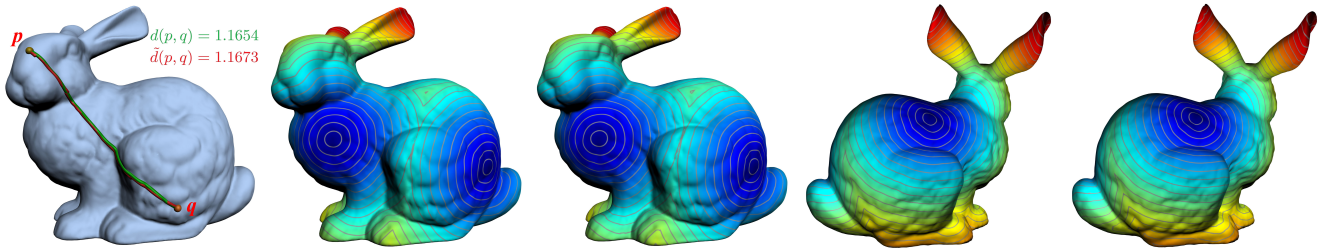
Constant-Time All-Pairs Geodesic Distance Query On Triangle Meshes

Shi-Qing Xin * Xiang Ying † Ying He ‡
School of Computer Engineering, Nanyang Technological University, Singapore



(a) Geodesic distance query (b) Exact result (c) Our result (d) Exact result (e) Our result

Figure 1: *Our algorithm can approximate the geodesic distance between arbitrary pair of points in* ***constant time****. The user can control the approximation error by specifying the number of sample points on the input model. Furthermore, both the geodesic path and geodesic distance field can be approximated in* ***linear time****. With* 1000 *sample points on the BUNNY model* ($n = 72K$)*, the relative root-mean-square error is less than* 0.5%*. (a) shows an example of geodesic distance query and the corresponding exact (in green) and approximate (in red) geodesic paths. (b) and (d) show the front and back views of the exact geodesic distance field by [Xin and Wang 2009]. (c) and (e) show the approximate result by our method. Cold (resp. warm) colors represent short (resp. long) distances.*

Abstract

Computing discrete geodesics on polyhedral surfaces plays an important role in computer graphics. In contrast to the well-studied "single-source, all-destination" discrete geodesic problem, little progress has been reported to the all-pairs geodesic, i.e., computing the geodesic distance between arbitrary two points on the surface. To our knowledge, the existing all-pairs geodesic algorithms have very high computational cost, thus, can not be applied to real-world models, which usually contain thousands of vertices. In this paper, we propose an efficient algorithm to approximate the all-pairs geodesic on triangular meshes. The pre-processing step takes $O(mn^2 \log n)$ time for the input mesh with n vertices and m samples, where m ($\ll n$) is specified by the user, usually between a few hundred and several thousand. In the query step, our algorithm can compute the *approximate* geodesic distance between arbitrary pair of points (not necessarily mesh vertices) in $O(1)$ time. Furthermore, the geodesic path and the geodesic distance field can be approximated in linear time. Both theoretical analysis and experimental results on real-world models demonstrate that our algorithm is efficient and accurate. We demonstrate the efficacy of our algorithm on the interactive texture mapping by using discrete exponential map.

Keywords: Discrete geodesics, constant-time algorithm, all-pairs geodesic distance query, geodesic distance field, discrete exponential map.

*email: sqxin@ntu.edu.sg
†email: ying0008@e.ntu.edu.sg
‡email: yhe@ntu.edu.sg

I3D 2012, Costa Mesa, CA, March 9 – 11, 2012.
© 2012 ACM 978-1-4503-1194-6/12/0003 $10.00

1 Introduction

Geodesic plays an important role in geometric computation and analysis. There are many applications, such as remeshing [Peyré and Cohen 2006], non-rigid registration [Huang et al. 2008], surface parametrization [Lee et al. 2005] [Xia et al. 2011], shape editing [Zelinka and Garland 2004] and shape segmentation [Sander et al. 2003; Zhou et al. 2004], etc, requiring the query of geodesic distance between any pair of points on the given polygonal meshes. Rather than the widely studied "single source, all destination" discrete geodesic problem, very little work has been reported on the "all-pairs" geodesic distance query. So far, the best known result is due to Cook IV and Wenk [2009], who pre-computed the pairwise geodesic between any two mesh vertices in $O(n^5 2^{\alpha(n)} \log n)$ time complexity and $O(n^4)$ space complexity, where n is the number of mesh vertices and $\alpha(n)$ the inverse Ackermann function. Then the geodesic distance between any pair of points on the mesh edges can be computed in $O(m + \log n)$ time, where m is the number of edges crossed by the geodesic path. Although Cook IV and Wenk's algorithm is able to compute the *exact* pairwise geodesic, the high computational cost limits its applications to real-world models which usually contain thousands of vertices.

