can be assigned to any region of them and correspondingly, the light point of the node corresponding to e will be that of the region assigned. The region assigned to e is determined by which region "reaches" e first in the tree construction and this is relevant to the geometric factors of T.

Consider an arbitrary destination point t on the terrain surface T. There exists three cases of t: (i) t lies on a vertex v of T, (ii) t lies on an edge of T excluding its two end-points, and (iii) t lies on the interior of a face of T. We define the *corresponding node* of t in the Visibility Tree under the three cases next.

- *Case (i).* Since t is a vertex on T, the corresponding node of t is the corresponding node of the vertex t (which is stored in the vertex table).
- *Case (ii).* Consider a point t lying on an edge ε of T. There are two cases of ε in the planar unfolding: (I) there is exactly one edge in the planar unfolding with the same end-points as ε; and (II) ε is split into two edges in the unfolding. Consider two edges $\overline{v_1v_2}$ and $\overline{v_{11}v_{13}}$ in Figure 2(a). $\overline{v_1v_2}$ falls into Case (I) and $\overline{v_{11}v_{13}}$ belongs to Case (II). Note that it is not possible that there are more than two edges in the planar unfolding with the same end-points as ε since ε is adjacent to exactly two faces. As such, there are at most two edge segments containing ε and we consider the edge segment e (whose corresponding node is o in the tree) with the minimum distance $d_o + d(\overline{s}_o, t)$. We call the node which corresponds to e in the tree the *corresponding node* of t.
- *Case (iii).* Given an edge segment e and the node corresponding to e (which is denoted by o), we call the face having both e and $c(r_o)$ on its boundary *eigen face* of e. In the example shown in Figure 2(c), the face $v_5v_6v_7$ is the eigen face of e_a since it has both e_a and its parent e_h on its boundary. Given a point t in the interior of a face of T, we call that an edge segment e *contains* t if (1) t is visible to s_o and (2) t lies on the eigen face of e. Consider the point t in Figure 2(b). t lies on the eigen face of e_a and it is also visible to the light point v_2 of $c(e_a)$ (i.e., o_{45}). Thus, we obtain that e_a contains t. We call the node which corresponds to the edge segment containing t the *corresponding node* of t. In this example, the corresponding node of t is o_{45}. It is worth mentioning that there is exactly one node o whose corresponding edge segment e contains t. The reason is as follows. Consider the face f containing t and the region R containing t. The edge segment e containing t is uniquely defined by the intersection of R and one edge of f. Note that all regions are disjoint. In Figure 2(c), t lies on the face $v_5v_6v_7$ and also the green region. The edge segment e_a containing t is uniquely defined by the intersection of the green region and the edge $\overline{v_5v_6}$ (i.e., an edge of the face).

Now, we are ready to present the *Back-Tracing algorithm*. Given an arbitrary point t on T, we could first find the corresponding node o of t in the *Visibility Tree*. Then, we find the path $\mathcal{P}$(=$(o_a, o_b, ..., o_k)$) from the root node to r_o in the tree and we denote the sequence of the corresponding vertices or edge segments of the nodes in $\mathcal{P}$ by P (i.e., $P = (c(o_a), c(o_b), ..., c(o_k))$). We extract all vertices in P and denote them by $v_a, v_b, ..., v_k$ in the order of their depth in the tree. We obtain that the path $P' = (s, v_a, v_b, ..., v_k, t)$ is the shortest Euclidean path from s to t. We proceed to find the intersection between P' and each vertex or edge segment in P and denote all intersection points as $p_a, p_b, ..., p_k$. The shortest geodesic path from s to t is $(s, p_a, p_b, ..., p_k, t)$.

Example 3 (Back-Tracing Algorithm). In Figure 2, consider the point t and the edge segment e_a containing t (i.e., o_{45} is the corresponding node of t in the Visibility Tree). The set P containing the corresponding vertices or edge segments of all nodes from the root node to o_{45} in the tree consists of $\overline{v_{10}v_{13}}$, v_1, v_2, $\overline{v_3v_8}$, $\overline{v_4v_8}$, $\overline{v_4v_7}$ and e_h. v_1 and v_2 are the two only vertices in P. As could be observed from Figure 2(b), the shortest Euclidean path from s to t is (s, v_1, v_2, t) and we indicate the intersection between the shortest Euclidean path and each node in P by a solid point. The sequence of the solid points forms the shortest geodesic path from s to t. □

We proceed to demonstrate how we construct the Visibility Tree. We dynamically maintain a priority queue $\mathcal{Q}$ which contains all created/visited nodes in the Visibility Tree. $\mathcal{Q}$ is initialized to be the root node of the tree which corresponds

to s. Its light point is assigned to be s and its associated distance is assigned to be 0. Then, we iteratively perform the following operations. We extract the top element o in $\mathcal{Q}$ and if o is the corresponding node of t, we safely terminate the algorithm and find the shortest geodesic path from s to t by using the Back-Tracing algorithm. Otherwise, we propagate all children of o in the Visibility Tree and insert them into $\mathcal{Q}$. Now, the only two issues left are (1) how to calculate the priority of each node in the Visibility Tree, and (2) how to propagate the children of each node o in the tree. The two issues will be illustrated in the next two sections, respectively.

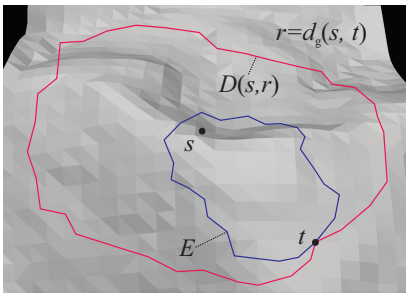

Fig. 3: Comparison of Search Spaces

In a high level intuition, our priority renders our algorithm being aware of both the source and the destination but each existing algorithm is only aware of the source and thus, ours is supposed to have better performance. Specifically, as shown in Figure 3, the final visited region of our algorithm is a narrow ellipse E with s and t on its two sides but the final visited region of each existing algorithm is a disk $D(s,r)$ centered at s with t on its boundary (where r is equal to the geodesic distance between s and t) and thus, our proposed priority will largely narrow down the search region visited and thus accelerate the shortest path algorithm. As could be noticed, A^* search [39] in the literature of graph also used the source- and destination-oriented lower bound but in the context of the terrain surface, the lower bound estimation is challenging and more technically involved as shown before since the most fundamental element of the search on the terrain surface involves the edge segment instead of vertex on graph.

Despite that *DIO* and *Polyanya* [35] share the high-level idea of A^* search in spirit, *DIO* involves a lot of non-trivial techniques specifically designed for the 3D terrain surfaces. We summarize the differences and the corresponding challenges as follows. (i) Different from the 2D Euclidean shortest path finding where the problem is considered in the 2D plane and the 2D polygon is already given, the shortest path finding on terrain surfaces must involve the unfolding procedure of the terrain surface (i.e., the processing of unfolding the 3D surface into a 2D polygon) which is incrementally maintained in our visibility tree. Note that there can be multiple ways of unfolding a terrain surface and thus, it is non-trivial for the shortest path finding on terrain surfaces with the correctness guarantee. In our algorithm, the visibility tree contains both unfolding information and the shortest path information and we also develop a non-trivial direction-aware query processing algorithm which returns the correct shortest path from s to t (with both the unfolding and the shortest path issues considered). (ii) Since the shortest path finding involves both the unfolding and the shortest path finding in the interior of the unfolded 2D polygon, the theoretical analysis is more technically involved and more challenging than that of the Euclidean shortest path finding.

4.3 Priority Estimation: Geometry-Based Lower Bound Estimation Algorithm

Consider a node o in the Visibility Tree. The priority of o in $\mathcal{Q}$ is equal to our estimated lower bound of the shortest geodesic path from s to t passing through its corresponding vertex or edge segment $c(o)$. Consider the case where $c(o)$ is an edge segment e. Let $\Pi_g(s,t|e)$ denote the s-t shortest path on the terrain surface passing through the edge segment e (i.e., it is the one with the minimum length among all paths from s to t passing through e on the terrain surface). Consider the example in Figure 2(a). It shows a terrain surface and there are two points s and t and two edge segments e_a and e_b on the surface shown in this example. The path $\Pi_g(s,t|e_a)$ (resp. $\Pi_g(s,t|e_b)$) is the shortest geodesic path from s to t passing through e_a (resp. e_b). In this example, $\Pi_g(s,t|e_a)$ is the shortest path $\Pi_g(s,t)$ from s to t on the terrain surface. Consider the case where $c(o)$ is a vertex v. Similarly, we denote the s-t shortest path on the terrain surface passing through a vertex v by $\Pi_g(s,t|v)$.

