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

Qianwei Xia, Juyong Zhang[†], *Member, IEEE,* Zheng Fang, Jin Li, Mingyue Zhang,
Bailin Deng, *Member, IEEE,* Ying He, *Member, IEEE*

Abstract—In this paper, we develop a novel method for fast geodesic distance queries. 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 the large number of variables and the fact that the embedding problem is highly nonlinear. We overcome the challenges with 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 the purely 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 precomputation data, our approach can determine the geodesic distance between any two vertices in near-constant time. Computational testing results show that our method is more desirable than previous geodesic distance queries methods.

Index Terms—Geodesic Distance Queries, Saddle Vertices, High Dimension Embedding, Cascaded Optimization.

✦

1 INTRODUCTION

T HE discrete geodesic problem has been investigated extensively since the seminal paper published by Mitchell et al. [2]. Many elegant and efficient algorithms are known for computing the single-source-all-destinations (SSAD) geodesic distances in discrete domains. With the significant progress of precomputation methods, such as the heat method [3] and the graph-based method [1], [4], 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 queries (GDQ), is a common scenario in shape analysis, such as intrinsic symmetry detection [5], [6], surface parameterization [7], and shape correspondence [8]. Some applications, such as the shape matching algorithm in [9], require recomputation of the geodesic distances between arbitrary points in each iteration, which would benefit from an efficient method for GDQ. Unlike SSAD, only a handful of methods have been designed for answering GDQ, such as geodesic triangle unfolding (GTU) [10], the Euclidean embedding metric (EEM) [11], and the pointwise distance metric (PDM) [12]. These methods, however, either have high space complexity or large error potential that limits their usage in accuracy-critical applications. For example, the

- *Q. Xia, J. Zhang and M. Zhang are with the School of Mathematical Sciences, University of Science and Technology of China.*
- *J. Li is with the School of Computer Science and Technology, University of Science and Technology of China.*
- *Z. Fang and Y. He are with the School of Computer Science and Engineering, Nanyang Technological University.*
- *B. Deng is with the School of Computer Science and Informatics, Cardiff University.*

[†]*Corresponding author. Email:* `juyong@ustc.edu.cn`.

naive way is to compute and store all pointwise distances. However, such an approach will consume large storage when saving the geodesic distance between all vertex pairs.

In this paper, we propose a novel method for fast geodesic distance queries. The key idea is to lift the model into a high-dimensional Euclidean space $\mathbb{R}^d$, so that the distance between 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) [13]. However, directly solving this optimization problem is challenging due to the potentially huge number of objective function terms and the high nonlinearity of the objective function. In addition, a purely Euclidean embedding often does not approximate the pairwise geodesic distances 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 [1], saddle vertices are crucial in solving the discrete geodesic problem, since they serve as the backbone of saddle vertex graphs (SVG). Based on the embedding of saddle vertices and the resulting fast evaluation of geodesic distances 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 can achieve near-constant time query of geodesic distance with high accuracy only with very small storage consumption.

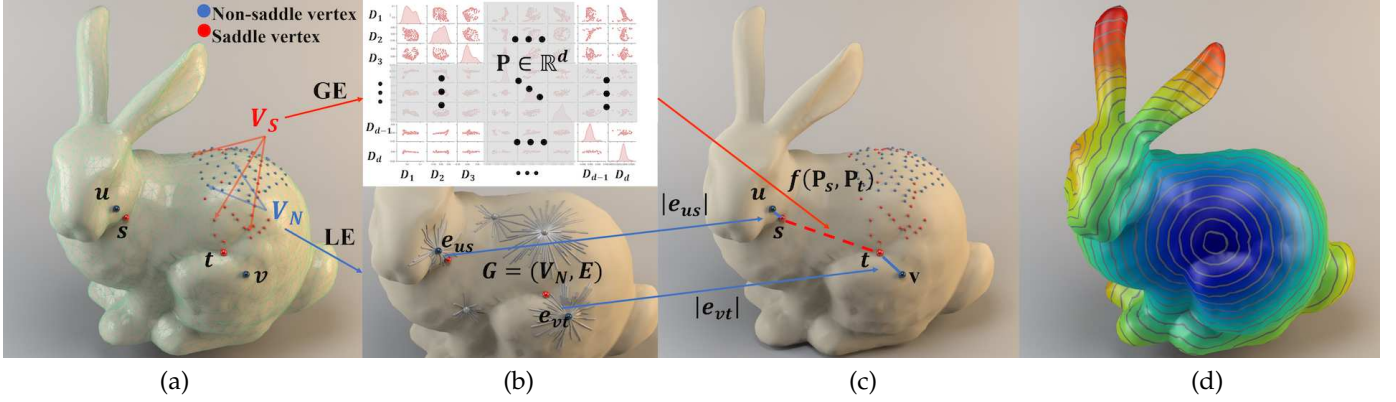1. A vertex is called a saddle if it has negative Gaussian curvature.