In this paper, we present an efficient algorithm for approximating all-pairs geodesic distances on triangle meshes. Our algorithm consists of two steps. In the preprocessing step, we construct a geodesic triangulation on m sample points, where m is specified by the user, usually, between a few hundred and several thousand. The preprocessing step takes $O(mn^2 \log n)$ time and outputs an $O(m^2 + n)$-sized file that encodes 1) the exact geodesic distances between every pair of sample points and 2) the exact geodesic distances between a mesh vertex and the three vertices of the enclosing geodesic triangle. Then in the query step, we can approximate the geodesic distance between any two points[1] (not necessarily mesh vertices) on the surface in constant time $O(1)$. We show that the upper bound

[1] We assume each query point q is given in the form of the mesh triangle containing q and a Barycentric coordinate to determine the location of q. Thus, point location also takes $O(1)$ time.

of the approximation error is related to the geodesic triangulation, which in turns can be controlled by the user-specified number of samples m. Experiments on real-world models demonstrate that our method can generate accurate results with a reasonable number of sample points and preprocessing time. For example, the relative root-mean-square error is less than 0.5%, for the BUNNY model with $m = 1000$ samples, see Figure 1. Furthermore, our method has two by-products. First, the approximate geodesic path can be computed in $O(k)$ time, where k is the number of edges crossed by the path. Second, the approximate multi-source geodesic distance field can be computed in $O(ln)$ time, where l is the number of source points.

The remaining of the paper is organized as follows: Section 2 briefly surveys the related work in discrete geodesic and Delaunay triangulation. Section 3 presents the notations and overview of our algorithm, followed by the details of pre-processing in Section 4 and run-time query in Section 5. Section 6 shows the experimental results and discusses the strengths and limitations of the proposed method. Finally, Section 7 concludes the paper with several future research directions. The error analysis is presented in the Supplementary Material.

2 Related work

Computing discrete geodesics on polygonal meshes is a classical problem in computational geometry and computer graphics. Loosely speaking, it has three variants, i.e., "single source, any destination", "geodesic path" and "all-pairs geodesic", which are also closely related to each other. The readers are suggested to refer [Mitchell 2004] for a complete survey.

Single source, any destination geodesic. Sharir and Schorr [1986] presented an $O(n^3 \log n)$ algorithm to compute the geodesic field for convex polyhedra with n faces. Mitchell et al. [1987] (MMP) improved the time complexity bound to $O(n^2 \log n)$ by using the "continuous Dijkstra" technique. Later, Chen and Han [1990] (CH) suggested building a binary tree to encode all the edge sequences that reduced the time complexity to $O(n^2)$. Surazhsky et al. [2005] extended MMP algorithm to compute approximate geodesics with bounded error. Xin and Wang [2009] improved the CH algorithm by exploiting a filtering theorem; their proposed method outperforms both the MMP and CH algorithms for real-world models. Schreiber and Sharir [2006] presented an optimal $O(n \log n)$-time algorithm for convex polyhedral surfaces. However, the optimal time bound for general polyhedral surfaces is still an open problem. Recently, by using parallel source windows, Xin et al. [2011] extended CH algorithm to compute geodesic offsets on triangle meshes. In addition, many algorithms [Barbehenn 1998; Kimmel and Sethian 1998; Polthier and Schmies 1999; Spira and Kimmel 2002; Weber et al. 2008] have been proposed to compute approximate discrete geodesics.

Geodesic path. The ϵ-approximation algorithms [Papadimitriou 1985; Clarkson 1987; Choi et al. 1994] are well studied to compute an approximate path at most $(1+\epsilon)d(s,t)$ in length, where $d(s,t)$ denotes the exact geodesic distance between two points on the input polyhedral surface. These algorithms make a trade-off between computation cost and accuracy. Kanai and Suzuki [2000] computed an approximate path between two points by locally refining the initial path on the discrete graph induced by the input polyhedral surface. Xin et al. [2012] developed an efficient algorithm to iteratively evolve an initial closed curve into an exact geodesic loop. The algorithm is guaranteed to converge in finite steps.

All-pairs geodesic. Agarwal et al. [1997] proved that the $\Theta(n^4)$ shortest path edge sequences can be computed in $O(n^6 2^{\alpha(n)} \log n)$-time and $O(n^5)$-space complexity, where $\alpha(n)$ is the inverse Ackermann function. Recently, Cook IV and Wenk [2009] improved this bound to $O(n^5 2^{\alpha(n)} \log n)$-time and $O(n^4)$-space. These algorithms are computationally expensive, thus, can hardly be applied to real-world applications.

Besides all pairs geodesic, there are also some related work on computing all pairs shortest paths on weighted or unweighted graphs, such as [Dor et al. 2000], [Zwick 1998], [Baswana et al. 2003], [Zwick 2002]. Using these graph-based approaches, the resultant shortest paths always follow the mesh edges, thereby resulting in a "zigzag" structure on the mesh. As a result, shortest paths highly depend on the mesh resolution and tessellation. Therefore, shortest paths are usually not robust and can lead to inaccurate results on polygonal meshes, especially with poor triangulation.

Our work focuses on approximating all pairs geodesics on triangle meshes in constant time. However, our method can be easily extended to compute the approximate geodesic paths and geodesic distance fields in linear time.

