Geodesic Embedding (GE): A High-Dimensional Embedding Approach for Fast Geodesic Distance Query

Qianwei Xia, Juyong Zhang, Jin Li
University Of Science And Technology Of China
Hefei, China
xqw000@mail.ustc.edu.cn, juyong@ustc.edu.cn, jarvis@mail.ustc.edu.cn

Zheng Fang, Ying He
Nanyang Technological University
Singapore, Singapore
fz0420@hotmail.com, YHe@ntu.edu.sg

Bailin Deng
Cardiff University
Cardiff, Wales
DengB3@cardiff.ac.uk

Abstract

In this paper, we develop a novel method for fast geodesic distance query. The key idea is to embed the mesh into a high-dimensional space, such that the Euclidean distance in the high-dimensional space can induce the geodesic distance in the original manifold surface. However, directly solving the high-dimensional embedding problem is not feasible due to large number of variables and high non-linearity of the embedding problem. We overcome the challenges by two novel ideas. First, instead of taking all vertices as variables, we embed only the saddle vertices, which greatly reduces the problem complexity. We then compute a local embedding for each non-saddle vertex. Second, to reduce the large approximation error resulting from pure Euclidean embedding, we propose a cascaded optimization approach that repeatedly introduces additional embedding coordinates with a non-Euclidean function to reduce the approximation residual. Using the pre-computed data, our approach can determine the geodesic distance between any two vertices in near constant time. Computational results show that our method is more accurate than the other geodesic distance query methods.

1. Introduction

The discrete geodesic problem has been investigated extensively since the seminal paper published by Mitchell et al. [18]. There are many elegant and efficient algorithms for computing the single-source-all-destinations (SSAD) geodesic distances in discrete domains. With significant progress of pre-computation methods, such as the heat method [9] and the graph-based method [34, 1], SSAD can now be solved in empirically linear time.

From the perspective of applications, computing the geodesic distance between two arbitrary points, a.k.a. geodesic distance query (GDQ), is a common scenario in shape analysis, such as intrinsic symmetry detection [33, 23], surface parameterization [37], shape correspondence [6], etc. Unlike SSAD, only a handful of methods have been designed for answering GDQ, including geodesic triangle unfolding (GTU) [31], Euclidean embedding metric (EEM) [20] and pointwise distance metric (PDM) [25]. These methods, however, either have high space complexity or large error that limits their usage to accuracy critical applications. For example, the naive way is to compute and store all point-wise distance. However, such approach will consume large storage to save the geodesic distance between all vertex pairs.

In this paper, we propose a novel method for fast geodesic distance query. The key idea is to lift the model into a high-dimensional Euclidean space $\mathbb{R}^d$, so that the distance of two points in $\mathbb{R}^d$ can induce the geodesic distance on the input mesh. The high-dimensional embedding is typically formulated as multidimensional scaling (MDS) [16]. However, directly solving this optimization problem is challenging due to the potentially huge number of objective function terms and the high non-linearity of the objective function. In addition, pure Euclidean embedding often does not approximate the pairwise geodesic distance well. We tackle these issues using two novel ideas. First, instead of lifting all vertices, we embed only the saddle vertices[1] in the high-dimensional space. As pointed out in [34], saddle vertices are crucial in solving the discrete geodesic problem, since they serve as backbone of saddle vertex graphs (SVG). Based on the embedding of saddle vertices and the

[1]A vertex is called saddle if it has negative Gaussian curvature.

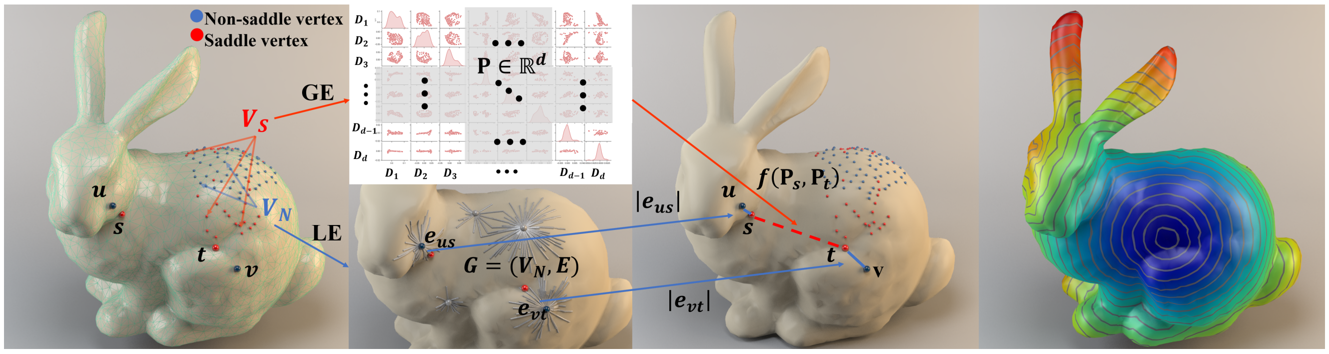

Figure 1. Our method performs distance query between an arbitrary pair of vertices in near constant time. (a) Distance query for (u, v). $\mathcal{V}_N$ is the set of non-saddle vertices and $\mathcal{V}_S$ is the set of saddle vertices. $u, v \in \mathcal{V}_N$ and $s, t \in \mathcal{V}_s$. (b) Top: Geodesic embedding (GE) for $\mathcal{V}_S$ and the scatter plot matrix of $\mathbf{P} \in \mathbb{R}^d$. $\mathbf{P}$ is the matrix for the embedded coordinates of $\mathcal{V}_S$. Bottom: Local embedding (LE) for $\mathcal{V}_N$ using Saddle Vertex Graph (SVG) [34]. s and t are the local neighbors of u and v respectively. The bidirectional SVG edge $e_{us}, e_{vt} \in E$ store the exact geodesic distance of (u, s) and (v, t). (c) The geodesic distance $d(u, v) = |e_{us}| + f(\mathbf{P}_s, \mathbf{P}_t) + |e_{vt}|$, where $\mathbf{P}_s, \mathbf{P}_t \in \mathbb{R}^d$ are the GE results of s and t from $\mathbf{P}$. (d) An illustration of single-source-all-destinations distance field. For this bunny model ($\#V = 34835$), the average query time for a vertex pair is 0.02ms, and the relative mean error of this query is $\varepsilon = 0.7367\%$.

resulting fast evaluation of geodesic distance between them, we adapt the SVG to achieve fast GDQ for any pair of vertices. Second, we propose a cascaded non-Euclidean embedding to gradually reduce the distance approximation error. Compared with pure Euclidean embedding, our approach can effectively improve the distance accuracy by increasing the dimensionality of the embedding vectors. An example is shown in Fig. 1. Computational results show that our method is able to achieve near-constant time query of geodesic distance with high accuracy only with very small storage consumption.

2. Related Works

There is a large body of literature on the discrete geodesic problem. Most algorithms focus on computing SSAD on triangle meshes. Representative algorithms are the wavefront propagation methods [18, 5, 27, 30, 35, 15, 32, 21, 22], the partial differential equation (PDE) methods [24, 11, 17, 9, 3, 28], and the graph-based methods [34, 29, 1]. As mentioned above, applying the SSAD methods for GDQ is too expensive. In the following, we review only the related works on GDQ.