We first consider the case where $c(o)$ is an edge segment, denoted by e, and present our estimation method of the lower bound of $\Pi_g(s,t|e)$ for a given edge segment. Our method is based on the two observations: (1) for any point p on e, the geodesic distance $d_g(s,p)$ from s to p is equal to $d_o + d(\overline{s}_o, p)$ (based on the properties of traceable edge segment) and (2) for any point p on e, the geodesic distance $d_g(p,t)$ from p to t is lower bounded by $d(p,t)$. In a nutshell, we could derive the lower bound $\tilde{d}_g(s,t|e)$ by the distances in Euclidean space and we adopt the position information of $\overline{s}_o$, e and t in this lower bound estimation which is detailed as follows. Given an edge segment e, our estimation $\tilde{d}_g(s,t|e)$ is equal to $\min_{p\in e}\{d_g(s,p) + d(p,t)\}$. We will present that the estimation is a lower bound of the length of $\Pi_g(s,t|e)$ (i.e., the correctness) later in Lemma 4.

Now, we proceed to present how to estimate $\min_{p\in e}\{d_g(s,p) + d(p,t)\}$. The estimation consists of two steps, namely *Plane Rotation* and *Distance Computing*.

Step 1: Plane Rotation. Let Ł denote the straight line where the edge segment e lies on. As shown in Figure 4, the first step finds (1) the plane P_1 containing $\overline{s}_o$ and e and (2) the plane P_2 containing t and e and then rotate P_2 along Ł to make it coincide with P_1. Let $\overline{t}$ denote the position of t after the rotation. Note that if $\overline{s}_o$ and t are on Ł, we simply skip this first step (i.e. Plane Rotation) and go to the second step (i.e., Distance Computing) directly.

Step 2: Distance Computation. Then, consider the second step. In this step, we first find the proxy point $\overline{t}'$ of $\overline{t}$ such that the length of the shortest Euclidean path from $\overline{s}_o$ to $\overline{t}$ is equal to that of the shortest Euclidean path from $\overline{s}_o$ to $\overline{t}'$. We compute $\overline{t}'$ for the ease of the distance computation in some cases shown below. There are two cases of $\overline{t}'$.

- Case 1: $\overline{t}'$ locates at the same position of $\overline{t}$ if (1) $\overline{s}_o$ is on Ł or $\overline{s}_o$ and (2) $\overline{t}$ are on the different sides of Ł.

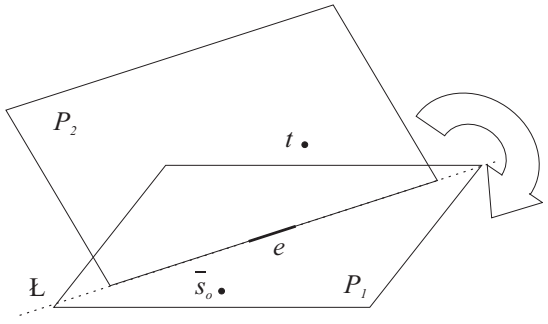

Fig. 4: An Illustration of Plane Rotation

- Case 2: Otherwise, the position of $\overline{t}'$ is symmetric to that of $\overline{t}$ w.r.t. Ł.

Let Ł′ denote the straight line containing $\overline{s}_o$ and $\overline{t}'$. There are two cases of Ł:

- Case (a): Ł′ intersects with e.
- Case (b): Ł′ does not intersect with e.

Thus, there are totally four disjoint cases, namely, Case 1(a), Case 2(a), Case 1(b) and Case 2(b). We proceed to present how to make the estimation in each case as we need which will be illustrated with some examples later. In Case 1(a) and Case 2(a), $\min_{p\in e}\{d_g(s,p)+d(p,t)\} = d_o + \min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\} = d_o + d(\overline{s}_o,\overline{t}')$. In Case 1(b) and Case 2(b), $\min_{p\in e}\{d_g(s,p)+d(p,t)\} = d_o + \min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\} = d_o + \min_{p\in\{p_1,p_2\}}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. We will elaborate this next with examples.

Example 4. (Four Cases in Step 2: Distance Computation) Figure 5, Figure 6, Figure 7 and Figure 8 show the examples of the four cases, respectively. We elaborate the position of $\overline{t}'$ in each case. Besides, we also show the correctness of our estimation of $\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ in each example and will formally prove it later.

Consider the example as shown in Figure 5 (i.e., Case 1(a)). Since $\overline{t}$ and $\overline{s}_o$ are on the different sides of Ł, the light point $\overline{t}'$ of $\overline{t}$ is located on $\overline{t}$. Since the line Ł′ intersects with e (note that Ł′ is defined to be the line passing through $\overline{s}_o$ and $\overline{t}'$), the shortest distance from $\overline{s}_o$ to $\overline{t}'$ is $d(\overline{s}_o,\overline{t}')$. Besides, by the rotation procedure, we obtain that for any point p on e, $d_g(s,p) = d_o + d(\overline{s}_o,p)$ and $d(p,t) = d(p,\overline{t}')$. Thus, we obtain that $\min_{p\in e}\{d_g(s,p)+d(p,t)\} = d_o + d(\overline{s}_o,\overline{t}')$ and our estimation in this case is $d_o + d(\overline{s}_o,\overline{t}')$.

Consider the example as shown in Figure 6 (i.e., Case 2(a)). Since $\overline{s}_o$ is not on Ł and $\overline{s}_o$ and $\overline{t}$ are on the same side of Ł, the light point $\overline{t}'$ of $\overline{t}$ is the reflection of $\overline{t}$ w.r.t. Ł. The estimation of $\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ and its correctness of the estimation are the same as those of Case 1(a). Note that in this case, although $\overline{t}'$ is not located on $\overline{t}$, it still holds that for any point p on e, $d(p,t) = d(p,\overline{t}')$ since $\overline{t}'$ and $\overline{t}$ are symmetrical w.r.t. Ł and Ł contains e and the rotation takes Ł as the axis, where Ł contains e.

Consider the example as shown in Figure 7 (i.e., Case 1(b)). Since $\overline{t}$ and $\overline{s}_o$ are on the different side of Ł, the light point $\overline{t}'$ of $\overline{t}$ is located on $\overline{t}$. In this case, the estimation is $\min_{p\in\{p_1,p_2\}}\{d_o+d(\overline{s}_o,p_1)+d(p_1,\overline{t}'), d_o+d(\overline{s}_o,p_2)+d(p_2,\overline{t}')\}$. In other words, $\arg\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ is one end-point of the edge segment e.

Consider the example as shown in Figure 8 (i.e., Case 2(b)). Since $\overline{s}_o$ is not on Ł and $\overline{s}_o$ and $\overline{t}$ are on the same side of Ł, the light point $\overline{t}'$ of $\overline{t}$ is the reflection of $\overline{t}$ w.r.t. Ł. The estimation in this case is the same as that of Case 1(b). □

Consider the case where $c(o)$ is a vertex v. Our estimation method of the lower bound of $\Pi_g(s,t|v)$ is simply equal to $d_o + d(v,t)$. We will present that the estimation is a lower bound of the length of $\Pi_g(s,t|v)$ (i.e., the correctness) later in Lemma 3.

4.4 Children Propagation Method for A Node in Visibility Tree

Given a node o (o may correspond to a vertex or an edge segment) in the Visibility Tree, in this section, we present how to find all children of o in the tree (which corresponds to several newly propagated vertices and edge segments). In a nutshell, the newly propagated edge segments and vertices are the visible ones that the light point of the current node o could see through its corresponding vertex or edge segment $c(o)$ in the planar unfolding of the terrain T. If one newly propagated edge segment e has overlap with one existing edge segment e', we divide the overlap part from each of these two edges segments into two disjoint edge segment parts. Consider a newly created node o'. In the end of the propagation, our algorithm makes sure that for any point p contained by $c(o')$, o' is the one in all existing nodes with the minimum value of $d_{o'} + d(\overline{s}_{o'},p)$.

Since this propagation problem is an incremental visibility maintenance problem, we adopt a standard technique [4] for this. In a nutshell, the newly propagated edge segments and vertices are the visible ones that the light point of the current node o could see through its corresponding vertex or edge segment $c(o)$ in the planar unfolding of the terrain T. If one newly propagated edge segment e has overlap with one existing edge segment e', the algorithm in [4] divides the overlap part from each of these two edges segments into two disjoint edge segment parts. Consider a newly created node o'. In the end of the propagation, the algorithm in [4] makes sure that for any point p contained by $c(o')$, o' is the one in all existing nodes with the minimum value of $d_{o'}+d(\overline{s}_{o'},p)$. Note that the vertex in our paper corresponds to the so-called 'pseudo-source window' in [4] and the edge segment in our paper corresponds to the 'interval window' in [4]. Thus, in this paper, we propagate from a node which corresponds to a vertex by using the propagation method of 'pseudo-source' window in [4]. The associated light point (resp. associated distance) of this type of newly propagated nodes is assigned to be the vertex (resp. the distance from s to the vertex) obtained by the propagation method in [4]. We propagate from a node which corresponds to an edge segment by using the propagation method of 'interval window' in [4]. The associated light point (resp. associated distance) of this type of newly propagated nodes is assigned to be the *image* of e (resp. the distance from s to this image) obtained by the propagation method in [4].