Fig. 1. Our method performs distance queries between an arbitrary pair of vertices in near constant time. (a) Distance queries 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) [1]. s and t are the local neighbors of u and v respectively. The bidirectional SVG edge $e_{us}, e_{vt} \in E$ stores 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 a 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 $\varepsilon = 0.7367\%$.

2 RELATED WORKS

A large body of literature exists on the discrete geodesic problem. Most algorithms focus on computing SSAD on triangular meshes. Representative algorithms are wavefront propagation methods [2], [14], [15], [16], [17], [18], [19], [20], [21], partial differential equation (PDE) methods [3], [22], [23], [24], [25], [26], and graph-based methods [1], [4], [27]. As mentioned above, applying these SSAD methods for GDQ is too expensive. In the following, we review only the related works on GDQ.

The GTU method [10] 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 flattened distance on $\mathbb{R}^2$, GTU can answer the GDQ using an unfolding technique. GTU is conceptually simple and elegant, but it is sensitive to geometric features and only works for small-scale meshes due to its quadratic space complexity for precomputation.

Xin et al. [28] proposed a fast query method for geodesic distance. They first construct a proximity graph using sample points from the surface. When querying the distance between two points, they augment the proximity graph with the source and target points for the query, then compute their shortest distance on the augmented graph. Our method is also based on augmenting a graph during the query, but we use an SVG for the underlying graph instead.

Panazzo et al. [11] proposed a Euclidean embedding metric that samples a representative subset S of the mesh vertices and computes pairwise approximate geodesic distances for these samples. They then embed S in a high-dimensional Euclidean space using metric multidimensional scaling (MMDS) [29]. After that, they embed the remaining vertices using least-square meshes [30]. EMM generates smooth distances quickly. However, their results often have large approximation errors, and their iso-distance contours

are not close to geodesic distances.

A few spectral distance techniques have been published, such as diffusion distances [31], commute-time distances [32] and biharmonic distances [33]. These distances are smooth, shape aware, and have a closed-form solution. However, they are not isotropic, so their level sets are not evenly spaced.

Using an 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 [12]. Taking all the eigenvectors of the Laplacian matrix, the resulting distance is the 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 errors.

The idea of this high-dimension embedding is not new. Given an initial polygonal mesh, a user-specified desired number of vertices, and a parameter that specifies the desired amount of anisotropy, Levy and Bonnee [34] generated a curvature-adapted anisotropic mesh. Their 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. [35] 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 triangular 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
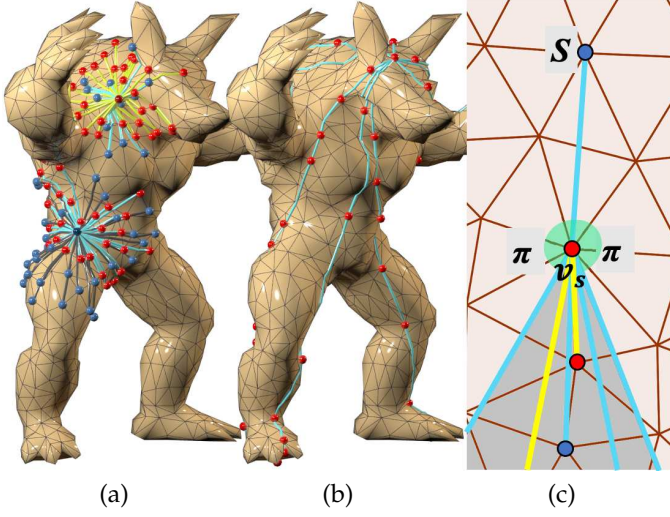
Fig. 2. Illustrations of SVG construction and the properties of saddle vertex. (a) Two subsets of G generated from two source vertices. The red and blue spheres indicate the saddle and non-saddle vertices, respectively. The upper subset contains E_{SS} in yellow and the cyan rendered E_{NS}. The lower subset also includes E_{NN} in black. 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.