The GTU method [31] solves GDQ in constant time using preprocessed distance information. In the preprocessing step, GTU constructs the geodesic Delaunay triangulation for a fixed number of sample vertices. It computes the pairwise geodesic distance of the samples. For each mesh vertex, the distance to the three vertices of its belonging geodesic triangle is recorded. In the query step, with the precomputed distance flatten on $\mathbb{R}^2$, GTU can answer the GDQ using an unfolding technique. GTU is conceptually simple and elegant, however, it is sensitive to geometric features and only works for small-scale meshes due to its quadratic space complexity for precomputation.

Panazzo et al. [20] proposed Euclidean embedding metric that samples a representative subset S of the mesh vertices, and computes pairwise approximate geodesic distances for these samples. They then embedded S in a high-dimensional Euclidean space using metric multidimensional scaling (MMDS) [8]. After that, they embedded the remaining vertices using least-square meshes [26]. EMM generates smooth and evenly-spaced smooth distances quickly. However, their results often have large approximation errors and their iso-distance contours are not close to geodesic distances.

There are also a few spectral distances, such as diffusion distances [7], commute-time distances [10] and biharmonic distances [13]. These distances are smooth, shape aware, and have closed-form solution. However, they are not isotropic, therefore their level sets are not evenly spaced.

Using optimal transport map between distributions that are only nonzero at individual vertices, Solomon et al. defined a family of smooth distances on a surface transitioning from biharmonic distance to geodesic distance [25]. Taking all the eigenvectors of the Laplacian matrix, the resulting distance is exact geodesic distance. However, it is not practical to compute all the eigenvectors unless the mesh is small. Therefore, they suggest using only a few low-frequency spectral terms to approximate geodesic distances, which often yields large error.

The idea of the high-dimension embedding is not new. Given an initial polygon mesh, a user-specified desired number of vertices and a parameter that specifies the desired amount of anisotropy, Levy and Bonnee [12] generated a curvature-adapted anisotropic mesh. The main idea is to transform the 3D anisotropic space into a higher-dimensional isotropic space, where the mesh is optimized by computing a Centroidal Voronoi Tessellation. Zhong et al. [36] presented a new method to compute a self-intersection free high dimensional Euclidean embedding for surfaces and volumes equipped with an arbitrary Riemannian metric. Given an input metric, they compute a smooth intersection-free high-dimensional embedding of the input such that the pullback metric of the embedding matches the input metric.

3. Proposed Model

Suppose we are given a connected triangle mesh $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$ denote the set of vertices, edges, and faces, respectively. The basic idea of our approach is to compute a high-dimensional vector $\mathbf{P}_k$ for each vertex v_k, so that the true geodesic distance d_{ij} between two vertices v_i, v_j can be well approximated by a simple function $f(\mathbf{P}_i, \mathbf{P}_j)$. If the function f is given, then the embedding vectors $\{\mathbf{P}_k\}$ can be determined by solving an optimization problem

$$\min_{\{\mathbf{P}_k\}} \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \Phi(f(\mathbf{P}_i, \mathbf{P}_j), d_{ij}), \tag{1}$$

where $\Phi(f(\mathbf{P}_i, \mathbf{P}_j), d_{ij})$ penalizes the difference between $f(\mathbf{P}_i, \mathbf{P}_j)$ and d_{ij}. However, performing such optimization for all vertex pairs will involve $O(|\mathcal{V}|^2)$ terms in the target function, which will require a significant amount of storage and computational time. For better efficiency, we note that the saddle vertices can serve as relays for geodesic wavefront propagation, and a saddle vertex can be shared by multiple discrete geodesic paths connecting mesh vertices [34]. Therefore, we perform optimization (1) only on the saddle vertices to enable fast query of geodesic distance between them in $O(1)$ time. Utilizing the distance between saddle vertex pairs, we adapt the Saddle Vertex Graph (SVG) proposed in [34] for geodesic distance query between an arbitrary pair of vertices with low time complexity.

In the following, we first explain our adaptation of SVG and how we use it for distance query. Afterwards, we present our method for solving the embedding problem for saddle vertices.

3.1. SVG Construction

The mesh vertex set $\mathcal{V}$ can be partitioned into two sets $\mathcal{V}_S$ and $\mathcal{V}_N$ containing all saddle vertices and all non-saddle vertices respectively. Consider a geodesic wavefront propagating from a source vertex. It will split at $\mathcal{V}_S$ and continue propagating within the fan-shape region of each saddle vertex, which is shown as the shadow region in Figure 2. Mitchell et al. [18] proved that a geodesic path cannot pass through a spherical vertex unless it is an endpoint or a boundary point. We define a geodesic path as *direct* if

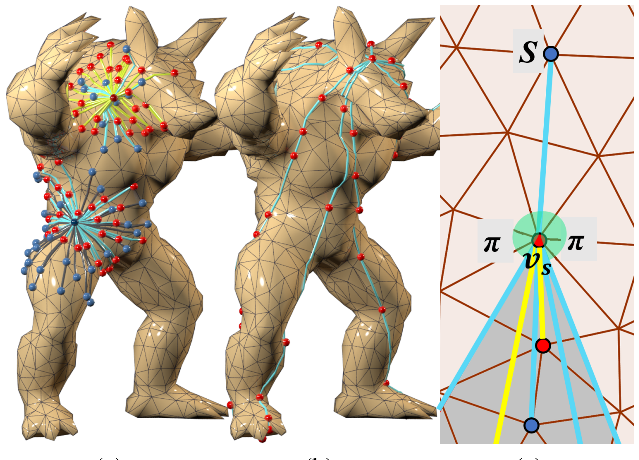

Figure 2. Illustrations of SVG construction and the properties of saddle vertex. (a) Two subsets of G generated from two source vertices. The red and blue spheres indicate the saddle and non-saddle vertices respectively. The upper subset contains E_{SS} in yellow and the cyan-color-rendered E_{NS}. The lower subset also includes E_{NN} in black color. All edges are direct geodesics. (b) shows that each long discrete geodesic path passes through several saddle vertices. (c) Close-up view of a saddle vertex v_s. The geodesic paths within the shadow region of v_s adopt v_s as a relay vertex.

it passes through no saddle vertex and *indirect* otherwise. The longer a path is, the more likely it is indirect. But it may consist of multiple *direct* paths. The saddle vertices along the discrete geodesic paths are called pseudosources in these algorithms and act as relay vertices connecting each direct geodesic path. So we can construct a graph which only contains direct paths, with saddle vertices acting as relays in the graph. The SVG proposed in [34] is a sparse undirected graph that utilizes saddle vertices to encode discrete geodesic paths between mesh vertices. The SVG for a given mesh contains all the mesh vertices as its nodes, while its edges represent direct geodesic paths connecting the nodes. Each edge has a weight that equals to the geodesic distance between the two nodes it connects. An exact SVG contains all direct geodesic paths as its edges, so that the geodesic distance between any two mesh vertices can be computed by solving a shortest path problem between their corresponding nodes on the SVG. However, constructing an exact SVG is computationally expensive. To improve performance, Ying et al. [34] construct an approximate SVG by limiting the maximal degree to a parameter K. In their construction, the incident SVG edges for each vertex v connect it to other vertices within a local geodesic disc centered at v, and are determined by propagation from v. It is shown in [34] that such an approximate SVG still achieves high accuracy for geodesic distance computation. The approximate SVG construction is a worst-case $O(nK^2 \log K)$ and empirical $O(nK^{1.5} \log K)$ time complexity where $n = |\mathcal{V}|$