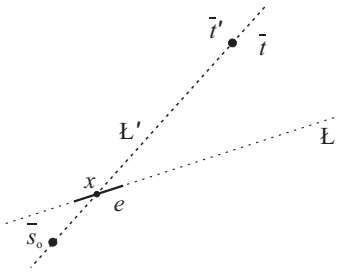

Fig. 5: Distance Computation (Case 1(a))

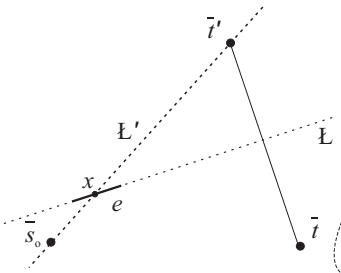

Fig. 6: Distance Computation (Case 2(a))

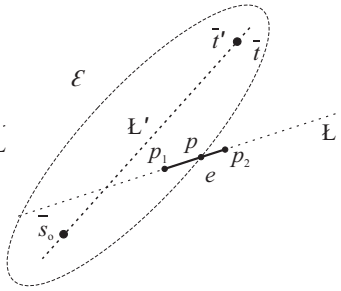

Fig. 7: Distance Computation (Case 1(b))

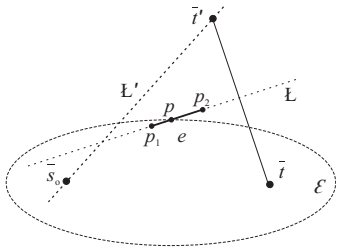

Fig. 8: Distance Computation (Case 2(b))

4.5 Putting Things All Together

Our algorithm expands the terrain surface by visiting all nodes in the Visibility Tree starting from the root node until the corresponding node of t is visited. In our algorithm, a priority queue (i.e., a min-heap) is maintained to store the visited nodes (which is similar to the Dijkstra's algorithm in graphs). The priority of each node o is equal to our estimated lower bound, denoted by $\tilde{d}_g(s,t|c(o))$, of the length of $\Pi_g(s,t|c(o))$ instead of the distance from s to $c(o)$. Our algorithm visits the nodes in their estimated lower bound as illustrated in Section 4.3. Algorithm 1 shows our proposed shortest geodesic path finding algorithm. Lines 1-5 present the initialization of our algorithm. Initially, we create a priority queue (i.e., a min-heap) $\mathcal{Q}$ and assign it to be $\emptyset$ (Line 1). Line 2-5 create the first node and insert it into $\mathcal{Q}$. In Line 2, we create the root node of the Visibility Tree which is a copy of s. In Line 3, we assign its associated light point to be s and assign its associated distance to be 0. In Line 4, we assign its priority to be 0. After that, we push the node into $\mathcal{Q}$ (Line 7). Line 6-18 contain the steps of the path finding. In each iteration of the while loop, we pop the top element in the queue (Line 7). If it is the corresponding node of t, we simply utilize the Back-Tracing method to find the shortest path from s to t and return it (Lines 9-10). If not, we propagate more nodes and compute their associated distances and images by using the children propagation method. Line 12 propagates several children from x. Lines 14-16 estimate the priority of each node propagated and put it into $\mathcal{Q}$.

Algorithm 1: DIO Algorithm

```
   Data: A Terrain Surface T(V, E, F) and a source point s and a
         destination point t on T
   Result: The geodesic shortest path from s to t on the terrain
           surface T
   /* Step 1: Initialization                                  */
   /* Step 1.1: Initialize the priority queue                 */
1  Initialize an empty priority queue Q = ∅;
   /* Step 1.2: Create the first node                         */
2  Create the root node of the Visibility Tree which corresponds
     to s;
3  Assign its associated image to be s and assign its associated
     distance to be 0;
4  Assign its priority to be 0;
5  Push the node into Q;
   /* Step 2: Path Finding                                    */
6  while True do
7      Pop the top element x from Q;
       /* Step 2.1: Find the path if x is the
          corresponding node of t                              */
8      if x is the corresponding node of t then
9          Find the shortest geodesic path Π_g(s,t) by using the
             Back-Tracing algorithm;
10         return Π_g(s,t)
11     end
       /* Step 2.2: Propagate more nodes                       */
12     Propagate a set S of new nodes from x and compute their
         associated distances and images by using the children
         propagation method;
13     for each element o in S do
14         Assign x to be the parent of o;
15         Estimate d̃_g(s,t|c(o)) by using the method present in
             Section 4.3 and assign them to be the priorities of o;
16         Push o into Q;
17     end
18 end
```

4.6 Theoretical Analysis

In this section, we formally prove the correctness and time complexity of our algorithm.

Given a node o in the Visibility Tree, we call that o is a *traceable node* if o satisfies the three properties: for any point p on $c(o)$, (1) $d_g(s,p) = d_o + d(\bar{s}_o, p)$, where d_o is the associated distance of o, (2) the light point of o lies on one vertex of the corresponding path of $\Pi_g(s,p)$ in the unfolding space, and (3) $c(r_o)$ in the Visibility Tree has intersection with $\Pi_g(s,p|c(o))$, where r_o is the parent of o in the Visibility Tree.

Lemma 2. Each node created by our algorithm is a traceable node and satisfies the visible property, the traceable property and the co-located property.

Proof: Since we propagate the edge segments and vertices by using the propagation method in [4], by Lemma 2.1 and the follow-up analysis of this lemma in [4], we obtain that each node created in our algorithm satisfies the first and the second properties of the traceable node. Next, we prove that o created by our algorithm satisfies the third property of the traceable edge segment. By the first property, we obtain that the triangle defined by e and the light point $\bar{s}_o$ of e is on the plane of the planar unfolding of the traced sequence of e and the triangle is in the inferior of the polygon of the planar unfolding. Thus, we obtain that the shortest path from $\bar{s}_o$ to p is a straight edge segment in the unfolding. Besides, by the first property, we obtain that $\bar{s}_o$ is on the path $\Pi_g(s,p)$ in this unfolding. By our algorithm, we obtain that o is propagated from r_o and by the propagation method in [4], we obtain that r_o has an intersection with $\overline{\bar{s}_o p}$ (which is part of $\Pi_g(s,p)$ in the planar unfolding). Thus, we obtain that r_o has an intersection with $\Pi_g(s,p)$ and o satisfies the third property. □

Lemma 3. Given a vertex v considered in our algorithm

whose corresponding node is o, $d_o + d(v,t)$ is at most the length of $\Pi_g(s,t|v)$.

Proof: The path $\Pi_g(s,t|v)$ consists of two parts. The first one is the shortest geodesic path $\Pi_g(s,v)$ from s to v and the second one is the shortest geodesic path $\Pi_g(v,t)$ from v to t. By our algorithm d_o is equal to the length of $\Pi_g(s,v)$ and since $d(v,t)$ is equal to the Euclidean distance between v and t, we obtain that $d(v,t)$ is at most the length of $\Pi_g(v,t)$. Thus, we finally obtain that $d_o+d(v,t)$ is at most the length of $\Pi_g(s,t|v)$. □

Lemma 4. Consider a given edge segment e considered in our algorithm. $\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ is at most the length of $\Pi_g(s,t|e)$.

Proof: Let p^* denote any point on $\Pi_g(s,t|e)\cap e$ (note that the intersection of $\Pi_g(s,t|e)$ and e may not be a point and an extreme case is that $\Pi_g(s,t|e)$ contains e and here, p^* is any point on $\Pi_g(s,t|e)\cap e$.). Thus, the length of $\Pi_g(s,t|e)$ is equal to $d_g(s,p^*)+d_g(p^*,t)$.

Let p' denote $\arg\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ (and thus, $d_g(s,p')+d_g(p',t) = \min_{p\in e}\{d_g(s,p)+d(p,t)\}$). By the definition of p', $d_g(s,p')+d(p',t) \le d_g(s,p^*)+d(p^*,t)$. Since $d_g(p^*,t)\ge d(p^*,t)$, we obtain that $d_g(s,p')+d(p',t)\le d_g(s,p^*)+d_g(p^*,t)$ which is the desired result. □

Theorem 1. Our estimation of $\min_{p\in e}\{d_g(s,p)+d(p,t)\}$ is correct, where e is an edge segment considered in our shortest path algorithm.