the function f is given, then the embedding vectors $\{\mathbf{P}_k\}$ can be determined by solving an optimization problem

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

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 an impractical 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 [1]. 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 [1] for geodesic distance queries 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 queries. Afterward, 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-shaped region of each saddle vertex, which is shown as the shadow region in Figure 2. Mitchell et al. [2] 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 it passes through no saddle vertex and *indirect* otherwise. The longer a path is, the more likely it is to be 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 that contains only direct paths, with saddle vertices acting as relays in the graph. The SVG proposed in [1] 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 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. [1] 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 [1] that such an approximate SVG still achieves high accuracy for geodesic distance computation. The approximate SVG construction has a worst-case $O(nK^2 \log K)$ and empirical $O(nK^{1.5} \log K)$ time complexity where $n = |\mathcal{V}|$ is the number of vertices, which is linear in mesh size when K is fixed.

We adopt the method from [1] to benefit from its low complexity, with a slight modification to suit our needs. 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 desire a sufficient number 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 the process of 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 Queries

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

- Subgraph $G_{SS} = (\mathcal{V}_S, E_{SS})$ forms the skeleton of the whole SVG. Each edge of E_{SS} is a direct geodesic path connecting two saddle vertices.
- Subgraph $G_{NS} = (\mathcal{V}_S \cup \mathcal{V}_N, E_{NS})$ connects non-saddle vertices to saddle vertices.
- Subgraph $G_{NN} = (\mathcal{V}_N, E_{NN})$ 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
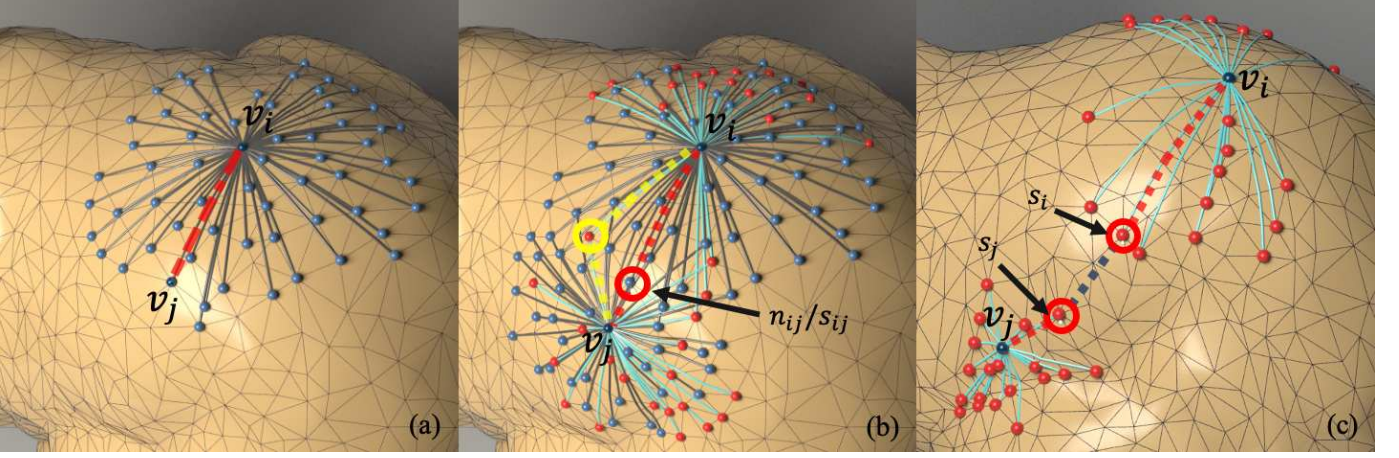
Fig. 3. Solving NN-distance for (v_i, v_j) using SVG. (a) The "direct" case. The red dotted curve indicates the shortest path, which is a direct edge in G_{NN}. (b) v_i, v_j are "near" each other. n_{ij} and s_{ij} belong to the common set of the saddle and non-saddle neighbors of v_i and v_j. The set 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.