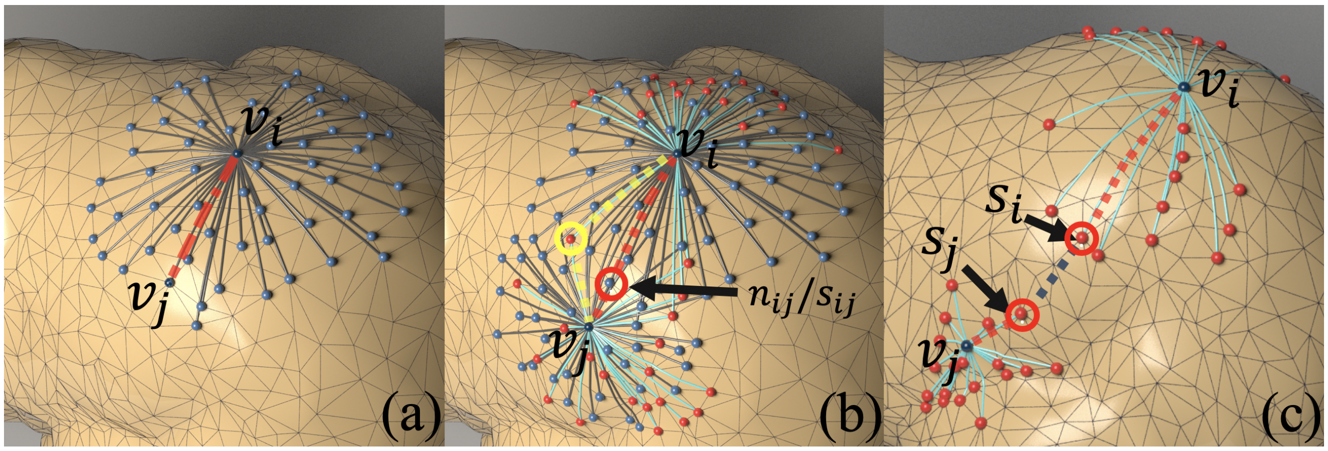

Figure 3. Solving NN-distance for (v_i, v_j) using SVG. (a) The 'direct' case. The red dotted curve indicates the shortest path, which is a direct edge in G_{NN}. (b) v_i, v_j are 'near' to each other. n_{ij} and s_{ij} belongs to the common set of the saddle and non-saddle neighbors of v_i and v_j. The set intersection is computed on $G_{NN} + G_{NS}$. The yellow path illustrates the detour effect. (c) shows the 'far' case. The black dotted line represents the embedding distance between s_i and s_j. Only G_{NS} is computed in this case.

is the number of vertices, which is linear in mesh size when K is fixed.

We adopt the method from [34] to benefit from its low complexity, with a slight modification to suit our need. Specifically, our query method attempts to find neighboring saddle vertices for both the source and target vertices on the SVG to relay the geodesic path, and to use the distance between saddle vertices where possible to benefit from its fast evaluation via embedding. Therefore, we would like to have a sufficient amount of saddle vertex neighbors for each node in the SVG, while still limiting its maximal degree. To this end, we set another threshold parameter $K_S < K$ for the number of neighboring saddle vertices. We set $K_S = K/3$ in our implementation. For each vertex v, we terminate the propagation for finding neighboring SVG nodes if we have found K neighbors or K_s saddle vertex neighbors. Hence the maximal degree of the resulting SVG is $O(K)$.

3.2. Geodesic Distance Query

The SVG constructed as mentioned above is a union of three sub-graphs $G = G_{SS} \cup G_{NS} \cup G_{NN}$:

- Sub-graph $G_{SS} = (\mathcal{V}_S, E_{SS})$ forms the skeleton of the whole SVG. Each edge of E_{SS} is a direct geodesic path connecting two saddle vertices.
- Sub-graph $G_{NS} = (\mathcal{V}_S \cup \mathcal{V}_N, E_{NS})$ connects non-saddle vertices to saddle vertices.
- Sub-graph $G_{NN} = (\mathcal{V}_N, E_{NN})$ connects non-saddle vertices to non-saddle vertices.

Based on the three-tier structure of SVG, we compute the distance between any pair of vertices in different ways according to the tier it belongs to. We label the distance between two saddle vertices as *SS-distance*, the distance between two non-saddle vertices as *NN-distance* and the distance between a non-saddle vertex and a saddle vertex as *NS-distance*.

3.2.1 SS-distance

We pre-compute the embedding of $\mathcal{V}_S$ in a high-dimensional space, so that the geodesic distance between any two saddle vertices v_i, v_j can be approximated using their embedded coordinates $\mathbf{P}_i, \mathbf{P}_j \in \mathbb{R}^d$ via a simple function $f(\mathbf{P}_i, \mathbf{P}_j)$. Afterwards, the geodesic distance between two saddle vertices are evaluated by retrieving their embedded coordinates to evaluate the function f, which can be done in $O(1)$ total time. The definition of f and the numerical method for computing the embedding will be explained in Section 3.3.

3.2.2 NN-distance.

If two non-saddle vertices v_i, v_j are neighbors on G_{NN}, then their geodesic distance can be directly retrieved from their edge length. Using an associative container such as the STL map to store the neighbors for each vertex, we can determine whether v_i and v_j are neighbors in $O(\log K)$ time, and retrieve the distance in $O(1)$ time if this is the case. If v_i and v_j are not neighbors on the SVG, then the geodesic path between them consists of SVG edges. We use the following steps to determine the geodesic distance between v_i and v_j:

- When v_i and v_j are close, it is possible that the geodesic path is relayed by their common neighbors. Therefore, we first extract the set of all common neighbors between v_i and v_j on the SVG. If such common neighbor set is not empty, then we compute the length of each path (v_i, n_k, v_j) that consists of two edges connecting v_i and v_j to a common neighbor n_k, and take the shortest length of all such paths as the geodesic distance between v_i and v_j. Since the SVG has a maximal degree K, this procedure can be performed in $O(K)$ time.

- If there is no common neighbor between v_i and v_j, then we assume that v_i, v_j are far apart enough and the geodesic path between them is relayed by saddle vertices. Thus we first extract the set S_i of saddle vertex neighbors for v_i, and the set S_j of saddle vertex neighbors for v_j. If neither of them is empty, we compute for shortest distance on the graph $S_i \cup S_j \cup S_{ij}$ between v_i and v_j as their geodesic distance, where S_{ij} is a dense graph that connects every node in S_i to every node in S_j. We compute the shortest distance using Dijkstra's algorithm, without explicitly constructing the dense graph S_{ij}. Since there are $O(K)$ nodes and $O(K^2)$ edges in $S_i \cup S_j \cup S_{ij}$, and each edge length of S_{ij} can be evaluated from the embedding in $O(1)$ time, the time complexity of the geodesic distance computation between v_i and v_j is $O(K^2)$.
- Finally, if at least one of S_i and S_j is empty, then we determine the geodeisc distance between v_i and v_j using Dijkstra's algorithm on G, with a time complexity $O(Kn \log n)$.

