Computer Aided Geometric Design 52–53 (2017) 262–284

Contents lists available at ScienceDirect

Computer Aided Geometric Design

www.elsevier.com/locate/cagd

Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces

Xiaoning Wang [a], Zheng Fang [a], Jiajun Wu [a], Shi-Qing Xin [b], Ying He [a,*]

[a] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[b] School of Computer Science and Technology, Shandong University, China

ARTICLE INFO

Article history:
Available online 29 March 2017

Keywords:
Geodesic distances
Polyhedral surfaces
Discrete geodesic graph
Saddle vertex graph

ABSTRACT

We present a new graph-based method, called discrete geodesic graph (DGG), to compute discrete geodesics in a divide-and-conquer manner. Let M be a manifold triangle mesh with n vertices and $\varepsilon > 0$ the given accuracy parameter. Assume the vertices are uniformly distributed on the input mesh. We show that the DGG associated to M has $O(\frac{n}{\sqrt{\varepsilon}})$ edges and the shortest path distances on the graph approximate geodesic distances on M with relative error $O(\varepsilon)$. Computational results show that the actual error is less than 0.6ε on common models. Taking advantage of DGG's unique features, we develop a DGG-tailored label-correcting algorithm that computes geodesic distances in empirically linear time. With DGG, we can guarantee the computed distances are true distance metrics, which is highly desired in many applications. We observe that DGG significantly outperforms saddle vertex graph (SVG) – another graph based method for discrete geodesics – in terms of graph size, accuracy control and runtime performance.

1. Introduction

Measuring geodesic distances on discrete domains is important to many graphics applications. To date, most of the discrete geodesic algorithms focus on manifold triangle meshes. Two notable classes of such algorithms are the discrete wavefront methods and the partial differential equation (PDE) methods. The discrete wavefront methods maintain wavefront on mesh edges and propagate it across the faces in a Dijkstra-like sweep. Representative works include the Mitchell–Mount–Papadimitriou (MMP) algorithm (Mitchell et al., 1987) and the Chen–Han (CH) algorithm (Chen and Han, 1990). These methods are able to compute the exact geodesic distances, if the numerical computation is exact. However, they are computationally expensive and difficult to implement. The PDE methods, such as the fast marching method (FMM) (Kimmel and Sethian, 1998) and the heat method (HM) (Crane et al., 2013), aim at numerically solving the Eikonal equation on discrete domains. The PDE methods have many advantages over the discrete wavefront methods: they are easy to implement, efficient and highly flexible to work on a wide range of discrete domains. But they compute only the first order approximation, which is sensitive to mesh resolution and tessellation.

Recently, a new class of technique, called saddle vertex graph (SVG) (Ying et al., 2013), has drawn attention. Unlike the above-mentioned methods which compute geodesic distances on meshes directly, the SVG method is based on a divide-

* Corresponding author.
E-mail address: yhe@ntu.edu.sg (Y. He).

http://dx.doi.org/10.1016/j.cagd.2017.03.010
0167-8396/

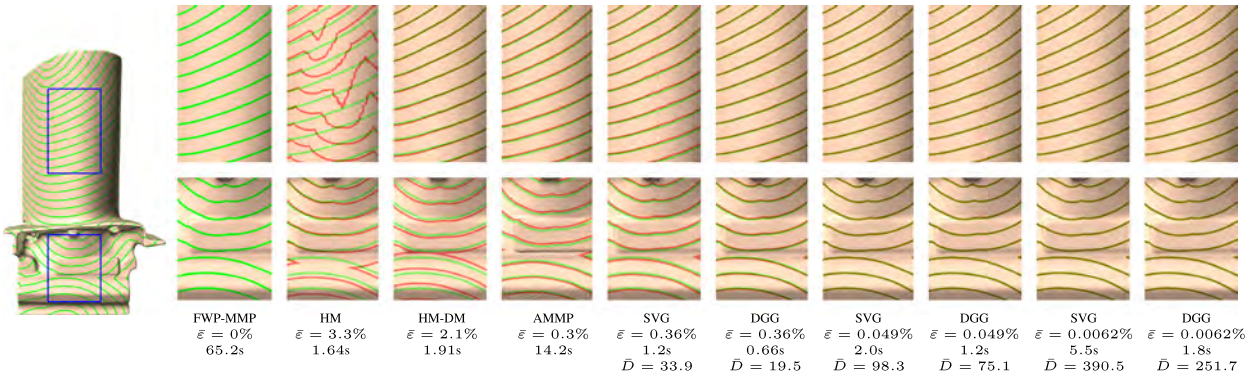


Fig. 1. Computing the single-source geodesic distances on the 1M-vertex Blade model. It takes the FWP-MMP method – an exact discrete geodesic algorithm – 65.2 seconds, whereas the pre-computation methods, such as SVG, DGG and HM, are much more efficient. DGG outperforms SVG in terms of accuracy, runtime performance and space complexity. Given the same accuracy measure, the DGG is only 2/3 the size of the SVG and it runs 2 to 5 times faster than SVG. DGG is also more accurate and efficient than the heat method and its variant HM-DM that adopts a Delaunay mesh as preconditioner. To visualize the quality of the results, we overlay the exact isolines of the geodesic distances (in green) with the isolines obtained by the approximate algorithms (in red). $\bar{D}$ is the average degree of the graphs and $\bar{\varepsilon}$ is the relative mean error. The timings of SVG, DGG, HM and HM-DM do not include the pre-computation time. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and-conquer strategy. It partitions long geodesic paths into shorter segments terminating at saddle vertices, and then forms a sparse graph G using these segments. As a result, a geodesic path on the input mesh is equivalent to a shortest path on G, which can be computed easily by Dijkstra's algorithm (Dijkstra, 1959). The SVG method allows information reuse, making it an effective tool for discrete geodesic. Since the shortest-path distances are a metric, the computed geodesic distances (regardless of exact or approximate) are guaranteed to be a metric. Moreover, the denser the SVG, the higher accuracy the algorithm has, and of course, the longer time it takes to construct the graph.

In spite of many nice properties, the SVG method has three limitations: First, users cannot specify the desired accuracy of the computed geodesic distances. Instead, users have to control the graph size by bounding the maximal degree, which is indirect and non-intuitive. Second, its space complexity and runtime performance highly depend on the number of saddle vertices and their distribution. Computational results show that adding noise to vertices can significantly reduce the graph size, hereby improve the runtime performance. However, the change of geometry compromises the quality of the computed geodesic distances. It is also unclear whether SVG can be generalized to other discrete domains, such as point clouds, which lacks a clear definition of saddle vertices. Third, SVG is built upon mesh vertices, hereby it can compute geodesic distances between vertices only. However, some applications require geodesic distances between arbitrary points on the input mesh. The commonly-used linear interpolation often produces poor results for meshes with non-uniform triangles, since geodesic distances are highly non-linear.

In this paper, we propose a new graph-based method, called discrete geodesic graph (DGG), which has all of the desired properties. Motivated by parallel transport on smooth manifolds, DGG approximates a long geodesic path γ by a series of *short* geodesic paths that are close enough to γ. Unlike SVG which heavily uses saddle vertices as relays, DGG can be flexibly constructed using any kinds of vertices, hence working for arbitrary polyhedral surfaces. This seemingly minor change produces a fundamentally different graph scheme that allows us to directly specify the approximation error. Let M be a manifold triangle mesh with n vertices and $\varepsilon > 0$ the given accuracy parameter. Assume that the vertices are uniformly distributed on M, we prove that the DGG associated to M has $O(\frac{n}{\sqrt{\varepsilon}})$ edges and the shortest path distances on the graph approximate geodesic distances with relative error $O(\varepsilon)$. Computational results show that the actual error does not exceed 0.6ε on common mesh models.

Taking advantage of DGG's unique features, we develop a DGG-tailored label-correcting algorithm for computing geodesic distances, which empirically runs in linear time. Although DGG is built upon mesh vertices, we can easily construct a locally augmented graph to compute geodesic distances and paths between two arbitrary points (not necessarily being mesh vertices) in the same level of accuracy as the original graph. We evaluate DGG on a wide range of triangle meshes and observe that DGG significantly outperforms SVG in terms of space complexity, accuracy control and runtime performance. See Fig. 1 for an example. To ease reading, Table 1 lists the main notations.

2. Related works

The algorithms for computing discrete geodesics can be broadly categorized as discrete wavefront propagation methods, PDE methods, $(1+\varepsilon)$-approximation algorithms and graph-based methods. This section briefly reviews the first three classes of approaches and the next section introduces the existing graph-based methods in details.

Table 1
Main notations.

M	a connected, manifold triangle mesh
V, E, F	the sets of vertices, edges and faces of M
$n\ (=\|V\|)$	the number of vertices
τ	anisotropy measure
$\gamma(p,q)$	the geodesic path between p and q
$\mathcal{F}(\gamma(p,q))$	the face sequence containing $\gamma(p,q)$
$\hat{x}$	the unfolded 2D image of an element $x \in M$
$\varepsilon\ (>0)$	the accuracy parameter
$\bar{\varepsilon}$	the actual error
K	the maximal degree of SVG
$\bar{D}$	the average degree of SVG/DGG
$w \subseteq e$	a window associated to the oriented half-edge e
$\mathcal{E}_\varepsilon(x,y)$	the ellipse with focal points x and y and eccentricity $\frac{1}{1+\varepsilon}$
$G_s = (V, E_s)$	the saddle vertex graph with edge set E_s
$G_\varepsilon = (V_\varepsilon, E_\varepsilon)$	the discrete geodesic graph with accuracy ε
$d_g(\cdot,\cdot)$	geodesic distance on the mesh M
$d_s(\cdot,\cdot)$	shortest path distance on the graph G
$\|AB\|$	the Euclidean distance between A and B

2.1. Discrete wavefront propagation methods

The discrete wavefront methods maintain wavefront on mesh edges and propagate it across the faces in a Dijkstra-like sweep. The MMP algorithm (Mitchell et al., 1987) and the CH algorithm (Chen and Han, 1990) are two representative methods. They represent discrete wavefronts in a similar way by partitioning mesh edges into intervals, called windows, where each window encodes the geodesic paths that are in the same face sequence. Then they propagate windows across mesh faces and update the geodesic distance of vertices when they are swept by some window. The MMP algorithm maintains a priority queue for windows and propagates the window closest to the source at a time, whereas the CH algorithm organizes windows in a tree and processes them in a breadth-first search order. Given an n-face mesh, the MMP algorithm runs in $O(n^2 \log n)$ time and the CH algorithm has an $O(n^2)$ time complexity. Both algorithms have many variants, aiming at improving runtime performance (Surazhsky et al., 2005; Xin and Wang, 2009; Liu, 2013; Liu et al., 2007; Xu et al., 2015; Qin et al., 2016), parallelization (Ying et al., 2014), computing polyline-sourced distances (Campen et al., 2013; Xin et al., 2011), and handling degeneracy (Liu et al., 2007). Mitchell et al. (1987) showed that there are at most $O(n^2)$ windows on an n-face mesh, so the window propagation algorithms have a worst-case $O(n^2)$ time complexity on general polyhedral surfaces, which is known as the theoretical quadratic time barrier. As observed in Surazhsky et al. (2005), the MMP algorithm has an empirical $O(n^{1.5} \log n)$ time complexity on real-world meshes.

Schreiber and Sharir (2008) proposed an elegant algorithm for computing geodesic distances on a convex polytope in $O(n \log(n))$ time, reaching the theoretical lower bound. Later, Schreiber (2010) generalized the optimal-time algorithm for convex polytopes to terrains, uncrowded polyhedrons and self-conforming models, which can be concave. Although these algorithms have significant theoretical values, the implementation is highly non-trivial and their application domains are quite limited. To date, the problem of computing *exact* geodesic distances on general polyhedral surfaces in sub-quadratic time still remains open.

2.2. PDE methods

The PDE methods solve the Eikonal equation $\|\nabla u(x)\| = 1$ with boundary condition $u(s) = 0$ on discrete domains, where s is the source. In contrast to the exact methods which are computationally expensive, the PDE methods are very easy to implement and efficient.

Using an upwind finite difference approximation to the gradient and a Dijkstra-like sweep, Sethian (1996) proposed the fast marching method (FMM), which provides a solution to the Eikonal equation in $O(n \log n)$ time on a regular grid. A similar algorithm based on a different discretization of the Eikonal equation was developed independently by Tsitsiklis (1995). Later, FMM was generalized to arbitrary triangulated surfaces (Kimmel and Sethian, 1998), unstructured meshes (Sethian and Vladimirsky, 2000), implicit surfaces (Mémoli and Sapiro, 2001), parametric surfaces (Spira and Kimmel, 2004) and broken meshes (Campen and Kobbelt, 2011). Due to a strict updating order and a priority queue to manage the narrow band containing the wavefront, it is non-trivial to parallelize the fast marching method. Using raster scan on a completely regular structure, Weber et al. (2008) developed a parallel FMM on geometry images with an $O(n)$ time complexity, significantly outperforming the original FMM. Other parallel methods to solve the Eikonal equation include Zhao (2007), Fu et al. (2011) and Detrixhe et al. (2013).

There are also methods to *indirectly* solve the Eikonal equation. Initialized with Euclidean distances, Xin et al. (2012a) iteratively improved the distances from their normalized gradients. Although their method works fairly well on broken meshes, it lacks theoretical guarantee of convergence. Crane et al. (2013) proposed the heat method, which is based on a different strategy. It places heat at the source vertex and then diffuses it in a very short time. Since the normalized

gradient of the heat function coincides with the gradient of the geodesic distance function according to Varadhan's formula (Varadhan, 1967), the heat method computes the geodesic distance by solving a Poisson equation. Using Cholesky factorization of the Laplacian matrix, both the heat flow and the Poisson equation can be solved in linear time. Solomon et al. (2014) considered the more general problem of computing earth mover's distances (EMD) between probability distributions supported on meshes. Rather than the brute force linear programming approach, they proposed an alternative differential formulation of EMD that can be discretized using finite elements (FEM). Their method leads to a family of distances that ranges from the geodesic distance to bi-harmonic distance (Lipman et al., 2010). Recently, Belyaev and Fayolle (2015) introduced a variational approach for computing the distance to a surface and proposed efficient iterative schemes for minimizing the energy functionals. They also investigated several PDE-based distance function approximation schemes, including Poisson distance, p-Laplacian distance and L_p-distance.

The PDE methods work for a wide range of discrete domains, including regular grids, point clouds, and unstructured triangular and tetrahedral meshes. However, they provide only the first-order approximation, which is highly sensitive to mesh tessellation.