3 Overview

Motivations. Our algorithm is motivated by the fact that geodesic distance field is G^1 continuous except at the conjugate points where the continuity is only C^0. Given some samples uniformly distributed on the input model and the exact geodesic distances among these samples, we can approximate the geodesic distance between arbitrary points with some interpolation technique. A naïve interpolation is the linear interpolation, which, however, leads to very poor results (see Section 6.3 for the discussions). This is mainly because the geodesic distance is not linear. In this paper, we develop an "unfolding" technique, which flattens the curved geodesic triangles and the query points to the Euclidean plane by keeping some pre-computed geodesic distances unchanged. Then the geodesic distance between the query points is given by the Euclidean distance between their unfolded images. This unfolding technique can guarantee the accurate result if the surface is developable. Experimental results on real-world models show that our technique is very effective in practice, and significantly outperforms the naïve linear interpolation.

Notations. The input triangle mesh M is represented by $M = (V, E, F)$, where V, E, and F are the sets of vertices, edges and faces respectively. Let $n = |V|$ denote the number of vertices.

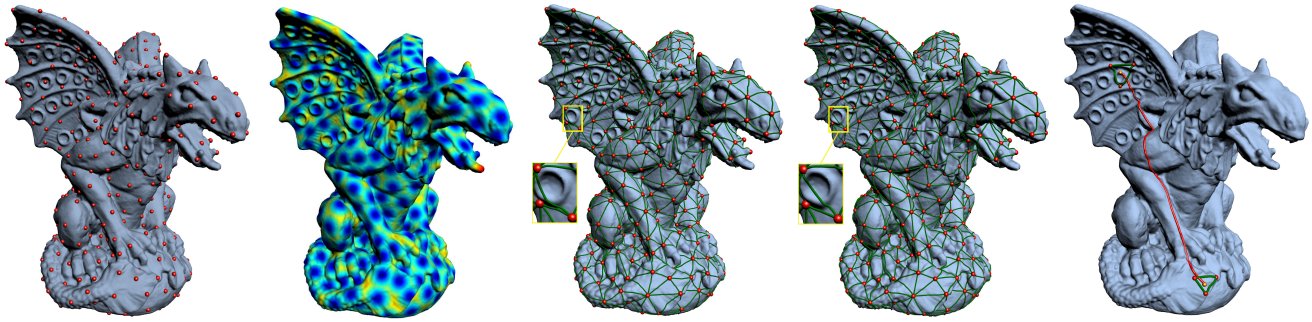
Throughout this paper, we distinguish *vertex*, *sample point*, and *query point* as follows:

- a *vertex* v is an element of V;
- a *sample point* s is a point on M, which is computed using farthest point sampling algorithm.
- a *query point* q is an arbitrary point on M, i.e., it can be a vertex, or on an edge or inside a triangle.

Furthermore, we use the following notations to denote the path and various distances between two query points p and q:

- $\|pq\|$ the Euclidean distance;
- $d(p,q)$ the exact geodesic distance;
- $\tilde{d}(p,q)$ the approximate geodesic distance computed by our algorithm;

Finally, let $[i] = i \bmod 3$ be the modulo 3 operator.



(a) (b) (c) (d) (e)

Figure 2: *Algorithm pipeline. Given the input mesh M, we first sample m points distributed uniformly on M (see (a)); Then taking the sample points as sources, we compute the geodesic distance field on M (see (b)) and construct an approximate Delaunay triangulation (see (c)), which induces a geodesic triangulation by replacing each Delaunay edge with a geodesic (see (d)). Next, the exact geodesic distances between any pair of sample points are saved in a file. Finally, with the pre-computed geodesic distance file, we can approximate the geodesic distance between any pair of query points in constant time $O(1)$ and find the corresponding geodesic path in linear time $O(k)$, where k is the number of edges crossed by the path (see (e)).*

Algorithmic pipeline. Our algorithm contains two steps, the pre-processing step and the query step, as shown in Figure 2.

The pre-processing step first samples the given surface with m points, where m is specified by the user (see (a)) and then computes the geodesic distance fields with the samples as sources (see (b)), which induces a Delaunay triangulation (see (c)). Next, the Delaunay triangulation edges are replaced by geodesics (see (d)). Finally, we save two kinds of exact geodesic distances into a file, i.e., the distances between any pair of sample points, and the distances between each mesh vertex and the three vertices of the geodesic triangle containing it. The pre-processing step takes $O(mn^2 \log n)$ time and outputs a file of size $O(m^2 + n)$.

In the query step, given two query points q_1 and q_2 on the surface, we first find the geodesic triangles containing them, say T_1 and T_2 (see (e)). Then, we *unfold* the geodesic triangles T_1 and T_2 to $\mathbb{R}^2$ such that the Euclidean side lengths coincide with the geodesic distances of the curved geodesic triangle. The query points q_1 and q_2 are also mapped to $\mathbb{R}^2$ using unfolding operations. The 2D images are denoted by $q_1^{'}$ and $q_2^{'}$, respectively. Finally, the geodesic distance between q_1 and q_2 is approximated by the Euclidean distance between $q_1^{'}$ and $q_2^{'}$. Note that this approximation can be done in constant time $O(1)$ as unfolding is a simple and local operation.