3.2.3 NS-distance.

The strategy to solve NS-distance is similar to solving *NN-distance*. The difference is that the construction of local graphs only executes on the non-saddle vertex side.

3.3. Geodesic Embedding

Central to our approach is to embed the geodesic distance between saddle vertices by solving the problem (1). To effectively solve this problem, we first apply high-dimensional Euclidean embedding to approximate the geodesic distance, and then apply a cascaded non-Euclidean embedding to further reduce the approximation error.

3.3.1 Euclidean Embedding

We first compute an embedding vector $\mathbf{q}_k \in \mathbb{R}^m$ for each vertex v_k, such that the Euclidean distance $\|\mathbf{q}_i - \mathbf{q}_j\|$ between two embedding vectors $\mathbf{q}_i, \mathbf{q}_j$ is as close as possible to the ground-truth geodesic distance d_{ij} between their corresponding vertices v_i, v_j. To this end, we perform an optimization based on the Shape-Up formulation from [4]:

$$\min_{\{\mathbf{q}_k\}} \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \omega_{ij} E_{ij}, \tag{2}$$

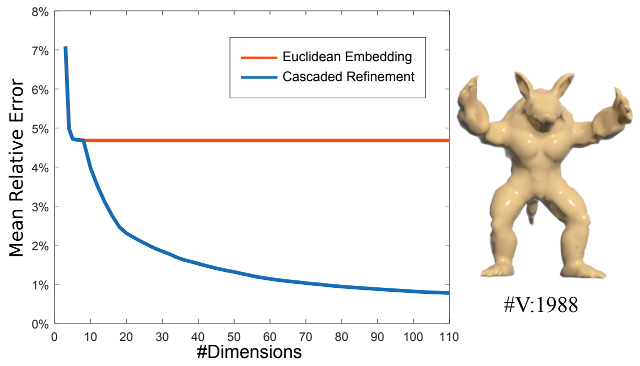

Figure 4. The mean relative errors of geodesic distance (see Eq. (3)) using Euclidean embedding, and using our approach with cascaded refinement after an initial eight-dimensional Euclidean embedding. The plots show the errors with respect to the dimensionality of the final embedding space. Our cascaded refinement can effectively reduce the approximation error with increasing dimensionality, while for pure Euclidean embedding the error stagnates after a certain threshold of dimensionality.

where

$$E_{ij} = \min_{\|\mathbf{z}_{ij}\| = d_{ij}} \|\mathbf{q}_i - \mathbf{q}_j - \mathbf{z}_{ij}\|^2$$

is the squared distance from the difference vector $\mathbf{q}_i - \mathbf{q}_j$ to the set of vectors in $\mathbf{R}^m$ with length d_{ij}, and the weight $\omega_{ij} = 1/d_{ij}^2$ is used to normalize the approximation error. We adopt the quasi-Newton solver from [14] for an efficient solution to (2).

3.3.2 Cascaded Non-Euclidean Embedding

The Euclidean embedding alone may not approximate the geodesic distance very well even with a large m. Fig. 4 shows the accuracy of Euclidean embedding on an Armadillo mesh using different values of m. To evaluate the accuracy, we follow [27] and compute the mean relative error

$$\varepsilon = \frac{1}{|\mathcal{P}|} \sum_{(v_i, v_j) \in \mathcal{P}} \varepsilon_{ij}, \tag{3}$$

where $\mathcal{P}$ is the set of sampled vertex pairs, and ε_{ij} is the relative error for the geodesic distance between two vertices v_i and v_j:

$$\varepsilon_{ij} = \frac{|d'_{ij} - d_{ij}|}{d_{ij}}, \tag{4}$$

with d_{ij} and d'_{ij} being the ground-truth distance and the distance computed from the embedding, respectively. In Fig. 4, we use all vertex pairs on the model to evaluate the mean relative error. Tab. 1 also provides the values of ε for different values of m. We can see that beyond a certain threshold, increasing the dimensionality for Euclidean embedding becomes ineffective for reducing the mean relative error. Therefore, we only perform the Euclidean embedding with a relatively small value of m. Afterwards, we increase the dimensionality of the embedding vectors and apply a non-Euclidean function on the extra dimensions to reduce the approximation error. Our key observation is that a non-zero residual

$$r_1^{ij} = \|\mathbf{q}_i - \mathbf{q}_j\| - d_{ij}$$

can be positive or negative. Therefore, we introduce a simple function for the extra embedding vector components which can attain both positive and negative values to approximate the residual, and subtract it from the existing embedding distance function to improve the approximation accuracy. Specifically, we add two components $s_1^{(k)}, t_1^{(k)}$ to the embedding vector of v_k, and approximate the residual r_1^{ij} for the distance between v_i and v_j using the following term:

$$(s_1^{(i)} - s_1^{(j)})^2 - (t_1^{(i)} - t_1^{(j)})^2.$$

We determine the values of $\{(s_1^{(k)}, t_1^{(k)})\}$ by solving a non-linear least squares problem:

$$\min_{\{(s_1^{(k)}, t_1^{(k)})\}} \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \omega_{ij} \left((s_1^{(i)} - s_1^{(j)})^2 - (t_1^{(i)} - t_1^{(j)})^2 - r_1^{ij} \right)^2. \tag{5}$$

We adopt the L-BFGS algorithm [19] in the ceres library [2] to solve this problem. Then the distance function for the embedding vectors of v_i and v_j becomes

$$\|\mathbf{q}_i - \mathbf{q}_j\| - \left(s_1^{(i)} - s_1^{(j)}\right)^2 + \left(t_1^{(i)} - t_1^{(j)}\right)^2. \tag{6}$$

Note that both problems (2) and (5) can be considered as an instance of the following optimization problem that minimizes the distance approximation error:

$$\min \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \omega_{ij} (f(\mathbf{P}_i, \mathbf{P}_j) - d_{ij})^2 \tag{7}$$

where $f(\mathbf{P}_i, \mathbf{P}_j)$ denotes the function that approximates the geodesic distance between two vertices v_i, v_j based on their embedding vectors $\mathbf{P}_i, \mathbf{P}_j$. The target function of this problem measures the overall normalized approximation error for the geodesic distance. In problem (2), the vectors $\{\mathbf{P}_k\}$ are the Euclidean embedding coordinates in $\mathbf{q}_k \in \mathbf{R}^m$, and the function f is the Euclidean distance. In problem (5), each vector $\mathbf{P}_k = (\mathbf{q}_k, s_1^{(k)}, t_1^{(k)})$ contains Euclidean and non-Euclidean components, and the function f is evaluated according to Eq. (6). Moreover, as the Euclidean embedding coordinates $\{\mathbf{q}_k\}$ are the solution to problem (2), setting $s_1^{(k)} = t_1^{(k)} = 0$ for all vertices will result in the same overall approximation error as the pure Euclidean embedding. Therefore, the solution to problem (5) is guaranteed to

Table 1. Mean relative error ε (Eq. (3)) of geodesic distance using Euclidean embedding (Eq. (2)) with different dimensionality m on the Armadillo model in Fig. 4.

m	ε	m	ε
3	5.909%	7	4.687%
4	4.899%	8	4.686%
5	4.739%	9	4.684%
6	4.700%	10	4.680%

reduce the overall approximation error compared with Euclidean embedding.