Proof: By Lemma 2, for any point p on e, $d_g(s,p) = d_o + d(\overline{s}_o,p)$, where o is the corresponding node of e. Thus, we obtain that $\min_{p\in e}(d_g(s,p)+d(p,t) = d_e + \min_{p\in e}(d(\overline{s}_o,p)+d(p,t)$. Since the Plane Rotation step rotate P_2 on Ł which contains e, we obtain that for any point p on e, $d(p,t)=d(p,\overline{t})$. With the two equations above, we obtain that $\min_{p\in e}(d_g(s,p)+d(p,t) = d_o+\min_{p\in e}(d(\overline{s}_o,p)+d(p,\overline{t})$. Then, it suffices to prove that $\min_{p\in e}(d(\overline{s}_o,p)+d(p,\overline{t})$ is equal to $d(\overline{s}_o,\overline{t}')$ (resp. $\min_{p\in\{o_1,o_2\}}(d(\overline{s}_o,p)+d(p,\overline{t}))$ in Case 1(a) and Case 2(a) (resp. Case 1(b) and Case 2(b)).

Consider Case 1(a) and Case 2(a). By Triangle inequality, we obtain that for any point p, $d(\overline{s}_o,p)+d(p,\overline{t}')\ge d(\overline{s}_o,\overline{t}')$. By the definition of $\overline{t}'$ (i.e., $\overline{t}'$ locates at either that of $\overline{t}$ or the position symmetric to that of $\overline{t}$ w.r.t. Ł), we obtain that for any point p on Ł, $d(p,\overline{t}) = d(p,\overline{t}')$. Thus, it must be true that $\min_{p\in Ł}(d(s,p)+d(p,t) = \min_{p\in Ł}(d(s,p)+d(p,t')$. Since x is on the edge segment $\overline{s\overline{t}'}$ and on the line Ł, we obtain that $d(\overline{s}_o,x)+d(x,\overline{t}') = d(\overline{s}_o,\overline{t}')$ and $d(\overline{s}_o,x)+d(x,\overline{t}) = d(\overline{s}_o,x)+d(x,\overline{t}')$. Thus, we obtain that $\min_{p\in Ł}(d(\overline{s}_o,p)+d(p,\overline{t}')) = d(\overline{s}_o,x)+d(x,\overline{t}') = d(\overline{s}_o,\overline{t}') = \min_{p\in Ł}(d(\overline{s}_o,p)+d(p,\overline{t}))$. Together with the fact that x is on the edge segment e, we obtain that $\min_{p\in e}(d(\overline{s}_o,p)+d(p,\overline{t}) = d(\overline{s}_o,x)+d(x,\overline{t}') = d(\overline{s}_o,\overline{t}')$.

Consider Case 1(b) and Case 2(b). Let $\mathcal{E}$ denote the ellipse whose focuses are $\overline{s}_o$ and $\overline{t}$ such that the length of its major axis is $\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. We first present three useful lemmas to show some intermediate results as follows. Note that $\mathcal{E}\cap e$ and $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$ are two sets containing some points.

Lemma 5. The ellipse $\mathcal{E}$ intersects with e (i.e., $\mathcal{E}\cap e\neq\emptyset$) and their intersection is $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$.

Proof:

We first prove that any point p' in $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$ must be in $\mathcal{E}\cap e$. Since the focuses of $\mathcal{E}$ are $\overline{s}_o$ and $\overline{t}$ and the length of its major axis is $\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$, p' must be on $\mathcal{E}$ and p' must be in e by the definition of $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. Thus, p' is in $\mathcal{E}\cap e$.

Then, we prove that any point p'' in $\mathcal{E}\cap e$ must be in $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. Since p'' is in $\mathcal{E}$, we obtain that $d(\overline{s}_o,p'')+d(p'',\overline{t}) = \min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. Besides, since p'' is in e, we obtain that p'' is in $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$.

□

Lemma 6. $\mathcal{E}\cap e$ (i.e., $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$) contains exactly one point.

Proof:

By Lemma 5, we obtain that $\mathcal{E}\cap e$ contains at least one point. Then, it suffices to prove that $\mathcal{E}\cap e$ contains at most one point.

We then prove this (i.e., $\mathcal{E}\cap e$ contains at most one point) by contradiction. Suppose that $\mathcal{E}\cap e$ (i.e., $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$) contains at least two points. Let p_1 and p_2 denote two points in $\mathcal{E}\cap e$. Since $p_1,p_2\in\mathcal{E}$, the edge segment $\overline{p_1,p_2}$ excluding p_1 and p_2 is in the interior of $\mathcal{E}$. From the property of an ellipse, $d(\overline{s}_o,p')+d(p',\overline{t})$ is less than the length of its major axis (i.e., $\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$), where p' is a point in $\overline{p_1,p_2}$ excluding p_1 and p_2. Besides, since $p_1,p_2\in e$, e contains the edge segment $\overline{p_1,p_2}$. This meas that $p'\in e$ and $d(\overline{s}_o,p')+d(p',\overline{t}) < \min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$. Contradiction.

□

Lemma 7. In Case (b), Ł is not a tangent line of $\mathcal{E}$.

Proof:

We first prove this by contradiction. Suppose Ł is a tangent line of $\mathcal{E}$ as Figure 9 shows. Consider the three lines $\overline{\overline{s}_o,o}$, $\overline{o,\overline{t}'}$ and $\overline{o,\overline{t}}$. Since Ł is a tangent line of $\mathcal{E}$ where the intersection is o, by [40] (see the Section 'The Normal bisects the angle between the lines to the foci'), we obtain that Ł bisects the angle the lines $\overline{\overline{s}_o,o}$ and $\overline{o,\overline{t}}$. Since $\overline{t}'$ is a reflection point of $\overline{t}$ w.r.t. Ł, we obtain that Ł bisects the angle $\angle\overline{t}o\overline{t}'$. Thus, we obtain that Ł coincides with the line $\overline{\overline{s}_o,o}$ (i.e., o is on the line Ł). Then, Ł′ must intersect with e. Contradiction.

□

By Lemma 5 and Lemma 6, we obtain that $\mathcal{E}\cap e = \arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$ and $|\mathcal{E}\cap e| = 1$. We prove the correctness of our estimation in Case 1(b) and Case 2(b) by contradiction. Suppose that (i.e., $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}$) is neither $\{o_1\}$ nor $\{o_2\}$. Let x denote the point contained in $\mathcal{E}\cap e$.

By Lemma 7, we obtain that Ł is not a tangent line of $\mathcal{E}$. Then, there must be a point x' on e such that x' is in the interior of $\mathcal{E}$ (as shown in Figure 7 and Figure 8). By the definition of ellipse, we obtain that $d(\overline{s}_o,x')+d(x',\overline{t})\le d(\overline{s}_o,x)+d(x',\overline{t})$ since the length of the major axis of $\mathcal{E}$ is $d(\overline{s}_o,x)+d(o,\overline{t})$. But this contradicts with the assumption that $\arg\min_{p\in e}\{d(\overline{s}_o,p)+d(p,\overline{t})\}=\{x\}$. □

Theorem 2. Our algorithm returns the shortest geodesic path from s to t correctly.

Proof: Let o denote the corresponding node of t and by the termination condition of our algorithm, in the last iteration, we find the corresponding node o of t. By Lemma 3, Lemma 4 and Theorem 1, the priority of o in $\mathcal{Q}$ is an lower bound of the length of the path $\Pi_g(s,t|c(o))$.

If t lies on a vertex or an edge segment (i.e., Case (i) and Case (ii) of t), then our Back-Tracing algorithm must find the shortest geodesic path from s to t by Lemma 2. Then, consider the last possible case (i.e., Case (iii)) of t, where t lies on the interior of a face. In this case, $c(o)$ is an edge segment and $c(o)$ contains t (i.e., t is visible to $s_{c(o)}$). Consider the point p on $c(o)$ such that $\overline{s_{c(o)}p}$ passes through t. Thus, we obtain that the geodesic shortest path $\Pi_g(s,p)$ from s to p passes through t. By Lemma 2, our Back-Tracing algorithm finds the intersections between each vertex or edge segment and the path $\Pi_g(s,p)$ which consists of $s, p_1, p_2, ..., p_k, p$. Together with the face that $\Pi_g(s,p)$ passes through t and p and t are located on the same face, we obtain that the path (i.e., $(s, p_1, p_2, ..., p_k, t)$) that our algorithm returns is the shortest geodesic path from s to t.

□

Let $\mathcal{E}_{s,t}$ denote the ellipse on the x-y plane whose focuses are s and t such that the length of its major axis is $d_g(s,t)$. Let $\mathcal{N}_{s,t}$ =#$\{p \in V|$ the projection of p on x-y plane is inside $\mathcal{E}_{s,t}\}$.