2.3. $(1+\varepsilon)$-approximation algorithms

As mentioned above, measuring *exact* geodesic distances is expensive. Although the existing PDE methods and the SVG method are efficient, they lack rigorous error analysis. In computational geometry, given an accuracy parameter $\varepsilon > 0$, the $(1+\varepsilon)$-approximate algorithm aims at computing a geodesic path between two given points, whose length is at most $(1+\varepsilon)$ times the length of the exact path. Such a path is called an ε-approximate path.

Hershberger and Suri (1995) presented an algorithm that computes a 2-approximate path on a *convex* polyhedron in $O(n)$ time. However, their algorithm does not seem to extend to yield better approximation factors. Agarwal et al. (1997) presented an algorithm that computes an ε-approximate path on a convex polyhedron in $O(n \log 1/\varepsilon + 1/\varepsilon^3)$ time for any prescribed $\varepsilon > 0$. Har-Peled (1997) showed that Agarwal et al.'s algorithm can be used to preprocess a convex polytope P with n faces in time $O(n)$, so that an ε-approximate path between any two query points on ∂P can be computed in time $O(1/\varepsilon^{1.5} \log n + 1/\varepsilon^3)$. Unfortunately, these algorithms heavily exploit the convexity of the polyhedron and do not extend to nonconvex polyhedra.

Har-Peled (1998) proposed an algorithm for computing $(1+\varepsilon)$-approximations of geodesic paths on general polyhedral surfaces. Given a source point s on a polyhedral surface P, his algorithm computes a subdivision of P based on Voronoi diagrams on P. The subdivision has size $O(n/\varepsilon \log 1/\varepsilon)$ and is computed in $O(n^2 \log n + n/\varepsilon \log(1/\varepsilon) \log(n/\varepsilon))$ time. After this preprocessing step, a $(1+\varepsilon)$-approximation of the geodesic path between s and any point $p \in P$ can be reported in $O(\log(n/\varepsilon))$ time. Therefore, the single-source geodesic distance can be computed in $O(n \log(n/\varepsilon))$ time.

Varadarajan and Agarwal (2000) developed two algorithms for approximating geodesic paths on a nonconvex polyhedron. Both algorithms are based on the same computational framework: given a parameter r, they first partition ∂P into $O(n/r)$ patches, each consisting of at most r faces. For each patch, they introduce Steiner points on the boundary of each patch and construct a graph to approximate the actual shortest path within the desired accuracy. Finally, they merge these graphs into a single graph and compute shortest paths using Dijkstra's algorithm. The first algorithm runs in $O(n^{5/3} \log^{5/3} n)$ time to compute an approximation to the shortest path with approximation ratio $7(1+\varepsilon)$. The second algorithm takes only $O(n^{8/5} \log^{8/5} n)$ time, but the approximation ratio increases to $15(1+\varepsilon)$. Varadarajan and Agarwal's algorithms are the first to break the quadratic time complexity, however they are not practical due to the large approximation factor.

3. Preliminaries

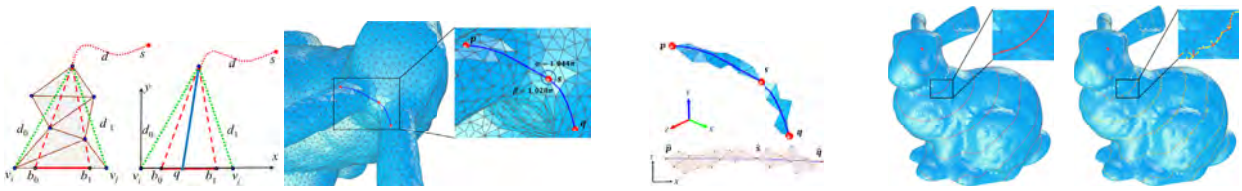
3.1. Geodesic paths & discrete wavefronts

Let us denote a connected, manifold triangle mesh by $M = (V, E, F)$, where V, E, F are the vertex, edge and face sets respectively. A vertex is called spherical, Euclidean or saddle if its cone angle is less than, equal to, or greater than 2π. Given two points $p, q \in M$, the geodesic path between p and q is denoted by $\gamma(p,q)$. Since geodesic paths are not unique in general, unless otherwise specified, $\gamma(p,q)$ refers to the global shortest geodesic path in this paper. The following theorem describes discrete geodesic paths.

Theorem 1 *(Mitchell et al. 1977). A geodesic path on a mesh is an alternating sequence of vertices and (possibly empty) edge sequences such that the unfolded image of the path along any edge sequence is a straight line segment and the angle of γ passing through a vertex is greater than or equal to π.*

Let us denote $\mathcal{F}(\gamma(p,q)) \subseteq F$ the set of faces that contains geodesic path $\gamma(p,q)$. Throughout the paper, we use the hat symbol $\hat{x} \in \mathbb{R}^2$ to denote the unfolded 2D image of an element $x \in M$. For example, given a vertex $v \in \mathcal{F}(\gamma(p,q))$ on the face sequence containing $\gamma(p,q)$, $\hat{v} \in \mathbb{R}^2$ is the 2D image after unfolding the faces to $\mathbb{R}^2$. Similarly, given a face sequence $\mathcal{F}(\gamma(p,q))$, $\hat{\mathcal{F}}(\gamma(p,q)) \subset \mathbb{R}^2$ is the unfolded 2D faces containing $\hat{\gamma}(p,q)$.

Let $s \in V$ be the source vertex. A wavefront consists of points which are equidistant to source s. Unlike Euclidean plane $\mathbb{R}^2$, where wavefronts are concentric circles sharing the same center s, wavefronts on polyhedral surfaces consist of arcs,



(a) A window and its 2D parameterization (b) Geodesic path $\gamma(p,q)$ passing through a saddle vertex s (c) Face sequence $\mathcal{F}(\gamma(p,q))$ and its 2D unfolded image $\hat{\mathcal{F}}(\gamma(p,q))$ (d) Continuous wavefronts (e) Discrete wavefronts

Fig. 2. Windows, geodesic paths and wavefronts. (a) shows a window $w = (e_{ij}, b_0, b_1, d_0, d_1, d)$ associated to the edge $e_{ij} = (v_i, v_j)$. Parameterizing w to $\mathbb{R}^2$, we can compute the geodesic distance $d_g(s,q)$ for any point $q \in w$. (b) shows a geodesic path $\gamma(p,q)$ on the Bunny model. Since $\gamma(p,q)$ passes through a saddle vertex s, it is an indirect geodesic path. Both angles of γ passing through s are no less than π. (c) shows the face sequence $\mathcal{F}(\gamma) \subseteq F$ containing γ and its unfolded image $\hat{\mathcal{F}}(\gamma)$ in $\mathbb{R}^2$. (d) shows continuous wavefronts at a few time steps and (e) shows the corresponding discrete wavefronts of the MMP algorithm. Each colored segment in (e) is a window, which encodes the geodesic paths going through the same face sequence.

which are centered at either source s or saddle vertices. Since it is difficult to maintain a continuous wavefront, both the MMP and CH algorithms simulate *discrete* wavefront propagation.

Imagine s is the source of light, and geodesic paths are visualized as light rays emanating from s in all tangent directions. The basic idea of discrete wavefront propagation is to track together groups of shortest paths that can be parameterized atomically. This is achieved by partitioning each mesh edge into a set of intervals called windows (Surazhsky et al., 2005). A window, as the word literally means, is a segment on an edge such that the geodesic path to any point on the segment trespasses the same sequence of faces and vertices. See Fig. 2(a). The last saddle vertex (if exists) on the face/vertex sequence is called *pseudo-source*.

Mathematically speaking, a window w associated to an oriented edge $e_{ij} = (v_i, v_j)$ is a 6-tuple $(e_{ij}, b_0, b_1, d_0, d_1, d)$, where

- b_0, b_1 are the endpoints of w;
- d_0, d_1 are the distances from the pseudo-source to v_i and v_j;
- d is the distance from the source to the pseudo-source.

A window is called *direct* if it is lit directly from source s, i.e., $d = 0$.

The MMP algorithm propagates the windows out from the source in a Dijkstra-like sweep. The windows, when created, are placed in a priority queue $\mathcal{Q}$, sorted by their distances back to the source. When a window w is popped off the queue, it is propagated outward across its adjacent face, producing new windows on the opposite edge(s). If an edge already contains some previously propagated windows that are overlapping with the new window, the MMP algorithm intersects the new window with the old one to determine the minimal distance. The new windows are then inserted to $\mathcal{Q}$. The algorithm terminates when the priority queue becomes empty. Mitchell et al. (1987) showed that each edge has up to $O(n)$ windows, implying that the window complexity is $O(n^2)$. Since it takes $O(\log n)$ time for the priority queue to sort windows in each iteration, the MMP algorithm has an $O(n^2 \log n)$ time complexity.

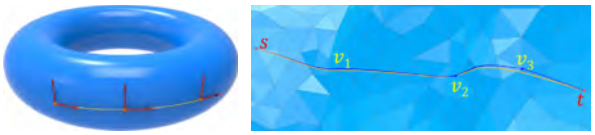
3.2. Graph-based methods

Graph-based methods solve the discrete geodesic problem by constructing an undirected graph G so that a geodesic path on the mesh M is (approximately) equal to a shortest path on G. Throughout this paper, we denote by $d_g(\cdot,\cdot)$ to be the geodesic distance on the mesh M, $d_s(\cdot,\cdot)$ the shortest path distance on the associated graph G, and $\|pq\|$ the Euclidean distance between p and q.

Obviously, a weighted complete graph $G_c = (V, E_c)$ with the edge set $E_c = \{e_{ij} | w_{ij} = d_g(v_i, v_j),\ \forall v_i, v_j \in V\}$ computes the exact single-source geodesic distances in $O(n)$ time. However, it is extremely expensive to construct and store G_c. Therefore, practical graph-based methods aim at computing some sort of *subgraph* of G_c, which balances accuracy and computational cost.

The first graph-based method is due to Aleksandrov et al. (1998), who proposed a $(1+\varepsilon)$-approximation algorithm for computing weighted shortest paths on polyhedral surfaces. The cost of travel through each face is the distance traveled multiplied by the face's weight. Given a polyhedron $\mathcal{P}$, their idea is to discretize $\mathcal{P}$ by placing Steiner points along the edges of the polyhedron. They construct a graph G containing the Steiner points as vertices and edges as those interconnections between Steiner points that correspond to segments which lie completely in the triangular faces of the polyhedron $\mathcal{P}$. Their algorithm introduces a logarithmic number of Steiner points along each edge of $\mathcal{P}$ and they are placed in a geometric progression along an edge. It takes $O(mn \log(mn) + nm^2)$ time to compute a $(1+\varepsilon)$-approximation geodesic path, where m is the total number of Steiner points.

The method proposed by Xin et al. (2012b) is based on a different strategy. It uniformly distributes m (specified by the user) samples $S = \{s_1, \cdots, s_m\}$ on M and computes a Delaunay triangulation so that each Delaunay edge is a geodesic path. The graph $G_g = (V_g, E_g)$ is then defined as follows: the vertex set consists of both mesh vertices and the samples



(a) (b)

Fig. 3. (a) On a smooth manifold, a geodesic is a curve whose tangent vectors remain parallel if they are transported along it. (b) On a triangle mesh, $\gamma(s,t)$ (in orange) is a direct geodesic path, i.e., it does not pass through any saddle vertices. Since $\gamma(s,t)$ is long, it can be well approximated by four short direct geodesic paths $\gamma(s,v_1)\bigcup\gamma(v_1,v_2)\bigcup\gamma(v_2,v_3)\bigcup\gamma(v_3,t)$. Each segment (v_i,v_{i+1}) approximates a part of $\gamma(s,t)$ very well so that the angle between their tangents is less than θ, where θ is a threshold specified by the user. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$V_g = V \bigcup S$. There are two tiers in the edge set $E_g = E_{g1} \bigcup E_{g2}$, where the first tier – the "backbone" of G_g – is a *complete* graph of sample points, $E_{g1} = \{e_{ij} | w_{ij} = d_g(s_i, s_j),\ \forall s_i, s_j \in S\}$, and the second tier consists of the short geodesic paths connecting the mesh vertices and the nearest samples. Consider a vertex $v \in V$ in a geodesic triangle, say $s_{i_1}s_{i_2}s_{i_3}$, the three geodesic paths $\gamma(v, s_{i_j})$ form the edges of E_{G2}. Constructing the graph G_G takes $O(mn^2 \log n)$ time and $O(m^2 + n)$ space. To compute the geodesic distance between arbitrary points p and q (not necessarily mesh vertices), it flattens the *curved* geodesic triangles containing p, q to Euclidean plane $\mathbb{R}^2$ by preserving the geodesic distances (which are stored as the edge weights in G_g). Then, the geodesic distance $d_g(p,q)$ is approximated by the Euclidean distance between their unfolded images. Since the unfolding is completely local, geodesic distance between any pair of points can be computed in $O(1)$. As a result, the single-source-all-destination (SSAD) geodesic problem is solved in $O(n)$ time. Xin et al.'s method, called Geodesic Triangle Unfolding (GTU), computes the geodesic distances with error bounded by the sizes of geodesic triangles, which in turn are determined by $O(\frac{1}{\sqrt{m}})$. Due to the quadratic space complexity, it is not practical to choose a large m in the GTU method.

Theorem 1 implies that a geodesic path passes through only Euclidean vertices and saddle vertices. A geodesic path is called *direct* if it does not pass through any saddle vertices. Ying et al. (2013) observed that the majority of geodesic paths on real-world meshes are *indirect*: the longer the geodesic path, the more likely it can be partitioned into smaller segments, where each segment is a direct geodesic path. Taking advantage of such a *local* characteristic of geodesic path, they proposed the saddle vertex graph to compute geodesic distances in a divide-and-conquer manner. A saddle vertex graph $G_s = (V, E_s)$ is undirected graph whose vertex set coincides with the mesh vertex set V and edge set E_s consists of three tiers: the first tier contains the direct geodesic paths between saddle vertices; the second tier contains the direct geodesic paths between non-saddle vertices and saddle vertices; and the third tier contains the direct geodesic paths between non-saddle vertices.

Unlike the GTU graph G_g, the saddle vertex graph G_s is a sparse graph. However, constructing an exact SVG (i.e., with *all* direct geodesic paths) is computationally expensive. Ying et al. (2013) proposed an algorithm for computing an approximate SVG, whose vertex degree is less than or equal to a user-specified constant K. For a vertex $v \in V$, they computed a geodesic disk with K vertices and then took only the direct geodesic paths in the disk as the SVG edges. Therefore, an approximate SVG with graph size $O(Kn)$ can be constructed in $O(nK^2 \log K)$ time. SVG allows users to control the accuracy by choosing a proper K. Ying et al. observed that a typical value $K \in [50, 500]$ produces a result with relative mean error 0.1% on common 3D models. In many application scenarios, however, it is highly desired to directly specify the maximal error rather than bounding the size of graph.

