Geodesic Tracks: Computing Discrete Geodesics With Track-Based Steiner Point Propagation

Wenlong Meng, Shiqing Xin, Changhe Tu, Shuangmin Chen, Ying He, *Member, IEEE*, and Wenping Wang, *Fellow, IEEE*

Abstract—This article presents a simple yet effective method for computing geodesic distances on triangle meshes. Unlike the popular window propagation methods that partition mesh edges into intervals of varying lengths, our method places evenly-spaced, source-independent Steiner points on edges. Given a source vertex, our method constructs a Steiner-point graph that partitions the surface into mutually exclusive tracks, called geodesic tracks. Inside each triangle, the tracks form sub-regions in which the change of distance field is approximately linear. Our method does not require any pre-computation, and can effectively balance speed and accuracy. Experimental results show that with 5 Steiner points on each edge, the mean relative error is less than 0.3% for common 3D models used in the graphics community. We propose a set of effective filtering rules to eliminate a large amount of useless broadcast events. For a 1000K-face model, our method runs 10 times faster than the conventional Steiner point method that examines a complete graph of Steiner points in each triangle. We also observe that using more Steiner points increases the accuracy at only a small extra computational cost. Our method works well for meshes with poor triangulation and non-manifold configuration, which often poses challenges to the existing PDE methods. We show that geodesic tracks, as a new data structure that encodes rich information of discrete geodesics, support accurate geodesic path and isoline tracing, and efficient distance query. Our method can be easily extended to meshes with non-constant density functions and/or anisotropic metrics.

Index Terms—Discrete geodesics, shortest paths, geodesic tracks, Steiner points, ridge points

1 Introduction

Computing geodesic distances and paths on triangle meshes is a fundamental operation in digital geometry processing, and it plays a central role in various tasks including remeshing [1], shape segmentation [2], shape retrieval [3], [4], parameterization [5], geometric deep learning [6], [7] and many others. The discrete geodesic problem has four versions, namely single-source-all-destinations (SSAD), single-source-single-destination, multiple-sources-all-destinations, and all-pairs. Among them, SSAD has been investigated most extensively, since it serves as the basis of the other three.

There is a large body of literature on computing SSAD distances [8]. The exact algorithms [9], [10], [11], [12], [13], [14] discretize wavefronts by a so-called "window" data structure that encodes a set of shortest paths sharing a common face sequence, and then dynamically generate candidate windows that may define a shortest path to a certain vertex. The algorithms differ in the ways of avoiding exponential window explosion. A common strategy is to detect useless windows based on known distances and prevent them from propagating. Using a set of window filtering rules, these algorithms can enforce windows to propagate along tree-like tracks, which is based on the fact that the shortest paths from the source point to all the destination points form a tree-like structure. The exact algorithms can work for manifold meshes with arbitrary triangulations, however they are often limited to small-scale meshes due to high computational cost.

In time-critical applications, the user often adopts approximate algorithms that trade accuracy for speed and memory usage. Graph-based algorithms are popular due to their efficiency and accuracy [15], [16], [17], [18], [19], [20]. Since the existing graph-based methods require pre-computation, they are ideal for frequent geodesic distance query. In this paper, we develop a new graph-based method that does not need pre-computation, and is able to compute high-quality geodesic distances and paths on the fly.

Our key idea is to construct graphs using Steiner points placed on mesh edges. Since forming a complete sub-graph for each triangle is expensive, we take each Steiner point as a simplified window and adapt the existing window filtering rules to propagate geodesic distances along intersection-free tracks, which are called *geodesic tracks*. As a result,

- *Wenlong Meng, Shiqing Xin, and Changhe Tu are with the School of Computer Science and Technology, Shandong University, Jinan, Shandong 250355, China. E-mail: longwuya@163.com, {xinshiqing, chtu}@sdu.edu.cn.*
- *Shuangmin Chen is with the School of Information and Technology, Qingdao University of Science and Technology, Qingdao, Shandong 266061, China. E-mail: csmqq@163.com.*
- *Ying He is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798. E-mail: yhe@ntu.edu.sg.*
- *Wenping Wang is with the Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: wenping@cs.hku.hk.*

Manuscript received 29 March 2021; revised 26 August 2021; accepted 26 August 2021. Date of publication 1 September 2021; date of current version 27 October 2022.
(Corresponding authors: Shiqing Xin and Changhe Tu.)
Recommended for acceptance by K. Hormann.
Digital Object Identifier no. 10.1109/TVCG.2021.3109042

we do not need to exhaust every possibility of distance propagation from a Steiner point to all of its neighbors. Specifically, we keep down only the circular order of Steiner points along each directed mesh edge and maintain intersection-free tracks for each pair (e, f), where e is a directed edge incident to the triangle face f. When the distance at a Steiner point, say p, is updated, we prioritize propagating the distance to the successors located in p' track, and check if the tracks on both sides of p need to update.

Geodesic tracks partition each triangle into sub-regions so that the geodesic distances change linearly in each subregion. This allows us to trace both geodesic paths and isolines in an accurate and robust manner, even for meshes with poor triangulations. We observe that geodesic tracks algorithm runs many times faster than the complete Steiner-point graph algorithms. For example, with 5 Steiner point on each edge of a 1000K-face model, our method takes only 1.39 seconds to compute the single-source-all-destination distances, while the naïve complete Steiner-point graph algorithm takes 12.89 seconds. Note that both methods have an average relative error of 0.13%.

Our method works well for meshes with poor triangulation and non-manifold configuration, which often poses challenges to the existing PDE methods. We show that geodesic tracks, as a new data structure that encodes rich information of discrete geodesics, support accurate geodesic path and isoline tracing, and efficient distance query. Our method is also flexible to handle meshes with density functions and/or anisotropic metrics, with which the existing window based methods cannot deal.

2 Related Work

There is a large body of literature on computing geodesics on discrete polygonal surfaces. In this section, we briefly review the works that are mostly relevant to ours, including the window propagation-based exact geodesic algorithms, graph-based methods and other approximate algorithms.

2.1 Exact Algorithms

Exact geodesic algorithms commonly use a window to encode the geodesic paths sharing the same edge sequence, which enables us to represent infinitely many geodesic paths by finite discrete elements. They generally propagate wavefronts across triangle faces in a Dijkstra-sweeping style [21], and distinguish in the way of filtering useless windows. The MMP algorithm [9], [11] and the CH algorithm [10] are two representative methods: the former maintains a partitioning scheme for each directed edge while the latter uses a "one angle one split" rule so that at most $O(n)$ windows are allowed to have two children. Both methods have many variants and improvements [11], [12], [13], [22], [23], [24], [25], [26]. Qin *et al.* [14] examined the situations that two windows arrive at the same triangle and proposed a comprehensive set of pruning rules. Their method, called VTP, is the state of the art for computing exact geodesic distances without preprocessing. Recently, Du *et al.* [27] developed parallel and approximate variants of VTP.

It is known that the shortest paths from a source point to all destinations form a tree. Two different shortest paths, rooted at the same source point, generally do not have intersections except at saddle points. The window filtering rules check whether a window carries useful information before propagating it, hereby reducing the number of windows generated. In this paper, we generalize the existing window filtering rules to prune Steiner points and guarantee that the distances are propagated along intersection-free tracks.

2.2 Graph Algorithms Based on Steiner Point Insertion

It is a natural idea to construct a sufficiently dense undirected graph from the input manifold mesh and use some graph based search techniques to deal with the distance/path query. There is a large body of literature on inserting Steiner points into edges of mesh triangles. Lanthier *et al.* [15], [28] adopted such a strategy to construct a dense graph that can balance the accuracy and the timing cost by tuning the number of inserted Steiner points. Later, Lanthier *et al.* [29] implemented a parallel algorithm to boost their previous Steiner-insertion approach. The algorithm utilizes a spatial indexing structure, called multidimensional fixed partition, and thus achieves load balancing and reduces the idle time of processors. Aleksandrov *et al.* [30], [31] proposed to add denser Steiner points around vertices than towards mid-edges such that the length of the approximated path is within a factor $(1 + \epsilon)$ of the shortest geodesic path. Later, they [32] found adding Steiner points along the bisectors of triangles also works. Based on the observation that the geodesic cones used in [31] can be replaced by a set of non-overlapping intervals, Sun *et al.* [33] suggested maintaining a small set of incident edges for each Steiner point that are likely to be used to improve the current shortest path. Kanai and Suzuki [34] proposed an iterative approximation algorithm to compute the shortest path between a pair of points. The algorithm repeatedly refines the edge sequence where the path passes through by adding Steiner points until the error between two successive paths is sufficiently small. See [8] for a comprehensive survey.

The existing algorithms on this side, in their nature, need to construct a graph G w.r.t. the Steiner points and use an edge to connect any pair of Steiner points in the same triangle. Let k be the number of Steiner points inserted on each mesh edge. The complexity of G climbs in a quadratic growth w.r.t. k, thus greatly increasing the query cost when k is large. In this paper, we observe that the shortest path tree extends along tracks and thus encourage the distance to propagate along tracks throughout the algorithm, instead of broadcasting the distance from one Steiner point to any of its neighboring Steiner points.

2.3 Graph Algorithms Based on Precomputed Geodesic Bridges

There are some research works on preprocessing the input mesh into a more compact graph by bridging non-adjacent vertices with precomputed geodesic paths. Mata *et al.* [35] presented an algorithm based on constructing a sparse

graph, named pathnet, which links selected pairs of subdivision vertices with locally optimal paths. It allows users to control the sparsity of the pathnet that facilitates quick search of paths. Varadarajan *et al.* [36] proposed partitioning the the whole surface into a set of patches. For each patch, one can select some representative points on the boundary edges and connect them into a subgraph. All the subgraphs can be combined into a single graph that serves as the basic data structure to support path query. Xin *et al.* [16] precomputed an all-pair geodesic distance matrix for a set of uniformly distributed sample points such that the distance between any pair of points can be estimated using an unfolding technique. Later, Xin *et al.* [19] suggested building a sparse proximity graph with regard to a set of sample points, where each graph edge is associated with the exact geodesic distance between the two endpoints. In the query stage, their algorithm augments the proximity graph by temporarily adding the source point and the target point on-the-fly. Ying *et al.* [17] precomputed the geodesic distance between any pair of close saddle vertices and use the saddle vertex graph (SVG) to report a possibly long geodesic path. Wang *et al.* [18] presented a graph-based method, called discrete geodesic graph (DGG), to compute discrete geodesics in a divide-and-conquer manner. Different from the SVG, DGG allows paths to go through non-saddle vertices, thus overcoming the limitations of the SVG method. In order to reduce the preprocessing cost of the original DGG method, Adikusuma *et al.* [20] proposed an improved scheme to directly compute DGG with $O(\frac{n}{\sqrt{\varepsilon}})$ edges in empirical $O(\frac{n}{\varepsilon^{0.75}} log\frac{1}{\varepsilon})$ time. Nazzaro *et al.* [37] proposed a method to construct a sparse graph where nodes correspond to mesh vertices while arcs are given by one-ring mesh edges and their dual edges. Compared with the original DGG, the size of the constructed graph is greatly reduced, making the improved algorithm run at a remarkable speed. Aiello *et al.* [38] used a recursive Voronoi-based approach that works by building a hierarchical partitioning of the input mesh and precompuing the exact geodesic distances within each patch. In this way, one can perform fast geodesic query between any vertex pair by exploiting the precomputed data.

The above-mentioned graph-based algorithms require pre-computation, while our geodesic track algorithm can compute high-quality geodesic distances and paths without preprocessing.