according to the tier it belongs to. We label the distance between two saddle vertices as the *SS-distance*, the distance between two non-saddle vertices as the *NN-distance*, and the distance between a non-saddle vertex and a saddle vertex as the *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 $f(\mathbf{P}_i, \mathbf{P}_j)$. Afterward, the geodesic distance between two saddle vertices is 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 a 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 geodesic distance between v_i and v_j using Dijkstra's algorithm on G, with 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
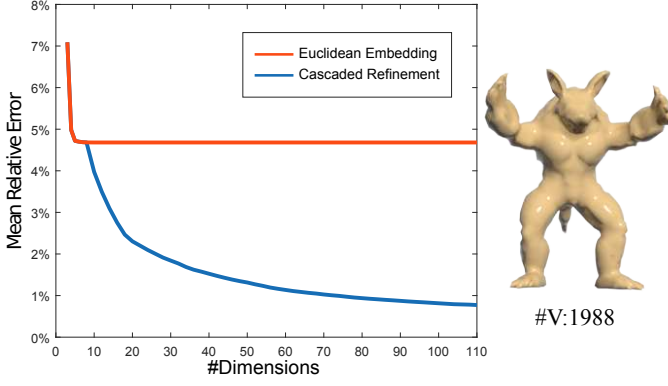
#V:1988

Fig. 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.

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%

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 [36]:

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

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 [37] for to efficiently solve (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 [15] and compute the mean relative error

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

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:

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

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. Afterward, 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 nonzero 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 that can attain both positive and negative values to approximate the residual; we 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 nonlinear least squares problem:

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

$$\qquad (5)$$

We adopt the L-BFGS algorithm [38] in the Ceres library [39] to solve this problem. Then the distance function for the embedding vectors of v_i and v_j becomes

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

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

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

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 reduce

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%

Fig. 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 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 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 solving an optimization problem similar to (5):

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

Then the embedding distance function becomes

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

Following the same argument used in the previous paragraph, we can see that the introduction of the components $\left(s_2^{(k)}, t_2^{(k)} \right)$ 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

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

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

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

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 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 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.

Fig. 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.

4 RESULTS

We implemented our algorithm in C++ and used OpenMP for parallelization. All experiments were run on a workstation with an AMD Epyc 7402P at 2.8GHz 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 [21]. Our optimization procedure requires $O(|\mathcal{V}_\mathcal{S}|^2)$ storage space for the ground-truth geodesic distances, where $\mathcal{V}_\mathcal{S}$ 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 $O(|\mathcal{V}_\mathcal{S}|)$ 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 [11]. 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, query time, and storage. In general, a larger K_S results in better accuracy 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. Unfortunately, a larger K_S leads to longer computational time and higher storage requirements. 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 the SVG method [1] (with $K = 40$), and the method from [11], which we denote as EMM. EMM first samples a user-specified number of points from the mesh model, then applies metric multidimensional scaling (MMDS) [29] to perform embedding in $\mathbb{R}^d$. To complete the embedding for the remaining vertices, EMM interpolates a smooth mesh in the d-dimensional space by solving a Poisson equation. Following that paper, we set the number of sample points to 1000 and the embedding dimension to $d = 8$ by default.

We compare the methods using the models shown in Fig. 6, and the results are shown in Tab. 3. We evaluate each method's mean relative error (as defined in Eq. (3)), average distance queries 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 queries time on the same set of $10K$ vertex pairs, and we use the VTP method [21] 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 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 queries 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 the embedding dimension increased to 100. Also included in Fig. 7 for comparison is the method from [12] based on the earth mover's distance (EMD). We compute the distance using the code provided by the authors of [12] and use 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 triangular meshes in a high-dimensional space so that the Euclidean distances

TABLE 3
Comparison between different methods in terms of storage, mean relative error (ε), and average computational time 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_{SVG}, T_{VTP} and T_{op} are the time for constructing SVG, computing ground-truth distance, and performing embedding in our method, respectively