4. Discrete geodesic graphs

4.1. Motivation & definition

As mentioned in Section 1, this paper aims at overcoming the limitations of the SVG method. Observe that the backbone of an SVG is its tier-1 sub-graph consisting of saddle vertices and *short* direct geodesic paths among them. The condition of using saddle vertices, however, is too strict, since some models may have very few saddle vertices or unbalanced distribution of saddle vertices.

The discrete geodesic graph is motivated by parallel transport on smooth manifolds, where tangent vectors remain parallel when transporting along a geodesic. See Fig. 3(a). In the discrete setting, a *long* geodesic path γ can be well approximated by a set of *short* geodesic paths which are close enough to γ. The "closeness" can be quantitatively measured by the angles between them. See Fig. 3(b).

We define discrete geodesic graph as follows.

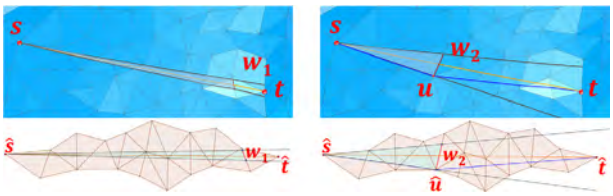
Definition. Given a manifold triangle mesh $M = (V, E, F)$ and the desired accuracy $\varepsilon > 0$, an undirected graph $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$ is called discrete geodesic graph, if

1. the vertex set V_ε includes all mesh vertices and possibly pseudo vertices, i.e., $V_\varepsilon \supseteq V$;
2. each edge in E_ε is a direct geodesic path; and
3. the shortest path distances on G_ε are $(1+\varepsilon)$-approximation of the geodesic distances on M.

Algorithm 1 Constructing Discrete Geodesic Graph on Manifold Triangle Meshes

Input: A manifold triangle mesh $M = (V, E, F)$ and the desired accuracy $\varepsilon > 0$
Output: The discrete geodesic graph $G_\varepsilon = (V_\varepsilon, E_\varepsilon)$

```
E_ε ← ∅
for all s ∈ V do
    E_ε ← E_ε ⋃ ComputeCandidateEdges(M, s, ε)
end for
for all s ∈ V do
    PruneRedundantEdges(G_ε, s, ε)
end for
for all s ∈ V do
    AddPseudoVerticesAndEdges(G_ε, M, s, ε)
end for
```



(a) Window propagation in the exact method (b) Window propagation in our method

Fig. 4. Let $\gamma(s,t)$ be a direct geodesic path. (a) To compute $\gamma(s,t)$, the exact methods (e.g., MMP, CH and ICH) propagate windows until a window, say w_1, finally reaching the destination t. (b) In contrast, our method allows early termination. A window w_2 has traveled for a sufficiently long distance so that the lit region becomes narrow enough. As a result, all geodesic paths in the lit region (including $\gamma(s,t)$) can be well approximated by some vertex $u \in \mathcal{F}(\gamma(s,t))$. A large ε is chosen for better illustration.

DGG distinguishes itself from SVG by approximating a long geodesic path using any kinds of relays, including auxiliary points added on the mesh edges or in the faces. As we will show later, this seemingly minor change produces a fundamentally different graph scheme with different properties than SVG. In a nutshell, relaxing saddle relays allows us to flexibly construct the graph, rigorously analyze the error bound and space complexity.

Intuitively speaking, we are looking for a *subset* of direct geodesic paths that can approximate *all* geodesic paths well. To ease analyzing space complexity and approximation error, we assume that the vertices are uniformly distributed on M, so that all vertices can be used as relays.

In Section 4.2, we construct DGG on meshes with fairly regular triangulations. Then we present practical techniques to deal with anisotropic meshes in Section 4.3.

4.2. DGG construction algorithm

To construct DGG, we first compute a set of direct geodesic paths, called candidate edges, which can guarantee the user-specified accuracy. Then we iteratively delete the redundant edges that can be well approximated by the other candidate edges. For anisotropic meshes, we also check the graph quality and add pseudo vertices and edges if necessary. We outline DDG construction in Algorithm 1 and detail the proposed techniques in pseudocodes.

4.2.1. Computing candidate edges

As we will show later, a *long* direct geodesic path is likely to be replaced by several short direct geodesic paths, hence will be removed from the graph. Our goal is to detect those geodesic paths at early stages to enable early termination of wavefront propagation. Observe that if a direct geodesic path $\gamma(s,t)$ is sufficiently long, the face sequence $\hat{\mathcal{F}}(\gamma(s,t))$ becomes very elongated. As a result, the 2D path $\hat{\gamma}(s,t)$ can be well approximated by a relay vertex $\hat{u} \in \hat{\mathcal{F}}(\gamma(s,t))$. See Fig. 4. We associate each window w with a threshold d_{max}, indicating the maximum distance it is allowed to travel. Initially, we set $d_{max} = \infty$. The d_{max} is gradually reduced during window propagation: when propagating a window w, we compute the d_{max} for the child windows of w according to the following theorem.

Theorem 2. *Let s be the source point and $w = (e_{ij}, b_0, b_1, d_0, d_1, 0)$ a direct window associated to an oriented half-edge $e_{ij} = (v_i, v_j)$. Denote by l the length of edge e_{ij}. For any geodesic path $\gamma(s,t)$ through window w, if $d_g(s,t) \geq \frac{l^2}{8\varepsilon(\min\{d_0,d_1\}-l/2)} + \frac{l}{2} + \max\{d_0, d_1\}$ ($\triangleq d_{max}$), then $\gamma(s,t)$ can be approximated by some vertex $u \in \mathcal{F}(\gamma(s,t))$ with the approximation ratio $1+\varepsilon$, i.e., $d_g(s,u) + d_g(u,t) \leq (1+\varepsilon) d_g(s,t)$.*

function ComputeCandidateEdges(M, s, ε)
  for all edge $e_{ij} = (v_i, v_j)$ opposite to s **do**
    Create a window $w = (e_{ij}, v_i, v_j, \|sv_i\|, \|sv_j\|, 0)$
    Place window w into a priority queue $\mathcal{Q}$ keyed by its distance to source s
  end for
  repeat
    Extract the top window w from $\mathcal{Q}$
    // $w = (e_{ij}, b_i, b_j, d_0, d_1, 0)$, $l = \|v_i v_j\|$
    $d_{max} \leftarrow \frac{l^2}{8\varepsilon(\min\{d_0, d_1\} - l/2)} + \frac{l}{2} + \max\{d_0, d_1\}$
    Propagate window w across its adjacent triangle
    Place its child windows into an array L
    if window w lights a vertex $v \in V$ **then**
      Update geodesic distance $d_g(s, v)$
      Create a candidate edge (s, v) if it does not exist yet
    end if
    for all child window $w' \in L$ **do**
      // $w' = (e', b'_0, b'_1, d'_0, d'_1, d')$
      if $d'_0 < d_{max}$ and $d'_1 < d_{max}$ and $d' = 0$ **then**
        // Only *direct* windows are added to $\mathcal{Q}$
        $\mathcal{Q}$.push(w')
      end if
    end for
  until $\mathcal{Q}$ is empty
return the computed DGG edges incident to s
end function

4.2.2. Pruning redundant edges

After obtaining the candidate edges, we need to identify redundant edges and remove them from the graph. We say a DGG edge $\gamma(p, q) \in E_\varepsilon$ is *redundant*, if it can be approximated by two other DGG edges $\gamma(p, r), \gamma(r, q) \in E_\varepsilon$ with an approximation ratio $1 + \varepsilon$.

Let $\mathcal{E}_\varepsilon(x, y)$ denote an ellipse with focal points x and y and eccentricity $\frac{1}{1+\varepsilon}$. The area of $\mathcal{E}_\varepsilon(x, y)$ is $\frac{1}{4}\sqrt{\varepsilon(2+\varepsilon)}(1 + \varepsilon)\pi\|xy\|^2$. For an arbitrary point z on the ellipse, the sum of its distances to the two focal points is $(1 + \varepsilon)\|xy\|$.

Now consider a geodesic path $\gamma(p, q)$. If the ellipse $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$ contains a vertex, then Lemma A2 (see Appendix) guarantees that $\gamma(p, q)$ can be approximated by some relay vertex with an approximation ratio $1 + \varepsilon$. Intuitively speaking, the longer the path $\gamma(p, q)$, the larger the area of the ellipse $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$, hence, more likely $\mathcal{E}_\varepsilon(\hat{p}, \hat{q})$ containing some mesh vertex, and the higher the probability that $\gamma(p, q)$ is redundant.

Theorem 3. *Assume that the vertices are uniformly distributed on the mesh M. For an accuracy parameter $\varepsilon > 0$, the following statements are true:*

1. *For a direct geodesic path γ with length l, the probability of γ being a DGG edge is $O(e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$, where A is the area of mesh M.*
2. *The graph G_ε has $O(\frac{n}{\sqrt{\varepsilon}})$ edges.*
3. *For any vertices $p, q \in V$, the shortest path distance $d_s(p, q)$ on G_ε approximates the geodesic distance $d_g(p, q)$ on M with an approximation ratio $1 + O(\varepsilon)$.*

4.3. Handling anisotropic meshes

To ease the complexity and error analysis in Theorem 3, we assume the vertices are uniformly distributed on the input meshes. However, the assumption does not hold for anisotropic meshes, on which vertices are distributed highly non-uniformly. In this subsection, we present a simple yet effective method to handle meshes with anisotropic triangulations. Observe that for meshes whose vertices are randomly and uniformly distributed, the DGG edges incident to a vertex are roughly evenly spaced (see Fig. 5(a)). For anisotropic meshes, however, the "gap" between two adjacent DGG edges may be big (see Fig. 5(c)). Our idea is to add pseudo edges to fill in those gaps so that the polar angle between any adjacent DGG edges is no more than $\sqrt{\varepsilon}$. Let v_i be a vertex and $\gamma(v_i, v_j)$ and $\gamma(v_i, v_k)$ the two adjacent DGG edges with included angle $\alpha > \sqrt{\varepsilon}$. We then add $\lfloor \frac{\alpha}{\sqrt{\varepsilon}} \rfloor$ pseudo edges to partition the included angle evenly. Each pseudo edge is a geodesic path emanating from v and with length l_ε. Denote the other endpoint of such a geodesic path by v', which is called pseudo vertex. Since pseudo vertex $v' \notin V$, we add it to M and re-triangle the face containing v'. Then for each pseudo vertex, we call ComputeCandidateEdge to construct the direct geodesic paths incident to it. We illustrate this idea in Fig. 5(c)–(e) and outline the pseudo code in AddPseudoVerticesAndEdges().

4.4. Property and time complexity

Theorem 4. *The graph G_ε is a connected metric graph.*

Proof. Since every mesh edge is a direct geodesic path, the computed candidate edges include all mesh edges. In function PruneRedundantEdges(), a candidate DGG edge $\gamma(p, q)$ (possibly a mesh edge) is redundant only if it can be approximated

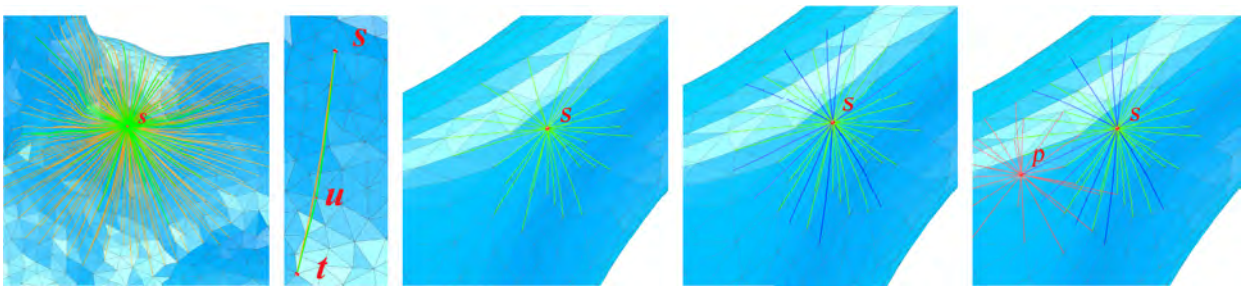


Fig. 5. Pruning redundant edges and adding pseudo edges. (a) shows the output of ComputeCandidateEdges(M, s, ε). Among them, the orange edges are redundant and the green edges are the final DGG edges. (b) shows a redundant edge $\gamma(s,t)$, which can be approximated by other DGG edges $\gamma(s,u)$ and $\gamma(u,t)$ (in green). (c)–(d) If the vertices are not distributed uniformly, we add pseudo edges of length l_ε (in blue) to the graph so that any two adjacent DGG edges have a polar angle no more than $\sqrt{\varepsilon}$. (e) Let $\gamma(s,p)$ be a pseudo edge whose endpoint p is not a vertex. Using p as the source, we compute the direct geodesic paths (in red) with length no more than l_ε and add them to the graph. A large $\varepsilon = 0.5\%$ is used for better visualization. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

```
1: function PruneRedundantEdges(G_ε, s, ε)
2:     V_s ← ∅
3:     for all edge (s, v) ∈ E_ε do
4:         V_s ← V_s ⋃ {v} // V_s contains s's neighboring vertices
5:     end for
6:     // Run a local Dijstra's algorithm at s
7:     d(s) ← 0
8:     Q.push(s) // Q is a priority queue keyed by d(v)
9:     while Q is not empty do
10:        u ← Q.pop()
11:        for all edge (u, v) ∈ E_ε and v ∈ V_s do
12:            //w(u, v) is the edge length between u and v
13:            if d(u) + w(u, v) ≤ (1 + ε) × w(s, v) then
14:                d(v) ← d(u) + w(u, v)
15:                Delete edge (u, v) from E_ε
16:                Q.push(v)
17:            end if
18:        end for
19:    end while
20: end function
```

by other edges. Hence, deleting $\gamma(p,q)$ does not break the connectedness. As a result, if M is connected, the graph G_ε is also connected.

Consider three arbitrary vertices $s,t,u \in V$ where $\gamma(s,t), \gamma(t,u), \gamma(s,u) \in E_\varepsilon$. If $d_s(s,u) + d_s(u,t) < d_s(s,t)$, then the edge (s,t) must be deleted during the edge pruning procedure, since $d_s(s,u) + d_s(u,t) < d_s(s,t) < (1+\varepsilon)d_s(s,t)$. Therefore, the graph (V, E_ε) is a metric graph. □

Now we analyze the *empirical* time complexity of Algorithm 1. The function ComputeCandidateEdges(M, s, ε) computes direct geodesic paths incident to vertex s. Thanks to Theorem 2, it guarantees the accuracy and it also terminates earlier than the MMP algorithm. Therefore, its time complexity is no worse than that of the MMP algorithm. As shown in Theorem 3, the average degree of G_ε is $O(\frac{1}{\sqrt{\varepsilon}})$, implying that the function sweeps $O(\frac{1}{\sqrt{\varepsilon}})$ vertices in $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ time. Therefore, it takes $O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon})$ time to compute all candidate edges.