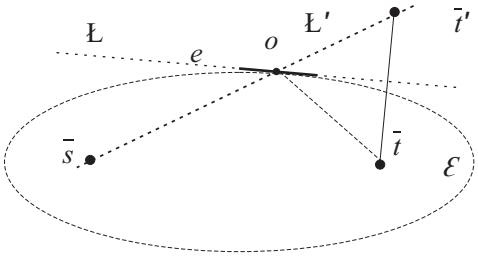

Fig. 9: An Illustration of Proof of Lemma 7

Theorem 3. The running time and space consumption of DIO algorithm are $O(\mathcal{N}_{s,t}^2 \log \mathcal{N}_{s,t})$ and $O(\mathcal{N}_{s,t}^2)$, respectively.

Proof:

We first show the number of visited vertices on the terrain surface by our algorithm in the following lemma.

Lemma 8. The number of vertices visited by our algorithm is $\mathcal{N}_{s,t}$.

Proof: We first prove that the projection of each vertex visited by our algorithm on the x-y plane are inside the ellipse $\mathcal{E}_{s,t}$. Consider any vertex v that is visited by our shortest path algorithm. Since the key of the root of the priority queue $\mathcal{Q}$ maintained in our algorithm is at most $d_g(s,t)$, it must be true that $d_g(s,v) + d(v,t) \leq d_g(s,t)$. Since for any two arbitrary points o_1 and o_2 on the terrain surface, it must be true that $d_g(o_1,o_2) \geq d(o_1,o_2) \geq d_{xy}(o_1,o_2)$, we obtain that for any visited vertex v, $d_{xy}(s,v) + d_{xy}(v,t) \leq d_g(s,t)$, where $d_{xy}(\cdot)$ denote the distance between the projections of two points on the x-y plane. Thus, we obtain that the projection of each vertex visited by our algorithm on the x-y plane lies in the ellipse $\mathcal{E}_{s,t}$ by the definition of the ellipse $\mathcal{E}_{s,t}$.

Reversely, we proceed to prove that each vertex, denoted by v, whose projection on the x-y plane is outside the ellipse $\mathcal{E}_{s,t}$ is not visited by our algorithm. According to the definition of the ellipse $\mathcal{E}_{s,t}$, we obtain that $d_{xy}(s,v) + d_{sy}(v,t) > d_g(s,t)$. Note that the length of the major axis of $\mathcal{E}_{s,t}$ is $d_g(s,t)$. Since for any two arbitrary points o_1 and o_2 on the terrain surface, it must be true that $d_g(o_1,o_2) \geq d(o_1,o_2) \geq d_{xy}(o_1,o_2)$, we obtain that for any visited vertex v, $d_g(s,v) + d_g(v,t) > d_g(s,t)$. By our algorithm, the key of each element considered in our priority queue $\mathcal{Q}$ is smaller than or equal to $d_g(s,t)$. Thus, we obtain that v is not visited by our algorithm. □

By [4], for a terrain surface with N vertices, each heterogeneous edge can only have at most N edges segments. Besides, each homogeneous edge has exactly one edge segment. Since the vertices and the edges of a terrain surface form a planar graph, we obtain that the number of edges on the terrain surface is $O(N)$ by [41]. Thus, the total number of nodes in the Visibility Tree is $O(N^2)$. By Lemma 8, the number of vertices visited by our algorithm is $\mathcal{N}_{s,t}$ which is equivalent to the fact that our algorithm only visited a terrain with $\mathcal{N}_{s,t}$ vertices. We obtain that the number of edge segments (i.e., the elements considered in our priority queue $\mathcal{Q}$) is $O(\mathcal{N}_{s,t}^2)$. Since each push or pop operation takes $O(\log \mathcal{N}(s,t))$ time for a priority queue with at most $O(\mathcal{N}_{s,t}^2)$ elements, we obtain that the running time of our algorithm is $O(\mathcal{N}_{s,t}^2 \log \mathcal{N}_{s,t})$.

□

5 Empirical Studies

Dataset	No. of Vertices	Resolution	Region Covered
BH (L)	146,547	30 meters	$14km \times 10km$
EP (L)	164,238	30 meters	$10.7km \times 14km$
SF (L)	172,186	30 meters	$14km \times 11.1km$
BH (H)	1,318,844	10 meters	$14km \times 10km$
EP (H)	1,392,236	10 meters	$10.7km \times 14km$
SF (H)	1,539,082	10 meters	$14km \times 11.1km$

TABLE 3: Dataset Statistics

5.1 Experimental Setup

We conducted our experiments on a Linux machine with 2.67 GHz CPU and 48GB memory. All algorithms were implemented in C++.

Datasets. Following some existing studies on terrain data [3], [5], [21], [31], we used three real terrain surfaces, namely Bearhead (in short, BH), Eaglepeak (in short, EP) and San Francisco South (in short, SF) and these datasets can be downloaded from this link [42]. Table 3 shows the dataset statistics. Each of the three terrain surfaces has two different versions with different resolution and sizes. Thus, there are totally six different datasets considered, namely *BH (L)*, *EP (L)*, *SF (L)*, *BH (H)*, *EP (H)* and *SF (H)*.

Algorithms. We tested our DIO algorithm and four existing exact shortest path algorithms on the terrain surface, namely MMP [15], VS [22], CH [16] and ICH [17]. Note that other existing algorithms can only find approximate geodesic shortest paths and thus, we do not test them in the experiment since they have different problem settings from ours. We obtained the source code of MMP from [43] and also the source code of CH and ICH from the webpage of the author

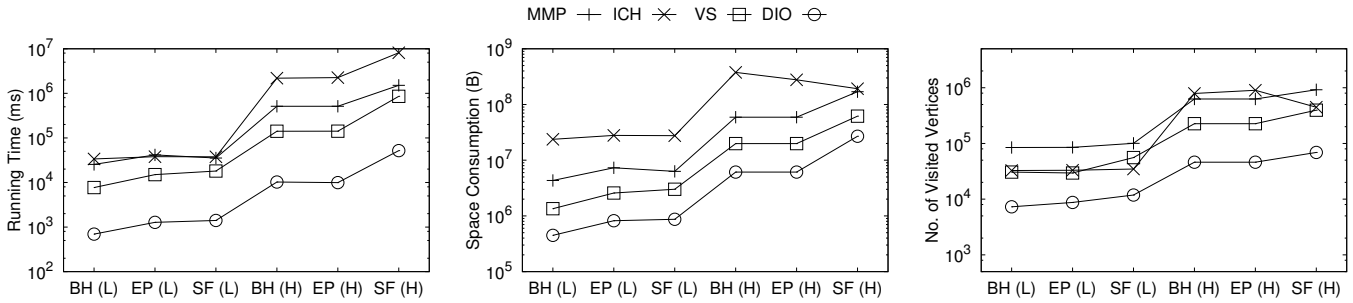

Fig. 10: Running Time, Space Consumption and No. of Visited Vertices of Each Algorithm on All Datasets

of [17]. We implemented the VS algorithm [22] by ourselves. The four baseline algorithms considered in the experiment were written in C++. Since the CH algorithm [16] is proved to be inferior to ICH and CH has a significantly larger running time than MMP and ICH according to the result of [4], we safely exclude CH for the better clarity.

Query Generation. Each shortest path query contains two query points, one as the source and the other as the destination. To study the effect of the distance between the two query points (i.e., the source and the destination), we generate 10 different groups of queries, namely $Q_1, Q_2, ..., Q_{10}$, for each dataset as follows. We first obtain the maximum (resp. minimum) pairwise geodesic distance between all vertices and we denote the distance as d_{max} (resp. d_{min}). Then, $\forall i \in [1, 10]$, we insert 100 pairs of vertices (s, t) into Q_i and we make sure that the geodesic distance $d_g(s, t)$ between s and t is in the range $[d_{min} + \frac{(d_{max} - d_{min}) * i}{10}, d_{min} + \frac{(d_{max} - d_{min}) * (i+1)}{10}]$. As such, the geodesic distance between each pair in Q_i is larger than Q_{i-1}, where $i \in [2, 10]$.

Factors & Measurements. Two factors, namely the query distance (the geodesic distance between the source and the destination) and N (the number of vertices on a terrain surface), were studied. Three measurements, namely (1) *running time* (which is the time for answering a shortest path query), (2) *space consumption* (which is the memory cost for running the algorithm) and (3) *No. of visited vertices* (which is the number of vertices visited by the algorithm and measures the size of the regions that each algorithm explores on the terrain surface) were used for evaluating the algorithms. For the query time, 100 queries were answered and the average running time was returned.