2.4 Other Algorithms

In fact, there are many other approximation algorithms that can be used to estimate geodesic distances, such as the fast marching method (FMM) [39] and the heat based approach [40]. The heat based method inherits the spirit of the heat diffusion equation and requires solving a pair of linear systems in implementation. In spite of the simplicity, the major disadvantage is being inaccurate and non-trivial to approach the real geodesic distance. Xin and Wang [41] found that it is possible to make an transition between the exact geodesic algorithm and Dijkstra's algorithm with a single parameter λ - a smaller λ tends to produce more accurate results but requires a higher cost. Eivind *et al.* [42] proposed the Discrete geodesic polar coordinates (DGPC) such that the distances at vertices can be updated from a neighbouring triangle, instead of a neighbouring vertex. Campen *et al.* [43] proposed a Short-Term Vector Dijkstra algorithm (STVD) for handling intrinsic anisotropic metrics, which can intuitively be understood as Dijkstra's classical shortest path algorithm equipped with a vector-valued short-term memory. Liu *et al.* [44] introduced an optimization driven framework to shorten the path between two given points. Xin *et al.* [45] gave a fast edge sequence constrained shortest path algorithm and suggested updating the edge sequence if a shorter path is available in the neighboring edge sequence. Ye *et al.* [46] developed a method based on differential evolution (DE) for computing optimal solutions and then propose a simple strategy to encode paths and define path operations so that the discrete paths can fit into the DE framework. Recently, Cao *et al.* [47] formulated the problem of computing a smooth geodesic distance field into a quadratic programming problem. Their algorithm makes a trade-off between accuracy and smoothness. Sharp *et al.* [48] introduced an approach to compute geodesics by iteratively performing edge flips and finally connecting the source with each destination vertex with a geodesic path.

Besides, there are many parallel methods that focus on improving the performance of the traditional sequential algorithms. In fact, the transition from sequential algorithms to parallel algorithms is not easy. Ying *et al.* [25] developed a GPU based implementation of Xin & Wang's geodesic algorithm by replacing the priority queue by buckets and propagating k top-priority windows at the same time. Ying *et al.* [49] designed an active strategy such that the vertices and half-edges on the wavefront take the initiative to collect their input data and then propagate windows and update geodesic information in their own memory space. As a result, all the read and write operations can be carried out simultaneously. Tao *et al.* [50] employed a breadth-first strategy to implement a parallel Gauss-Seidel solver for the diffusion step in the heat method. Farias *et al.* [51] introduced a new method for computing optimal path maps on the GPU. They explored GPU rasterization as a way to propagate optimal costs on a polygonal 2D environment, producing optimal path maps which can efficiently be queried at run-time. Calla *et al.* [52] presented a minimalistic parallel algorithm by taking into account memory coalescence issues, where a breadth-first search traversal works harmoniously with the parallel front propagation approach.

3 Background Knowledge

In this section, we shall briefly review the window pruning rules in the existing exact geodesic algorithms [9], [10], [14] and these pruning rules provide a theoretical base for our algorithm design. After that, we review how Steiner based graph search algorithms work and why there is redundant computation.

3.1 Window Propagation and Pruning

In the discrete setting, a geodesic path is a local shortest path lying on a polygonal surface. From the application point of view, we are interested in global shortest paths. It is obvious that a global shortest path Γ enters and leaves a triangle face f at most once, and the intersection $\Gamma \cap f$ is a point, a line

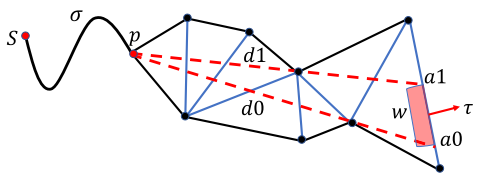


Fig. 1. Illustration of windows. A window $w = (a_0, a_1, d_0, d_1, \sigma, \tau)$ is an interval on an oriented edge that encodes a family of geodesic paths sharing the same edge sequence. p is the closest saddle vertex on the edge sequence, and d_0 and d_1 are the distances from p to the endpoints a_0 and a_1. σ is the distance from the source point s to the pseudo source p and τ is a binary direction specifying the side of the edge from which the paths arrive.

segment, or an empty set. Furthermore, a geodesic path that does not pass through any vertex becomes a straight line segment when the corresponding face sequence is unfolded onto a plane. A geodesic path may pass through one or more saddle vertices. The most recent one is called pseudo source. In computational geometry methods, the geodesic paths sharing the same face sequence are grouped in a data structure, called window [9], which is used to parameterize face sequence and compute geodesic distances. See Fig. 1 for an example of window.

When a window is propagated across a triangle, it may produce one or more children. Given a triangle mesh with n faces, there are at most $O(n^2)$ windows produced [9], [10], [53]. It is worth noting that the quadratic upper bound is of purely theoretical interest. A window becomes useless if there is a new-born window carries shorter distance than it. A useless window can be safely discarded without compromising the correctness of the result. Since the real performance of an algorithm heavily depends on the actual windows generated, various strategies have been proposed to reduce the number of useless windows. Mitchell *et al.* [9] organized windows in a priority queue, and always propagated the one with the top priority, i.e., the window closest to the source.

The rationale behind is that the shortest paths from the same source do not intersect each other except at the starting point or some saddle vertices. Maintaining windows in increasing order of distance reduces the number of comparisons among them and thus makes the pruning operation more effective. The MMP algorithm also requires overlapping-free windows by window clipping. When a new window w arrives at an oriented e, the MMP algorithm clips w with its neighbors if intersection occurs and inserts the clipped window in a sorted list

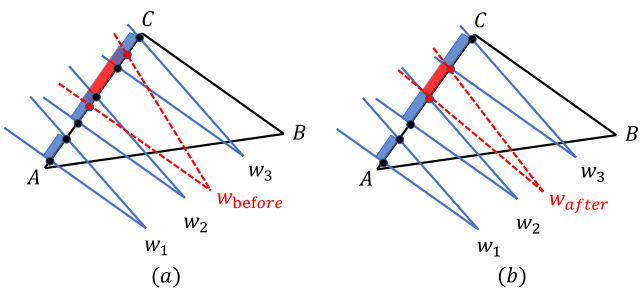


Fig. 2. Window clipping in the MMP algorithm [11]. (a) A new candidate window w arrives at e and overlaps with some existing windows. (b) After clipping, the windows on an oriented edge are overlapping-free and stored in a sorted list.

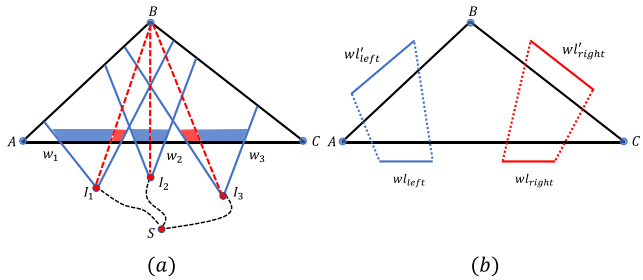


Fig. 3. Window pruning. (a) shows the "one angle one split" rule of the CH algorithm. Among all the windows covering angle $\angle B$, only one is allowed to produce two children. (b) Window list propagation in the VTP algorithm. The blue windows do not bring shorter distance to edge BC, hence they are propagated towards AB only. Similarly, the red windows are propagated towards BC only.

(see Fig. 2). The MMP algorithm works in $O(n^2 \log n)$ time and $O(n^2)$ space.

Chen and Han [10] reduced the quadratic space complexity to $O(n)$. In contrast to the MMP algorithm, the CH algorithm allows overlapping windows. It adopts pruning rule called "one angle one split" (see Fig. 3a): among the windows of edge e covering the same vertex v, only one can propagate two child windows. The VTP algorithm [14] gave an exhaustive list of scenarios for pairwise window pruning inside a triangle. Furthermore, their algorithm deals with an ordered list of windows arriving at some directed edge, instead of window by window, to enforce the orderliness of distance propagation (see Fig. 3b).

3.2 Steiner Points

Steiner points are used in the graph-based methods [15], [30], [32] to achieve desired accuracy. By inserting k Steiner points into each mesh edge, one can obtain a graph $\mathcal{G}$ - any pair of Steiner points in the same triangle is connected. The size of the augmented graph $\mathcal{G}$ is $O(n \times k^2)$. Suppose that the distance between two neighboring Steiner points on the same edge is no more than l. It is easy to show that $\|\Pi'(s,t)\| \leq \|\Pi(s,t)\| + ml$, where $\Pi(s,t)$ is the real geodesic path between s and t, $\Pi'(s,t)$ is the shortest path induced from the graph, and m is the number of edges crossed by $\Pi(s,t)$.

However, the increasing of accuracy is at the cost of more running time. For traditional sweeping algorithms on graphs, when a Steiner point p gets a shorter distance, it has to broadcast the update event to all of the neighboring Steiner points (or vertices), causing a sharp rise of the overall computational cost due to the fact that the complexity of $\mathcal{G}$ is proportional to k^2. This leads to a dilemma - it is hard to make a good balance between accuracy and performance, which inspires us to reduce useless broadcast events by maintaining an orderly propagation like that achieved in exact algorithms.

4 Steiner Point Based Propagation

4.1 Motivations

Generally speaking, the design of exact/approximate geodesic algorithms has to take into account two aspects. On the one hand, filtering rules have to be carefully designed for reducing the number of generated discrete events. On the other hand, the priority in which discrete events are

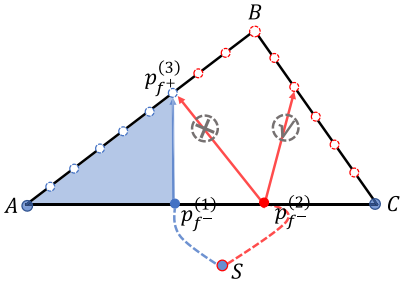


Fig. 4. Pruning Steiner-point broadcasting events. If $p_{f^-}^{(2)}$ cannot give a shorter distance to $p_{f^+}^{(3)}$ than $p_{f^-}^{(1)}$, then we do not need to propagate the distance from $p_{f^-}^{(2)}$ to the Steiner points lying on the left side of $\overrightarrow{p_{f^-}^{(1)}p_{f^+}^{(3)}}$. The detailed rules are summarized in Fig. 5.

processed has to be carefully maintained such that the wavefronts extend from near to far.

In this paper, we intend to use Steiner points to replace windows to seek a balance between accuracy and speed. In detail, the motivations lie in two aspects. First, we inherit the existing filtering rules [14]. Second, we use buckets, instead of a priority queue, to maintain the priorities of discrete events, like that achieved in [13].

4.2 Data Structures

In our method, the role of inserted Steiner points is much like the windows in exact geodesic methods. We classify Steiner points into two kinds, i.e., vertex-type Steiner points and edge-type Steiner points. Each edge-type Steiner point q on the boundary of the triangle f has two roles. We use q_{f^+} to denote the instance of q leaving the face f, and q_{f^-} to denote the instance of q entering f. We use $g(q_{f^-})$ to denote the geodesic distance arriving at the Steiner point q and entering f. Likewise, $g(q_{f^+})$ is to denote the geodesic distance leaving the Steiner point q and exiting f. In implementation, an edge-type Steiner point can be represented by the following data structure:

```
class EdgeSteinerPoint : SteinerPoint
{
int idOfEdge; // the identity of the directed edge
double distance;// one-way distance g(q_{f+})
SteinerPoint *parent; // the current distance is given by parent
EdgeSteinerPoint *reverseSteinerPoint;
}
```

Similarly, each vertex v has many instances. The instance of v incident to the triangle face f is denoted by v_{f^+}. We use $g(v_{f^+})$ to denote the minimum distance to the vertex v from the way of the triangle f. Therefore, there are $\mathrm{Deg}(v)$ distances at the vertex v, each coming from an incident face, where $\mathrm{Deg}(v)$ denotes the degree of v. The following data structure is used to represent a vertex-type Steiner point:

```
class VertexSteinerPoint : SteinerPoint
{
int idOfVert; // the identity of the vertex v
int idOfFace; // the identity of the face f
double distance;// one-way distance g(v_{f+})
SteinerPoint *parent; // the current distance is given by parent
}
```

There is a set of Steiner points along the boundary of each triangle face f. We build two circular lists for f to store the Steiner points. In the first circular list, each Steiner point is entering f, while in the other circular list, each Steiner point is leaving f. Suppose that a Steiner point q on the boundary of f gets a shorter distance at one time. Rather than broadcast the distance update event to every Steiner point in f, our intention is to quickly locate the candidate children of q, which shall be discussed in the next subsections.