4 Pre-processing

4.1 Constructing geodesic triangulation

As shown in Sec. 5.3, the approximation error is closely related to the maximal side length of geodesic triangles. Thus, an isotropic sampling, which induces geodesic triangulation of similar sizes, is desired. In this paper, we adopt a variant of the farthest point sampling method [Peyré and Cohen 2006], which iteratively updates the sampling set with the farthest point to the existing sample points. The difference is that we use the improved CH algorithm [Xin and Wang 2009] rather than the fast marching method [Kimmel and Sethian 1998] for geodesic distance computation. The time complexity for sampling m points is $\sum_{i=1}^{m} f(\frac{n}{i}) = O(n^2 \log n)$, where $f(x) = O(x^2 \log x)$.

Taking m sample points as source points, we compute the multi-source geodesic distance field using [Xin and Wang 2009], then construct an approximate Delaunay triangulation using [Xin and Wang 2010]. It is shown that the resultant Delaunay triangulation is well defined in the sense that all the Delaunay edges do not intersect [Xin and Wang 2010] under moderate condition. However, the edges are not "straight" in general. Finally, we need to replace each curved Delaunay edge with a geodesic path.

Note that it takes $O(mn^2 \log n)$ time to run the exact geodesic algorithm [Xin and Wang 2009] with each sample point as source point. Also, it takes $O(n)$ time to replace each curved Delaunay edge with a geodesic. Considering the number of edges in the Delaunay triangulation is $O(m)$, the time complexity for constructing geodesic triangulation is $O(mn^2 \log n)$.

4.2 Generating geodesic distance file

After constructing the geodesic triangulation, we are ready to generate a file containing the following information, which will be used our all-pairs geodesic distance query algorithm.

- The exact geodesic distances between any pair of sample points.
- For each vertex v, output the geodesic triangle $\triangle s_1 s_2 s_3$ containing v and the exact geodesic distances $d(v, s_i)$, $i = 1, 2, 3$.

Note that all the required geodesic distances are readily available when we constructed the geodesic triangulation in the previous step.

Putting them all together, the entire pre-processing takes $O(mn^2 \log n)$ time and outputs a file of size $O(m^2 + n)$.

5 All-Pairs Geodesic Distance Query

5.1 Unfolding geodesic triangles

We first present two simple *unfolding* operations that play key roles in our constant-time query algorithm. Observing that geodesic distances follow triangle inequality, thus, we can flatten the curved geodesic triangle to a unique triangle (up to rigid motion) in $\mathbb{R}^2$.

Given two curved geodesic triangles $\triangle psr$ and $\triangle qrs$, we can unfold them to 2D triangles $\triangle p's'r'$ and $\triangle q'r's'$, where the corresponding Euclidean edge lengths are equal to those of the geodesic edges. Let $u(p, q|rs)$ denote such unfolding operation with respect to edge rs. We call it one-side unfolding if p and q are on the same side of rs and two-side unfolding, otherwise (see Figure 3(a)).

5.2 Algorithm

We assume that the query point q is given in the form of a triangle $t = (t_1, t_2, t_3)$ and a Barycentric coordinate (α, β, γ) such that $q \in t$ and $q = \alpha t_1 + \beta t_2 + \gamma t_3$. Thus, the point location takes constant time. This assumption is reasonable, since many sampling algorithms (e.g. [Osada et al. 2002]) do output the samples in this format.

Given two arbitrary query points $p, q \in M$, we consider the following cases to compute the approximate geodesic distance $\tilde{d}(p, q)$:

Case 1: both p and q are sample points. Their exact geodesic distance can be reported immediately by retrieving the information from the pre-computed geodesic file.

Case 2: one of p and q is a sample point, while the other is a vertex. Without loss of generality, say p is a sample point, and q is a vertex.

- **Case 2.1: p and q are in the same geodesic triangle.** Then p is the vertex of the geodesic triangle containing q. So, the exact geodesic distance $d(p, q)$ can be obtained directly from the geodesic file.
- **Case 2.2: p and q are in different geodesic triangle.** Let $\triangle s_1 s_2 s_3$ be the geodesic triangle containing q, then the exact geodesic distances $d(p, s_i)$, $i = 1, 2, 3$, are known. We apply the two-side unfolding operation with respect to each edge of geodesic triangle $\triangle s_1 s_2 s_3$, i.e., $u(p, q|s_i s_{[i]+1})$, $i = 1, 2, 3$. Let p_i and q_i denote the images of p and q. Then the approximate geodesic distance is $\tilde{d}(p, q) = \min_{1 \le i \le 3} \|p_i q_i\|$ (see Figure 3(b)).

Case 3: both p and q are vertices, rather than sample points.

- **Case 3.1: p and q are in the same geodesic triangle $\triangle s_1 s_2 s_3$.** We apply one-side unfolding $u(p, q|s_i s_{[i]+1})$, $i = 1, 2, 3$, with respect to each geodesic edge. Let p_i and q_i denote the 2D images of p and q. Then the approximate geodesic distance is $\tilde{d}(p, q) = \min_{1 \le i \le 3} \|p_i q_i\|$ (see Figure 3 (c)).
- **Case 3.2: p and q are in different geodesic triangles, say $p \in \triangle s_1 s_2 s_3$ and $q \in \triangle s_4 s_5 s_6$.** Note that the sample point s_4 and query point p are in different geodesic triangles, we can compute $\tilde{d}(p, s_4)$ according to Case 2.2. Similarly, we compute $\tilde{d}(p, s_5)$ and $\tilde{d}(p, s_6)$. Then with the approximate geodesic distances $\tilde{d}(p, s_i)$, $4 \le i \le 6$, we can apply Case 2.2 again to compute $\tilde{d}(p, q)$.