The function PruneRedundantEdges$(G_\varepsilon, s, \varepsilon)$ runs a Dijkstra-like sweep at vertex s. For a graph with $|E|$ edges, the priority-queue based Dijkstra's algorithm runs in $O(|E|\log|E|)$ time. Since there are $O(\frac{1}{\sqrt{\varepsilon}})$ edges incident to s, pruning them takes $O(\frac{1}{\sqrt{\varepsilon}}\log\frac{1}{\varepsilon})$ time, implying that the time complexity of the entire edge pruning process is $O(\frac{n}{\sqrt{\varepsilon}}\log\frac{1}{\varepsilon})$.

For meshes with fairly regular triangulations, the vertices are distributed uniformly. As a result, no pseudo vertices and edges are added.

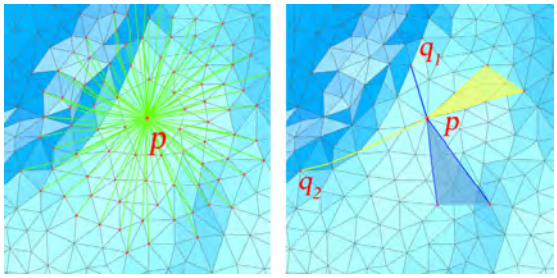
Putting it all together, Algorithm 1 runs in $O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon})$ time empirically.

5. Computing discrete geodesics using DGG

5.1. DGG-tailored label-correcting algorithm

Once the DGG G_ε is available, we compute geodesic distances by finding the shortest distances on G_ε. Dijkstra's algorithm is a popular method to solve the shortest path problem with nonnegative arc lengths. It assigns every vertex a

function AddPseudoVerticesAndEdges(G_ε, M, s, ε)
  Take an arbitrary DGG edge incident to s as the reference edge and sort the other incident edges according to their polar angles to the reference direction. Denote by γ_i the i-th DGG edge incident to s and θ_i the polar angle, $i = 1, \cdots, k$.
  for $i = 1$; $i \le k$; i ++ **do**
    if $\theta_{i+1} - \theta_i \ge \sqrt{2\varepsilon}$ **then**
      for $j = 1$; $j \le \lfloor \frac{\theta_{i+1} - \theta_i}{\sqrt{\varepsilon}} \rfloor$; j ++ **do**
        Construct a geodesic path $\gamma(s, p_j)$ with polar angle $\theta_i + j\sqrt{\varepsilon}$ and length l_ε
        Add p_j to M as a pseudo vertex and triangulate the face containing p_j
        $E_{p_j} \leftarrow$ ComputeCandidateEdges($M \bigcup \{p_j\}$, p_j, ε)
        Add the pseudo vertex p_j and the pseudo edges E_{p_j} to G_ε
      end for
    end if
  end for
end function



(a) (b)

Fig. 6. For each vertex, we take an arbitrary incident edge as the reference edge and sort the other incident edges according to their polar angles to the reference direction. Each DGG edge is then associated with a fan-shaped region, in which distance update is performed. (a) shows the DGG edges incident to the central vertex p. In (b), we pick two edges $\gamma(p, p_1)$ and $\gamma(p, p_2)$, and show their fan-shaped regions. We adopt a large ε for better illustration.

tentative distance value: set it zero to the source vertex and infinity for all other vertices. It also marks the source vertex as *current* and all other vertices *unvisited*. Then it performs two steps repeatedly: vertex selection and distance update. The vertex selection step selects a vertex, say v_i, with the smallest distance label for examination. The distance update step scans each unvisited neighbor of v_i and updates its distance label if necessary. Placing vertices in a priority queue keyed by the distances to the source, Dijkstra's algorithm extracts the vertex with the smallest label in an iteration. The overhead for entering and leaving a priority queue is $O(\log|V|)$, hence Dijkstra's algorithm takes $O(|E_\varepsilon| \log|V|) = O(\frac{n}{\sqrt{\varepsilon}} \log n)$ time on G_ε. Since each vertex enters the queue exactly once and its label remains unchanged after leaving the queue, Dijkstra's algorithm is usually referred to as a label *setting* method.

To improve the performance of Dijkstra's algorithm, we borrow a label correction scheme (Bertsekas, 1998), which removes the $O(\log n)$ overhead per iteration *empirically*. We also develop the *DGG-tailored* distance update to further reduce the time complexity to $O(n)$, making it insensitive to the user-specified accuracy parameter ε.

5.1.1. Label-correcting based vertex selection

In contrast to the label setting method, the label correcting methods maintain the vertices in a first-in-first-out queue Q and adopt a faster strategy for vertex selection, at the expense of multiple entrances of vertices in Q. Different label correcting methods are distinguished by the strategy for choosing the queue position to insert a vertex. A popular strategy is *Small Label to the Front* (SLF) (Bertsekas, 1998): whenever a vertex v_j enters Q, its label d_j is compared with the label d_i of the top vertex v_i of Q. If $d_j \le d_i$, vertex v_j is inserted at the top of Q; otherwise at the bottom of Q. Although the SLF scheme lacks a theoretical time complexity, extensive computational results show that it significantly outperforms the label-setting algorithm on real-world graphs (Bertsekas, 1998), since a vertex enters the queue Q only a few times.

5.1.2. DGG-tailored distance update

Recall that an approximate geodesic path $\gamma(s,t)$ is a union of DGG edges $v_0(=s)$, v_1, $\cdots$, $v_k(=t)$, so that the angle between consecutive edges $v_{i-1}v_i$ and v_iv_{i+1} is no less than $\pi - \sin\sqrt{\varepsilon}$. Based on the above observation, we design the DGG data structure as follows: For every vertex v_i, we store its incident edges $\{e_{ij}\}$ and a set $\{\mathfrak{f}_{ij}\}$ of fan-shaped regions. Each incident edge $e_{ij} \in E_\varepsilon$ (i.e., an approximate geodesic path ending at v_i) corresponds to a region $\mathfrak{f}_{ij}$, whose boundaries form an angle no less than $\pi - \arcsin\sqrt{\varepsilon}$ to e_{ij}. See Fig. 6.

In the conventional Dijkstra's algorithm, when a vertex, say v_j, is visited, the algorithm considers *all* of its unvisited neighbors and calculates their distances. The DGG-tailored algorithm improves the performance by considering only *part* of v_j's unvisited neighbors that provide the shortest distances. Assuming v_j is visited via a DGG edge $e_{ij} = (v_i, v_j)$, the algorithm searches only the neighboring vertices in the fan-shaped region $\mathfrak{f}_{ij}$. Intuitively speaking, the conventional Dijkstra's

Table 2
The average time that a vertex enters and leaves the FIFO-queue.

Model	$\|V\|$	$\varepsilon = 0.1\%$	$\varepsilon = 0.01\%$	$\varepsilon = 0.001\%$
Bunny	72K	1.31	1.57	1.73
Armadillo	173K	1.27	1.51	1.72
Lucy	263K	1.27	1.46	1.63
Gargoyle	350K	1.29	1.47	1.79
Buddha	500K	1.34	1.54	1.79
Thai Statute	625K	1.39	1.58	1.77
Ramesses	825K	1.4	1.54	1.69
Isidore Horse	1.1M	1.26	1.49	1.73
Blade	2M	1.23	1.41	1.71
Dragon	3M	1.28	1.47	out of memory

Algorithm 2 DGG-Tailored Label-Correcting Algorithm

Input: DGG $G_\varepsilon = (V, E_\varepsilon)$, source vertex $s \in V$
Output: $d_g(s, v)$ for all $v \in V$

for all $v \in V$ **do**
  $d_g(s, v) \leftarrow \infty$
  Mark v *unvisited*
end for
$d_g(s, s) = 0$
Set s *current* and place it in a first-in-first-out queue $\mathcal{Q}$
$\bar{d} \leftarrow \frac{\Sigma_{v \in \mathcal{Q}} d_g(s,v)}{|\mathcal{Q}|}$
while $\mathcal{Q}$ is not empty **do**
  $u \leftarrow \mathcal{Q}$.top()
  while $d_g(s, u) > \bar{d}$ **do**
    Move u to the bottom of $\mathcal{Q}$
    $u \leftarrow \mathcal{Q}$.top()
  end while
  $u \leftarrow \mathcal{Q}$.pop()
  Mark u *visited*
  for all $v \in$ [v.fan.starting_edge, v.fan.ending_edge] **do**
    //$w(u, v)$ is the edge length between u and v
    if v is *unvisited* and $d_g(s, v) + w(u, v) < d_g(s, v)$ **then**
      $d_g(s, v) \leftarrow d_g(s, u) + w(u, v)$
      $b(v) \leftarrow u$
      if $v \in \mathcal{Q}$ **then**
        Update $\Sigma_{v \in \mathcal{Q}} d_g(s, v)$
      else
        Insert v to the bottom of $\mathcal{Q}$
      end if
      Update $\bar{d}$
    end if
  end for
end while

algorithm explores nodes in all directions uniformly, whereas the DGG-tailored label-correcting algorithm considers only the directions that lead to the shortest distances, i.e., the smaller the expected error ε, the narrower the searching range.

With the SLF strategy, it takes $O(1)$ time to enter a vertex to the queue $\mathcal{Q}$ and select a vertex from $\mathcal{Q}$. Computational results show that on average, a vertex enters and leaves $\mathcal{Q}$ no more than 2 times. See Table 2. Assuming that the vertices are uniformly and randomly distributed, the number of DGG edges falling in the fan-shaped region is proportional to the included angle $2\sqrt{\varepsilon}$. Therefore, distance update takes $O(\sqrt{\varepsilon})$ time. Putting it all together, for meshes with fairly regular triangulations, the DGG-tailored label correcting algorithm (Algorithm 2) runs in $O(|E_\varepsilon|\sqrt{\varepsilon}) = O(n)$ empirically.

6. Results & discussions

Since both the DGG and SVG methods are graph-theoretic, they share some common features:

1. Both methods solve the discrete geodesic problem by taking advantage of its local feature, therefore, they allow information reuse and can compute geodesic distances efficiently.
2. Both graphs are metric graphs, thus, the computed geodesic distances are guaranteed to be true metrics.
3. Both methods can compute either the exact or approximate geodesic distances, depending on the user-specified parameter. For the SVG method, if the user specifies a sufficiently large K (which bounds the maximal vertex degree in the graph), the computed distances are exact. For the DGG method, the result is exact if the desired accuracy ε is sufficiently small.

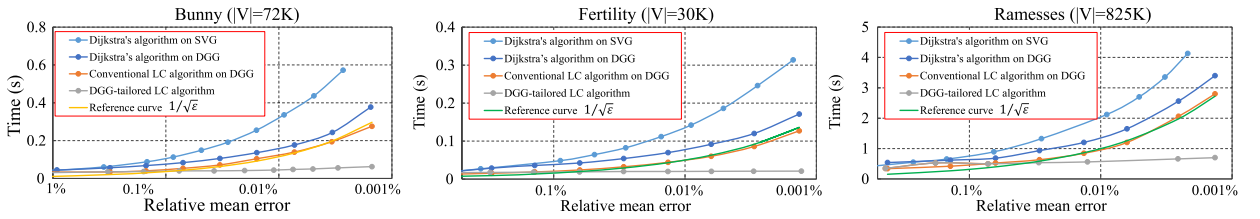


Fig. 7. For small- and middle-scale meshes, the DGG-tailored label-correcting algorithm runs in $O(n)$ time empirically, which is insensitive to the user-specified accuracy ε. In contrast, the runtime performance of SVG heavily depends on its parameter K, which bounds the maximal degree. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

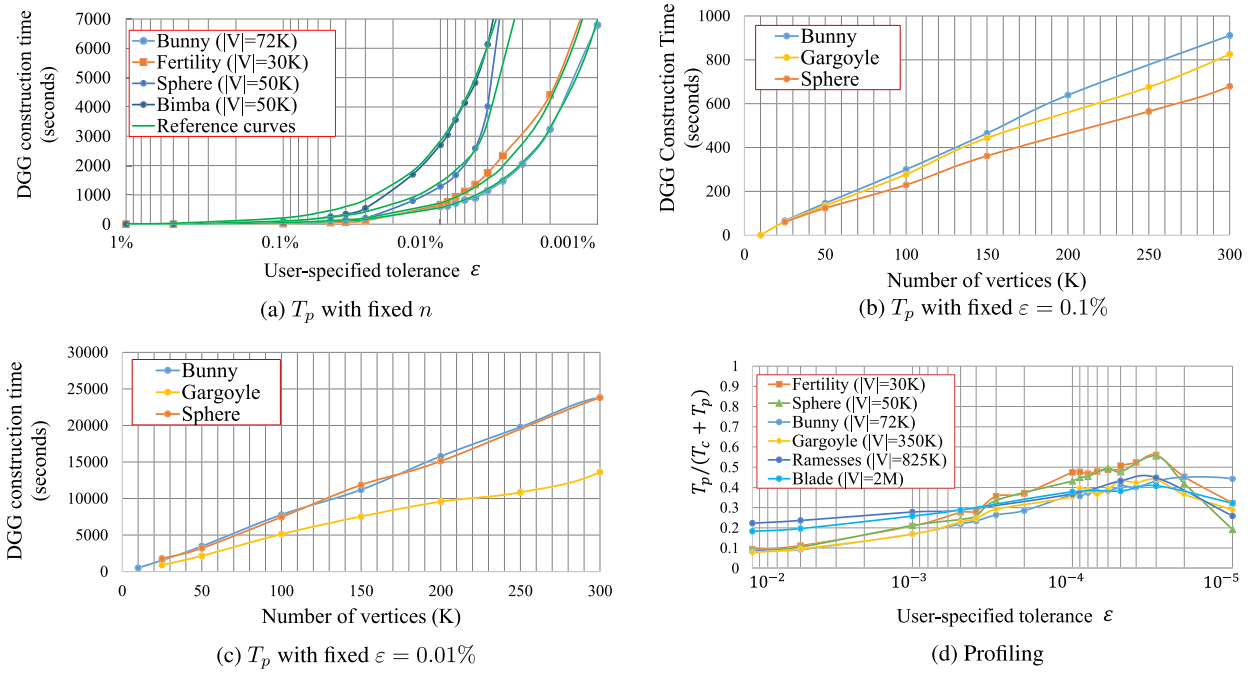


Fig. 8. The DGG construction algorithm runs in $O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon})$ empirically. (a) shows the construction time T_p with fixed n and varying ε. (b) and (c) show T_p with fixed ε but varying n. (d) Profiling of the DGG construction algorithm. The vertical axis is the ratio $\frac{T_p}{T_p+T_c}$, where T_p is running time of edge pruning and T_c is the time of computing candidate edges. Timing was obtained on an Intel Xeon 2.66GHz CPU using a single core. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

However, the two methods differ fundamentally in graph construction, space complexity, error control, and runtime performance, which are thoroughly evaluated in Section 6.1 to 6.4. We also compare DGG with other discrete geodesic algorithms in Section 6.5.