| Model | SVG | | | EMM-1 | | | EMM-2 | | | GE | | | | | |
($|V|$, $|V_S|$)	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_q (ms)	ε	Storage (MB)	T_{SVG} (s)	T_q (ms)	ε	T_{VTP} (s)	T_{op} (s)
Sumotori (9842, 5140)	10.80	1.79	**0.354%**	**0.3004**	**0.0002**	4.723%	0.3004	0.0002	4.535%	13.71	1.024	0.0106	0.7555%	61.23	181.2
Kangaroo (9861, 5889)	9.46	1.53	**0.410%**	**0.3009**	**0.0002**	4.533%	0.3009	0.0002	3.897%	12.19	0.837	0.0109	0.6960%	74.14	249.4
Click (9857, 5219)	10.49	1.50	**0.385%**	**0.3008**	**0.0002**	5.811%	0.3008	0.0002	5.057%	14.14	1.068	0.0189	0.6965%	63.74	183.8
Aligator (9792, 5738)	9.49	1.57	**0.399%**	**0.2988**	**0.0002**	3.549%	0.2988	0.0002	3.114%	12.11	0.937	0.0131	0.5887%	69.03	234.7
Elephant (9979, 5401)	10.37	1.84	**0.537%**	**0.3045**	**0.0002**	4.820%	0.3045	0.0002	4.482%	13.15	1.090	0.0108	0.8204%	63.81	212.3
Human (8559, 4757)	8.75	1.78	**0.465%**	**0.2612**	**0.0002**	4.798%	0.2612	0.0002	4.270%	11.24	0.922	0.0190	0.5905%	47.03	163.8
Santa (20000, 11015)	21.28	4.72	**0.275%**	**0.6104**	**0.0001**	5.473%	0.6104	0.0003	5.179%	26.85	1.767	0.0234	0.7557%	313.27	858.4
Hand1 (10022, 5601)	10.39	2.14	**0.319%**	**0.3058**	**0.0002**	6.682%	0.3058	0.0002	5.798%	14.32	1.032	0.0179	0.7565%	82.48	214.1
Hand2 (10022, 5636)	10.42	2.12	**0.295%**	**0.3058**	**0.0002**	5.627%	0.3058	0.0002	5.188%	14.71	0.921	0.0177	0.7725%	91.69	214.2
Weedle (9927, 4495)	12.73	2.18	**0.283%**	**0.3029**	**0.0002**	4.55%	0.3029	0.0002	4.52%	14.96	1.054	0.0214	0.667%	54.37	140.6
Gargoyle (6002, 3264)	5.91	1.30	**0.434%**	**0.1832**	**0.0001**	5.604%	0.1832	0.0001	5.287%	7.365	0.595	0.0124	1.098%	18.18	69.5
Dragon (5250, 2841)	4.90	0.82	**0.551%**	**0.1602**	**0.0001**	3.976%	0.1602	0.0001	3.737%	5.964	0.538	0.0162	0.9902%	13.56	56.9
Homohabilis (5002, 2734)	4.93	0.95	**0.428%**	**0.1526**	**0.0001**	6.382%	0.1526	0.0001	6.224%	6.133	0.522	0.0158	0.8201%	13.19	50.8
Vase (10002, 5857)	10.04	2.04	**0.395%**	**0.3052**	**0.0002**	6.115%	0.3052	0.0002	5.84%	12.76	0.949	0.0105	0.6767%	84.17	213.6
Angels (7527, 3787)	8.47	1.89	**0.316%**	**0.2297**	**0.0001**	5.04%	0.2297	0.0002	4.786%	10.34	0.800	0.0123	0.8233%	31.35	100.1
Rockerarm (9423, 4340)	12.06	1.42	**0.357%**	**0.2876**	0.0002	7.162%	0.2876	**0.0001**	6.847%	18.89	1.086	0.0185	0.8233%	76.67	149.7
TwoHeadedBunny (18111, 9503)	10.74	1.92	**0.571%**	**0.2924**	**0.0001**	5.759%	0.2924	0.0002	5.576%	14.71	1.813	0.0228	1.091%	283.51	667.0
Thundercrab (9584, 4975)	19.63	3.54	**0.429%**	**0.5527**	**0.0001**	4.804%	0.5527	0.0002	4.586%	24.54	1.007	0.0176	0.8927%	47.93	181.3
Armadillo (1988, 973)	2.090	0.50	**0.346%**	**0.0607**	**0.0001**	4.602%	0.0607	0.0001	4.525%	2.441	0.259	0.0194	0.7106%	1.75	2.9
Lucy (11906, 6146)	12.64	2.23	**0.420%**	**0.3633**	**0.0001**	4.868%	0.3633	0.0002	4.57%	15.38	1.294	0.0140	0.823%	83.08	272.0

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. For both the global and local embeddings, we can quickly compute the geodesic distance between any two vertices in near-constant time. Our computational results show that this new method is more accurate than previous geodesic distance queries methods. Tab. 4 compares the characteristics of our method with representative methods for SSAD and GDQ.