4.3 Pruning

When a Steiner point, say q, gets a shorter distance at the moment, the most important task is to determine which Steiner points (maybe just a few) need to update their shortest distances from q. We can suppress useless broadcast events by inheriting the key spirit of existing window pruning rules; See the following theorem.

Theorem 1. *Let $p_{f^-}^{(1)}$ and $p_{f^-}^{(2)}$ be two Steiner points entering the face f, and $p_{f^+}^{(3)}$ a Steiner point leaving f. Without loss of generality, assume $p_{f^-}^{(2)}$ lies in the right side of the directed line $\overrightarrow{p_{f^-}^{(1)}p_{f^+}^{(3)}}$ (see Fig. 4). Let $g(\cdot)$ be the distance function. If*

$$g(p_{f^-}^{(1)}) + \|p_{f^-}^{(1)}p_{f^+}^{(3)}\| \le g(p_{f^-}^{(2)}) + \|p_{f^-}^{(2)}p_{f^+}^{(3)}\|,$$

then $p_{f^-}^{(2)}$ cannot give a shorter distance to the Steiner points on the left side of the directed line $\overrightarrow{p_{f^-}^{(1)}p_{f^+}^{(3)}}$, hereby no broadcasting the distance from $p_{f^-}^{(2)}$ to them.

In the following, we discuss how to filter out useless propagation events based on Theorem 1. Suppose that the distance at the Steiner point q is updated at this moment and the propagation enters $f \triangleq \triangle ABC$. The filtering key lies in quickly excluding those Steiner points that are neighboring to q but impossibly get their shortest distances from q. For this purpose, we summarize the pruning rules into 3 situations, including totally 8 cases in detail, as is shown in Fig. 5. Let p_{f^-}, p'_{f^+} be a pair of Steiner points in $\triangle ABC$ and p_{f^-} be p'_{f^+}'s parent. Here, we abbreviate p_{f^-} as p and p'_{f^+} as p'. In the following, we elaborate the pruning rules, one by one, assuming $g(p) + \|pp'\| \le g(q) + \|qp'\|$.

Situation #1. $p, q \in AC$. It can be further divided into 4 cases; See the first two columns of Fig. 5. Take Case 1 for an example, where q is on the left side of p. If $g(p) + \|pp'\| \le g(q) + \|qp'\|$, then q cannot provide shorter distance for those Steiner points on the right side of $\overrightarrow{pp'}$, which is a special case of Theorem 1.

Situation #2. $q \in AC, p \in BC, p' \in AB$ or $q \in AC, p' \in BC, p \in AB$. Take Case 5 for an example, where q is on the left side of the straight line $\overrightarrow{pp'}$. According to Theorem 1, if $g(p) + \|pp'\| \le g(q) + \|qp'\|$, then q cannot give shorter distances to those Steiner points on the right side of $\overrightarrow{pp'}$ (see the blue circled Steiner points).

Situation #3. q coincides with the vertex C. It can be regarded as the degenerate case of *Situation #1*. Similarly, q cannot help determine the shortest distances to those blue circled Steiner points if $g(p) + \|pp'\| \le g(q) + \|qp'\|$ holds.

We further label the Steiner points clockwise around the boundary of $\triangle ABC$. In this way, we can unify the 8 pruning cases mentioned above in Algorithm 1.

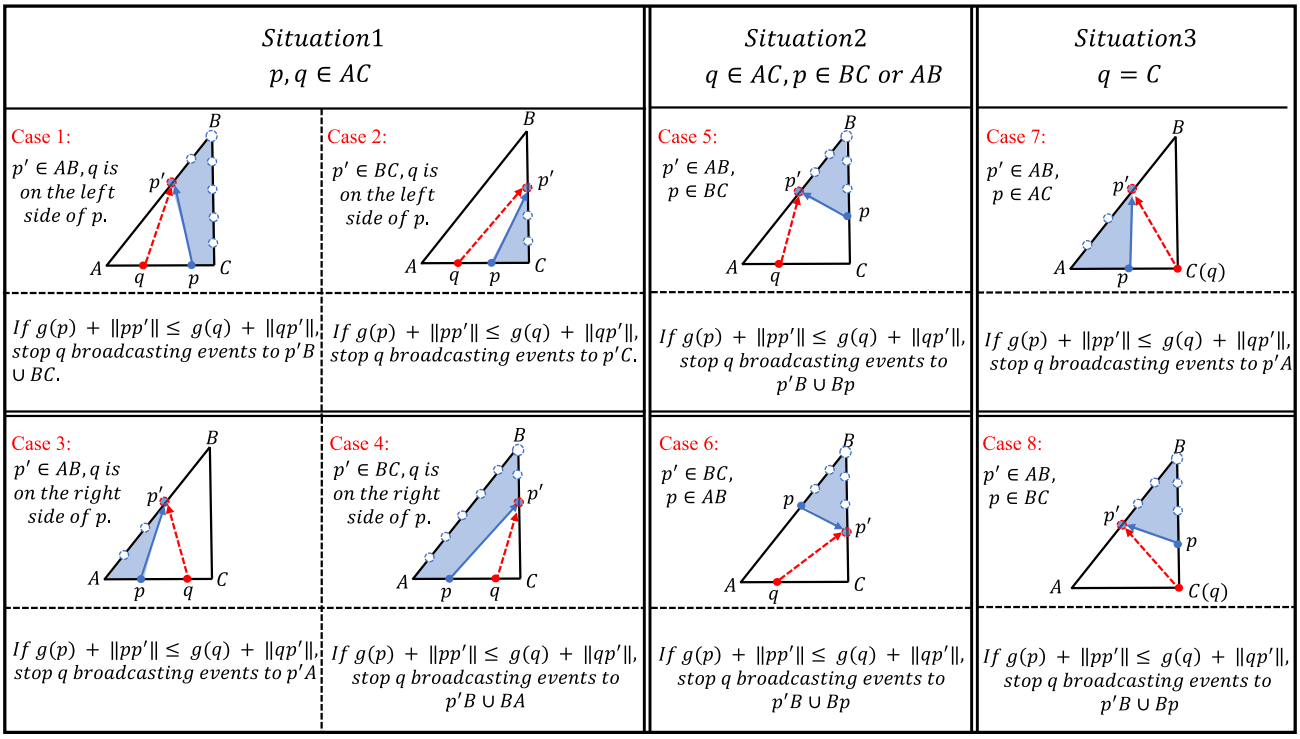


Fig. 5. There are 3 situations, covering 8 cases, to elaborate the pruning rules assuming $g(p) + \|pp'\| \leq g(q) + \|qp'\|$, where $g(\cdot)$ denotes the distance function, and p is the parent of p', i.e., the distance of p' is given by p. The propagating events of q cannot broadcast in the shaded regions.

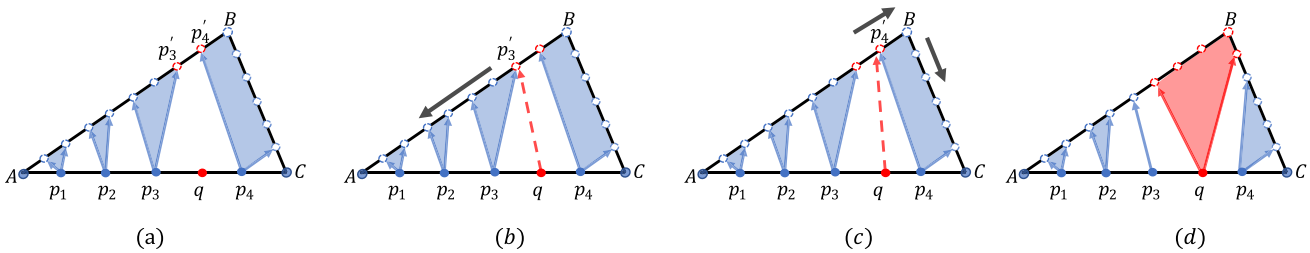


Fig. 6. When the distance at the Steiner point q is updated, we locate its children, in two separate ways, by utilizing the existing track. (a) The existing track. (b-c) We find q's children, in two separate ways, by checking if it can defeat the existing parent based on Theorem 1. (d) All of q's children are found.

4.4 Two-Way Location of Children

We assume that the distance at the Steiner point q, on the boundary of $f \triangleq \triangle ABC$, is updated at one time. As Fig. 6 shows, the Steiner point q_{f^-}, or abbreviated as q, gets a shorter distance, the new distance $g_{\text{new}}(q_{f^-})$ is less than the old distance $g_{\text{old}}(q_{f^-})$. It can be further divided into two cases: (1) $g_{\text{new}}(q_{f^-}) > g_{\text{old}}(q_{f^+})$ and (2) $g_{\text{new}}(q_{f^-}) \leq g_{\text{old}}(q_{f^+})$. For the first case, we don't need to find q_{f^-}'s children. In the following, we focus on how to locate its children under the second situation.

Without loss of generality, we assume that the current track configuration is as Fig. 6a shows. Let q_{left} and q_{right} be respectively the left neighboring Steiner point and the right neighbor of q. Rather than test every Steiner point in $\triangle ABC$, we propose a two-way fast location technique to locate its children. As Fig. 6b shows, we come to see if q can provide a shortest distance to q_{left}'s right-most child. If q can defeat q_{left}, then we continue to test more Steiner points to the left side until q fails to update the shortest distance. Similarly, we can find q's children to the right side as Fig. 6c shows. Finally, all of q's children are found (see Fig. 6d) and the track configuration is naturally updated during the process. We summarize the process in Algorithm 2.

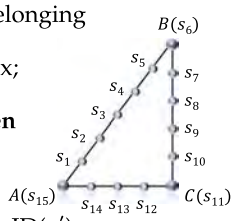
Remark. Note that in the VTP algorithm [14], the authors suggest taking the window list on one edge as a whole and propagating them from one edge to the next edge. The rational behind is that it is very helpful in reducing the cost spent in locating a newly generated window in an existing window list. However, the Steiner points in our algorithm are pre-inserted and fixed. Therefore, we cleverly solve the location problem by a two-way strategy. In fact, it is easy to show that at most two useless trials are made when we locate q's children, which shows that our strategy is very effective. Traditional Dijkstra's algorithm needs to propagate the distance to any other Steiner points in the same triangle when a Steiner point gets a shorter distance at one time, making the required timing cost increasing basically quadratically with regard to k, i.e., the number of Steiner points on each edge. By contrast, our algorithm reduces the timing cost from $O(k^2)$ to $O(k)$ empirically. See Section 6 for more experimental results.

4.5 Buckets Based Priority Maintenance

Dijkstra's algorithm and Dijkstra-like algorithms often use a priority queue to control the processing order, which needs $O(\log n)$ time per insertion/removal operation. In fact, the maintenance cost for each single operation can be reduced to $O(1)$ if using buckets [13]. Therefore, we replace the priority queue by buckets in our implementation.