Case 4: one of p and q is a vertex, while the other is an arbitrary point. Without loss of generality, assume $q \in V$ is a vertex, but p is not. Let $\triangle v_1 v_2 v_3$ be the mesh triangle containing p. By using Case 3, we can compute $\tilde{d}(v_i, q)$,$i = 1, 2, 3$. Then, the approximate geodesic distance $\tilde{d}(p, q)$ is given by applying Case 3 again, i.e., two-side unfolding with respect to each edge $v_i v_{[i]+1}$, $i = 1, 2, 3$.

Case 5: neither of p and q is a vertex. Assume p is in a mesh triangle $\triangle v_1 v_2 v_3$. Following Case 4, we can compute $\tilde{d}(v_i, q)$, $i = 1, 2, 3$, immediately. Note that $d(p, v_i)$, $i = 1, 2, 3$, can be computed using Euclidean distance. Again, we can apply two-side unfolding operation in Case 3 to each edge of $\triangle v_1 v_2 v_3$, i.e., $u(p, q|v_i v_{[i]+1})$, $i = 1, 2, 3$, to obtain the final estimate $\tilde{d}(p, q)$.

The above all-pairs geodesic distance query algorithm contains only three kinds of operations, retrieving geodesic distance from pre-computed geodesic file, unfolding operations and computing the Euclidean distance between two 2D points, which are all constant-time operations. Thus, each query can be answered in constant time $O(1)$.

5.3 Error Analysis

If two query points are in the *different* geodesic triangles, the upper bound of approximation error is given by the following Theorem, where the proof is provided in the Supplementary Material.

Theorem 1. *Given two query points q_1 and q_2 in different geodesic triangles, the approximation error $|d(q_1, q_2) - \tilde{d}(q_1, q_2)| \le 2L + 2l$, where L (resp. l) is the maximum edge length of geodesic (resp. mesh) triangles containing q_1 and q_2.*

6 Results & Discussions

6.1 Experimental Results

We implemented our algorithm in C++ on a 64-bit Workstation with Xeon 2.66GHz CPU and 8GB memory. Table 1 shows the query speed of the BUDDHA and the LUCY models.

Table 1: *Query speed (# queries/second).*

	BUDDHA ($n = 50K$)			LUCY ($n = 263K$)		
m	1000	2000	3000	1000	2000	3000
speed	$27.3K$	$36.7K$	$45.4K$	$8.2K$	$15.5K$	$21.0K$

To evaluate the accuracy of our algorithm, we compared the approximate geodesic distance with the exact geodesic distance. Specifically, we randomly sampled k query points[2] using [Osada et al. 2002] for each test model. Then for any pair of query points p and q, we computed the *exact* geodesic distance $d(p, q)$ using [Xin and Wang 2009] and the *approximate* geodesic distance $\tilde{d}(p, q)$ using our algorithm. Next, we computed the relative error $\varepsilon(p, q) = \frac{|d(p,q) - \tilde{d}(p,q)|}{d(p,q)}$. Finally, we computed the relative root-mean-square error

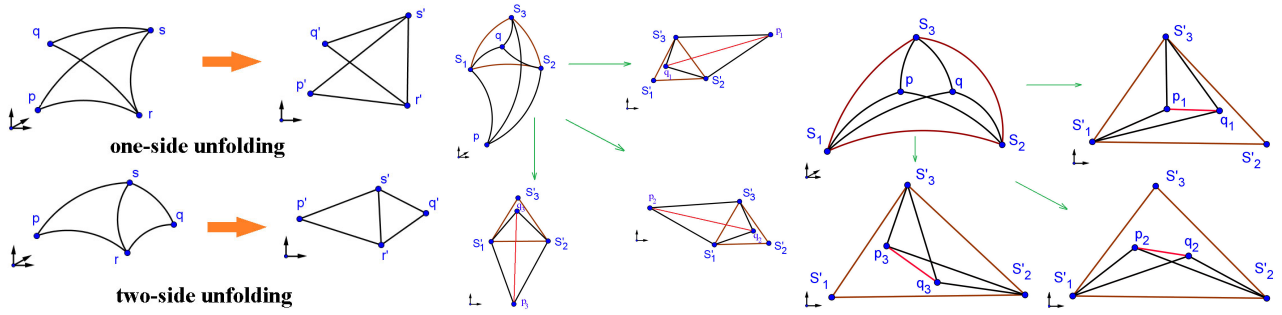
$$\epsilon = \frac{1}{\binom{k}{2}} \sqrt{\sum_{p,q} \varepsilon^2(p, q)}.$$

As the upper bound of approximation error is closely related to the longest edge of the geodesic triangles that containing the query points (see Theorem 1), which in turn depends on the number of sample points m. Clearly. the more number of sample points, the more accurate geodesic distances one obtains, but the more pre-processing time one has to take. As shown in Figure 5(top), for smooth model such as FERTILITY and BUNNY (see Figure 1), we can obtain query results with $\epsilon \le 0.5\%$ with just a few hundred samples. For large models with detailed features, such as ARMADILLO and LUCY (see Figure 7), we observed that a few thousand samples usually lead to good results. As shown in Figure 5(bottom), the pre-processing time is roughly a linear function of m. The error distribution histogram in Figure 4 also demonstrates that increasing m is helpful to reduce the approximation error. Thus, this is a tradeoff between the accuracy and pre-processing time, which is usually application dependent.