6.1. Graph construction

As analyzed in Section 4.4, the DGG construction algorithm runs in $O(\frac{n}{\varepsilon}\log\frac{1}{\varepsilon})$ time empirically, which is confirmed by experimental results (see Fig. 7). The algorithm consists of three steps, i.e., computing candidate edges, pruning the redundant edges which can be well approximated by others, and adding pseudo edges if necessary. As Fig. 8(d) shows, computing candidate edges is dominant. When $\varepsilon = 0.01\%$, edge pruning takes roughly half of the time as of computing candidate edges. For even higher accuracy $\varepsilon < 0.001\%$, most of the candidate edges become DGG edges, hence the ratio $\frac{T_p}{T_p+T_c}$ drops.

In contrast, SVGs can be constructed in a straightforward manner. For each vertex, one simply computes the direct geodesic paths within a geodesic disk of K vertices, which takes $O(K^2 \log K)$ time. Table 5 shows the runtime performance of the two construction algorithms. We observe that for the accuracy parameter $\varepsilon > 0.5\%$, the computational cost of constructing DGG is roughly the same as that of SVG, mainly because for relatively large parameter ε, the average degree is small, so is the number of redundant edges. So it takes little time to identify redundant edges. However, for very small ε (e.g., $< 0.01\%$), constructing DGG is more expensive than that of SVG, since DGG computes more candidate edges than SVG in order to ensure the user-specified accuracy. As Table 3 shows, more than 80% of the candidate edges are redundant. Identifying those edges also takes time.

Table 3
Statistics of edge pruning.

Model ($\|V\|$)	$\varepsilon = 0.1\%$				$\varepsilon = 0.01\%$			
	Before pruning		After pruning		Before pruning		After pruning	
	$\bar{\varepsilon}$	$\bar{D}$	$\bar{\varepsilon}$	$\bar{D}$	$\bar{\varepsilon}$	$\bar{D}$	$\bar{\varepsilon}$	$\bar{D}$
Sphere (20K)	0.0067%	300.7	0.027%	83.2	0.00016%	2305.9	0.0028%	269.7
Fertility (30K)	0.0074%	298.1	0.029%	83.3	0.00013%	2066.6	0.0030%	266.7
Bunny (72K)	0.020%	229.8	0.036%	80.3	0.0045 %	1113.8	0.0040%	259.6
Armadillo (173K)	0.024%	180.8	0.042%	81.2	0.00096%	948.3	0.0033%	266.1
Lucy (263K)	0.054%	129.6	0.063%	67.9	0.0047 %	586.9	0.0054%	237.9
Buddha (500K)	0.021%	224.1	0.035%	79.8	0.0012 %	1662.6	0.0047%	261.4
Isidore Horse (1104K)	0.019%	234.3	0.031%	75.8	0.0011%	1564.2	0.0032%	261.2

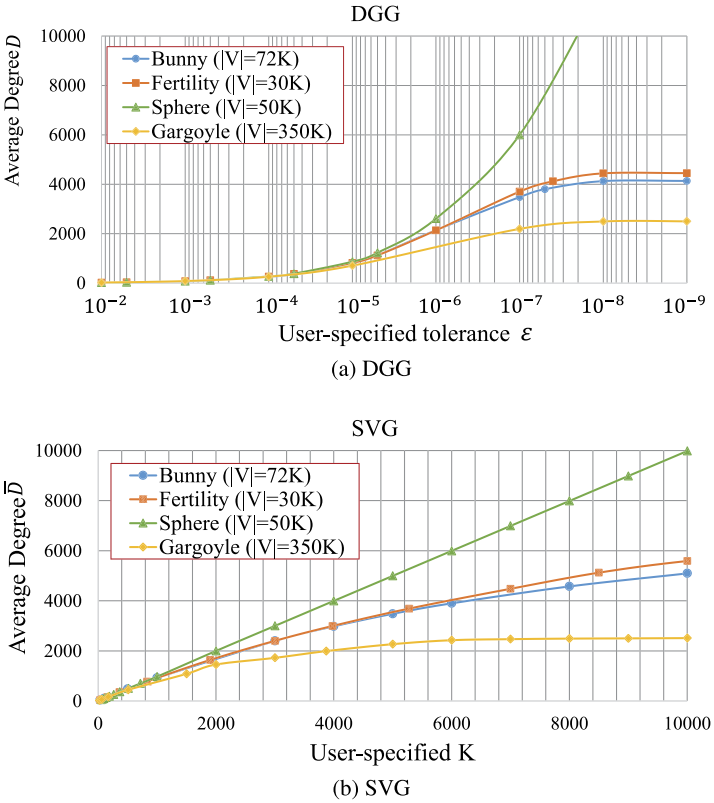


Fig. 9. Both DGG and SVG have a degree limit for non-convex, real-world models. As a result, the approximate graphs become exact when ε is sufficiently small or K is sufficiently large. For the sphere model, the exact DGG and SVG are complete graphs.

6.2. Space complexity

SVG partitions long geodesic paths using saddle vertices, thus, the size of an SVG is highly sensitive to the number of saddle vertices and their distributions. We observe that when the number of saddle vertices increases, the direct geodesic paths tend to be shorter and the vertex degree drops, hereby the runtime performance improves. Conversely, if a mesh has few saddle vertices, the vertex degree of SVG could be very high. In contrast, DGG approximates long geodesic paths using any kind of relay vertices, hence its size is less sensitive to the number of saddle vertices than SVG. See Fig. 10(b).

As shown in Section 4.4, the average degree of a DGG is $O(\frac{1}{\sqrt{\varepsilon}})$. It is worth noting that the upper bound is *not* tight. Given a sufficiently small ε, ComputeCandidateEdges() returns *all* direct geodesic paths and none of them can be removed in the edge pruning stage. As a result, the resulting graph is exact. We also observe that for non-convex, real-world meshes, the average degree is bounded by a model-dependent constant. See Fig. 9.

Fig. 11 compares SVG and DGG in terms of vertex degrees. The standard deviation of DGG's degree is 60% smaller than that of SVG, meaning that DGG is more balanced than SVG. Fig. 10(a) compares SVG and DGG in terms of graph size and quality. Take the 350K-vertex Gargoyle model as an example. Given a DGG and an SVG with similar sizes, the geodesic distances computed by DGG is nearly twice as accurate as those by SVG. Conversely, given the graphs producing geodesic

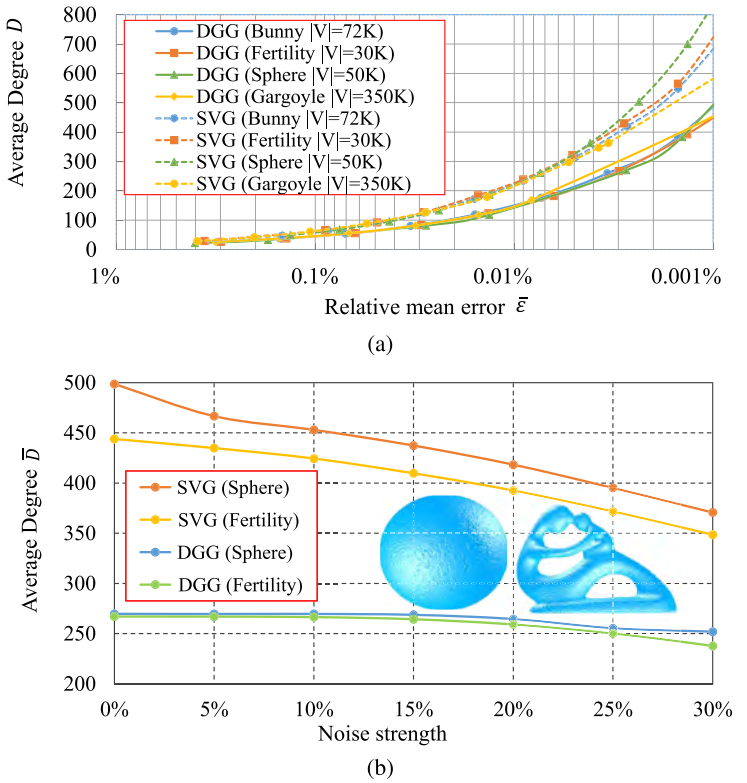


Fig. 10. (a) Comparing SVG and DGG in terms of accuracy and space complexity. For an SVG and a DGG with accuracy $\bar{\varepsilon} \leq 0.01\%$, DGG is no more than $\frac{2}{3}$ the size of the SVG. Conversely, given SVG and DGG with similar sizes, the geodesic distances computed by DGG are 2–5 times more accurate than those of SVG. (b) Adding noise to the model increases the number of saddle vertices, hereby reducing the size of SVG. The size of DGG, however, is less sensitive to saddle vertices. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

distances of similar accuracy, the DGG is only $\frac{2}{3}$ the size of the SVG. Table 3 reports the statistics of edge pruning. We observe that a large amount of candidate edges are redundant, which also justifies the compactness of DGG.

6.3. Runtime performance

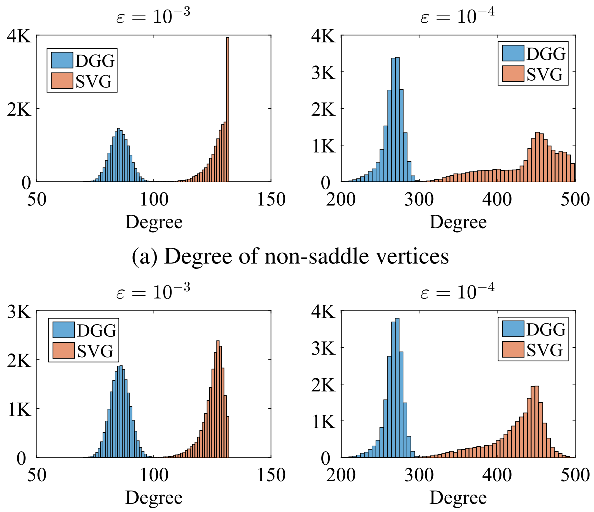
Table 5 shows the statistics of mesh complexity and runtime performance. SVG adopts Dijkstra's algorithm to compute geodesic distances in $O(Kn \log n)$ time, where K is the maximal degree. For small- and middle-scale meshes, we observe that the DGG-tailored label-correcting algorithm empirically runs in $O(n)$ time, which is insensitive to the user-specified accuracy parameter ε (see Fig. 7). For large-scale meshes, the runtime performance of DGG slightly depends on ε. Take the 1M-vertex Blade model (Fig. 1) as an example. It takes the DGG with $\varepsilon = 0.36\%$ 0.6 s to compute a single-source geodesic distance, while the DGG with $\varepsilon = 0.0062\%$, which is two orders of magnitude more accurate, takes only 1.8 s.

To make a fair comparison between SVG and DGG, we construct the graphs with similar accuracy. We first constructed the DGG with the user-specified error. Then we tune the parameter K for SVG so that it has the same quality. For common models with up to 3 million vertices, we observe that DGG runs 2 to 5 times faster than SVG. The improvement can be even higher for larger models, since DGG is more compact than SVG. See Fig. 13.

6.4. Approximation error

With SVG, users control the accuracy via specifying the maximal degree of the graph, which is non-intuitive. As mentioned above, direct error control is one of the unique features of DGG. Given a desired tolerance $\varepsilon > 0$, Theorem 3 guarantees that the DGG G_ε approximates geodesic distances with a relative mean error $O(\varepsilon)$. Through extensive evaluations on both isotropic and anisotropic meshes, we observe that the actual error $\bar{\varepsilon}$ never exceeds the user-specified tolerance ε. See Fig. 14.

Fig. 12 shows the results on anisotropic meshes. The Ellipsoid, Cyclide and Fertility models have very high degree of anisotropy with $\tau > 5$, implying that the DGG edges are distributed highly non-uniformly. After adding pseudo vertices and edges, we can ensure that for an arbitrary vertex $v \in V$, its incident edges can cover *all* radial directions, i.e., any two adjacent edges have an included angle no more than $\arcsin\sqrt{\varepsilon}$. As shown in the table, the accuracy of the graph is improved significantly, demonstrating the efficacy of procedure AddPseudoVerticesAndEdges on highly anisotropic meshes.



(a) Degree of non-saddle vertices

(b) Degree of saddle vertices

		DGG		SVG	
		$\varepsilon = 0.1\%$ $\bar{\varepsilon} = 0.027\%$	$\varepsilon = 0.01\%$ $\bar{\varepsilon} = 0.0028\%$	$K = 135$ $\bar{\varepsilon} = 0.027\%$	$K = 500$ $\bar{\varepsilon} = 0.0028\%$
$\bar{D}$	saddle	85.5	266.7	125.1	425.8
	non-saddle	85.3	265.8	127.4	438.3
	all	85.4	266.3	126.2	431.8
σ	saddle	4.29	14.17	4.37	34.93
	non-saddle	4.37	14.73	4.38	43.37
	all	4.33	14.46	4.52	39.73

Fig. 11. Statistics of vertex degrees on the Bimba model with 74.5K vertices. DGG is more balanced than SVG, due to the small variance of vertex degree. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Model	$\|V\|$	τ	DGG ($\varepsilon = 10^{-2}$)					DGG ($\varepsilon = 10^{-3}$)					DGG ($\varepsilon = 10^{-4}$)					The Heat Method				
			w/o		w/			w/o		w/			w/o		w/			original mesh		Delaunay mesh		
			$\bar{\varepsilon}$	Tg (s)	$\bar{\varepsilon}$	Tg (s)	$\frac{\|Vp\|}{\|V\|}$	$\bar{\varepsilon}$	Tg (s)	$\bar{\varepsilon}$	Tg (s)	$\frac{\|Vp\|}{\|V\|}$	$\bar{\varepsilon}$	Tg (s)	$\bar{\varepsilon}$	Tg (s)	$\frac{\|Vp\|}{\|V\|}$	$\bar{\varepsilon}$	Tg (s)	$\bar{\varepsilon}$	Tg (s)	$\frac{\|Vp\|}{\|V\|}$
Ellipsoid	500	5.25	8.59%	0.00013	0.095%	0.0039	8.41	0.19%	0.00016	0.012%	0.0059	13.36	0.0046%	0.00032	0.00058%	0.011	23.25	33.2%	0.0007	1.97%	0.0092	9.78
Cyclide	4K	5.28	2.47%	0.0031	0.18%	0.026	6.41	0.13%	0.0039	0.029%	0.038	9.99	0.011%	0.0044	0.0056%	0.078	11.88	29.6%	0.0094	3.42%	0.10	10.14
Fertility	12K	5.54	7.3%	0.0053	0.62%	0.032	9.45	0.60%	0.0042	0.071%	0.037	14.21	0.067%	0.0063	0.0074%	0.083	21.53	12.1%	0.042	6.6%	0.079	6.48
Eight	8K	1.38	0.70%	0.0033	0.31%	0.0056	0.62	0.044%	0.0036	0.028%	0.0067	0.29	0.0031%	0.0056	0.00072%	0.0060	0.012	2.04%	0.0091	1.04%	0.012	0.48

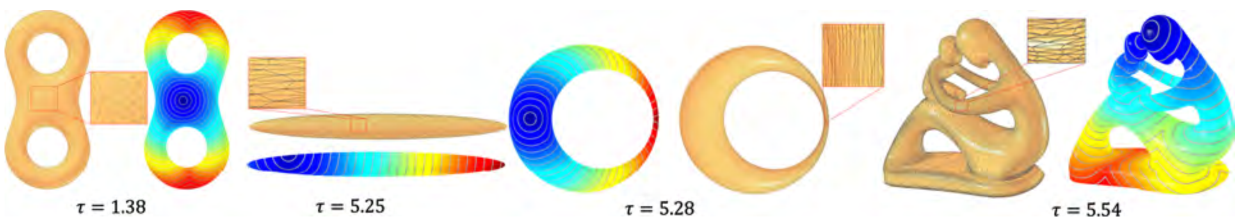


Fig. 12. For anisotropic meshes, adding pseudo vertices and edges to the graph can significantly improve the accuracy. τ measures the anisotropy of the triangulation. The larger the value τ, the high the degree of anisotropy the mesh has. $\frac{|V_p|}{|V|}$ is pseudo vertex ratio. "w/" and "w/o" mean with and without pseudo vertices and edges. T_g is the time of computing geodesic distances.