ALGORITHM 1: Pruning rules based on clockwise ordering

```
//p, p', q are the Steiner points belonging to same triangle f;
//ID(p) is the Steiner point index;
if g(p) + ||pp'|| ≤ g(q) + ||qp'|| then
    if ID(p)<ID(p')<ID(q) then
        q cannot give a shorter distance to the Steiner point s with ID(p)<ID(s)<ID(p').
    if ID(p)>ID(p')>ID(q) then
        q cannot give a shorter distance to the Steiner point s with ID(p)>ID(s)>ID(p').
    if ID(p)<ID(q)<ID(p') then
        q cannot give a shorter distance to the Steiner point s with ID(s)<ID(p) or ID(s)>ID(p').
    if ID(p)>ID(q)>ID(p') then
        q cannot give a shorter distance to the Steiner point s with ID(s)>ID(p) or ID(s)<ID(p').
    if ID(p')<ID(p)<ID(q) then
        q cannot give a shorter distance to the Steiner point s with ID(p')<ID(s)<ID(p).
    if ID(p')>ID(p)>ID(q) then
        q cannot give a shorter distance to the Steiner point s with ID(p)<ID(s)<ID(p').
```



Algorithm 2. Two-Way Location of Children

```
// q_left is the left neighboring Steiner point of q_f-
// q_right is the right neighboring Steiner point of q_f-
if (g_new(q_f-) < g_old(q_f-)) && (g_new(q_f-) < g_old(q_f+)) then
    Locate the right-most child of q_left;
    Check if q_f- can be the new parent, one by one from right to left;
    Locate the left-most child of q_right;
    Check if q_f- can be the new parent, one by one from left to right;
Push the updated children to buckets.
```

Let L be the upper bound of the geodesic distance. We set totally $\lceil L/h \rceil$ buckets, denoted by $\mathcal{B}$, where h is the average edge length. The ith bucket is used to process those Steiner points whose distances are between ih and $(i+1)h$. During the distance propagation, we push the Steiner point q to the $\lfloor g(q)/h \rfloor$th bucket. We maintain two indices, I_{begin} and I_{end}, that respectively record the first non-empty bucket and the last non-empty bucket. When a new Steiner point q is pushed to $\mathcal{B}$, we update I_{end} if $\lfloor g(q)/h \rfloor > I_{\text{end}}$. Similarly, when a Steiner point q is removed from $\mathcal{B}$, we update I_{begin} if the I_{begin}th bucket becomes empty. Each time we take out a single Steiner point from $\mathcal{B}$ from the I_{begin}th bucket. The algorithm comes to termination when the buckets are empty, i.e., $I_{\text{begin}} > I_{\text{end}}$. With the support of buckets, our algorithm runs about twice faster than the implementation with a priority queue.

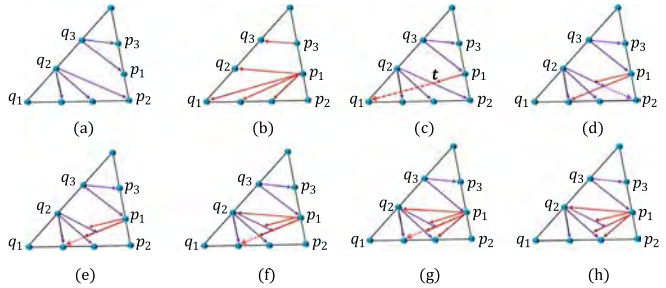


Fig. 7. Combining tracks. (a-b) two track configurations in different directions. (c-g) The 5 directed connections in (b) are combined into (a) one by one. (h) Combined track.

5 Geodesic Tracks

In this section, we first discuss how to merge tracks triangle by triangle and then report how geodesic tracks benefit some important geodesic related problems including path tracing, isoline extraction and path query on non-uniform density metric or non-manifold surfaces.

5.1 Track Combination and Triangle Decomposition

Recall that each edge-type Steiner point q has two instances, q_{f^-} entering f and q_{f^+} leaving f. Each instance of q may have its own parent. When the distance propagation terminates, each triangle face may have at most three tracks, depending on which edge is taken as the entering edge; See Figs. 7a and 7b shows. Suppose that q_{f^-}, entering f, is the parent of p_{f^+} that is leaving f. We have $g(q_{f^-}) < g(p_{f^+})$. If $g(q_{f^-}) \geq g(q_{f^+})$, it shows that the connection from q_{f^-} to p_{f^+} does not help at all for determining any shortest path. Therefore, we consider only the following two types of child-parent connections:

1) $g(q_{f^-}) < g(q_{f^+})$ and $g(p_{f^+}) < g(p_{f^-})$;
2) $g(q_{f^-}) < g(q_{f^+})$ and $g(p_{f^+}) \geq g(p_{f^-})$.

It's worth noted that for the second case, $q_{f^-}p_{f^+}$ can be viewed as two segments. For any point closer in the segment closer to the point q_{f^-}, the shortest path passes through q_{f^-}. But the segment closer to the point p_{f^+} is of no use in determining a shortest path. Based on the observation, we discuss how to combine tracks that march forward in different directions.

In Figs. 7a and 7b, we show two tracks for the triangle, waiting for being combined. (Note that the third track is missing because no useful child-parent connections exists.) We add the 5 directed connections in Figs. 7b to 7a one by one. Now we consider the connection $\overrightarrow{p_1q_1}$. We find the first intersection point of $\overrightarrow{p_1q_1}$, say, t, with the existing track. If at the intersection point t, p_1 cannot give a shorter distance than $\overrightarrow{q_2p_2}$, then we cut $\overrightarrow{p_1q_1}$ at t and continue to test the second intersection point; See Fig. 7c.

The 5 directed connections in Fig. 7b can be processed in a random order. When there is a pair of reverse connections, for example, $\overrightarrow{p_3q_3}$ and $\overrightarrow{q_3p_3}$, we can keep one connection and

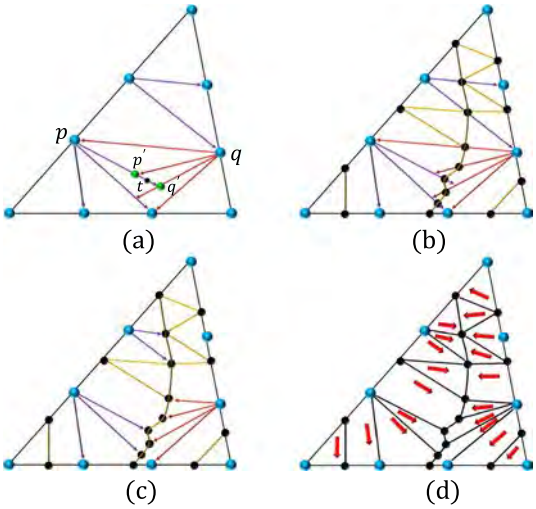


Fig. 8. On the basis of the combined track, we further decompose the triangle into a set of sub-regions. (a) Finding a pivot (black dot). (b) All the pivots (black dots) have been found and connected with yellow line segments. (c) Useless segments are removed. (d) The gradients of the distance field in each region are roughly uniform; See the red arrows for the gradients.

ignore the other. In this way, we get a combined track as Fig. 7h shows.

The combined track decomposes the triangle into a set of sub-regions. But we still need an extra step to finish the decomposition. Let $p'q'$ be a segment as Fig. 8a shows. It can be seen that the shortest distance of the endpoint p' is given by p, rather than by q, since the ray rooted at q is intercepted at p'. Let $g(p') = g(p) + \|pp'\|$. Similarly, the shortest distance of the other endpoint q' is given by q. Let $g(q') = g(q) + \|qq'\|$. Now we find a pivot t on the segment $p'q'$ such that

$$g(p') + \|p't\| = g(q') + \|q't\|.$$

The position of t is shown in Fig. 8a. It roughly means that the point t has two equal-length paths, one arriving at t from the left side and the other from the right side. Similarly, we can find more pivots, as shown in Fig. 8b, and we connect two pivots if they lie in the same region. It's worth noted that there are two special situations: (1) We can find a pivot between any two successive Steiner points. (2) In the combined track, if there are three pivots found on the boundary of a constituent cell, we connect any pair of them.

Next, we delete useless segments. For example, the segment tq' is deleted since any point in tq' cannot get a shortest path via the point t; See Fig. 8c. To this end, the triangle f is decomposed to a set of regions, and the gradients of the distance field in each region is roughly uniform. We can infer the gradients in each region by a least square technique. See Fig. 8d for illustration of gradients.

5.2 Implementation Details

The Number of Steiner Points. Recall that we need to insert some Steiner points on each mesh edge. In general, there are two sampling ways that are fixed and interval schemes referring [28]. The fixed scheme samples a certain number of Steiner points on each edge while the interval sampling scheme samples Steiner points at uniform distance on each edge,so that the number of Steiner points per edge depends on edge length. In [28], they tested two schemes that the latter scheme can achieve the same accuracy of the former one with a smaller number of Steiner points. So we use the interval scheme as our default sampling scheme that indicator k means the average number of Steiner points inserted per edge. We set $k = 5$ in most of our experiments.

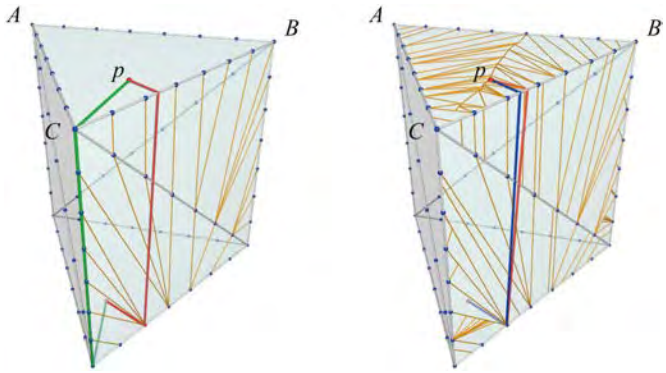


Fig. 9. Path backtracking on the surface of a regular triangular prism. The source is set at the interior of the bottom face. Left: green path is computed by traditional Dijkstra algorithm which cannot accurately infer the backtracking direction at a surface-interior destination point. Right: By contrast, blue path is computed by our algorithm which can give an accurate backtracking direction for the point in each region. We color the ground-truth paths in red.

Track Combination. At any time point of the track merging operation, the triangle of interest consists of a set of polygonal cells. When a new connection is inserted, we need to compute the intersection points between a ray, representing the new connection, and the polygonal cells. Currently we store the polygonal cells in an array and test whether each polygonal cell intersects the ray or not. There may be a better implementation way and we shall speed up the computation in the future.

Solving the Pivot Equation. Let p_1, p_2 be a segment, and the distances at the two endpoints be respectively d_1, d_2. The operation of finding the pivot is to solve the following quadratic equation

$$g(p_1) + t\|p_1p_2\| = g(p_2) + (1-t)\|p_1p_2\|,$$

which generally has two roots. It can be proved that as long as $|g(p_1) - g(p_2)| < \|p_1p_2\|$, only one of the roots meets $0 \leq t \leq 1$, which defines the pivot.

Fitting Gradients in Each Region. Recall that each triangle is decomposed into a set of regions such that the change of the distance field can be deemed to be linear. Let $p_1, p_2, \ldots, p_m$ be the vertices enclosing a region and $d_1, d_2, \ldots, d_m$ be respectively their distances. W.l.o.g., we take it as a 2D problem. In order to compute the gradients inside each region, we can fit the linear equation $d = ax + by + c$. When $m = 3$, there is a unique solution. When $m > 3$, we need to use the least square technique, which can be further transformed to solving a 3×3 linear system. Finally, (a, b) defines the gradients in this region.

5.3 Tracing Paths and Isolines Based on Tracks

Once each triangle is decomposed into a set of regions such that the change of the distance field can be deemed to be linear, one can trace paths/isolines more accurate.