The non-Euclidean embedding optimization (5) can be repeated to further reduce the approximation error. Concretely, using the solution to problem (5), we can compute a new residual for each the distance d_{ij} based on the distance formula (6):

$$r_2^{ij} = \|\mathbf{q}_i - \mathbf{q}_j\| - \left(s_1^{(i)} - s_1^{(j)}\right)^2 + \left(t_1^{(i)} - t_1^{(j)}\right)^2 - d_{ij}.$$

Then we introduce two extra embedding vector components $s_2^{(k)}, t_2^{(k)}$ for each vertex v_k, and determine their values by solve an optimization problem similar to (5):

$$\min_{\{(s_2^{(k)}, t_2^{(k)})\}} \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \omega_{ij} \left((s_2^{(i)} - s_2^{(j)})^2 - (t_2^{(i)} - t_2^{(j)})^2 - r_2^{ij} \right)^2. \tag{8}$$

Then the embedding distance function becomes

$$\begin{aligned} &\|\mathbf{q}_i - \mathbf{q}_j\| - \left(s_1^{(i)} - s_1^{(j)}\right)^2 + \left(t_1^{(i)} - t_1^{(j)}\right)^2 \\ &\quad - \left(s_2^{(i)} - s_2^{(j)}\right)^2 + \left(t_2^{(i)} - t_2^{(j)}\right)^2 \\ = &\|\mathbf{q}_i - \mathbf{q}_j\| - \sum_{k=1}^{2} \left(s_k^{(i)} - s_k^{(j)}\right)^2 + \sum_{k=1}^{2} \left(t_k^{(i)} - t_k^{(j)}\right)^2 \end{aligned} \tag{9}$$

Following the same argument as the previous paragraph, we can see that the introduction of the components $(s_2^{(k)}, t_2^{(k)})$ is guaranteed to reduce the overall approximation error for the geodesic distance. We repeat this process and perform l rounds of non-Euclidean embedding optimization. The p-th round of optimization ($1 \leq p \leq l$) solves a problem

$$\min_{\{(s_p^{(k)}, t_p^{(k)})\}} \sum_{\substack{v_i, v_j \in \mathcal{V} \\ i<j}} \omega_{ij} \left((s_p^{(i)} - s_p^{(j)})^2 - (t_p^{(i)} - t_p^{(j)})^2 - r_p^{ij} \right)^2, \tag{10}$$

where r_p^{ij} is the residual for distance d_{ij} after the previous round of optimization:

$$r_p^{ij} = \|\mathbf{q}_i - \mathbf{q}_j\| - \sum_{k=1}^{p-1} \left(s_k^{(i)} - s_k^{(j)}\right)^2 + \sum_{k=1}^{p-1} \left(t_k^{(i)} - t_k^{(j)}\right)^2 - d_{ij}.$$

Table 2. Mean relative error ε (Eq. (3)) of geodesic distance with our cascaded refinement approach on the Armadillo model in Fig. 4, using an initial Euclidean embedding with dimensionality $m = 8$ and different numbers of refinement steps l. Our method can effectively reduce the error with an increasing l.

l	ε	l	ε	l	ε
0	4.682%	17	1.453%	34	0.9738%
1	3.972%	18	1.412%	35	0.9575%
2	3.495%	19	1.364%	36	0.9441%
3	3.095%	20	1.322%	37	0.9288%
4	2.756%	21	1.292%	38	0.9159%
4	2.466%	22	1.264%	39	0.9024%
6	2.356%	23	1.235%	40	0.8888%
7	2.206%	24	1.206%	41	0.8724%
8	2.107%	25	1.176%	42	0.8610%
9	2.028%	26	1.149%	43	0.8503%
10	1.918%	27	1.124%	44	0.8385%
11	1.836%	28	1.100%	45	0.8274%
12	1.761%	29	1.075%	46	0.8163%
13	1.689%	30	1.054%	47	0.8038%
14	1.622%	31	1.029%	48	0.7939%
15	1.563%	32	1.011%	49	0.7839%
16	1.504%	33	0.9929%	50	0.7745%

After the l rounds of optimization, we obtain a $(m + 2l)$-dimensional embedding vector for each vertex:

$$\mathbf{P}_k = (\mathbf{q}_k, \mathbf{s}_k, \mathbf{t}_k),$$

where $\mathbf{q}_k \in \mathbb{R}^m$ is the Euclidean components, and

$$\mathbf{s}_k = (s_1^{(k)}, s_2^{(k)}, \ldots, s_l^{(k)}), \mathbf{t}_k = (t_1^{(k)}, t_2^{(k)}, \ldots, t_l^{(k)}) \in \mathbb{R}^l$$

are the non-Euclidean components. The embedding distance between two vectors is computed as

$$f(\mathbf{P}_i, \mathbf{P}_j) = \|\mathbf{q}_i - \mathbf{q}_j\| - \|\mathbf{s}_i - \mathbf{s}_j\|^2 + \|\mathbf{t}_i - \mathbf{t}_j\|^2. \quad (11)$$

Fig. 4 also plots the mean relative error of this embedding approach for the Armadillo model, using $m = 8$ for the initial Euclidean embedding and an increase value of l for the subsequent non-Euclidean refinement. Tab. 2 provides the mean relative error values for different values of l. We can see that the cascaded refinement effectively reduces the approximation error with increasing dimensionality of the embedding vectors. Fig. 5 further plots the histogram of the relative error (defined in Eq. (4)) for each type of geodesic distance (NN, NS, SS) on the Armadillo model using our method with $m = 8$ and $l = 46$. We can see that

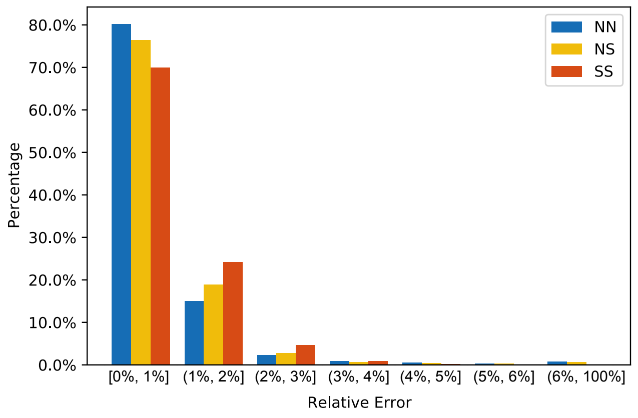

Figure 5. Distribution of the relative error ε_{ij} (defined in Eq. (4)) of geodesic distance for all vertex pairs on the Armadillo mesh in Fig. 4, using our method with $m = 8$ and $l = 46$.

the the distance for the majority of point pairs has a relative error smaller than 2%, with a high concentration of errors smaller than 1%. This indicates a good accuracy using our method.