Our method also has limitations. First, although the distance computed by our method is quite close to the 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, the numerical optimization algorithm currently used to solve this model is still computationally expensive for large meshes. Third, we use saddle vertices as embedding elements. However, some meshes may have very few saddle vertices, yielding a poor result. Discrete geodesic graph
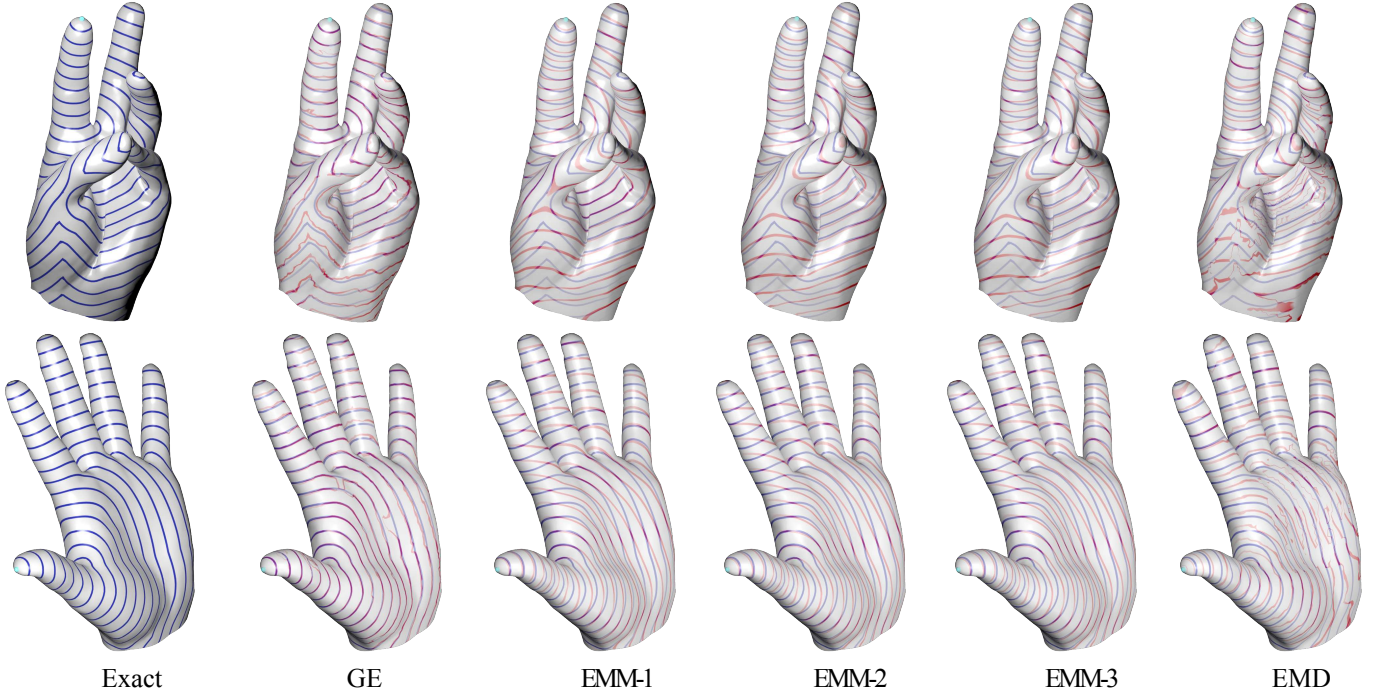
Fig. 7. Comparison between the iso-contours of geodesic distance from the same vertex, computed using different methods. The first column shows the ground-truth iso-contours computed using VTP [21]. To visually compare the quality, we overlay the iso-contours from each method with the ground truth. The 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%.

TABLE 4
Comparison between representative methods in terms of accuracy, need for precomputation (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 (precomputation), SC (space complexity) and TC (time complexity).