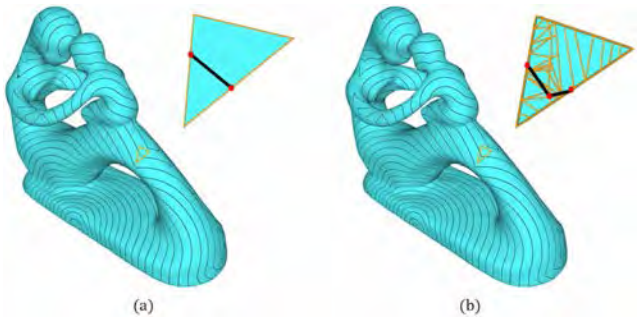


Fig. 10. (a) Traditional algorithms extract a straight-line segment inside each triangle to form the isolines. (b) Tracing isolines based on the combined tracks of our algorithm.

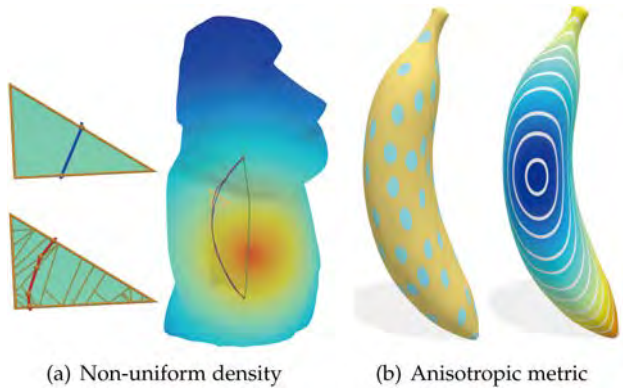


Fig. 11. Our method can support both non-uniform density functions and anisotropic metrics. (a) The Moai model is associated with a Gaussian density function and the geodesic paths tend to avoid the high density region. Our resulting path (red) has length 34.5 while Dijkstra's algorithm yields a path with length 39.9 (blue). The close-up view shows the geodesic tracks of a triangle. Each sub-region has an approximately linear-varying distance, so that we can get more accurate path result along backtracking direction. We also show the geodesic path under constant density (green) for reference. (b) Anisotropic metric and the resulting distance field.

Shortest Paths. Taking the Fig. 9 for an example, we select the interior of the bottom face as the source and compute the distance field. If we use existing approximate/exact geodesic algorithms, only the distance values at the vertices A, B, C are reported, without yielding any additional helpful information. However, since the distances of A, B, C are almost equal, making the interpolated gradients in $\triangle ABC$ vanish. Therefore, one cannot find a suitable direction at the destination point p to facilitate backtracking the shortest path from p to the source. By contrast, our algorithm concludes with a decomposition of any triangle. The top triangle is decomposed into a set of regions and in each region, there always exists a non-degenerate gradient. To summarize, traditional algorithms fail to report the decomposition configuration, and cannot accurately infer the backtracking direction at a surface-interior destination point, especially when the destination point is located nearby the ridge curve.

Isolines. Isolines are an important tool to visualize the distance field. In computer graphics, it is a common used trick to visualize a distance field by texture mapping. However, even if the distances at vertices are computed by exact geodesic algorithms, there are still conspicuous artifacts that often occur around the ridge curve; See Fig. 10a. By contrast, our algorithm is able to report higher-quality isolines; See Fig. 10b.

5.4 Non-Uniform Density and Anisotropic Metric

Our Steiner point propagation method is flexible enough to handle meshes with density function and/or anisotropic metric. Furthermore, each triangle is also decomposed into a set of linear regions in the case of density or anisotropic metric setting, so that we can accurately infer the backtracking direction.

Density Function. Given a mesh surface $\mathcal{M}$, a discrete density field defines a scalar at each vertex. Let $\triangle ABC \in \mathcal{M}$ be a triangle and p_1, p_2 be two Steiner points in this triangle. The density can be deemed to change linearly along the segment $\overline{p_1p_2}$. The weighted length of $\overline{p_1p_2}$ can be computed by integrating the linear density function along the segment. In this way, we can define the weighted length for any polygonal path constrained on the surface. In spite of the difference from the uniform case, the filtering rules still hold and thus our algorithm works well without any modification. As Fig. 11a shows, we set a Gaussian density function to the Moai model (10K faces), and the weighted shortest path colored in red is computed by our algorithm, which is very different from the shortest path under the setting of uniform density.

Anisotropic Metric. Our method can also handle meshes with anisotropic metric shown in Fig. 11b. The left figure visualizes the Riemannian metric using a set of anisotropic disks while the right figure shows the resulting geodesic distance field. In implementation, the metric tensor can be represented by a 2×2 matrix in each triangle face f, and the Riemannian length of the segment $\overline{p_1p_2} \in f$ is measured by

$$\|\overline{p_1p_2}\| \sqrt{\lambda_1 \cos^2\theta + \lambda_2 \sin^2\theta},$$

where λ_1, λ_2 respectively denote the two scaling factors of the metric tensor while θ is the angle between $\overline{p_1p_2}$ and the axis corresponding to λ_1.

Similarly, even under the anisotropic metric, any two shortest paths cannot intersect with each other except at the source point or any saddle vertex, which does not have any distinction from the euclidean metric. Because of this, our filtering rules are still effective; See Fig. 11b for the resulting distance field.

5.5 Non-Manifolds

In the past research, one usually assumes that the input surface is a manifold whose representation is half-edge structure in the discrete setting, which is due to the fact that most of the geodesic algorithms support only manifold surfaces. Currently there is no appropriate algorithm available that can accurately report a geodesic path between two points on a non-manifold surface, except for Steiner point based sweep algorithms. A typical application occasion is as follows: After the medial-axis (MA) surface has been computed, the skeleton of a 3D model can be obtained by a grassfire propagation, which requires estimating the geodesic distance between two points on the non-manifold surface [54].

Our algorithm can naturally deal with the dilemma with a slight modification. Let q be a Steiner point on a non-manifold edge that is shared by k faces $f_1, f_2, \ldots, f_k$ at the same

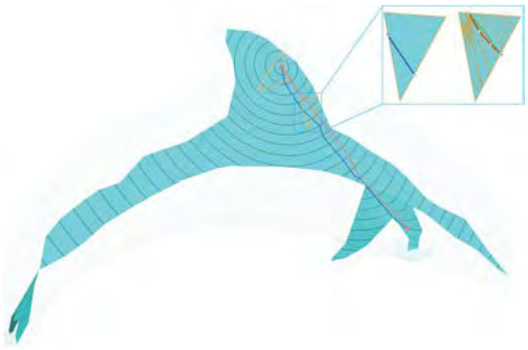


Fig. 12. Computing geodesic distances on the medial axis of the Dolphin model, which is a non-manifold structure. Owing to the sharding linear in each triangle face, we can get more accurate backtracking direction. The source is set inside of triangle. The red path is ours while the blue path is Dijkstra result. Their lengths are 7.52 and 8.45 respectively. Furthermore, we connect the start and end points of path directly in every triangles to refine our result shown in zoom window whose length is shorten to 7.13.

time. We maintain k instances for q, i.e., $q_{f_1^-}, q_{f_2^-}, \ldots, q_{f_k^-}$, where the instance $q_{f_i^-}$ means the Steiner point q that is leaving f_i. Since all the filtering rules still hold, our algorithm works well on non-manifold surfaces. In Fig. 12, we show a path lying on the MA surface of the Dolphin model.

6 Results

We implemented our algorithm in Microsoft Visual C++ 2015, without using any additional numeric packages. All the experiments were conducted on a computer with Intel (R) Core(TM) i7-9700K CPU 3.60GHz and 8GB memory. We evaluated our method on both commonly used 3D models in the graphics community and the Thingi10k 3D shape repository [55], which contains a large number of man-made anisotropic meshes and some have an extremely high degree of anisotropy.

6.1 Filtering Rules

In Fig. 13, we show how the filtering rules help reduce the number of distance broadcasting events. Let indicator k means the average number of Steiner points inserted per edge. It's no doubt that the total number of distance boarding events increases with k while the average relative error decreases with k. Since the naïve algorithm [15] propagates distances from a Steiner point to any of its neighbors, it is easy to show that the total number of required distance tests climbs in a quadratic growth with regard to k. The plot curve in the Fig. 13 shows that our algorithm is basically linear to k. Therefore, it is able to achieve more accurate results at only a little extra cost.

As an example, we use a 48K-face horse model for test. The state-of-the-art exact geodesic algorithm, i.e., VTP [14], reports the exact geodesic distance field in 0.31 seconds. For the comparison purpose, we set k to be 5 . With this setting, the naïve algorithm [15] costs 0.46 seconds. Our algorithm costs only 0.05 seconds, which is a big improvement in performance. It's worth noted that both the naïve algorithm and ours have an average relative error of 0.18%.

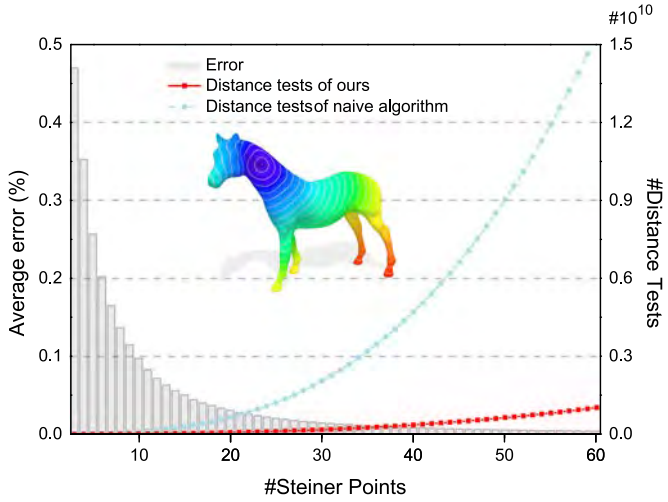


Fig. 13. Compared with the naïve Steiner point algorithm that forms a complete graph for each triangle. Our algorithm propagates distances along mutual exclusive tracks, thus greatly reducing the number of distance tests. The total number of distance tests of the naïve algorithm grows in a quadratic rate k^2.

6.2 Choice of the Number of Steiner Points

In order to choose a default value of k, the average number of Steiner points inserted per edge, we test $k = 3, 5$, and 8 respectively. Table 1 gives detailed statistics about the running time, average errors and maximum errors, where the number of faces ranges from 100K to 3,600K. It can be seen that when $k = 5$, our algorithm runs about 5 times faster than the VTP algorithm that is the state-of-the-art exact geodesic algorithm. At the same time, our algorithm has an average error of less than 0.2%, which is very accurate and able to meet the requirements for most geometry processing occasions.

6.3 Quantitative Comparison

In order to make quantitative comparisons with existing algorithms, we provide two plots in Fig. 14, measuring timing costs and errors respectively. Here the algorithms include:

1) Dijkstra's algorithm [21];
2) the Steiner-point graph method [15] which augments the triangle mesh by inserting Steiner points on each edge;
3) the VTP algorithm [14], which is the state-of-the-art exact geodesic algorithm;
4) the fast marching method (FMM) [39];
5) The heat method (HM) [40] with intrinsic Delaunay triangulations [56] to improve robustness and accuracy.
6) discrete geodesic polar coordinates (GDPC) [42];
7) the approximate ICH algorithm (AICH) [41]; and
8) geodesic tracks (this paper).