4. Results

We implement our algorithm in C++ and use OpenMP for parallelization. All experiments are run on a computer with an Intel Xeon E5-4657L v2 at 2.40GHz and 256 GB of RAM. To perform the embedding and evaluate the approximation error, we compute the exact geodesic distance d_{ij} using the VTP method [22]. Our optimization procedure requires $O(|\mathcal{V}_S|^2)$ storage space for the ground-truth geodesic distances, where $\mathcal{V}_S$ is the set of saddle vertices. After the optimization, these ground-truth distances can be discarded and we only need to store the embeddings using $O(|\mathcal{V}_S|)$ space.

In all our experiments, we set the dimensionality of the initial Euclidean embedding to $m = 8$, because a larger m provides little reduction of the approximation error. This is also the recommended setting of the method in [20]. For the cascaded refinement, more rounds of optimization (10) can reduce the approximation error but also increase the computational time. We set $l = 46$ in our implementation to achieve a balance between speed and accuracy. The parameter K_S also influences the approximation accuracy, the query time, and the storage. In general, a larger K_S results in better accuracy. This is because it increases the number of candidate paths between two non-saddle vertices via saddle vertices, which helps to improve the accuracy of NN-distances. On the other hand, a larger K_S leads to longer computational time and higher storage requirement. We set $K_S = 20$ in all experiments.

In the following, we compare the GDQ performance between our method and state-of-the-art methods, including

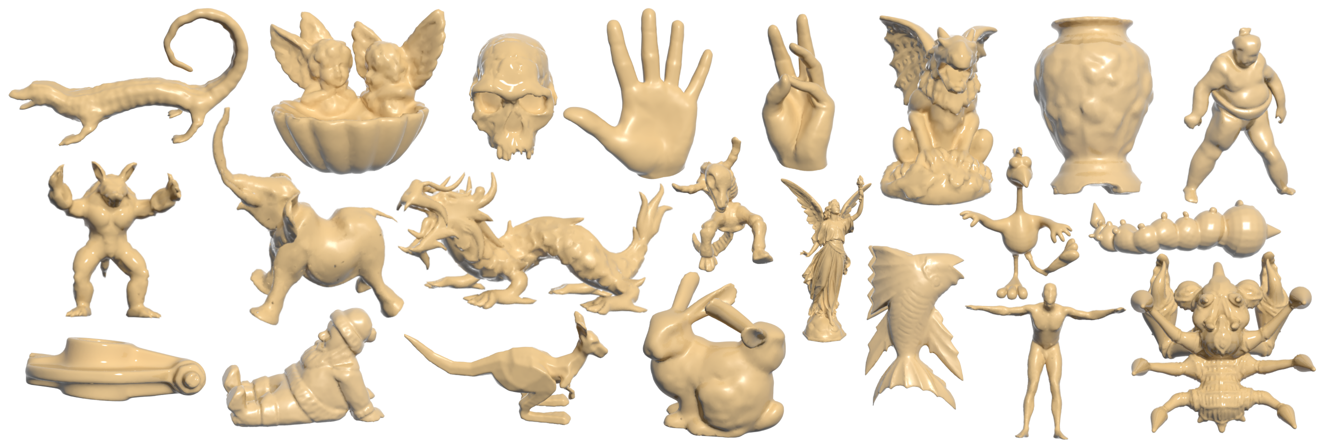
Figure 6. We test and compare our method with other methods on a variety of meshes as shown in the this figure. See Table 3 for the results.

Table 3. Comparison between different methods in terms of storage, mean relative error (ε), and the average computational time for for querying geodesic distance between a random pair of vertices (T_q). $|\mathcal{V}|$ and $|\mathcal{V}_S|$ represent the total number of vertices and the number of saddle vertices, respectively. T_{VTP} and T_{op} are the time for computing ground-truth distance and performing embedding in our method, respectively

Model ($\|V\|$, $\|V_S\|$)	SVG			EMM-1			EMM-2			GE		
	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_q (ms)	ε
Sumotori (9842, 5140)	10.80	1.836	0.3540%	0.3004	0.0002	4.723%	0.3004	0.0002	4.535%	13.71	0.0189	0.7555%
Kangaroo (9861, 5889)	9.46	1.684	0.4099%	0.3009	0.0002	4.533%	0.3009	0.0002	3.897%	12.19	0.0130	0.6960%
Click (9857, 5219)	10.49	1.672	0.3845%	0.3008	0.0002	5.811%	0.3008	0.0002	5.057%	14.14	0.0193	0.6965%
Aligator (9792, 5738)	9.49	1.646	0.3993%	0.2988	0.0002	3.549%	0.2988	0.0002	3.114%	12.11	0.0140	0.5887%
Elephant (9979, 5401)	10.37	2.052	0.5367%	0.3045	0.0002	4.820%	0.3045	0.0002	4.482%	13.15	0.01717	0.8204%
Human (8559, 4757)	8.75	1.832	0.4651%	0.2612	0.0001	4.798%	0.2612	0.0002	4.270%	11.24	0.0220	0.5905%
Santa (20000, 11015)	21.28	5.116	0.2747%	0.6104	0.0001	5.473%	0.6104	0.0003	5.179%	26.85	0.0224	0.7557%
Hand1 (10022, 5601)	10.39	2.307	0.3187%	0.3058	0.0002	6.682%	0.3058	0.0002	5.798%	14.32	0.0189	0.7565%
Hand2 (10022, 5636)	10.42	2.287	0.2949%	0.3058	0.0002	5.627%	0.3058	0.0002	5.188%	14.71	0.0229	0.7725%
Weedle (9927, 4495)	12.73	2.143	0.2831%	0.3029	0.0002	4.55%	0.3029	0.0002	4.52%	14.96	0.0239	0.667%
Gargoyle (6002, 3264)	5.91	1.277	0.4335%	0.1832	0.0001	5.604%	0.1832	0.0001	5.287%	7.365	0.0147	1.098%
Dragon (5250, 2841)	4.90	0.9879	0.5513%	0.1602	0.0001	3.976%	0.1602	0.0001	3.737%	5.964	0.0176	0.9902%
Homohabilis (5002, 2734)	4.93	1.002	0.4276%	0.1526	0.0001	6.382%	0.1526	0.0001	6.224%	6.133	0.0153	0.8201%
Vase (10002, 5857)	10.04	2.084	0.3953%	0.3052	0.0002	6.115%	0.3052	0.0002	5.84%	12.76	0.0130	06767%
Angels (7527, 3787)	8.47	2.708	0.3159%	0.2297	0.0001	5.04%	0.2297	0.0002	4.786%	10.34	0.0191	0.8233%
Rockerarm (9423, 4340)	12.06	1.585	0.3565%	0.2876	0.0002	7.162%	0.2876	0.0002	6.847%	18.89	0.0191	0.8233%
TwoHeadedBunny (18111, 9503)	10.74	2.001	0.571%	0.2924	0.0001	5.759%	0.2924	0.0002	5.576%	14.71	0.0237	1.091%
Thundercrab (9584, 4975)	19.63	3.902	0.429%	0.5527	0.0001	4.804%	0.5527	0.0003	4.586%	24.54	0.0176	0.8927%
Armadillo (1988, 973)	2.090	0.4019	0.3457%	0.0607	0.0001	4.602%	0.0607	0.0001	4.525%	2.441	0.0199	0.7106%
Lucy (11906, 6146)	12.64	2.485	0.4195%	0.3633	0.0001	4.868%	0.3633	0.0002	4.57%	15.38	0.0154	0.823%

the SVG method [34] (with $K = 40$), and the method from [20] which we denote as EMM. EMM first samples a user-specified number of points from the mesh model, and applies metric multidimensional scaling (MMDS) [8] to perform embedding in $\mathbb{R}^d$. To complete the embedding for the remaining vertices, they interpolate a smooth mesh