We also compare with the HM-DM method, which adopts Delaunay meshes (Liu et al., 2015) as preconditioner to improve robustness and accuracy. Delaunay meshes also add pseudo vertices to ensure that the local Delaunay condition holds everywhere. As Fig. 12 shows, the number of pseudo vertices added by DM is comparable to ours. However, our method consistently outperforms both HM and HM-DM in terms of accuracy and performance.

It is also worth noting that for the Eight model with anisotropy measure $\tau = 1.38$, pseudo vertices are not necessary. Even without them, the graphs can produce geodesic distances satisfying the user-specified tolerances.

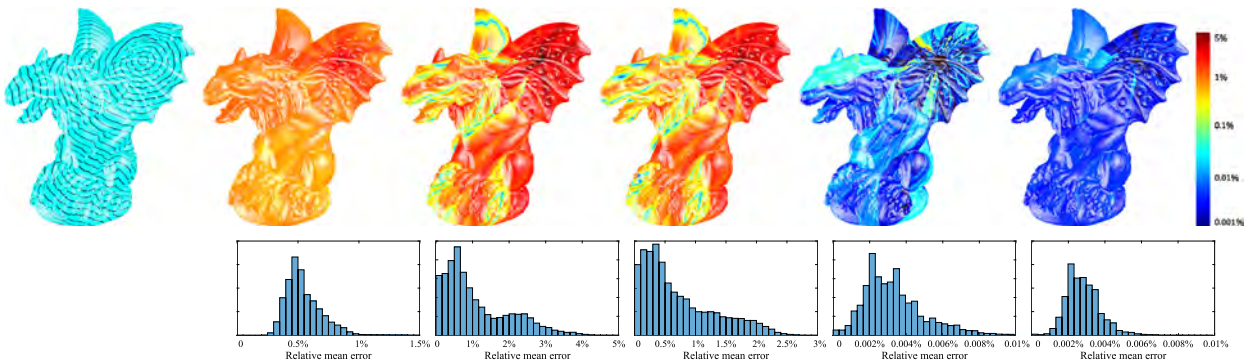


Fig. 13. Runtime performance and error distribution. (a) shows the level sets of exact geodesic distances and (b)–(f) visualize the error of various methods using color maps. Delaunay meshes (DM) can improve the accuracy of the heat method. In the error histograms, the scales of the horizontal axes are different. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

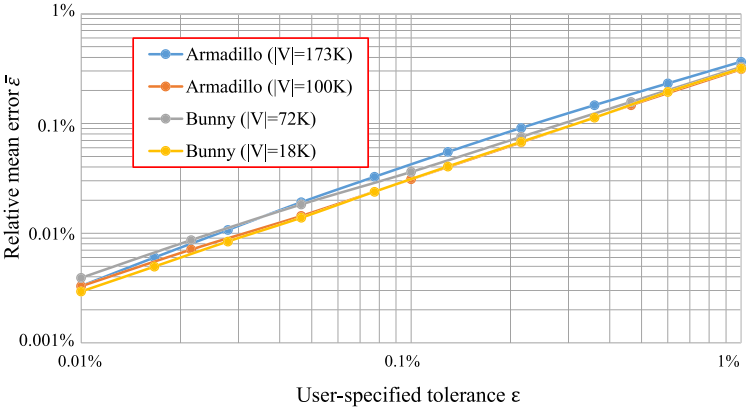


Fig. 14. Relation between the user-specified tolerance ε and the actual error $\bar{\varepsilon}$. We observe that the ratio $\bar{\varepsilon}/\varepsilon$ is in the range $[0.3, 0.6]$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
Comparison of representative discrete geodesic algorithms on polyhedral surfaces with n vertices. An algorithm that finds approximations to a geodesic path with approximation ratio λ returns a path of length at most λ times the exact geodesic path. Thus, exact algorithms have approximation ratio 1. m is the number of sample points in the GTU method and K is the maximal degree of saddle vertex graphs.

Method	Approximation Ratio	Pre-Computing	Computing Single-source Distances	Space Complexity
MMP (Mitchell et al., 1987)	1	–	$O(n^2 \log n)$	$O(n^2)$
FWP-MMP (Chen and Han, 1990)	1	–	$O(n^2)$	$O(n^2)$
CH (Chen and Han, 1990)	1	–	$O(n^2)$	$O(n)$
ICH (Xin and Wang, 2009)	1	–	$O(n^2 \log n)$	$O(n)$
FMM (Kimmel and Sethian, 1998)	N.A.	–	$O(n \log n)$	$O(n)$
HM (Crane et al., 2013)	N.A.	factoring Laplacian matrix	empirical $O(n)$	empirical $O(n)$
GTU (Xin et al., 2012b)	N.A.	$O(mn^2 \log n)$	$O(n)$	$O(m^2 + n)$
AMMP (Surazhsky et al., 2005)	N.A.	–	empirical $O(n \log n)$	empirical $O(n)$
SVG (Surazhsky et al., 2005)	N.A.	$O(nK^2 \log K)$	$O(Kn \log n)$	$O(Kn)$
DGG	theoretical $1 + O(\varepsilon)$ empirical $1 + \varepsilon$	empirical $O(\frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	empirical $O(n)$	$O(\frac{n}{\sqrt{\varepsilon}})$

6.5. Comparison with other methods

Table 4 summarizes the features of the existing discrete geodesic algorithms. Although DGG aims at computing approximate geodesic distances with bounded error, it is closely related to *exact* algorithms, by which we can compute the direct geodesic paths. Therefore, development of exact algorithms can improve the performance of DGG construction.

Table 5
Mesh complexity and runtime performance. To make a fair comparison, we specify the SVG parameter K carefully so that its computed geodesic distances have similar accuracy as those of the DGG. The timings are broken down into the pre-computing time T_p and the time T_g for computing geodesic distances, which were measured in seconds on an Intel Xeon 2.66 GHz CPU with 32GB memory. τ measures the degree of anisotropy of the input triangulation. To make fair comparison, all programs were tested on a single CPU core. $\bar{\varepsilon}$ is the relative mean error of the computed geodesic distances and $\bar{D}$ is the average degree of the graph.

Model	$\|V\|$	τ	FWP-MMP	HM-DM			DGG ($\varepsilon = 10^{-2}$)				SVG				DGG ($\varepsilon = 10^{-3}$)				SVG			
			T	T_p	T_g	$\bar{\varepsilon}$	$\bar{D}$	T_p	T_g	$\bar{\varepsilon}$	K	$\bar{D}$	T_p	T_g	$\bar{D}$	T_p	T_g	$\bar{\varepsilon}$	K	$\bar{D}$	T_p	T_g
Fertility	30K	1.27	1.1	1.09	0.034	0.90%	21.8	5.6	0.015	0.43%	34	32.7	6.3	0.021	81.9	70.9	0.022	0.030%	133	127.5	34.1	0.07
Bunny	72K	1.26	2.58	4.78	0.10	0.57%	25.3	20.4	0.034	0.32%	32	30.6	15.6	0.046	76.3	111.8	0.045	0.048%	117	111.2	68.4	0.14
Armadillo	173K	1.35	5.16	10.53	0.26	0.49%	26.0	24.5	0.091	0.36%	34	31.2	43.1	0.13	75.1	199.7	0.14	0.049%	123	112.0	190.4	0.44
Lucy	263K	1.60	7.9	14.31	0.37	1.31%	22.2	45.5	0.12	0.48%	31	29.4	52.4	0.15	67.9	176.9	0.18	0.063%	102	93.8	211.4	0.48
Gargoyle	350K	1.42	12.3	33.54	0.53	0.81%	22.5	65.6	0.17	0.43%	35	32.8	82.4	0.23	83.0	864.5	0.28	0.031%	132	119.7	396.5	0.81
Buddha	500K	1.31	29.4	58.3	0.98	0.89%	21.2	130.5	0.23	0.45%	32	30.0	120.7	0.35	79.8	992.5	0.44	0.035%	130	110.5	516.2	0.83
Thai Statute	625K	1.41	30.1	109.0	1.26	1.03%	23.0	137.6	0.53	0.38%	30	28.7	116.8	0.73	83.5	1661.8	0.66	0.028%	132	123.2	699.0	1.03
Ramesses	825K	1.16	23.5	325.8	1.85	0.69%	20.9	158.0	0.54	0.44%	29	27.4	152.8	0.89	82.6	2254.2	1.02	0.026%	123	121.3	1063.1	1.67
Blade	1M	1.32	65.2	265.3	1.91	2.1%	19.5	204.6	0.66	0.36%	35	33.9	232.1	1.2	75.1	2156.3	1.2	0.049%	100	98.3	801.2	2.0
Isidore Horse	1.1M	1.19	96.3	272.6	1.90	1.14%	20.5	176.3	0.59	0.39%	31	29.5	223.5	1.56	75.8	3648.5	0.95	0.031%	131	128.3	1957.2	1.93
Blade	2M	1.41	168.2	900.5	3.21	1.98%	19.5	347.2	1.15	0.59%	27	26.3	316.3	1.78	72.7	10940.2	1.84	0.057%	84	81.8	3214.2	2.93
Dragon	3M	1.38	200.3	out of memory			16.5	498.5	1.11	0.74%	31	29.4	498.3	2.76	61.8	2466.9	2.39	0.071%	87	85.3	6754.6	5.78

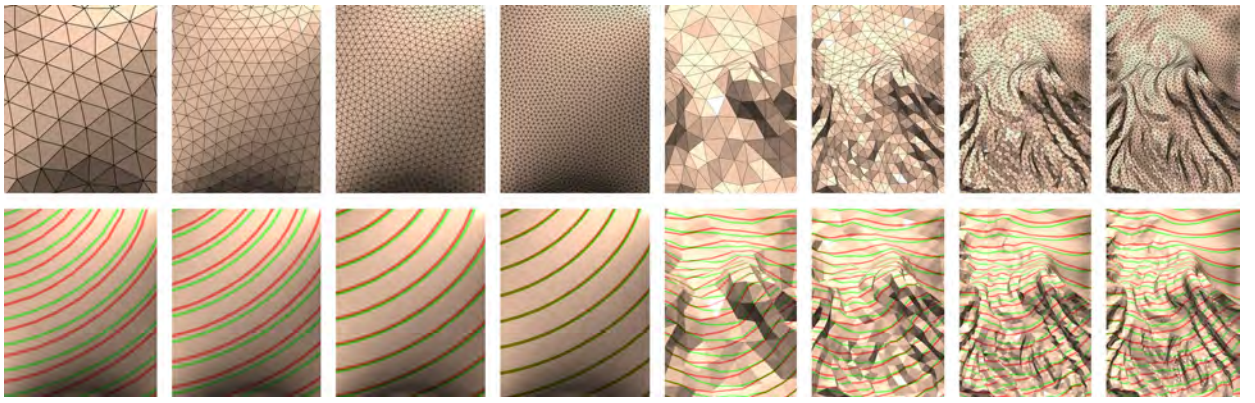


Fig. 15. The PDE based methods work well for smooth models with highly regular triangulations. However, they are sensitive to geometric details.

Surazhsky et al. (2005) extended the MMP algorithm with a window merging operation: before propagating a window, it tries to merge it with adjacent windows on the same edge. They proved that the merging operation has a *local* error bound when some conditions are met. Since the approximate MMP (AMMP) algorithm runs in $O(n \log n)$ time and it does not require precomputing, it is a good choice when only a small number of distance queries are required. However, their algorithm does not allow users to directly specify the *global* accuracy. Moreover, its runtime performance is not comparable to the pre-computation methods, such as the heat method, SVG and DGG, which do take advantage of information reuse.

The fast marching method and the heat method are popular approximate methods that work for a wide range of geometric domains, such as triangle meshes, point clouds, implicit functions and tetrahedral meshes. Both methods are conceptually simple and easy to implement, and they are much more efficient than the exact methods. However, FMM and HM do not allow error control and the results are sensitive to mesh triangulation and resolution. The computed distances are not metrics, due to violation of triangle inequality and the symmetric condition.

We also observe that FMM and HM work fairly well on *smooth* models. However, they produce poor results on models with rich geometric details, even the models have highly regular triangulations. See Fig. 15. In contrast, SVG and DGG do take advantage of local geometric features. The richer the geometric details, the more number of saddle vertices, the shorter the direct geodesic paths, hereby the stronger the local structure of the discrete geodesic problem and the better performance and accuracy SVG and DGG can provide. Such a feature is unique to the graph based methods.

It is worth noting that the PDE based methods such as HM and FM, can work on a wide range of discrete domains as well as anisotropic metrics (Bommes and Kobbelt, 2007). Those features are not available to the graph-based methods.

7. Conclusion & future work

This paper presents a discrete geodesic graph for computing approximate geodesic distances and paths. Comparing with the existing methods, DGG has several unique advantages, such as theoretical soundness, direct error control, compact size, linear-time complexity (independent of the accuracy) of computing the single-source or multiple-sources geodesic distances, the ability of computing geodesic distances and paths between arbitrary points, and extension to other discrete domains, such as point clouds.