From the two figures, we can see that our algorithm ($k = 5$) runs faster than all the other approximation algorithms except Dijkstra's algorithm. However, Dijkstra's algorithm runs on the original mesh that does not include any Steiner points. Among these approximation algorithms, our algorithm has the least average errors. For example, on the Armadillo model with 350K faces, the average relative error of our algorithm is only 0.15%. We further use Fig. 15 to visualize the error distributions. All the experimental

TABLE 1
Detailed Performance Statistics

Methods \ Models	Performance	Buddha (F:100K)	Armadillo (F:350K)	Gargoyle(F:700K)	Lucy (F: 1,000K)	Dragon (F: 3,600K)
Ours (K = 3)	Time (s)	0.075	0.368	0.594	0.873	4.974
	Average error (%)	0.422	0.314	0.338	0.263	0.182
	Max error (%)	1.532	1.881	2.461	2.098	1.190
	Peak memory usage (MB)	0.470	0.815	1.504	2.822	4.229
Steiner-point graph (K = 3)	Time (s)	0.377	1.648	3.848	5.274	28.744
	Average error (%)	0.422	0.314	0.338	0.263	0.182
	Max error (%)	1.532	1.881	2.461	2.098	1.190
	Peak memory usage (MB)	1.762	2.039	3.793	4.556	15.483
Ours (K = 5)	Time(s)	0.107	0.503	1.071	1.397	7.243
	Average error (%)	0.240	0.150	0.171	0.134	0.092
	Max error (%)	0.752	0.870	1.243	0.875	0.797
	Peak memory usage (MB)	0.859	1.474	2.734	5.850	6.613
Steiner-point graph (K = 5)	Time (s)	0.945	5.088	10.036	12.894	64.666
	Average error (%)	0.240	0.150	0.171	0.134	0.092
	Max error (%)	0.752	0.870	1.243	0.875	0.797
	Peak memory usage (MB)	2.147	3.686	6.834	10.706	29.310
Ours (K = 8)	Time (s)	0.194	0.846	2.206	2.832	13.382
	Average error (%)	0.109	0.089	0.072	0.069	0.043
	Max error (%)	0.357	0.431	0.982	0.441	0.382
	Peak memory usage (MB)	1.576	2.843	5.291	6.833	8.210
Steiner-point graph (K = 8)	Time (s)	2.707	12.107	29.212	36.985	172.94
	Average error (%)	0.109	0.089	0.072	0.069	0.043
	Max error (%)	0.357	0.431	0.982	0.441	0.382
	Peak memory usage (MB)	4.190	7.108	13.229	24.006	71.051
Dijkstra	Time (s)	0.048	0.252	0.391	0.454	2.020
	Average error (%)	4.770	4.215	4.337	4.020	3.801
	Max error (%)	19.300	21.320	18.779	19.201	18.425
	Peak memory usage (MB)	0.348	0.694	1.387	2.013	3.531
FMM	Time (s)	0.091	0.509	0.773	1.002	4.300
	Average error (%)	1.792	1.530	1.447	1.201	0.988
	Max error (%)	16.33	21.47	14.01	12.80	13.39
	Peak memory usage (MB)	0.887	1.492	2.807	4.117	13.471
DGPC	Time (s)	0.400	1.263	2.805	4.370	15.790
	Average error (%)	0.387	0.285	0.211	0.107	0.049
	Max error (%)	1.258	1.405	1.100	1.127	0.940
	Peak memory usage (MB)	0.870	2.892	5.531	7.940	26.788
AICH (λ = 0.1)	Time (s)	0.110	0.828	1.562	2.511	10.911
	Average error (%)	9.400	3.370	3.440	2.880	4.592
	Max error (%)	21.810	15.621	16.380	13.982	14.200
	Peak memory usage (MB)	0.861	2.686	5.373	4.088	14.712
Heat method [56] (t = h^2)	Time (s)	0.861 (t_p) + 0.052 (t_s)	4.369 (t_p) + 0.217 (t_s)	8.197 (t_p) + 0.497 (t_s)	12.184 (t_p) + 0.886 (t_s)	Out of memory
	Average error (%)	0.635	0.702	0.932	0.821	
	Max error (%)	4.200	3.432	3.270	2.229	
	Peak memory usage (MB)	109.210	375.387	842.769	1184.153	
IDGG	Time (s)	1.271 (t_p) + 0.020 (t_s)	2.758 (t_p) + 0.051 (t_s)	8.433 (t_p) + 0.113 (t_s)	11.600 (t_p) + 0.172 (t_s)	24.154 (t_p) + 0.262 (t_s)
	Average error (%)	0.870	0.914	0.720	1.040	1.139
	Max error (%)	2.179	2.330	1.961	2.647	2.536
	Peak memory usage (MB)	82.151	221.670	368.217	517.071	788.214
VTP	Time(s)	0.544	2.998	5.997	10.165	42.780
	error (%)	–	–	–	–	–
	Peak memory usage (MB)	0.861	2.686	5.373	7.088	14.712

We compared with nine state-of-the-art methods on running time, errors and peak memory usage.
[1] *In the implementation [56] of the heat method, intrinsic triangulation is used to improve robustness and accuracy.*
[2] t_p *is the precomputation timing cost and* t_s *is the real timing cost spent for querying the geodesic distance.*

results show that our algorithm has an advantage in terms of accuracy and speed and is able to report a very accurate geodesic distance field as the output.

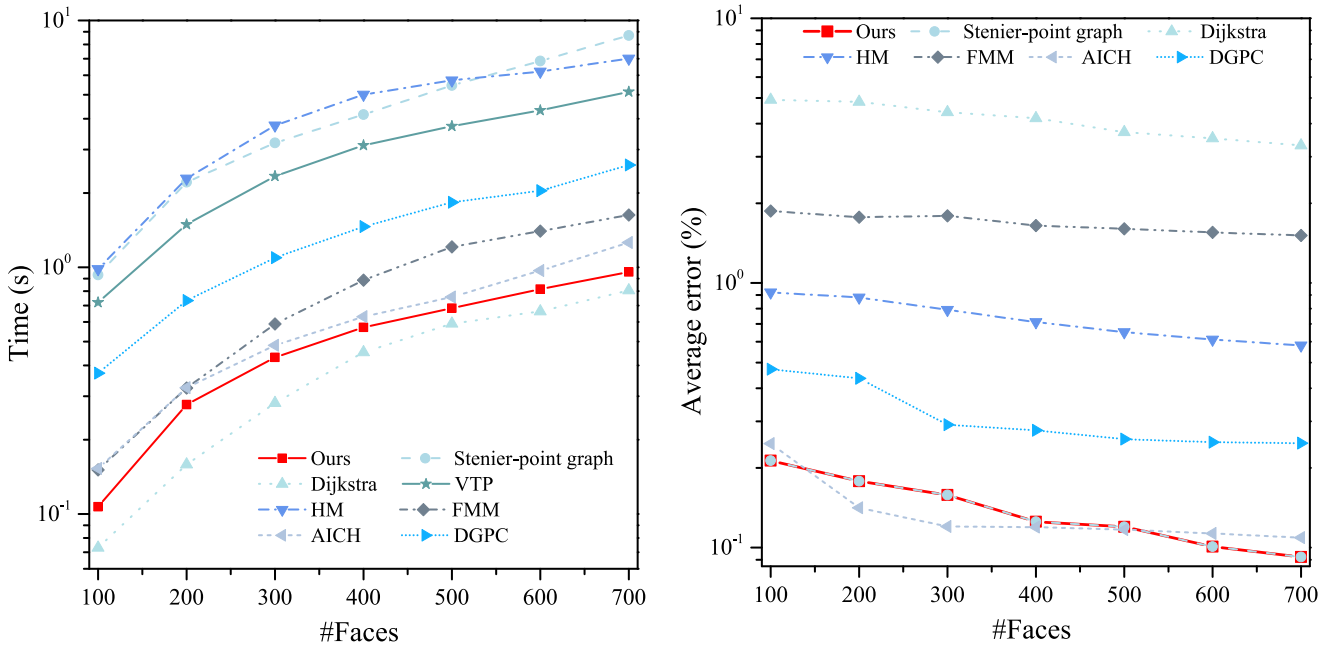
6.4 Qualitative Comparison

Dijkstra's algorithm is most widely used algorithm for computing shortest paths for graphs and triangle meshes due to its simplicity and high efficiency. However, since the resulting paths follow mesh edges, its accuracy is usually quite low especially for anisotropic meshes. Our algorithm, by contrast, is able to achieve a desirable trade-off between accuracy and performance. Users can specify the number Steiner points based on their accuracy and speed requirements.

The Steiner-Point Graph Method. Lanthier *et al.* [15] suggested adding a number of Steiner points on mesh edges to construct an augmented graph and used the Dijkstra's algorithm to compute shortest paths. The biggest difficulty this kind of algorithm faces with is the sharp rise of the complexity of the augmented graph - proportional to k^2. We use two techniques to improve the traditional Steiner method. One is to use a set of filtering rules to facilitate the orderly distance propagation, and the other is to use buckets based structure to reduce the maintenance cost of a priority queue. Our method runs 10 times faster than the conventional Steiner point method with $k = 5$ on a 100K-face model.

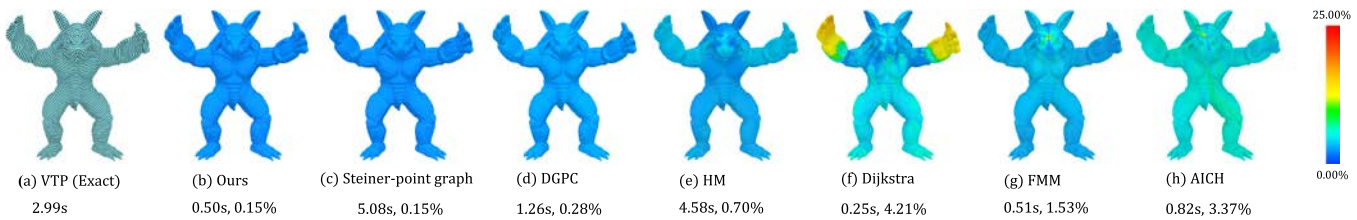
The fast marching method (FMM) [39] has an $O(n \log n)$ time complexity, where n is the number of vertices of the input mesh. Although FMM is asymptotically accurate when the triangulation of the input mesh is sufficiently dense and has a good quality, we observe that the resulting geodesic distance field by FMM has a large average relative error especially around the source point. By contrast, our algorithm is more flexible and works well even if the input mesh is poorly triangulated or has some very large triangles.

Vertex-Sorted Wavefront Propagation (VTP) [14]. The VTP algorithm is the most efficient exact geodesic method if precomputation is not allowed. It uses more filtering rules than the other exact geodesic approaches, and the filtering operation may occur between two windows that arrive at the



(a) Running time (b) Average error

Fig. 14. Plots of running time and accuracy. The horizontal axis is mesh complexity.



(a) VTP (Exact) 2.99s | (b) Ours 0.50s, 0.15% | (c) Steiner-point graph 5.08s, 0.15% | (d) DGPC 1.26s, 0.28% | (e) HM 4.58s, 0.70% | (f) Dijkstra 0.25s, 4.21% | (g) FMM 0.51s, 1.53% | (h) AICH 0.82s, 3.37%

Fig. 15. Comparison in terms of timings and errors on the Armadillo model with 350K faces. (a) shows the exact distances computed by the VTP algorithm, and (b)−(h) the results of representative methods. For the Heat Method, we adopt intrinsic triangulations to improve its robustness and accuracy.

same face but from different edges. In fact, we adapt their filtering rules and use them in our algorithm. Compared with VTP, our Steiner-point based algorithm is easier to implement and more computationally efficient. Taking the 700K Gargoyle model for test, our method are about 10 times faster than the VTP method under the condition that the average error is less than 0.34% (see Fig. 16). Meanwhile, the peak memory of ours is reduced to about one third. Compared with VTP which computed distance field on 700K Gargoyle model using about 5.9s, there is an interesting observation that our method uses about same time with $k = 17$ while the accuracy of our method is under 0.008%.