5.2 Experimental Results

Figure 10 shows the running time, space consumption and no. of visited vertices of each algorithm on all the six datasets. We used all queries contained in Q_1, Q_2, ..., Q_{10} and reported the average running time of the 1000 queries contained in all ten groups for each algorithm. In Figure 10(a), we could observe that (1) our algorithm has the smallest running time which is smaller than the best existing algorithm (i.e., VS) around 1 order of magnitude; (2) MMP and ICH are the slowest two algorithms and they have very similar performance and this result is consistent with that of [4]; (3) The running time of VS is 3-5 times smaller than that of MMP which is consistent with the result of [22]. In Figure 10(b), we find that (1) our algorithm is the most space-efficient and its space consumption is several times smaller than that of the best existing algorithm (i.e., VS); (2) MMP and ICH have the largest space consumption and this result is consistent with that of [22]; In Figure 10(c), we observe that (1) our algorithm has the smallest no. of visited vertices which is several times smaller than the best existing algorithm (i.e., VS). This result verifies that our destination-aware algorithm is very effective and only visited a small region of the terrain surface compared with existing algorithms; (2) The no. of visited vertices of VS is smaller than that of MMP and ICH in most cases which is consistent with the result of [22].

Effect of Query Distance. We studied the effect of query distance by testing the 10 groups of the queries on each high-resolution dataset. Note that the query distance is monotonically increasing from Q_1 to Q_{10}. The results on the BH (high resolution) datasets is shown in Figure 11. Figure 11(a) presents the running time of each algorithm in the 10 query groups. As could be observed from the figure, the running time of each algorithm grows up with the increase of the query distance. The running time of our algorithm is 1-3 orders smaller than the 3 baselines. Figure 11(b) presents the space consumption of each algorithm. Our algorithm significantly outperforms all the 3 baselines by a notable margin and is the most space-efficient one. Figure 11(c) demonstrates the number of vertices visited by each algorithm on the terrain surface. Our algorithm visited several times fewer vertices than the best existing algorithm. The result confirms that the destination-awareness of our algorithm is effective and the lower bound estimation in our algorithm provides a tight lower bound. As such, our algorithm only visited a much smaller region on the terrain surface than all existing algorithms and the query processing is highly boosted. The results on EP (H) and SF (H) are shown in Figure 12 and Figure 13, respectively. Their results are similar to those of BH (H).

Scalability Test. We tested the scalability of each algorithm considered on five synthetic datasets with sizes from $\{0.5M, 1M, 1.5M, 2M, 2.5M\}$. Each synthetic dataset with N vertices is a simplified terrain surface from an enlarged BH (H) dataset (4.2M vertices). Note that each simplified terrain surface covers the same region as the original BH dataset with a different simplification ratio. The enlarged BH dataset was generated from the BH (H) dataset as follows. On each face of BH (H), we added a new vertex on its geometric center and add a new edge between the new vertex and each of the three vertices on the face. Then, we adopted a terrain toolkit [5] to simplify the enlarged dataset.

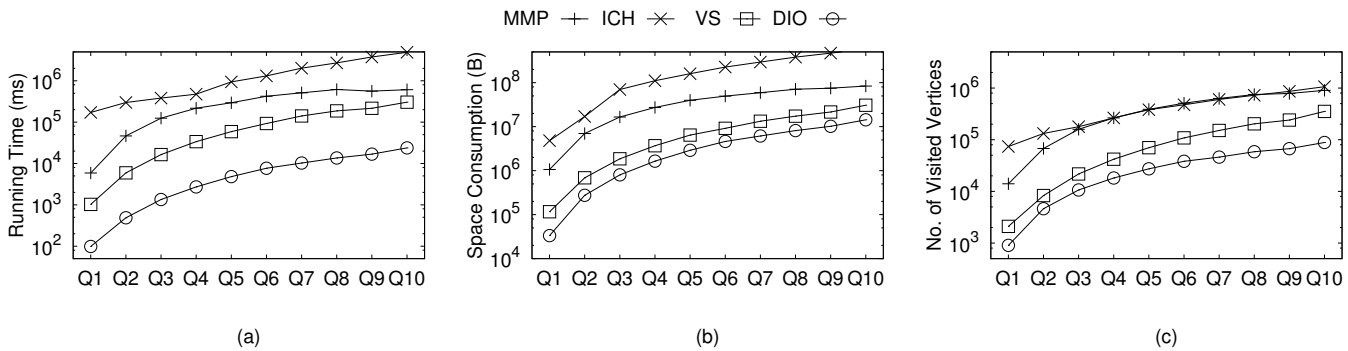

Fig. 11: Effect of Query Distance on BH (high resolution) dataset

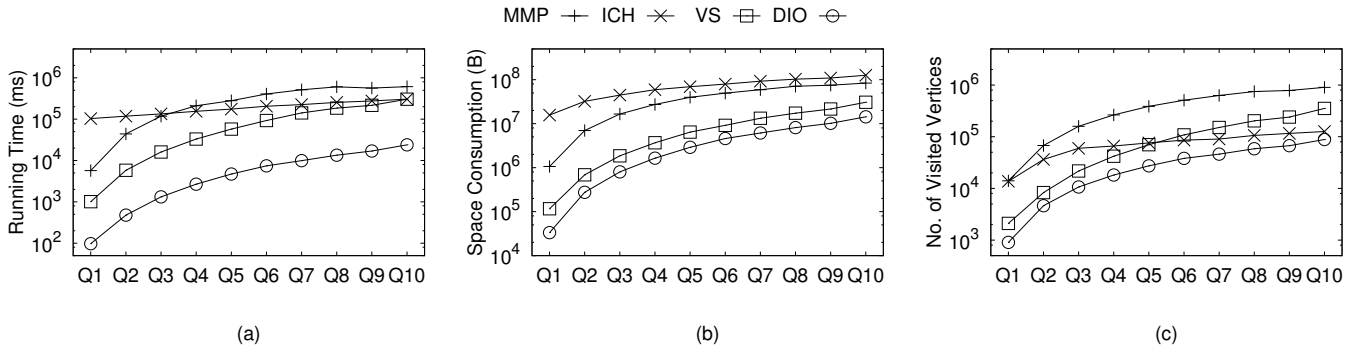

Fig. 12: Effect of Query Distance on EP (high resolution) dataset

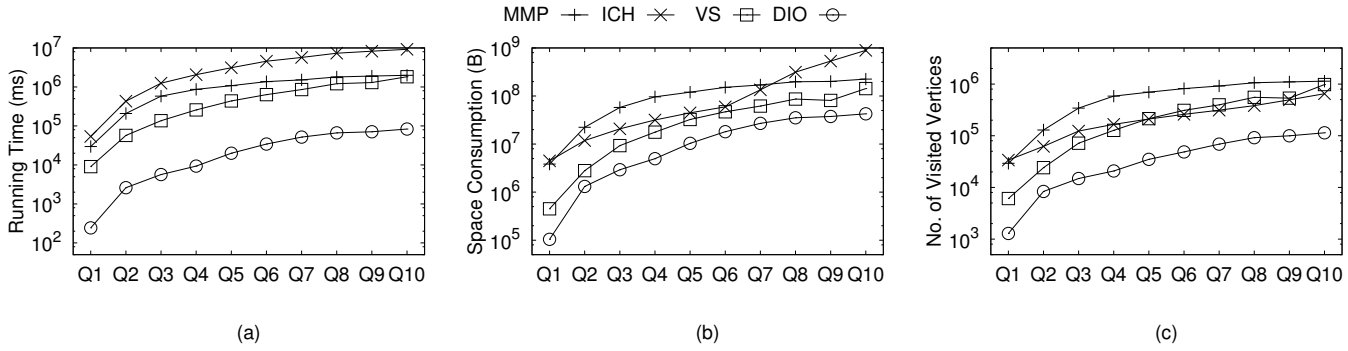

Fig. 13: Effect of Query Distance on SF (high resolution) dataset

Figure 14 shows the results of the scalability test. As the figure shows, the running time, space consumption and no. of visited vertices of each algorithm are all monotonically increasing with the increase of the data size. Our method has a running time smaller than that of all existing algorithms by more than one order of magnitude.

5.3 Case Study of Geodesic Shortest Path Queries

In this section, we conducted a case study in one application scenario of *Path Advisor* [44] which is a widely used web-based and mobile app in the Hong Kong University of Science and Technology (HKUST). This app provides location-based services on the campus of this university including the navigation and POI recommendation, etc. based on a 3D terrain surface in this university containing 1,789 vertices.