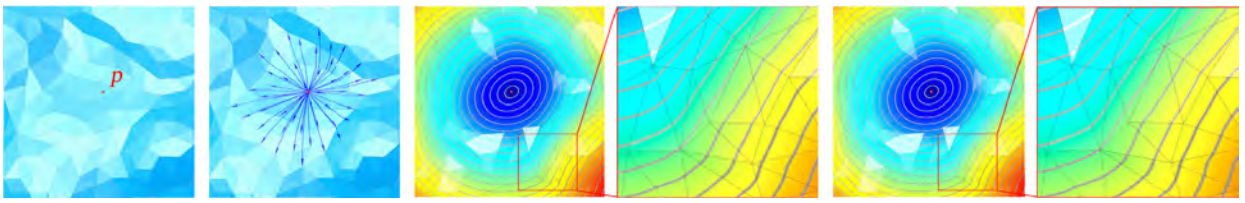
Assume that the vertices are uniformly distributed, we show that the shortest path distances on DGG approximate geodesic distances with a *theoretical* approximation ratio $1 + O(\varepsilon)$. Extensive results show that the actual approximation ratio does not exceed $1+\varepsilon$. Thus, it is of particular (theoretical) interest to refine the bound. On the practical side, there is a large room for improving the runtime performance of the DGG construction algorithm. As Table 3 shows, for $\varepsilon = 0.1\%$, more than 80% of candidate edges are redundant, implying that our current strategy for accuracy control in window propagation is very pessimistic. Therefore, it is highly desired to develop a more effective method that could identify redundant edges at early stage, hereby reducing the construction time.

Acknowledgements

We'd like to thank the anonymous reviewers for their detailed and constructive comments. This project is partially supported by MOE2013-T2-2-011 and RG23/15.

Appendix A. Computing geodesic between arbitrary points

The discrete geodesic graph introduced in the main text is built upon mesh vertices, thus it can compute geodesic distances between any pair of vertices. In some applications (e.g., sampling), it is highly desired to compute geodesic distances between non-vertex points. Let $p_1, p_2 \in M \setminus V$ be two non-vertex points that are contained in faces $f_1, f_2 \in F$. Since the



(a) Source p (b) Augmented edges (c) Our result (d) Exact result

Fig. A.16. Computing geodesic distance between arbitrary points. (a) Let $p \in M \setminus V$ be a non-vertex point and $f \in F$ the face containing p. (b) We split f at p, take p as a temporary vertex, and compute the direct geodesic paths from p. (c) The locally augmented graph allows us to accurately compute the geodesic distances with source p (see (d) for the exact distances).

Algorithm 3 Computing Geodesic Distance between Arbitrary Points

Input: A triangle mesh $M = (V, E, F)$ and its DGG $G_\varepsilon = (V, E_\varepsilon)$, the source and destination points $p_1, p_2 \in M$, the desired accuracy ε
Output: The geodesic distance $d_g(p_1, p_2)$

$\widetilde{E}_\varepsilon \leftarrow E_\varepsilon$
for i=1 to 2 **do**
  if $p_i \notin V$ **then**
    $M' \leftarrow M \cup \{p_i\}$ // Add p_i as a temporary vertex
    $\widetilde{E}_\varepsilon \leftarrow \widetilde{E}_\varepsilon \bigcup$ ComputeCandidateEdges(M', p_i, ε)
  end if
end for
$\widetilde{G} \leftarrow (V \bigcup \{p_1, p_2\}, \widetilde{E}_\varepsilon)$
$d_g(p_1, p_2) \leftarrow$ BidirectionalDijkstra($\widetilde{G}$, p_1, p_2)

geodesic distances between the vertices of f_1 and f_2 are available, one may simply adopt bilinear interpolation to approximate $d_g(p_1, p_2)$. Unfortunately, since geodesic distances are highly non-linear, bilinear interpolation often produces poor results when mesh resolution is low and/or mesh triangulation is poor. The geodesic triangle unfolding (GTU) method (Xin et al., 2012b) produces better results than bilinear interpolation, however, it computes only a first-order approximation of $d_g(p_1, p_2)$.

Here we present a simple-yet-effective technique to compute highly accurate geodesic distances between non-vertex points. We first add each point p_i to the mesh as a temporary vertex and then construct the direct geodesic paths incident to p_i. Adding those edges to the existing DGG, we obtain an augmented graph that computes $d_g(p_1, p_2)$ with the approximation ratio $1 + O(\varepsilon)$, since the direct geodesic paths from p_1 and p_2 are *exact*. See Fig. A.16 for an illustration and Algorithm 3 for the pseudo-code.

To improve the performance, we adopt the bidirectional Dijkstra's algorithm, which two simultaneous Dijkstra searches on the augmented graph – one forward from p_1 and one backward from p_2 stopping when the two meet in the middle. Since locally updating the graph at p_1 and p_2 takes $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ time, the procedure BidirectionalDijkstra(), taking $O(n)$ time, is the dominant term in terms of time complexity. Therefore, computing the augmented edges does not increase the overall time complexity.

Besides geodesic distances, our method is able to compute the geodesic path by triangle unfolding. Let $\gamma(s,t) \in M$ be the geodesic path on M and $\{p_0, p_1\}, \{p_1, p_2\}, \cdots, \{p_k, p_{k+1}\}$ be the corresponding shortest path on the DGG, where $p_0 = s$ and $p_{k+1} = t$. Then the procedure $\bigcup_{i=0}^{k}$ Backtrace($\{p_i, p_{i+1}\}$) can recover the geodesic path $\gamma(s,t)$ accurately.

Fig. A.17 shows computing geodesic path between non-vertex points using the augmented DGG. Fig. A.18 demonstrates the augmented DGG on farthest point sampling.

Appendix B. Proof of Theorem 2

Lemma A1. *Let $\triangle PQR$ be an obtuse triangle with $\angle Q > 90°$. Let S be the foot of the altitude from Q to PR. If $\|PR\| \geq \frac{\|QS\|^2}{2\varepsilon\|PS\|} + \|PS\|$, then PR can be approximated by PQ and QR with approximation ratio $1+\varepsilon$, i.e., $\|PQ\| + \|QR\| \leq (1+\varepsilon)\|PR\|$.*

Proof. Let us denote $\alpha = \angle QPR$, $\beta = \angle QRP$, $a = \|PS\|$ and $b = \|SR\|$.

$$\|PQ\| + \|QR\| \leq (1+\varepsilon)\|PR\| \Leftrightarrow \frac{a}{\cos\alpha} + \frac{b}{\cos\beta} \leq (1+\varepsilon)(a+b).$$

Observe that $\frac{1}{\cos\theta} = 1 + \frac{\theta^2}{2} + O(\theta^4)$. For small angles, we can safely omit the high order terms. Thus, we re-write the above inequality as

$$a\left(1 + \frac{\alpha^2}{2}\right) + b\left(1 + \frac{\beta^2}{2}\right) \leq (1+\varepsilon)(a+b),$$

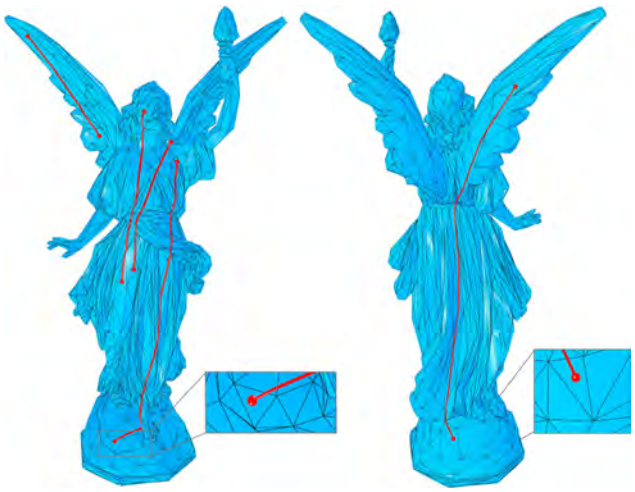
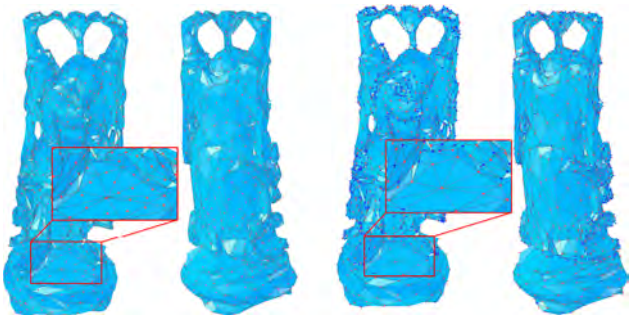


Fig. A.17. Computing geodesic path between non-vertex points using the augmented DGG.



(a) DGG based FPS (b) FMM/HM based FPS

Fig. A.18. We generate 1000 samples (in red) on the 2500-vertex Happy Buddha model using farthest point sampling (FPS), which, in each iteration, produces a new sample of the farthest distances to all existing samples. Note that the FMM and HM compute geodesic distances between vertices only, all generated samples are on mesh vertices, producing a result highly sensitive to mesh triangulation. Since the FMM and HM results are similar, we show only one of them. The blue vertices in (b) are the vertices which are not picked by FPS. In contrast, the DGG based FPS can distribute samples uniformly on the model. Images are of high resolution allowing close-up examination. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

which can be proven as follows:

$$\begin{aligned} &\|PR\| \geq \|PS\| + \frac{\|QS\|^2}{2\varepsilon\|PS\|} \\ \Rightarrow\ &\|QS\|^2 \leq 2\varepsilon ab \\ \Rightarrow\ &\frac{a\tan^2\alpha}{2} + \frac{b\tan^2\beta}{2} \leq \varepsilon(a+b) \\ \Rightarrow\ &a\alpha^2 + b\beta^2 \leq 2\varepsilon(a+b). \end{aligned} \tag{B.1}$$

The last inequality comes from the fact $\theta \leq \tan\theta$ for $0 \leq \theta < \pi/2$. □

Intuitively speaking, Lemma A1 implies that for a *narrow* window w which is sufficiently far from the source, any geodesic path going through w can be well approximated by the path going through the endpoint of w.

Lemma A2. *Let $\gamma(p,q)$ be a direct geodesic path and $\mathcal{E}_\varepsilon(\hat{p},\hat{q})$ the ellipse with foci $\hat{p}$ and $\hat{q}$ and eccentricity $\frac{1}{1+\varepsilon}$. If the ellipse $\mathcal{E}_\varepsilon(\hat{p},\hat{q})$ contains a vertex $\hat{r} \in \mathcal{F}(\gamma(p,q))$, we can always find a relay vertex $t \in V$, possibly coinciding with r, so that the approximation ratio is no more than $1+\varepsilon$.*

Proof. Since $\gamma(p,q)$ is direct, the geodesic distance between p and q equals the Euclidean distance between their unfolded images $\hat{p},\hat{q} \in \hat{\mathcal{F}}(\gamma(p,q))$, i.e., $d_g(p,q) = \|\hat{p}\hat{q}\|$. We consider three cases:

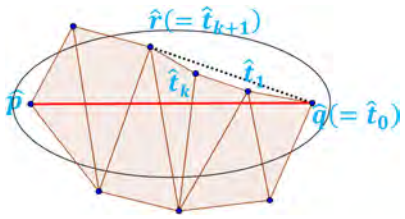


Fig. B.19. Case 2 in the proof of Lemma A2. The figure is not drawn to scale. For a typical ε (e.g., 0.1%), the ellipse E_ε is very elongated.

Case 1: Both geodesic paths $\gamma(p,r)$ and $\gamma(r,q)$ are direct. Then r is the relay vertex, because $d_g(p,r)+d_g(r,q) = \|\hat{p}\hat{r}\|+\|\hat{r}\hat{q}\| \le (1+\varepsilon)\|\hat{p}\hat{q}\| = \|(1+\varepsilon)d_g(p,q)$.

Case 2: One geodesic path, say $\gamma(r,q)$, is indirect, which passes through a sequence of saddle vertices $t_1,\cdots,t_k$. Denote by $t_0=q$ and $t_{k+1}=s$. Note that each angle $\angle\hat{t}_{i-1}\hat{t}_i\hat{t}_{i+1} > \pi$, $i=1,\cdots,k$, meaning that all saddle vertices $\{\hat{t}_i\}_{i=1}^k$ are also inside the ellipse C. Taking t_1 as the relay vertex, we have $d_g(p,t_1)+d_g(t_1,q) = \|\hat{p}\hat{t}_1\|+\|\hat{t}_1\hat{q}\| \le (1+\varepsilon)\|\hat{p}\hat{q}\| = (1+\varepsilon)d_g(p,q)$. See Fig. B.19(a).