6.2 Geodesic Path & Geodesic Distance Field

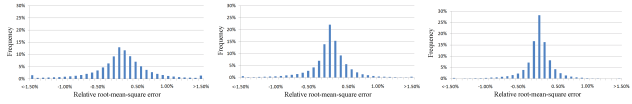
Our algorithm has two byproducts that 1) the approximate geodesic path can be computed in $O(k)$ time, where k is the number of edges crossed by the path; and 2) the approximate geodesic distance field with l source points can be computed in $O(ln)$ time.

[2] We observed that $k = 10000$ leads to very stable statistical result.



(a) Unfolding operation (b) Case 2.2 (c) Case 3.1

Figure 3: *(a) The unfolding operation flattens curved geodesic triangles to* $\mathbb{R}^2$ *such that the corresponding edge lengths are equal, i.e.,* $d(r,s) = \|r's'\|$, $d(p,r) = \|p'r'\|$, $d(p,s) = \|p's'\|$, $d(q,r) = \|q'r'\|$, $d(q,s) = \|q's'\|$. *(b) and (c) show two important cases of applying unfolding operation in our algorithm.*



$m = 1000$ $m = 2000$ $m = 3000$

Figure 4: *Error distribution of ARMADILLO model.* 40.27%, 58.45% *and* 68.40% *relative errors are in* $[-0.2\%, +0.2\%]$ *for* $m = 1000$, 2000 *and* 3000, *respectively.*

Geodesic distance field. Computing geodesic distance field with l source points is straight forward, i.e., for each source point, simply computing the geodesic distances with all mesh vertices, and then choosing the smallest distance. Thus, the time complexity is $O(ln)$. Figure 6 shows the approximate multi-source geodesic distance fields with various number of samples on BUDDHA model.

Geodesic path. Given two query points s and t on the surface, let $\gamma : [0,1] \to M$ be the geodesic path between s and t, where $\gamma(0) = s$ and $\gamma(1) = t$. Let $D(\cdot)$ denote the geodesic distance field with source point s. The geodesic path satisfies the following Eikonal equation [Kimmel and Sethian 1998]:

$$\frac{d\gamma}{dt} = -\nabla D,$$

where ∇D is the gradient of D. This ODE can be solved efficiently by tracing the gradient vector field ∇D on triangle mesh. Let $Nb(t)$

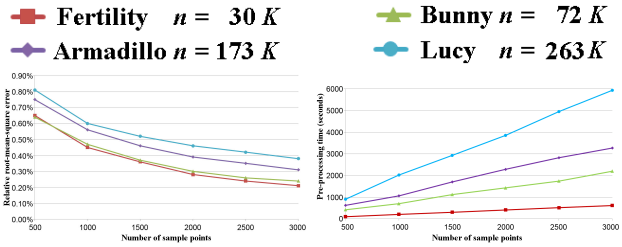
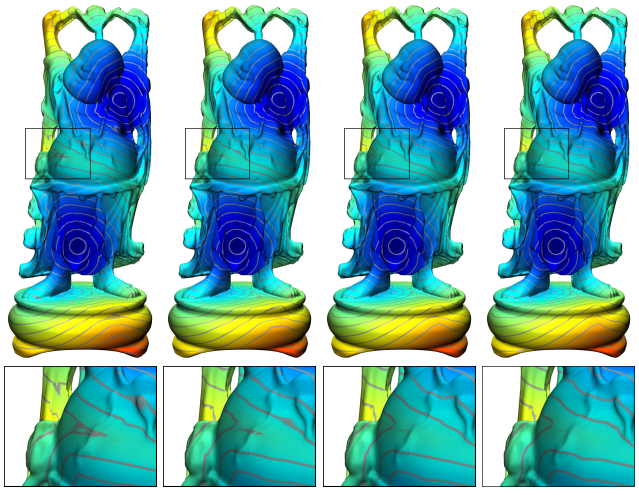


Figure 5: *Numerical results. Left: Relative root-mean-square error VS number of sample points; Right: Preprocessing time (seconds) VS number of sample points.*

be the set of one-ring neighbors of t, if t is a vertex; otherwise, $Nb(t)$ includes the three vertices of the mesh triangle containing t. We compute $\tilde{d}(s,t)$ and the geodesic distances between each element of $Nb(t)$ and s. Then we compute the gradient ∇D for t's neighboring triangles and find the one with the largest magnitude, which is used as the initial tracing direction.

Starting from t and following the initial tracing direction, we can reach one mesh edge, which induces a neighboring triangle f. Then we can compute the gradient ∇D in f, which further induces the next tracing direction. Repeat the tracing procedure until we reach the source point s.

Note that computing the gradient ∇D inside a mesh triangle f takes constant time, so is the tracing in f. Thus, the time complexity for tracing the geodesic path is $O(k)$, where k is the number of edges crossed by the path.