We consider the shortest path query from the location of the atrium, which is a landmark of the university, and also the lobby of the main building to the canteen 'Passion'. Figure 15 demonstrates the user interface of the Path Advisor and the details of this query mentioned above. In this figure, there is a menu on the left side which provides the search option and also legends. On the right side, it shows the 3D terrain model of the campus. The desired shortest path is shown in the green path in Figure 15. We report the experimental results of our algorithm and all baselines considered in Table 4 below. In our results, we observe that (1) all competitors suffer from their very large query time (i.e., larger than 2 seconds) which prevent from its usage in this Path Advisor system which requires real-time response; (2) our algorithm DIO has the smallest space consumption and the smallest query time which is smaller than that of others by more than one order of magnitude. It is quite light-weight and enjoys its neglectable storage overhead and instant query response.

Algorithm	Space Consumption (KB)	Query Response Time (ms)
MMP	253	3,281
ICH	4,563	2,934
VS	119	2,089
DIO	24	87

TABLE 4: Performances of All Algorithms in Our Case Study of Shortest Path Query

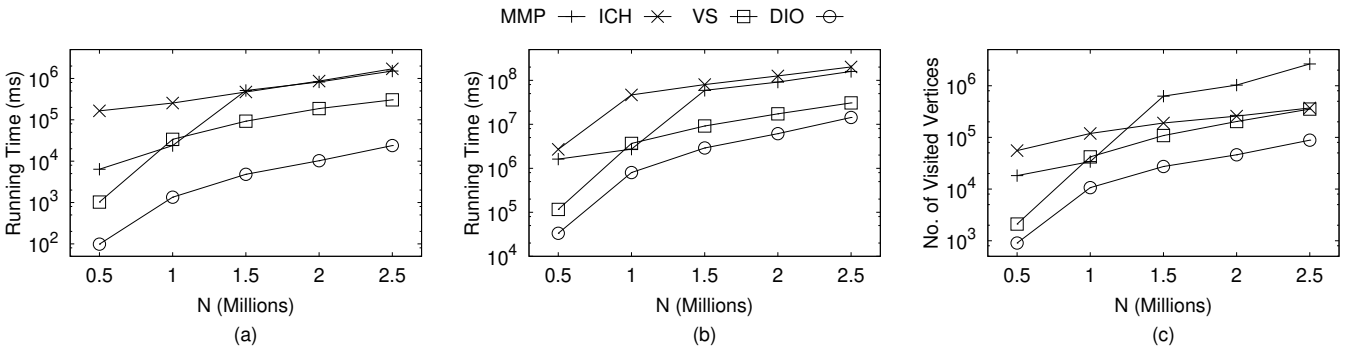

Fig. 14: Scalability Test

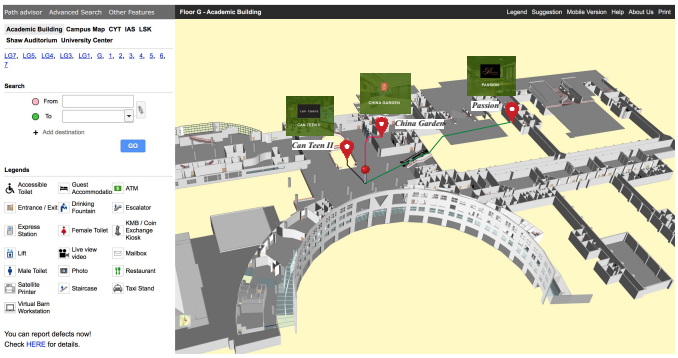

Fig. 15: An Illustration of Path Advisor in HKUST

6 Extension to Weighted Terrain Surface

Weighted terrain surface is a variant of the terrain surface studied in the paper. The same as the terrain surface which is studied in this paper, a weighted terrain surface also has a set of vertices, a set of edges and a set of faces. Each edge is adjacent to two vertices and each face has three adjacent vertices and three adjacent edges. Each vertex $v \in V$ has three coordinate values, denoted by x_v, y_v and z_v. But differently, in the weighted terrain surface, each face is assigned a positive real-valued weight and the weight information captures different travel cost (e.g., travel effort and energy consumption) in different faces and the weight information encodes many different features such as slope, terrain type (e.g., sand, wet land), obstacles, etc. [20], [25], [45]. Consider two points s, t and let $\pi_g(s,t)$ denote a path between them on a weighted terrain surface. Formally, $\pi_{wg}(s,t)$ consists of a sequence X of line segments and each segment lies on a unique face of the terrain surface. Given a line segment $x \in X$, we denote the unique face that x lies on by f_x and we denote the length of x by $l(x)$. The length of $\pi_{wg}(s,t)$ on the weighted terrain surface, denoted by $l(\pi_{wg}(s,t))$, is defined to be $\sum_{x \in X}(w(f_x) \cdot l(x))$ which is the sum over the product of the length of each line segment in X and the weight of the face it lies on. The *geodesic shortest path on the weighted terrain surface* between s and t, denoted by $\Pi_{wg}(s,t)$, is defined to be the path between s and t on the terrain surface with the smallest length (i.e., $\Pi_{wg}(s,t) = \arg\min_{\pi_{wg}(s,t)}\{l(\pi_{wg}(s,t))\}$). The *weighted geodesic distance* between s and t, denoted by $d_{wg}(s,t)$, is defined to be the length of $\Pi_{wg}(s,t)$.

Although our technique is originally designed for the unweighted terrain, it could be easily adapted to the weighted terrain surface with some minor modifications (by replacing each original geodesic distance involved with its weighted version). We also empirically compared our algorithm with the baselines and the result is shown in Table 5.

Algorithm	Space Consumption (MB)	Path Query Time (ms)
MMP	31.23	231
ICH	35.89	1,235
VS	5.64	186
DIO	0.32	16

TABLE 5: Performances of All Algorithms on Weighted Terrain Surface (BH, low resolution)

6.1 Experimental Result Summary

Our geodesic shortest path processing algorithm consistently outperforms the state-of-the-art algorithms, i.e., *MMP*, *ICH* and *VS*, in terms of all measurements (i.e., running time, space consumption, and the number of visited vertices) and enjoys excellent scalability. The speed-up of our algorithms compared with the existing algorithms is more than an order of magnitude. The space consumption and the no. of visited vertices of our algorithm are several times smaller than that of the state-of-the-art algorithm.

7 Conclusion

In this paper, we study the problem of shortest geodesic path query processing. We propose an efficient on-the-fly algorithm for the query studied in which a core component is the lower bound estimation of the length of the shortest geodesic path passing through each part of the terrain. The lower bound provides a priority of each part which is destination-aware and guides the search towards the destination. As such, the search space is highly reduced and the query processing is boosted. We theoretically prove that our algorithm is correct and our empirical study confirms that our algorithm significantly outperforms the state-of-the-art in terms of the running time and the number of vertices visited.

REFERENCES

[1] K. Deng, H. T. Shen, K. Xu, and X. Lin, "Surface k-nn query processing," in *IEEE International Conference on Data Engineering (ICDE)*, 2006, p. 78.

[2] K. Deng, X. Zhou, H. T. Shen, Q. Liu, K. Xu, and X. Lin, "A multi-resolution surface distance model for k-nn query processing," in *ACM International Journal on Very Large Data Bases (VLDBJ)*, 2008, pp. 1101–1119.

[3] C. Shahabi, L.-A. Tang, and S. Xing, "Indexing land surface for efficient knn query," in *ACM International Conference on Very Large Data Bases (VLDB)*, 2008, pp. 1020–1031.

[4] S. Xing, C. Shahabi, and B. Pan, "Continuous monitoring of nearest neighbors on land surface," in *ACM International Conference on Very Large Data Bases (VLDB)*, 2009, pp. 1114–1125.

[5] L. Liu and R. C.-W. Wong, "Finding shortest path on land surface," in *ACM Conference on Management of Data (SIGMOD)*, 2011, pp. 433–444.

[6] D. Yan, Z. Zhao, and W. Ng, "Monochromatic and bichromatic reverse nearest neighbor queries on land surfaces," in *The Conference on Information and Knowledge Management (CIKM)*, 2012, pp. 942–951.

[7] M. Kaul, R. C.-W. Wong, B. Yang, and C. S. Jensen, "Finding shortest paths on terrains by killing two birds with one stone," in *ACM International Conference on Very Large Data Bases (VLDB)*, 2013, pp. 73–84.

[8] M. Kaul, R. C.-W. Wong, and C. S. Jensen, "New lower and upper bounds for shortest distance queries on terrains," in *ACM International Conference on Very Large Data Bases (VLDB)*, 2015, pp. 168–179.

[9] L. T. Sarjakoski, P. Kettunen, H.-M. Flink, M. Laakso, M. Rönneberg, and T. Sarjakoski, "Analysis of verbal route descriptions and landmarks for hiking," in *Personal and Ubiquitous Computing*, 2012, pp. 1001–1011.