Case 3: Both $\gamma(p,r)$ and $\gamma(r,q)$ are indirect. Similar to Case 2, let us denote by $\{s_j\}_{j=1}^m$ the sequence of saddle vertices lying on $\gamma(p,r)$. If $\hat{t}_1 \in \triangle\hat{p}\hat{s}_1\hat{q}$, take $\hat{t}_1$ as the relay vertex; otherwise, take s_1. For either case, we can approximate $\gamma(p,q)$ with an approximation ratio no more than $1+\varepsilon$. □

Theorem 2. *Let s be the source point and $w=(e_{ij},b_0,b_1,d_0,d_1,0)$ a direct window associated to an oriented half-edge $e_{ij}=(v_i,v_j)$. Denote by l the length of edge e_{ij}. For any geodesic path $\gamma(s,t)$ through window w, if $d_g(s,t) \ge \frac{l^2}{8\varepsilon(\min\{d_0,d_1\}-l/2)}+\frac{l}{2}+\max\{d_0,d_1\}$, then $\gamma(s,t)$ can be approximated by some vertex $u\in\mathcal{F}(\gamma(s,t))$ with the approximation ratio $1+\varepsilon$, i.e., $d_g(s,u)+d_g(u,t) \le (1+\varepsilon)d_g(s,t)$.*

Proof. Let $v_t \in e_{ij}$ be the intersection between the geodesic path $\gamma(s,t)$ and window w. Without loss of generality, assume $\|\hat{v}_i\hat{v}_t\| \le l/2$. Let $\hat{v}_f$ be the foot of altitude from $\hat{v}_i$ to $\hat{s}\hat{t}$. Then, we have

$$\begin{aligned}&\frac{\|\hat{v}_i\hat{v}_f\|}{2\varepsilon\|\hat{s}\hat{v}_f\|}+\|\hat{s}\hat{v}_f\| \le \frac{(l/2)^2}{2\varepsilon(d_0-l/2)}+d_0+l/2\\ \le\ &\frac{l^2}{8\varepsilon(\min\{d_0,d_1\}-l/2)}+\max\{d_0,d_1\}+l/2\\ \le\ &d_g(s,t).\end{aligned}$$

By Lemma A1, $(1+\varepsilon)\|\hat{s}\hat{t}\| \ge \|\hat{s}\hat{v}_i\|+\|\hat{v}_i\hat{t}\|$, implying that $\hat{v}_i$ lies in the ellipse $\hat{\mathcal{F}}(\gamma(s,t))$. By Lemma A2, there exists some vertex $u\in\mathcal{F}(\gamma(s,t))$, not necessarily v_i, that can approximate $\gamma(s,t)$ with approximation ratio $1+\varepsilon$. □

Appendix C. Proof of Theorem 3

Theorem 3. *Assume that the vertices are uniformly distributed on the mesh M. For an accuracy parameter $\varepsilon>0$, the following statements are true:*

1. *For a direct geodesic path γ with length l, the probability of γ being a DGG edge is $O(e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$, where A is the area of mesh M.*
2. *The constructed graph G_ε has $O(\frac{n}{\sqrt{\varepsilon}})$ edges.*
3. *For any vertices $p,q\in V$, the shortest path distance $d_s(p,q)$ on G_ε approximates the geodesic distance $d_g(p,q)$ on M with an approximation ratio $1+O(\varepsilon)$.*

Proof. First, we compute the probability of a direct geodesic path survives the redundant edge pruning. For a geodesic path $\gamma(p,q)$ with length l, let A_ε denote the area of ellipse $\mathcal{E}_\varepsilon(\hat{p},\hat{q})$. By Lemma A2, the probability of $\gamma(p,q)$ being a DGG edge equals the probability of the ellipse $\mathcal{E}_\varepsilon(\hat{p},\hat{q})$ containing *no* vertex other than $\hat{p}$ and $\hat{q}$ in the unfolded face sequence. Since the mesh vertices are uniformly distributed on M, such a probability is $(1-\frac{A_\varepsilon}{A})^{n-2}$. By inequality $(1-1/t)^t < e^{-1} < (1-1/t)^{t-1}$ for $t>0$, we have

$$e^{-\frac{(n-2)A_\varepsilon}{A-A_\varepsilon}} < (1-\frac{A_\varepsilon}{A})^{n-2} < e^{-\frac{(n-2)A_\varepsilon}{A}}.$$

Observe that $A_\varepsilon = \frac{\pi}{4}d_g^2(p,q)\sqrt{2\varepsilon}+O(\varepsilon^2)$. Ignoring the high order terms, we have $p = O(e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$.

Next we compute the average degree of G_ε. Consider any arbitrary vertex p. Let $\mathcal{A}$ be an annulus centered at p with inner radius l and outer radius $l+dl$, where dl is the length element. Since the vertices are distributed uniformly, the number of vertices falling in $\mathcal{A}$ is proportional to $\frac{2\pi n l dl}{A}$. Since the probability of a direct geodesic path with length l being a DGG edge is $O(e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$, the number of DGG edges with endpoint p and falling in $\mathcal{A}$ is $O(\frac{nldl}{A}e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}})$. Integrating the length l, we obtain the number of DGG edges with endpoint p is

$$\begin{aligned}
&O\Big(\int_0^\infty \frac{nldl}{A}e^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}}\Big)\\
&= O\Big(\frac{n}{A}\int_0^\infty le^{-\frac{(n-2)l^2\sqrt{\varepsilon}}{A}}dl\Big)\\
&= O\Big(\frac{n}{A}\frac{A}{(n-2)\sqrt{\varepsilon}}\Big)\ \ \Big(\because \int xe^{-tx^2} = \frac{1}{2t}\Big)\\
&= O\Big(\frac{1}{\sqrt{\varepsilon}}\Big).
\end{aligned}$$

Therefore, G_ε has $O(\frac{n}{\sqrt{\varepsilon}})$ edges.

Finally, we show that for arbitrary vertices $p, q \in V$, the shortest path distance $d_s(p,q)$ on G_ε approximates the geodesic distance $d_g(p,q)$ on M with an approximation ratio $1+O(\varepsilon)$. We consider two cases:

Case 1) $\gamma(p,q)$ is a direct geodesic path. If the geodesic path $\gamma(p,q)$ is a DGG edge, then we can obtain the *exact* geodesic distance $d_g(p,q)$.

Now assume $\gamma(p,q) \notin G_\varepsilon$, i.e., $\gamma(p,q)$ is a redundant edge. Consider an ellipse $\mathcal{E}_\varepsilon(p,q)$ with focal points p and q. By Theorem 2, there exists a point u whose unfolded image $\hat{u}$ is in $\mathcal{E}_\varepsilon(\hat{p},\hat{q})$, otherwise the geodesic path $\gamma(p,q)$ cannot be deleted. Since vertex u is also on the *shortest* path between p and q on G_ε, we have $d_s(p,q) = d_s(p,u) + d_s(u,q)$. Since $d_s(p,q)$ is the shortest path distance between p and q, for any point r in the ellipse $\mathcal{E}$, $d_s(p,q) = d_s(p,u) + d_s(u,q) \leqslant d_s(p,r) + d_s(r,q)$.

For all m vertices inside the ellipse $\mathcal{E}$, we have $m\ d_s(p,q) \leqslant \Sigma_{r\in\mathcal{E}}(d_s(p,r) + d_s(r,q))$. Since the vertices are distributed uniformly, in the continuous form we have

$$d_s(p,q) \leqslant \frac{1}{A_\mathcal{E}}\iint_{r\in\mathcal{E}} (d_s(p,r) + d_s(r,q))\,dr,$$

where $A_\mathcal{E}$ is the area of the ellipse. Define a function $f(x)$, for a geodesic path x, $f(x)$ is the length of the shortest path on DGG with the same end points as the geodesic path x. Let the geodesic distance $d_g(p,q)$ be l, so $d_s(p,q) = f(l)$.

By Lemma A2, for any vertex $\hat{r}$ inside the ellipse, we can find a vertex t satisfying $d_g(p,t) + d_g(t,q) \leqslant (1+O(\varepsilon))d_g(p,q)$. So there exists a length t $(0 \leqslant t \leqslant l)$, and $d_g(p,t) = (1+O(\varepsilon))t$, $d_g(t,q) = (1+O(\varepsilon))(l-t)$. So we have $d_s(p,t) = f((1+O(\varepsilon))t)$, $d_s(t,q) = f((1+O(\varepsilon))(l-t)$. Rewriting the surface integral for r as a line integral for t, we get

$$\begin{aligned}
f(l) &\leqslant \frac{1}{l}\int_0^l f\left((1+O(\varepsilon))t\right) + f\left((1+O(\varepsilon))(l-t)\right)\,dt\\
&\leqslant O\left((1+\varepsilon)^2\right)l.
\end{aligned}$$

Omitting the second order term $O(\varepsilon^2)$, we have $f(l) \leqslant O(1+2\varepsilon)l$.

Case 2) The geodesic path $\gamma(p,q)$ is indirect. Then $\gamma(p,q)$ can be decomposed into k segments $\gamma(p,q) = \cup_{i=0}^{k-1}\gamma(v_i, v_{i+1})$, where $v_0 = p$, $v_k = q$ and each segment $\gamma(v_i, v_{i+1})$ is a direct geodesic path. Therefore, we have $d_g(p,q) = \sum_{i=0}^{k-1} d_g(v_i, v_{i+1})$. Applying the result of Case 1 to each segment yields $d_s(v_i, v_{i+1}) \leqslant (1+O(\varepsilon))d_g(v_i, v_{i+1})$. Therefore, we have $d_s(p,q) \leqslant (1+O(\varepsilon))d_g(p,q)$.

Putting it all together, the shortest path distance $d_s(p,q)$ approximates the geodesic distance $d_g(p,q)$ with an approximation ratio $1+O(\varepsilon)$. □

References

Agarwal, P.K., Har-Peled, S., Sharir, M., Varadarajan, K.R., 1997. Approximating shortest paths on a convex polytope in three dimensions. J. ACM 44 (4), 567–584.

Aleksandrov, L., Lanthier, M., Maheshwari, A., Sack, J.-R., 1998. An ε-approximation for weighted shortest paths on polyhedral surfaces. In: Proceedings of the 6th Scandinavian Workshop on Algorithm Theory. SWAT '98, pp. 11–22.
Belyaev, A.G., Fayolle, P.-A., 2015. On variational and PDE-based distance function approximations. Comput. Graph. Forum 34 (8), 104–118.
Bertsekas, D.P., 1998. Network Optimization: Continuous and Discrete Models. Athena Scientific.
Bommes, D., Kobbelt, L., 2007. Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. VMV 7, 151–160.
Campen, M., Heistermann, M., Kobbelt, L., 2013. Practical anisotropic geodesy. Comput. Graph. Forum 32 (5), 63–71.
Campen, M., Kobbelt, L., 2011. Walking on broken mesh: defect-tolerant geodesic distances and parameterizations. Comput. Graph. Forum 30 (2), 623–632.
Chen, J., Han, Y., 1990. Shortest paths on a polyhedron. In: Proceedings of the Sixth Annual Symposium on Computational Geometry. ACM, pp. 360–369.
Crane, K., Weischedel, C., Wardetzky, M., 2013. Geodesics in heat: a new approach to computing distance based on heat flow. ACM Trans. Graph. 32 (5), 152.
Detrixhe, M., Gibou, F., Min, C., 2013. A parallel fast sweeping method for the Eikonal equation. J. Comput. Phys. 237, 46–55.
Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1 (1), 269–271.
Fu, Z., Jeong, W.-K., Pan, Y., Kirby, R.M., Whitaker, R.T., 2011. A fast iterative method for solving the Eikonal equation on triangulated surfaces. SIAM J. Sci. Comput. 33 (5), 2468–2488.
Har-Peled, S., 1997. Approximate shortest paths and geodesic diameters on convex polytopes in three dimensions. In: Proceedings of the Thirteenth Annual Symposium on Computational Geometry, pp. 359–365.
Har-Peled, S., 1998. Constructing approximate shortest path maps in three dimensions. In: Proceedings of the Fourteenth Annual Symposium on Computational Geometry, pp. 383–391.
Hershberger, J., Suri, S., 1995. Practical methods for approximating shortest paths on a convex polytope in $\mathbb{R}^3$. In: Proceedings of the Sixth Annual ACM–SIAM Symposium on Discrete Algorithms, pp. 447–456.
Kimmel, R., Sethian, J., 1998. Computing geodesic paths on manifolds. Proc. Natl. Acad. Sci. 95, 8431–8435.
Lipman, Y., Rustamov, R.M., Funkhouser, T.A., 2010. Biharmonic distance. ACM Trans. Graph. 29 (3), 27.
Liu, Y.-J., 2013. Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures. Comput. Aided Des. 45 (3), 695–704.
Liu, Y.-J., Xu, C.-X., Fan, D., He, Y., 2015. Efficient construction and simplification of Delaunay meshes. ACM Trans. Graph. 34 (6), 174.
Liu, Y.-J., Zhou, Q.-Y., Hu, S.-M., 2007. Handling degenerate cases in exact geodesic computation on triangle meshes. Vis. Comput. 23 (9–11), 661–668.
Mémoli, F., Sapiro, G., 2001. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. J. Comput. Phys. 173 (2), 730–764.
Mitchell, J.S., Mount, D.M., Papadimitriou, C.H., 1987. The discrete geodesic problem. SIAM J. Comput. 16 (4), 647–668.
Qin, Y., Han, X., Yu, H., Yu, Y., Zhang, J., 2016. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. ACM Trans. Graph. 35 (4), 125.
Schreiber, Y., 2010. An optimal-time algorithm for shortest paths on realistic polyhedra. Discrete Comput. Geom. 43 (1), 21–53.
Schreiber, Y., Sharir, M., 2008. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. Discrete Comput. Geom. 39 (1–3), 500–579.
Sethian, J., 1996. A fast marching level set method for monotonically advancing fronts. Proc. Natl. Acad. Sci. 93, 1591–1595.
Sethian, J., Vladimirsky, A., 2000. Fast methods for the Eikonal and related Hamilton–Jacobi equations on unstructured meshes. Proc. Natl. Acad. Sci. 97, 5699–5703.
Solomon, J., Rustamov, R., Guibas, L., Butscher, A., 2014. Earth mover's distances on discrete surfaces. ACM Trans. Graph. 33 (4), 67.
Spira, A., Kimmel, R., 2004. An efficient solution to the Eikonal equation on parametric manifolds. Interfaces Free Bound. 6 (4), 315–327.
Surazhsky, V., Surazhsky, T., Kirsanov, D., Gortler, S.J., Hoppe, H., 2005. Fast exact and approximate geodesics on meshes. ACM Trans. Graph. 24 (3), 553–560.
Tsitsiklis, J., 1995. Efficient algorithms for globally optimal trajectories. IEEE Trans. Autom. Control 40 (9), 1528–1538.
Varadarajan, K.R., Agarwal, P.K., 2000. Approximating shortest paths on a nonconvex polyhedron. SIAM J. Comput. 30 (4), 1321–1340.
Varadhan, S., 1967. On the behavior of the fundamental solution of the heat equation with variable coefficients. Commun. Pure Appl. Math. 20 (2), 431–455.
Weber, O., Devir, Y.S., Bronstein, A.M., Bronstein, M.M., Kimmel, R., 2008. Parallel algorithms for approximation of distance maps on parametric surfaces. ACM Trans. Graph. 27 (4), 104.
Xin, S.-Q., Quynh, D.T., Ying, X., He, Y., 2012a. A global algorithm to compute defect-tolerant geodesic distance. In: SIGGRAPH Asia 2012 Technical Briefs, 23.
Xin, S.-Q., Wang, G.-J., 2009. Improving Chen and Han's algorithm on the discrete geodesic problem. ACM Trans. Graph. 28 (4), 104.
Xin, S.-Q., Ying, X., He, Y., 2011. Efficiently computing geodesic offsets on triangle meshes by the extended Xin–Wang algorithm. Comput. Aided Des. 43 (11), 1468–1476.
Xin, S.-Q., Ying, X., He, Y., 2012b. Constant-time all-pairs geodesic distance query on triangle meshes. In: Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, pp. 31–38.
Xu, C.-X., Wang, T.Y., Liu, Y.-J., Liu, L., He, Y., 2015. Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes. IEEE Trans. Vis. Comput. Graph. 21 (7), 822–834.
Ying, X., Wang, X., He, Y., 2013. Saddle vertex graph (SVG): a novel solution to the discrete geodesic problem. ACM Trans. Graph. 32 (6), 170.
Ying, X., Xin, S.-Q., He, Y., 2014. Parallel Chen–Han (PCH) algorithm for discrete geodesics. ACM Trans. Graph. 33 (1), 9.
Zhao, H., 2007. Parallel implementations of the fast sweeping method. J. Comput. Math. 25 (4), 421–429.