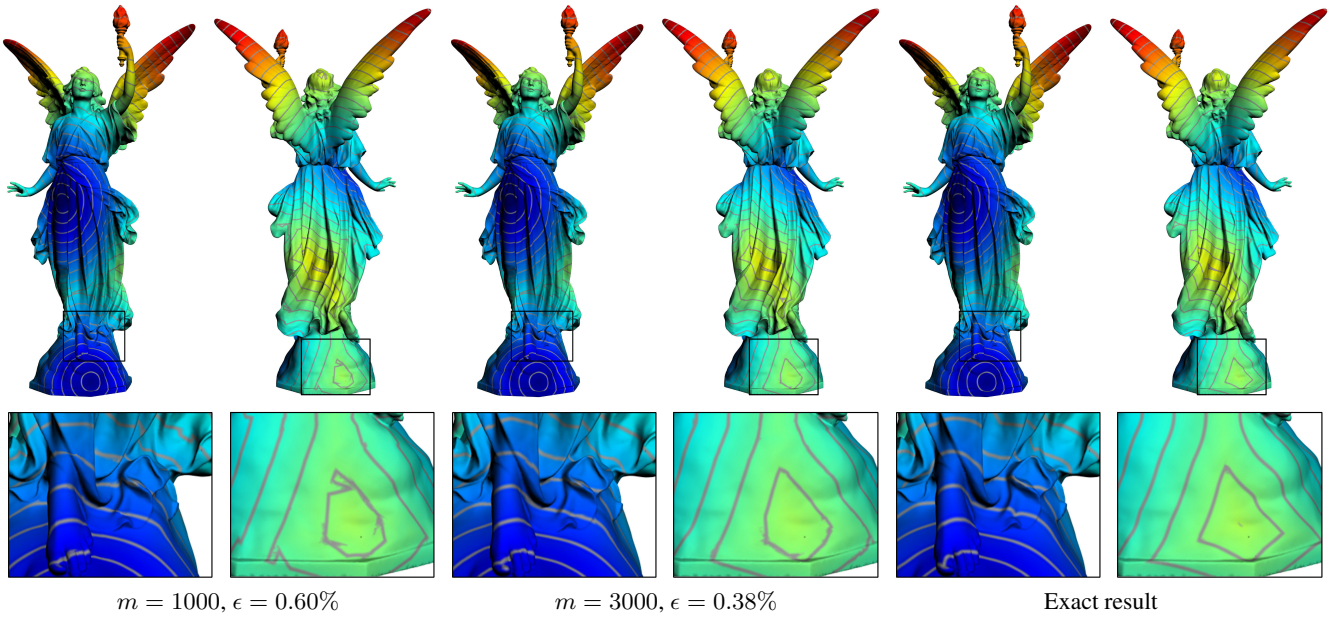


$m = 1000$ $\epsilon = 0.68\%$ | $m = 2000$ $\epsilon = 0.50\%$ | $m = 3000$ $\epsilon = 0.42\%$ | Exact result

Figure 6: *Computing geodesic distance field on BUDDHA (*$n = 50K$*). The user can specify source points arbitrarily on the 3D model, then our program is able to compute the approximate geodesic distance field in linear time. The first three columns show the approximate results by our algorithm with* $m = 1000$, 2000, *and* 3000 *respectively. The last column shows the exact result by [Xin and Wang 2009].*

6.3 Discussions

We compared our algorithm with a simple linear interpolation method: Considering two mesh vertices v_1 and v_2, which are located in geodesic triangles $\triangle s_1 s_2 s_3$ and $\triangle s_4 s_5 s_6$ respectively. As the exact geodesic distances $d(v_1, s_i)$, $i = 1, 2, 3$, are known (from the pre-computed geodesic file), we can compute the Barycentric-like coordinate $(\lambda_1, \lambda_2, \lambda_3)$ of v_1 with respect to geodesic triangle $\triangle s_1 s_2 s_3$. Similarly, compute $(\lambda_4, \lambda_5, \lambda_6)$ for v_2 with respect to $\triangle s_4 s_5 s_6$. Let $d_{ij} = d(s_i, s_j)$ denote the exact geodesic distance



$m = 1000, \epsilon = 0.60\%$ $m = 3000, \epsilon = 0.38\%$ Exact result

Figure 7: *Computing geodesic distance field on the LUCY model ($n = 263K$).*

between s_i and s_j, $1 \leq i \leq 3$, $4 \leq j \leq 6$, which can also be obtained from the pre-computed geodesic file. Then the linear interpolation method computes the geodesic between v_1 and v_2 by

$$\tilde{d}(v_1, v_2) = \begin{pmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{pmatrix} \begin{pmatrix} d_{14} & d_{15} & d_{16} \\ d_{24} & d_{25} & d_{26} \\ d_{34} & d_{35} & d_{36} \end{pmatrix} \begin{pmatrix} \lambda_4 \\ \lambda_5 \\ \lambda_6 \end{pmatrix}.$$

Although conceptually simple and easy to implement, this linear interpolation method performs poorly as shown in Figure 8. This is mainly because the geodesic distance field is not linear. In contrast to the linear interpolation method, our unfolding technique can guarantee the accurate result if the surface is developable. It also works very well for the real-world models with complicated geometry (see Figures 6 and 7).