Geodesics in Heat Method [40]. The heat method is an elegant approach for computing geodesic distances on a wide range of continuous and discrete domains. In the pre-computation stage, it factors the Laplacian matrix. Then given a set of source points, the heat method efficiently computes geodesic distances by solving a pair of linear systems using forward and backward substitutions. Thus, it has an empirical linear time complexity. The heat method controls the accuracy and smoothness of the resulting distances by a parameter of heat diffusion time. To improve accuracy and robustness, it is common to adopt intrinsic Delaunay triangulations [56] or Delaunay meshes [57], which however increases the cost of pre-computaion and model complexity. The HM results have smooth iso-distance lines, which are desired in segmentation, texture mapping, etc. Our method, in contrast, does not require pre-computation at all and is able to compute geodesic distances on the fly. Our method is also memory efficient and can deal with large-scale

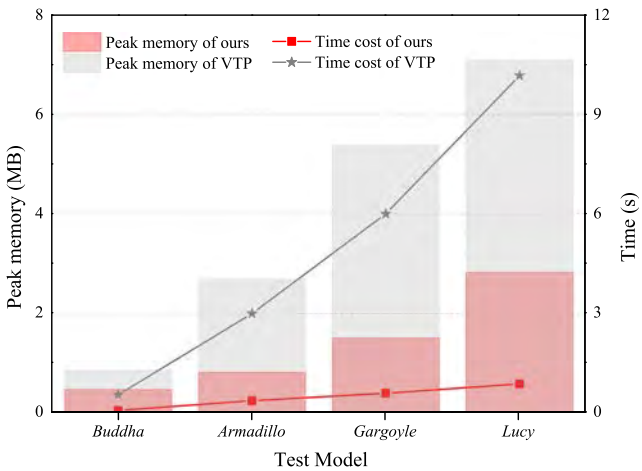


Fig. 16. Comparison with the VTP algorithm on typical models with faces ranging from 100K to 1000K. Setting $k = 3$, our method yields results with average error less than 0.45%, and has roughly linear memory and space complexities.

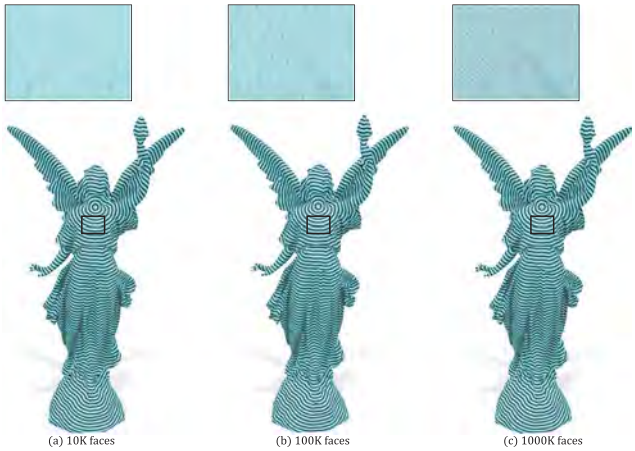


Fig. 17. Our method is insensitive to the input mesh quality and resolution. (a) 10K faces and the Lo value is 0.109, (b) 100K faces and the Lo value is 0.31, (c) 1000K faces and the Lo value is 0.82.

meshes. Moreover, our results are guaranteed to be distance metric. Such a feature is desired in accuracy-critical applications, such as path planning.

Discrete Geodesic Polar Coordinates [42]. Discrete geodesic polar coordinates (DGPC) is based on extending an existing algorithm for computing geodesic distance which is similar to Dijkstra's algorithm, but instead of updating a vertex distance from a neighbour vertex, they use the geometric information from a neighbouring triangle. Compared with DGPC, our method makes a good balance between accuracy and speed; See Table 1.

The Approximate ICH Algorithm [41]. The approximate ICH algorithm (AICH) can obtain a precision controlled approximation by importing a precision parameter λ, which makes the algorithm vary from the improved CH algorithm to Dijkstra's algorithm as λ increases from 0 to 1. Thus, an interesting relationship between them can be naturally established. Similar to the AICH algorithm, our method can make a trade-off between accuracy and performance. However, AICH is not so accurate as our method but more time consuming. As shown in Table 1, our method needs only about 0.5 seconds while the AICH method needs about 0.8 seconds under the requirement of a 3.3% average error.

The fast DGG algorithm [20] improves the original DGG method [18] by reducing both the pre-computation time and graph size. Adikusuma *et al.*'s algorithm can generate a sparse graph with $O(\frac{n}{\sqrt{\varepsilon}})$ edges in empirically $O(\frac{n}{\varepsilon^{0.75}}\log\frac{1}{\varepsilon})$ time. Using CH-relay vertices, their algorithm can efficiently compute short direct geodesic paths. Both the original DGG and the improved DGG algorithms are pre-computation methods, thereby are ideal for frequent geodesic distance query. Our method requires no pre-computation at all and still has good run-time performance and accuracy. For example, test on the Lucy model with 1000K faces shows that the fast DGG method takes 11.16 seconds to construct the search graph and 0.172 seconds to compute distance field with a mean relative error 0.1%. Our method takes totally 2.83 seconds (Steiner point insertion plus the computation of the whole distance field) to report a more accurate distance field (a mean relative error 0.069%). See Table 1 for more details. Our method is flexible to deal with anisotropic metrics and non-uniform density functions, whereas the DGG method cannot.

6.5 Robustness

Our algorithm is numerically stable and the accuracy of the resultant distance fields is insensitive to the mesh quality and the mesh resolution. To deal with long and thin triangles, we add Steiner points according to a specified step [15]. Here we use the Lo value [58] to evaluate the quality of triangles. The higher quality the mesh has, the closer to 1 the Lo value is. We prepare a set of Lucy models that have various meshing qualities (Lo values are 0.109, 0.31 and 0.82 respectively) and various mesh resolutions (10K faces, 100K faces and 1000K faces). As shown in Fig. 17, our method can produce reliable and accurate distance fields on these diverse models.

We particularly examine and compare the method with respect to the mesh's anisotropy degree. We adopt anisotropy measure definition of Xu *et al.* [13] that the anisotropy measure $\tau(f) = \frac{P \cdot H}{2\sqrt{3}S}$ for a triangular face f, where P, H and S are respectively the half-perimeter, the longest edge length and the area of f. The mesh's anisotropy measure is $\tau(f) = \frac{\sum_{f\in F}\tau(f)}{|F|}$, the average of all per face measure. In general, the larger τ, the higher the degree of anisotropy. We select a subset of Thingi10k repository with 200 models whose mean anisotropy measure τ ranges from 1.0 to 120. Herein, Figs. 18a, 18b and 18c illustrate the trade off between performance and accuracy which is controlled by the k for three models with different anisotropic degree, where the polylines denote mean and maximum errors using our method while the bars represent time performance. Fig. 18d represents the average error distribution for the models with different anisotropic degree. In the experiment, we use 200 models in Thingi10k 3D shape repository whose

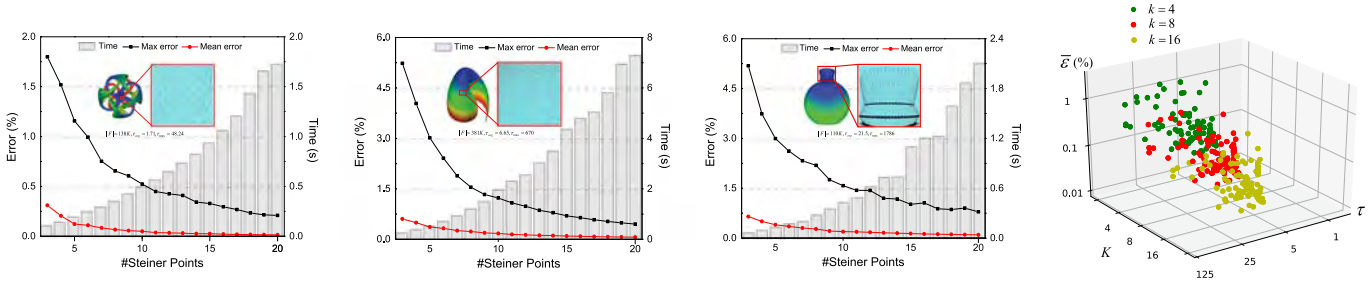


Fig. 18. The trade off between performance and accuracy. (a)-(c) The three models have anisotropic degrees τ_{avg} 1.7, 6.6 and 21.5, respectively. (d) shows 200 randomly selected models in the Thingi10k 3D Shape Repository with mean $\tau_{avg} \in [1.0, 120]$, and maximum $\tau_m \in [2.1, 53455]$. The average error distributions demonstrate our method is well adapted to the model with anisotropy measurement.

mean anisotropy measure τ_e ranges from 1.0 to 120 while maximum anisotropy measure τ_m ranges from 2.1 to 53455. From the results, when the k is set to 8, more than 95% models have the average error under 0.1% while the maximum error is under 1.5%. It shows that our method is well adapted to the model with anisotropy measurement. From the Fig. 18, we can find that the accuracy and performance are slightly influenced by the anisotropic degree. The bad performance appear in the model with extremely large anisotropy degree, that the average error is under 7%.

7 Conclusion and Future Works

In this paper, we improve the naïve Steiner-point based geodesic method by utilizing a set of filtering rules. Throughout the algorithm, we maintain a Steiner-point tree that partitions the surface into mutually exclusive tracks. The tracks can help quickly locate the children of a Steiner point, and thus reduce the total number of distance tests from $O(k^2)$ to empirically $O(k)$, where k is the number of Steiner points on longest length edge of each mesh.

Our algorithm has at least three nice features. First, it has an advantage in terms of speed and accuracy. Second, it works for meshes with poor triangulation and non-manifold meshes and can be extended to a wider class of variant geodesic problems. Finally, the tracks can decompose each triangle into smaller sub-regions such that the change of the distance field in each sub-region can be basic linear, which is helpful in increasing the accuracy in back-tracing shortest paths and extracting geodesic isolines.

In the future, we shall investigate whether a better strategy of adding Steiner points exists. In addition, currently we fail to prove that the time complexity of our algorithm is rigorously linear to the number of Steiner points pre-inserted on each mesh edge, which is also one of our future works.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the National Natural Science Foundation of China under Grants 61772016, 62002190, 61772318, and 62072284, in part by the NSF of Shandong Province, China, under Grant ZR2020MF036, the Key Research Development program of Shandong Province under Grant 2019GGX101021, in part by the Key Program of Shandong Natural Science Foundation under Grant 2020ZLYS01, and in part by the Singapore Ministry of Education under Grants RG20/20 and T2EP20220-0014.