Method	Accuracy	PC	SC	TC (SSAD)	TC (GDQ)
MMP [2]	Exact	No	$O(n^2)$	$O(n^2 \log n)$	$O(n^2 \log n)$
CH [14]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
ICH [16]	Exact	No	$O(n)$	$O(n^2 \log n)$	$O(n^2 \log n)$
FWP [19]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
VTP [21]	Exact	No	$O(n)$	$O(n^2)$	$O(n^2)$
FMM [23]	Approx	No	$O(n)$	$O(n \log n)$	$O(n \log n)$
DGPC [24]	Approx	No	$O(n)$	$O(n \log n)$	$O(n \log n)$
HM [3]	Approx	Yes	$O(n)$	$O(n)$	$O(n)$
PSHM [26]	Approx	No	$O(n)$	$O(n)$	$O(n)$
SVG [1]	Approx	Yes	$O(Kn)$	$O(Kn \log n)$	$O(Kn \log n)$
DGG [4]	Approx	Yes	$O(\frac{n}{\sqrt{\varepsilon'}})$	$O(n)$	$O(n)$
GTU [10]	Approx	Yes	$O(m^2 + n)$	$O(n)$	$O(1)$
GE (Ours)	Approx	Yes	$O(n)$	$O(n)$	$O(1)$

[27] (DGG), which uses any kind of vertex as relays to approximate long geodesic paths, may be used instead. A potential future project is to develop a lightweight numerical solver with improved efficiency for embedding.

REFERENCES

[1] X. Ying, X. Wang, and Y. He, "Saddle vertex graph (SVG): A novel solution to the discrete geodesic problem," *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 170:1–12, 2013.
[2] J. S. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.
[3] K. Crane, C. Weischedel, and M. Wardetzky, "Geodesics in heat: A new approach to computing distance based on heat flow," *ACM Trans. Graph.*, vol. 32, no. 5, pp. 152:1–152:11, 2013.
[4] Y. Y. Adikusuma, Z. Fang, and Y. He, "Fast construction of discrete geodesic graphs," *ACM Trans. Graph.*, vol. 39, no. 2, pp. 14:1–14:14, 2020.
[5] 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.*, vol. 28, no. 5, p. 138, 2009.
[6] D. Raviv, A. M. Bronstein, M. M. Bronstein, and R. Kimmel, "Full and partial symmetries of non-rigid shapes," *Int. J. Comput. Vis.*, vol. 89, no. 1, pp. 18–39, 2010.
[7] G. Zigelman, R. Kimmel, and N. Kiryati, "Texture mapping using surface flattening via multidimensional scaling," *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 2, pp. 198–207, 2002.
[8] A. Clements and H. Zhang, "Robust 3d shape correspondence in the spectral domain," in *Proc. of Shape Modeling International*, 2006, pp. 118–129.
[9] A. Tevs, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel, "Isometric registration of ambiguous and partial data," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1185–1192.

[10] 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*, 2012, pp. 31–38.

[11] D. Panozzo, I. Baran, O. Diamanti, and O. Sorkine-Hornung, "Weighted averages on surfaces," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 60:1–60:12, 2013.

[12] J. Solomon, R. M. Rustamov, L. J. Guibas, and A. Butscher, "Earth mover's distances on discrete surfaces," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 67:1–67:12, 2014.

[13] A. Mead, "Review of the development of multidimensional scaling methods," *Journal of the Royal Statistical Society: Series D (The Statistician)*, vol. 41, no. 1, pp. 27–39, 1992.

[14] J. Chen and Y. Han, "Shortest paths on a polyhedron," in *Proceedings of the Sixth Annual Symposium on Computational Geometry*, ser. SCG '90. New York, NY, USA: ACM, 1990, pp. 360–369.

[15] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 553–560, 2005.

[16] S.-Q. Xin and G.-J. Wang, "Improving Chen and Han's algorithm on the discrete geodesic problem," *ACM Trans. Graph.*, vol. 28, no. 4, pp. 104:1–104:8, 2009.

[17] X. Ying, S.-Q. Xin, and Y. He, "Parallel Chen-Han (PCH) algorithm for discrete geodesics," *ACM Transactions on Graphics*, vol. 33, no. 1, pp. 9:1–9:11, 2014.

[18] Y.-J. Liu, "Exact geodesic metric in 2-manifold triangle meshes using edge-based data structures," *Computer-Aided Design*, vol. 45, no. 3, pp. 695–704, 2013.