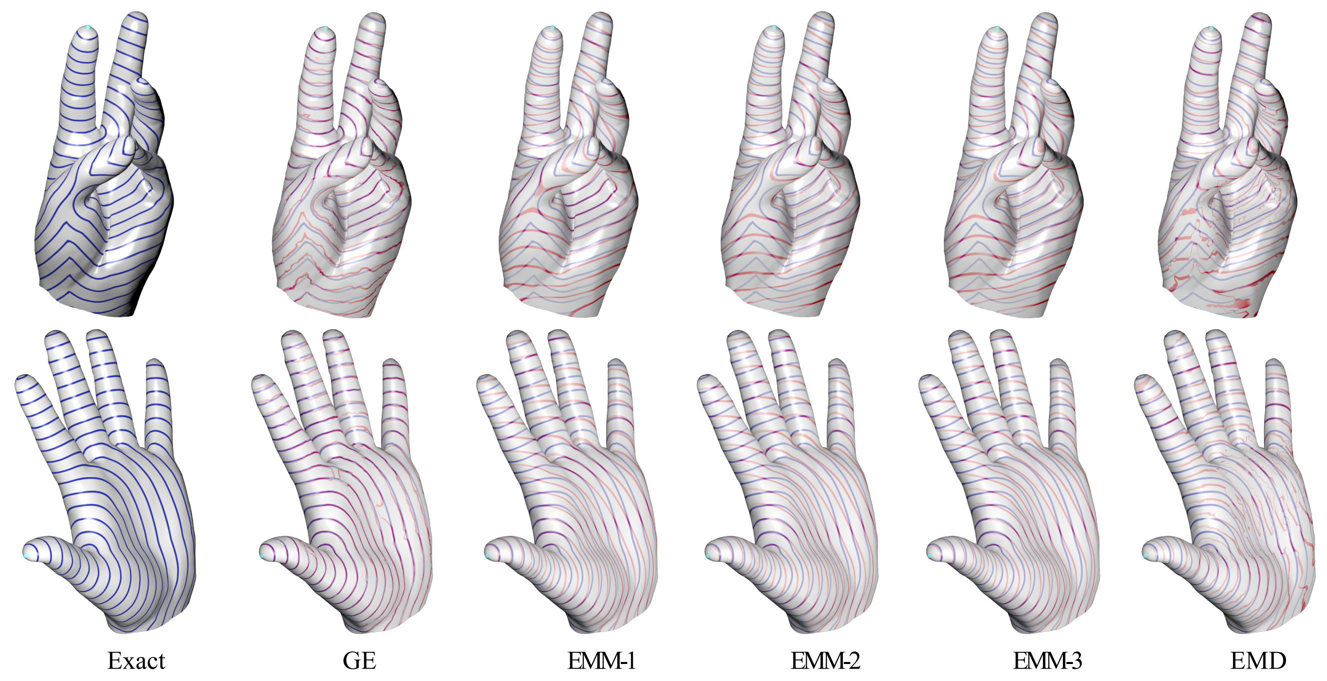

Figure 7. Comparison between the iso-contours of geodesic distance from the same vertex, computed using different methods. The first column shows the ground-truth iso-contours computed using VTP [22]. To visually compare the quality, we overlay the iso-contours from each method with the ground truth. The mean relative errors are: (top row) GE 0.9336%, EMM-1 5.8782%, EMM-2 4.990%, EMM-3 5.0134%, and EMD 26.7218%; (bottom row) GE 0.6414%, EMM-1 3.7507%, EMM-2 3.8316%, EMM-3 3.8322%, and EMD 11.4222%.

in the d-dimensional space by solving a Poisson equation. Following their paper, we set the number of sample points to 1000, and the embedding dimension $d = 8$ by default.

We compare the methods using the models shown in Fig. 6, and the results are shown in Tab. 3. For each method, we evaluate its mean relative error (as defined in Eq. (3)), average distance query time for a pair of vertices, and storage consumption. For our method, we also provide the computational time for the ground-truth distance and for the embedding. For each model, we evaluate the mean relative error and the average distance query time on the same set of $10K$ vertex pairs, and we use the VTP method [22] to compute the ground-truth geodesic distance for evaluating the mean relative error.

Tab. 3 shows that our method can evaluate the geodesic distance between an arbitrary pair of vertices in about 0.02ms, with a low storage consumption. Compared with our method, the SVG method requires less storage space for the graph information of GDQ and can achieve better accuracy, but its average distance query time is about two orders of magnitude longer than our method. We test two versions of the EMM method: EMM-1 which samples 500 vertices, and EMM-2 which samples the same number of vertices ($|\mathcal{V}_{\mathcal{S}}|$) as our method, with the embedding dimensionality set to 8 in both cases. We can see that EMM-2 can achieve slightly better results than EMM-1. The mean relative errors of the EMM methods are about 5%, while the mean relative error of our method is less than 1%.

In Fig. 7, we also show the iso-contours of exact geodesic distance and the distance functions computed by different methods. Besides the two variants EMM-1 and EMM-2 mentioned above for the EMM method, we also include another variant EMM-3 with the number of sample points being the same as the number of saddle points used in our method, and with the embedding dimension increased to 100. Also included in Fig. 7 for comparison is the method from [25] based on the earth mover's distance (EMD). We compute the distance using the code provided by the authors of [25] and with a basis size 100. Figure 7 shows that our results have low approximation errors and our iso-contours largely follow those of the exact contours, but our distance functions are less smooth than those from the other methods. The other methods achieve smoother iso-contours, but at the cost of higher approximation errors. Therefore, our method complements existing approaches and can be desirable in applications that require higher accuracy.

5. Conclusions & Future Work

We developed a new method to embed triangle meshes in a high-dimensional space so that the Euclidean distances

Table 4. Comparison between representative methods in terms of accuracy, the need for pre-computation (PC), space complexity (SC), and time complexities (TC) for SSAD and GDQ. Here K is the maximal degree of SVG; m is the number of samples on the input mesh of GTU; ε' is the accuracy parameter of DGG; Acronyms are PC (pre-computation), SC (space complexity) and TC (time complexity).