[10] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," in *Journal of field robotics*, 2006, pp. 839–861.

[11] N. Vandapel, R. R. Donamukkala, and M. Hebert, "Unmanned ground vehicle navigation using aerial ladar data," in *The International Journal of Robotics Research*, 2006, pp. 31–51.

[12] B. Koyuncu and E. Bostancı, "3d battlefield modeling and simulation of war games," in *Proceedings of the 3rd International Conference on Communications and information technology*, 2009, pp. 64–68.

[13] L.-H. Lee, T. Braud, P. Zhou, L. Wang, D. Xu, Z. Lin, A. Kumar, C. Bermejo, and P. Hui, "All one needs to know about metaverse: A complete survey on technological singularity, virtual ecosystem, and research agenda," *ArXiv*, vol. abs/2110.05352, 2021.

[14] L. Lee, Z. Lin, R. Hu, Z. Gong, A. Kumar, T. Li, S. Li, and P. Hui, "When creators meet the metaverse: A survey on computational arts," *ArXiv*, vol. abs/2111.13486, 2021. [Online]. Available: https://arxiv.org/abs/2111.13486

[15] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," in *SIAM Journal on Computing*, 1987, pp. 647–668.

[16] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *International Symposium on Computational Geometry (SoCG)*, 1990, pp. 360–369.

[17] S.-Q. Xin and G.-J. Wang, "Improving chen and han's algorithm on the discrete geodesic problem," in *ACM Transactions on Graphics (TOG)*, 2009, pp. 104:1–104:8.

[18] 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," in *Discrete & Computational Geometry*, 2010, pp. 762–801.

[19] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," in *Journal of ACM*, 2005, pp. 25–53.

[20] H. N. Djidjev and C. Sommer, "Approximate distance queries for weighted polyhedral surfaces," in *The European Symposium on Algorithms (ESA)*, 2011, pp. 579–590.

[21] V. J. Wei, R. C.-W. Wong, C. Long, and D. M. Mount, "Distance oracle on terrain surface," in *ACM Conference on Management of Data (SIGMOD)*, 2017, pp. 1211–1226.

[22] V. Verma and J. Snoeyink, "Reducing the memory required to find a geodesic shortest path on a large mesh," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009, pp. 227–235.

[23] V. J. Wei, R. C.-W. Wong, C. Long, D. M. Mount, and H. Samet, "Proximity queries on terrain surface," *ACM Transactions on Database Systems (TODS)*, 2022.

[24] P. A. Benton, "Unfolding polyhedra," Ph.D. dissertation, University of Cambridge, 2008.

[25] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating shortest paths on weighted polyhedral surfaces," in *Algorithmica*, 2001, pp. 527–562.

[26] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack, "An ε—approximation algorithm for weighted shortest paths on polyhedral surfaces," in *Algorithm Theory—SWAT'98*, pp. 11–22.

[27] T. Kanai and H. Suzuki, "Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications," in *Proceedings Geometric Modeling and Processing 2000. Theory and Applications (GMPTA)*, 2000, pp. 241–250.

[28] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," in *Journal of ACM*, 1995, pp. 67–90.

[29] J. Sankaranarayanan and H. Samet., "Distance oracles for spatial networks." *In* Proceedings of the 25th IEEE International Conference on Data Engineering*, pages 652–663, Shanghai, China, April 2009.*

[30] J. Sankaranarayanan and H. Samet, "Query processing using distance oracles for spatial networks," IEEE Transactions on Knowledge and Data Engineering*, 22(8):1158–1175, August 2010. (Best Papers of ICDE 2009 Special Issue.)*, pp. 1158–1175.

[31] K. Deng and X. Zhou, "Expansion-based algorithms for finding single pair shortest path on surface," in *International Conference on Web and Wireless Geographical Information Systems (WWGIS)*, 2004, pp. 151–166.

[32] M. Saad, A. I. Salameh, and S. Abdallah, "Energy-efficient shortest path planning on uneven terrains: A composite routing metric approach," in *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, 2019.

[33] N. Ganganath, C.-T. Cheng, and K. T. Chi, "Finding energy-efficient paths on uneven terrains," in *2014 10th France-Japan/8th Europe-Asia Congress on Mecatronics (MECATRONICS2014-Tokyo)*, 2014.

[34] M. Saad, A. I. Salameh, S. Abdallah, A. El-Moursy, and C.-T. Cheng, "A composite metric routing approach for energy-efficient shortest path planning on natural terrains," *Applied Sciences*, 2021.

[35] M. Cui, D. D. Harabor, and A. Grastien, "Compromise-free pathfinding on a navigation mesh." in *IJCAI*, 2017, pp. 496–502.

[36] R. Hechenberger, P. J. Stuckey, D. Harabor, P. Le Bodic, and M. A. Cheema, "Online computation of euclidean shortest paths in two dimensions," in *Proceedings of the International Conference on Automated Planning and Scheduling*, 2020.

[37] B. Shen, M. A. Cheema, D. D. Harabor, and P. J. Stuckey, "Euclidean pathfinding with compressed path databases," in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021.

[38] J. Du, B. Shen, and M. A. Cheema, "Ultrafast euclidean shortest path computation using hub labeling," 2023.

[39] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," in *IEEE transactions on Systems Science and Cybernetics*, 1968, pp. 100–107.

[40] https://en.wikipedia.org/wiki/Ellipse.

[41] https://en.wikipedia.org/wiki/Planar_graph.

[42] https://www.dropbox.com/s/ofa9ddk138x91w3/dataset.tar.gz?dl=0.

[43] https://code.google.com/archive/p/geodesic/.

[44] Y. Yan and R. C. wing Wong, "Path advisor: A multi-functional campus map tool for shortest path," *VLDB*, 2021.

[45] 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," in *Discrete & Computational Geometry*, 2010, pp. 762–801.

Victor Junqiu Wei is currently working as a research assistant professor in the Department of Computer Science and Engineering (CSE), the Hong Kong University of Science and Technology (HKUST). He obtained his bachelor degree from Nanjing University and PhD degree from Department of Computer Science and Engineering, the Hong Kong University of Science and Technology. He also has several years' working experience in the world-famous research labs in the AI industry including Baidu natural language processing (NLP) group, AI group of WeBank and Noah's Ark Lab of Huawei.

Raymond Chi-Wing Wong is a Professor in Computer Science and Engineering (CSE) of The Hong Kong University of Science and Technology (HKUST). He is currently the associate head of Department of Computer Science and Engineering (CSE). He was the associate director of the Data Science and Technology (DSCT) program (from 2019 to 2021), the director of the Risk Management and Business Intelligence (RMBI) program (from 2017 to 2019), the director of the Computer Engineering (CPEG) program (from 2014 to 2016) and the associate director of the Computer Engineering (CPEG) program (from 2012 to 2014). He received the BSc, MPhil and PhD degrees in Computer Science and Engineering in the Chinese University of Hong Kong (CUHK) in 2002, 2004 and 2008, respectively. In 2004-2005, he worked as a research and development assistant under an R&D project funded by ITF and a local industrial company called Lifewood. His research interests include database and data mining.

Cheng Long is currently an Assistant Professor at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU). From 2016 to 2018, he worked as a lecturer (Asst Professor) at Queen's University Belfast, UK. He got the PhD degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST) in 2015. His research interests are broadly in data management, data mining and big data analytics. He has served as a Program Committee member/referee for several top data management and data mining conferences/journals (TODS, VLDBJ, TKDE, ICDM, CIKM, etc.). He is a senior member of IEEE.

David M. Mount is a professor at the University of Maryland, College Park department of computer science whose research is in computational geometry. Mount received a B.S. and his Ph.D. in Computer Science at Purdue University. Mount's main area of research is computational geometry, which is the branch of algorithms devoted to solving problems of a geometric nature. In particular, Mount has worked on the k-means clustering problem, nearest neighbor search, and point location. Mount was named to the 2022 class of ACM Fellows, "for contributions to algorithms and data structures for geometric data analysis and retrieval".

Hanan Samet is a Distinguished University Professor of computer science. He is a fellow of the ACM, IEEE, and the International Association of Pattern Recognition (IAPR). From 1989 to 1991 he served as the Capital region representative on the ACM Council. He is the recipient of the 2009 UCGIS Research Award and received best paper awards in the 2008 SIGMOD Conference, the 2008 SIGSPATIAL ACMGIS'08 Conference, and the 2007 Computers & Graphics Journal. Samet's paper at the 2009 IEEE International Conference on Data Engineering (ICDE) was selected as one of the best papers for publication in the IEEE Transactions on Knowledge and Data Engineering. In his pioneering research since the 1980s on quadtrees and other data structures, as well as his well-received books, Samet has profoundly influenced the theory and application of multidimensional spatial data structures.