[19] 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*, vol. 21, no. 7, pp. 822–834, 2015.

[20] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang, "Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 125:1–125:13, 2016.

[21] ——, "Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 125:1–125:13, 2016.

[22] J. A. Sethian, "Fast marching methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, 1999.

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

[24] E. L. Melvær and M. Reimers, "Geodesic polar coordinates on polygonal meshes," *Computer Graphics Forum*, vol. 31, no. 8, pp. 2423–2435, 2012.

[25] A. G. Belyaev and P. Fayolle, "On variational and pde-based distance function approximations," *Comput. Graph. Forum*, vol. 34, no. 8, pp. 104–118, 2015.

[26] 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*, vol. 43, no. 2, pp. 579–594, 2021.

[27] 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.*, vol. 52, pp. 262–284, 2017.

[28] S. Xin, W. Wang, Y. He, Y. Zhou, S. Chen, C. Tu, and Z. Shu, "Lightweight preprocessing and fast query of geodesic distance via proximity graph," *Computer-Aided Design*, vol. 102, pp. 128–138, 2018, proceeding of SPM 2018 Symposium.

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

[30] 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*. IEEE Computer Society, 2004, pp. 191–199.

[31] 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*, vol. 102, no. 21, pp. 7426–7431, 2005.

[32] 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*, vol. 19, no. 3, pp. 355–369, 2007.

[33] Y. Lipman, R. M. Rustamov, and T. A. Funkhouser, "Biharmonic distance," *ACM Trans. Graph.*, vol. 29, no. 3, pp. 27:1–27:11, 2010.

[34] B. Lévy and N. Bonneel, "Variational anisotropic surface meshing with voronoi parallel linear enumeration," in *Proceedings of the 21st international meshing roundtable*. Springer, 2013, pp. 349–366.

[35] 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*, vol. 37, no. 4CD, pp. 1–16, 2018.

[36] S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, and M. Pauly, "Shape-Up: Shaping discrete geometry with projections," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1657–1667, 2012.

[37] T. Liu, S. Bouaziz, and L. Kavan, "Quasi-newton methods for real-time simulation of hyperelastic materials," *ACM Trans. Graph.*, vol. 36, no. 3, pp. 23:1–23:16, 2017.

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

[39] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.
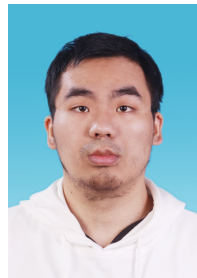
Qianwei Xia received the bachelor degree in Mathematical Sciences from University of Science and Technology of China in 2019 and is currently working toward the master degree in Mathematical Sciences from University of Science and Technology of China. His research interests include computer graphics and numerical optimization.

Juyong Zhang is an associate professor in the School of Mathematical Sciences at University of Science and Technology of China. He received the BS degree from the University of Science and Technology of China in 2006, and the PhD degree from Nanyang Technological University, Singapore. His research interests include computer graphics, computer vision, and numerical optimization. He is an associate editor of The Visual Computer.

Zheng Fang received the BS degree in computer science and technology from Tianjin University, China. He is currently working toward the PhD degree with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His current research interests include computational geometry and computer graphics.

Jin Li is currently an undergraduate student at School of Computer Science and Technology at the University of Science and Technology of China. His research interests include computer graphics and computer systems.

Mingyue Zhang received the Bachelor of Science degree in Mathematical Sciences from University of Science and Technology of China in 2020 and is currently working toward the PhD degree in Mathematical Sciences from University of Science and Technology of China. Her research interests include computer vision and numerical optimization.

Bailin Deng is a lecturer in the School of Computer Science and Informatics at Cardiff University. He received the BEng degree in computer software (2005) and the MSc degree in computer science (2008) from Tsinghua University (China), and the PhD degree in technical mathematics from Vienna University of Technology (Austria). His research interests include geometry processing, numerical optimization, computational design, and digital fabrication. He is a member of the IEEE.

Ying He is an associate professor at School of Computer Science and Engineering, Nanyang Technological University, Singapore. He received the BS and MS degrees in electrical engineering from Tsinghua University, China, and the PhD degree in computer science from Stony Brook University, USA. His research interests fall into the general areas of visual computing and he is particularly interested in the problems which require geometric analysis and computation. For more information, visit http://www.ntu.edu.sg/home/yhe/