References

[1] Y. Liu, D. Fan, C. Xu, and Y. He, "Constructing intrinsic Delaunay triangulations from the dual of geodesic Voronoi diagrams," *ACM Trans. Graph.*, vol. 36, no. 2, pp. 1–15, 2017.
[2] B. L. Price, B. Morse, and S. Cohen, "Geodesic graph cut for interactive image segmentation," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2010, pp. 3161–3168.
[3] J. Rabin, G. Peyré, and L. D. Cohen, "Geodesic shape retrieval via optimal mass transport," in *Proc. Eur. Conf. Comput. Vis.*, 2010, pp. 771–784.
[4] J. Ye and Y. Yu, "A fast modal space transform for robust nonrigid shape retrieval," *Vis. Comput.*, vol. 32, no. 5, pp. 553–568, 2016.
[5] G. Peyré and L. Cohen, "Geodesic computations for fast and accurate surface remeshing and parameterization," in *Elliptic and Parabolic Problems*. Springer, 2005, vol. 63, pp. 157–171.
[6] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on Riemannian manifolds," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, 2015, pp. 832–840.
[7] T. He *et al.*, "Geonet: Deep geodesic networks for point cloud analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 6881–6890.
[8] P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer, "A survey of geodesic paths on 3D surfaces," *Comput. Geometry*, vol. 44, no. 9, pp. 486–498, 2011.
[9] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM J. Comput.*, vol. 16, no. 4, pp. 647–668, 1987.
[10] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *Proc. 6th Annu. Symp. Comput. Geometry*, 1990, pp. 360–369.
[11] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 553–560, 2005.
[12] S. Xin and G. Wang, "Improving Chen and Han's algorithm on the discrete geodesic problem," *ACM Trans. Graph.*, vol. 28, no. 4, pp. 1–8, 2009.
[13] C. Xu, T. Y. Wang, Y. Liu, L. Liu, and Y. He, "Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes," *IEEE Trans. Vis. Comput. Graph.*, vol. 21, no. 7, pp. 822–834, Jul. 2015.
[14] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang, "Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–13, 2016.
[15] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating shortest paths on weighted polyhedral surfaces," *Algorithmica*, vol. 30, no. 4, pp. 527–562, 2001.
[16] S. Xin, X. Ying, and Y. He, "Constant-time all-pairs geodesic distance query on triangle meshes," in *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. Games*, 2012, pp. 31–38.
[17] X. Ying, X. Wang, and Y. He, "Saddle vertex graph (SVG): A novel solution to the discrete geodesic problem," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–12, 2013.
[18] X. Wang, Z. Fang, J. Wu, S. Xin, and Y. He, "Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces," *Comput. Aided Geometric Des.*, vol. 52–53, pp. 262–284, 2017.
[19] S. Xin *et al.*, "Lightweight preprocessing and fast query of geodesic distance via proximity graph," *Comput.-Aided Des.*, vol. 102, pp. 128–138, 2018.
[20] Y. Y. Adikusuma, Z. Fang, and Y. He, "Fast construction of discrete geodesic graphs," *ACM Trans. Graph.*, vol. 39, no. 2, pp. 1–14, 2020.
[21] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, 1959.
[22] Y. Liu, Q. Zhou, and S. Hu, "Handling degenerate cases in exact geodesic computation on triangle meshes," *Vis. Comput.*, vol. 23, no. 9, pp. 661–668, 2007.
[23] D. Bommes and L. Kobbelt, "Accurate computation of geodesic distance fields for polygonal curves on triangle meshes," in *Proc. 12th Int. Fall Workshop Vis., Model.,Vis.*, 2007, pp. 151–160.
[24] S. Xin, X. Ying, and Y. He, "Efficiently computing geodesic offsets on triangle meshes by the extended Xin-Wang algorithm," *Comput.-Aided Des.*, vol. 43, no. 11, pp. 1468–1476, 2011.
[25] X. Ying, S. Xin, and Y. He, "Parallel Chen-Han (PCH) algorithm for discrete geodesics," *ACM Trans. Graph.*, vol. 33, no. 1, pp. 1–11, 2014.
[26] X. Ying *et al.*, "Parallelizing discrete geodesic algorithms with perfect efficiency," *Comput.-Aided Des.*, vol. 115, pp. 161–171, 2019.
[27] J. Du, Y. He, Z. Fang, W. Meng, and S. Xin, "On the vertex-oriented triangle propagation (VTP) algorithm: Parallelization and approximation," *Comput.-Aided Des.*, vol. 130, 2021, Art. no. 102943.
[28] M. Lanthier, A. Maheshwari, and J.-R. Sack, "Approximating weighted shortest paths on polyhedral surfaces," in *Proc. 13th Annu. Symp. Comput. Geometry*, 1997, pp. 274–283.
[29] M. Lanthier, D. Nussbaum, and J.-R. Sack, "Parallel implementation of geometric shortest path algorithms," *Parallel Comput.*, vol. 29, no. 10, pp. 1445–1479, 2003.
[30] L. Aleksandrov, M. Lanthier, A. Maheshwari, and Sack, "An epsilon-approximation algorithm for weighted shortest paths on polyhedral surfaces," in *Proc. Scand. Workshop Algorithm Theory*, 1998, pp. 11–22.
[31] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Approximation algorithms for geometric shortest path problems," in *Proc. 32nd Annu. ACM Symp. Theory Comput.*, 2000, pp. 286–295.
[32] L. Aleksandrov, A. Maheshwari, and J. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," *J. ACM*, vol. 52, no. 1, pp. 25–53, 2005.
[33] Z. Sun and J. H. Reif, "On finding approximate optimal paths in weighted regions," *J. Algorithms*, vol. 58, no. 1, pp. 1–32, 2006.

[34] T. Kanai and H. Suzuki, "Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications," in *Proc. Geometric Model. Process. Theory Appl.*, 2000, pp. 241–250.
[35] C. S. Mata and J. S. Mitchell, "A new algorithm for computing shortest paths in weighted planar subdivisions," in *Proc. 13th Annu. Symp. Comput. Geometry*, 1997, pp. 264–273.
[36] K. R. Varadarajan and P. K. Agarwal, "Approximating shortest paths on a nonconvex polyhedron," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1321–1340, 2000.
[37] G. Nazzaro, E. Puppo, and F. Pellacini, "Decosurf: Recursive geodesic patterns on triangle meshes," 2020, *arXiv: 2007.10918*.
[38] R. Aiello, F. Banterle, N. Pietroni, L. Malomo, P. Cignoni, and R. Scopigno, "Compression and querying of arbitrary geodesic distances," in *Proc. Int. Conf. Image Anal. Process.*, 2015, pp. 282–293.
[39] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proc. Nat. Acad. Sci.*, vol. 95, no. 15, pp. 8431–8435, 1998.
[40] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in heat: A new approach to computing distance based on heat flow," *ACM Trans. Graph.*, vol. 32, no. 5, pp. 1–11, 2013.
[41] S. Xin and G. Wang, "Applying the improved Chen and Han's algorithm to different versions of shortest path problems on a polyhedral surface," *Comput.-Aided Des.*, vol. 42, no. 10, pp. 942–951, 2010.
[42] E. L. Melvr and M. Reimers, "Geodesic polar coordinates on polygonal meshes," *Comput. Graph. Forum*, vol. 31, no. 8, pp. 2423–2435, 2012.
[43] M. Campen, M. Heistermann, and L. Kobbelt, "Practical anisotropic geodesy," in *Proc. 11th Eur./ACM SIGGRAPH Symp. Geometry Process.*, 2013, pp. 63–71.
[44] B. Liu, S. Chen, S. Xin, Y. He, Z. Liu, and J. Zhao, "An optimization-driven approach for computing geodesic paths on triangle meshes," *Comput.-Aided Des.*, vol. 90, pp. 105–112, 2017.
[45] S. Xin and G. Wang, "Efficiently determining a locally exact shortest path on polyhedral surfaces," *Comput.-Aided Des.*, vol. 39, no. 12, pp. 1081–1090, 2007.
[46] Z. Ye, Y. Liu, J. Zheng, K. Hormann, and Y. He, "De-path: A differential-evolution-based method for computing energy-minimizing paths on surfaces," *Comput.-Aided Des.*, vol. 114, pp. 73–81, 2019.
[47] L. Cao *et al.*, "Computing smooth quasi-geodesic distance field (QGDF) with quadratic programming," *Comput.-Aided Des.*, vol. 127, 2020, Art. no. 102879.
[48] N. Sharp and K. Crane, "You can find geodesic paths in triangle meshes by just flipping edges," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–15, 2020.
[49] X. Ying *et al.*, "Parallelizing discrete geodesic algorithms with perfect efficiency," *Comput.-Aided Des.*, vol. 115, pp. 161–171, 2019.
[50] J. Tao, J. Zhang, B. Deng, Z. Fang, Y. Peng, and Y. He, "Parallel and scalable heat methods for geodesic distance computation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, pp. 579–594, Feb. 2021.
[51] R. Farias and M. Kallmann, "Optimal path maps on the GPU," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 9, pp. 2863–2874, Sep. 2020.
[52] L. A. R. Calla, L. J. F. Perez, and A. A. Montenegro, "A minimalistic approach for fast computation of geodesic distances on triangular meshes," *Comput. Graph.*, vol. 84, pp. 77–92, 2019.
[53] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," *SIAM J. Comput.*, vol. 15, no. 1, pp. 144–153, 1984.
[54] Y. Yan, K. Sykes, E. Chambers, D. Letscher, and T. Ju, "Erosion thickness on medial axes of 3D shapes," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–12, 2016.
[55] Q. Zhou and A. Jacobson, "Thingi10k: A dataset of 10,000 3D-printing models," 2016, *arXiv:1605.04797*.
[56] N. Sharp, Y. Soliman, and K. Crane, "Navigating intrinsic triangulations," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–16, 2019.
[57] Y. Liu, C. Xu, D. Fan, and Y. He, "Efficient construction and simplification of Delaunay meshes," *ACM Trans. Graph*, vol. 34, no. 6, pp. 1–13, 2015.
[58] S. H. Lo, "A new generation scheme for arbitrary planar domains," *Int. J. Numer. Methods Eng.*, vol. 21, no. 8, pp. 1403–1426, 1985.

Wenlong Meng is currently working toward the PhD degree in computer science with Shandong University. His research interests include computational geometry, computer graphics, and computer-aided design.

Shiqing Xin received the PhD degree from Zhejiang University, China, in 2009. He was a research fellow with Nangyang Technological University, Singapore, for three years. He is currently an associate professor with the School of Computer Science, Shandong University. He has authored or coauthored more than 60 papers in top journals and conferences, including the *IEEE Transactions on Visualization and Computer Graphics* and *ACM Transactions on Graphics*. His research interests include various geometry processing algorithms, especially geodesic computation approaches and Voronoi/power tessellation methods. He was the recipient of three best paper awards and many other academic awards.

Changhe Tu received the BSc, MEng, and PhD degrees from Shandong University, China, in 1990, 1993, and 2003, respectively. He is currently a professor with the School of Computer Science and Technology, Shandong University, China. He currently leads the CG-VIS Group, Shandong University. He has authored or coauthored more than 100 papers in international journals and conferences. His research interests include computer graphics, 3D vision, and computer-aided geometric design.

Shuangmin Chen received the PhD degree from Ningbo University in 2018. She is currently an assistant professor with the School of Information and Technology, Qingdao University of Science and Technology, China. She has authored or coauthored more than 40 research papers in famous journals and conferences. Her research interests include computer graphics and computational geometry.

Ying He (Member, IEEE) received the BS and MS degrees in electrical engineering from Tsinghua University, and the PhD degree in computer science from Stony Brook University. He is currently an associate professor with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include the problems that require geometric computing and analysis.

Wenping Wang (Fellow, IEEE) received the PhD degree in computer science from the University of Alberta in 1992. He is currently a chair professor of computer science with the University of Hong Kong. His research interests include computer graphics, computer visualization, computer vision, robotics, medical image processing, and geometric computing. He is an associate editor for several premium journals, including *Computer Aided Geometric Design* (CAGD), *Computer Graphics Forum* (CGF), *IEEE Transactions on Computers*, and *IEEE Computer Graphics and Applications*, and has chaired more than 20 international conferences, including Pacific Graphics 2012, ACM Symposium on Physical and Solid Modeling (SPM) 2013, SIGGRAPH Asia 2013, and Geometry Submit 2019. He was the recipient of the John Gregory Memorial Award for his contributions in geometric modeling.