Method	Accuracy	PC	SC	TC (SSAD)	TC (GDQ)
MMP [18]	Exact	No	$O(n^2)$	$O(n^2 \log n)$	$O(n^2 \log n)$
CH [5]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
ICH [30]	Exact	No	$O(n)$	$O(n^2 \log n)$	$O(n^2 \log n)$
FWP [32]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
VTP [22]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
FMM [11]	Approx	No	$O(n)$	$O(n \log n)$	$O(n \log n)$
DGPC [17]	Approx	No	$O(n)$	$O(n \log n)$	$O(n \log n)$
HM [9]	Approx	Yes	$O(n)$	$O(n)$	$O(n)$
PSHM [28]	Approx	No	$O(n)$	$O(n)$	$O(n)$
SVG [34]	Approx	Yes	$O(Kn)$	$O(Kn \log n)$	$O(Kn \log n)$
DGG [1]	Approx	Yes	$O(\frac{n}{\sqrt{\varepsilon'}})$	$O(n)$	$O(n)$
GTU [31]	Approx	Yes	$O(m^2 + n)$	$O(n)$	$O(1)$
GE (Ours)	Approx	Yes	$O(n)$	$O(n)$	$O(1)$

can approximate the geodesic distances well. Our method is based on two novel ideas. First, instead of taking all vertices as variables, we embed only the saddle vertices, which greatly reduces the problem complexity. We then compute a local embedding for each non-saddle vertex. Second, to solve the large residual issue, we propose a cascaded optimization method that can effectively reduce the residual in a step-by-step manner. With both the global and local embeddings, we can quickly compute the geodesic distance between any two vertices in near constant time. Computational results show that our method is more accurate than the other geodesic distance query methods. Tab. 4 compares the characteristics of our method with existing representative methods for SSAD and GDQ.

Our method also has some limitations. First, although the distance computed by our method is quite close to exact geodesic distance, its level sets are not smooth and there is no guarantee that it is a metric. How to add these constraints into the GE framework is an interesting research topic. Second, current numerical optimization algorithm to solve this model is still computationally expensive. Third, we use saddle vertices as embedding elements. However, some meshes may have very few saddle vertices such that the result is not very well. Discrete geodesic graph [29] (DGG) which uses any kind of vertices as relays to approximate long geodesic paths may be used instead. A potential future work is to develop a lightweight numerical solver with improved efficiency for embedding.

Acknowledgments

The authors are supported by National Natural Science Foundation of China (No. 61672481), Youth Innovation Promotion Association CAS (No. 2018495), and Singapore MOE RG26/17.

References

[1] Y. Y. Adikusuma, Z. Fang, and Y. He. Fast construction of discrete geodesic graphs. *ACM Trans. Graph.*, 39(2):14:1–14:14, 2020. 1, 2, 10

[2] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org. 6

[3] A. G. Belyaev and P. Fayolle. On variational and pde-based distance function approximations. *Comput. Graph. Forum*, 34(8):104–118, 2015. 2

[4] S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, and M. Pauly. Shape-Up: Shaping discrete geometry with projections. *Computer Graphics Forum*, 31(5):1657–1667, 2012. 5

[5] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, SCG '90, pages 360–369, New York, NY, USA, 1990. ACM. 2, 10

[6] A. Clements and H. Zhang. Robust 3d shape correspondence in the spectral domain. In *Proc. of Shape Modeling International*, pages 118–129, 2006. 1

[7] R. R. Coifman, S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences*, 102(21):7426–7431, 2005. 2

[8] T. F. Cox and M. Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition, 2000. 2, 8

[9] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.*, 32(5):152:1–152:11, 2013. 1, 2, 10

[10] F. Fouss, A. Pirotte, J. Renders, and M. Saerens. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):355–369, 2007. 2

[11] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences*, 95(15):8431–8435, 1998. 2, 10

[12] B. Lévy and N. Bonneel. Variational anisotropic surface meshing with voronoi parallel linear enumeration. In *Proceedings of the 21st international meshing roundtable*, pages 349–366. Springer, 2013. 2

[13] Y. Lipman, R. M. Rustamov, and T. A. Funkhouser. Biharmonic distance. *ACM Trans. Graph.*, 29(3):27:1–27:11, 2010. 2

[14] T. Liu, S. Bouaziz, and L. Kavan. Quasi-newton methods for real-time simulation of hyperelastic materials. *ACM Trans. Graph.*, 36(3):23:1–23:16, 2017. 5

[15] Y.-J. Liu. Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures. *Computer-Aided Design*, 45(3):695–704, 2013. 2

[16] A. Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39, 1992. 1

[17] E. L. Melvær and M. Reimers. Geodesic polar coordinates on polygonal meshes. *Computer Graphics Forum*, 31(8):2423–2435, 2012. 2, 10

[18] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987. 1, 2, 3, 10

[19] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer-Verlag New York, 2nd edition, 2006. 6

[20] D. Panozzo, I. Baran, O. Diamanti, and O. Sorkine-Hornung. Weighted averages on surfaces. *ACM Trans. Graph.*, 32(4):60:1–60:12, 2013. 1, 2, 7, 8

[21] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Trans. Graph.*, 35(4):125:1–125:13, 2016. 2

[22] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Trans. Graph.*, 35(4):125:1–125:13, 2016. 2, 7, 9, 10

[23] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Full and partial symmetries of non-rigid shapes. *Int. J. Comput. Vis.*, 89(1):18–39, 2010. 1

[24] J. A. Sethian. Fast marching methods. *SIAM Review*, 41(2):199–235, 1999. 2

[25] J. Solomon, R. M. Rustamov, L. J. Guibas, and A. Butscher. Earth mover's distances on discrete surfaces. *ACM Trans. Graph.*, 33(4):67:1–67:12, 2014. 1, 2, 9

[26] O. Sorkine and D. Cohen-Or. Least-squares meshes. In *2004 International Conference on Shape Modeling and Applications (SMI 2004), 7-9 June 2004, Genova, Italy*, pages 191–199. IEEE Computer Society, 2004. 2

[27] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005. 2, 5

[28] J. Tao, J. Zhang, B. Deng, Z. Fang, Y. Peng, and Y. He. Parallel and scalable heat methods for geodesic distance computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019. 2, 10

[29] X. Wang, Z. Fang, J. Wu, S. Xin, and Y. He. Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces. *Comput. Aided Geom. Des.*, 52:262–284, 2017. 2, 10

[30] S.-Q. Xin and G.-J. Wang. Improving Chen and Han's algorithm on the discrete geodesic problem. *ACM Trans. Graph.*, 28(4):104:1–104:8, 2009. 2, 10

[31] S.-Q. Xin, X. Ying, and Y. He. Constant-time all-pairs geodesic distance query on triangle meshes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 31–38, 2012. 1, 2, 10

[32] C.-X. Xu, T. Y. Wang, Y.-J. Liu, L. Liu, and Y. He. Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics*, 21(7):822–834, 2015. 2, 10

[33] K. Xu, H. Zhang, A. Tagliasacchi, L. Liu, G. Li, M. Meng, and Y. Xiong. Partial intrinsic reflectional symmetry of 3d shapes. *ACM Trans. Graph.*, 28(5):138, 2009. 1

[34] X. Ying, X. Wang, and Y. He. Saddle vertex graph (SVG): A novel solution to the discrete geodesic problem. *ACM Transactions on Graphics*, 32(6):170:1–12, 2013. 1, 2, 3, 4, 8, 10

[35] X. Ying, S.-Q. Xin, and Y. He. Parallel Chen-Han (PCH) algorithm for discrete geodesics. *ACM Transactions on Graphics*, 33(1):9:1–9:11, 2014. 2

[36] Z. Zhong, W. Wang, B. Levy, J. Hua, and X. Guo. Computing a high-dimensional euclidean embedding from an arbitrary smooth riemannian metric. *ACM Transactions on Graphics*, 37(4CD):1–16, 2018. 3

[37] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multidimensional scaling. *IEEE Trans. Vis. Comput. Graph.*, 8(2):198–207, 2002. 1