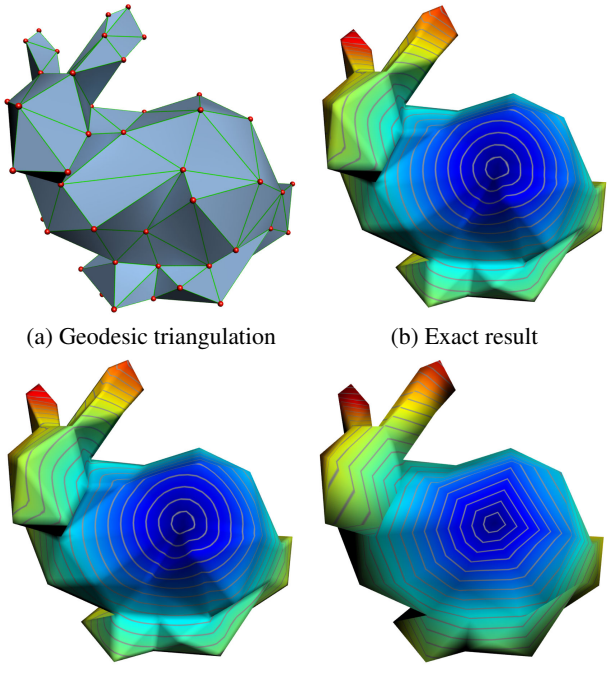
6.4 Application

The proposed $O(1)$ all-pairs geodesic distance query algorithm can be applied to a wide range of 3D graphics applications, such as sampling, parametrization, texture mapping, shape analysis, etc. In this paper, we demonstrate our algorithm on the discrete exponential map and texture mapping.

Let $\mathbf{p} \in S$ be a point on the surface S, the exponential map at point $\mathbf{p}$ takes geodesic curves originating at $\mathbf{p}$ to straight rays emanating from the origin of the tangent plane $\mathcal{T}_\mathbf{p}$. Conversely, for any unit vector $\mathbf{v} \in \mathcal{T}_\mathbf{p}$, there exists a unique geodesic $g_\mathbf{v}$ parameterized by arc length such that $g_\mathbf{v}(0) = \mathbf{p}$ and $g_\mathbf{v}'(0) = \mathbf{v}$ (see Figure 10). Exponential map is a very useful method to create a geodesic polar coordinate system around $\mathbf{p}$ on the curved surface.

Due to the large amount of geodesic computations involved, it is technically challenging to compute exponential maps on polygonal meshes in real-time. Schmidt et al. [2006] developed a method to approximate discrete exponential map which requires only a simple addition to Dijkstras algorithm [Dijkstra 1959]. Although it is easy to implement and works well for smooth surfaces, this approximation usually leads to large errors for surfaces with rich geometric details.

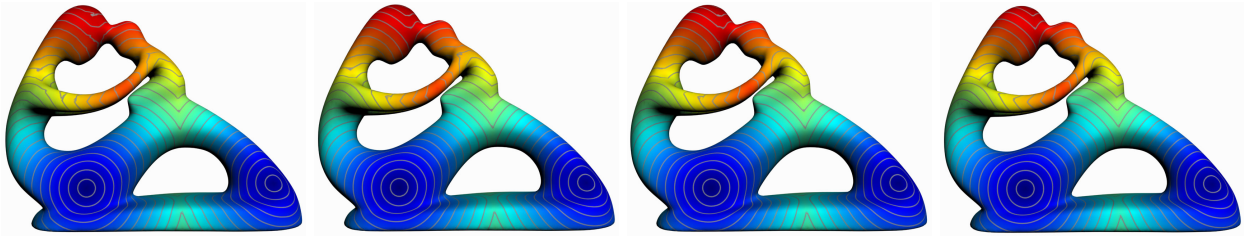
Our algorithm is able to approximate the geodesic distance between arbitrary two query points in constant time and also find the



(a) Geodesic triangulation (b) Exact result

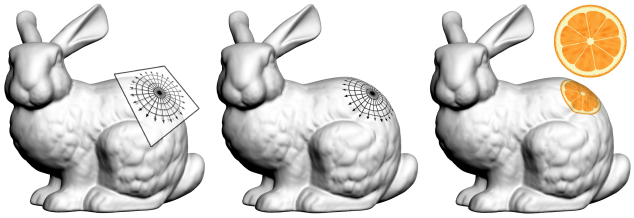
(c) Our result (d) Linear interpolation

Figure 8: *Comparison to the linear interpolation method. (a) Given the simplified BUNNY model with 100 vertices, we use the mesh vertices as the sample points (red points), thus, the mesh triangulation induces the geodesic triangulation. (b) The exact geodesic distance field. (c) The geodesic distance field computed by our unfolding technique. As the input surface is almost developable everywhere (except at the mesh vertices), the unfolding technique leads to highly accurate result. (d) The naïve linear interpolation method, however, leads to very poor results, since the geodesic distance field is not linear.*



$m = 500$ $\epsilon = 0.65\%$ | $m = 1000$ $\epsilon = 0.45\%$ | $m = 1500$ $\epsilon = 0.36\%$ | Exact result

Figure 9: *Computing geodesic distance field on the FERTILITY model ($n = 30K$).*



(a) (b) (c)

Figure 10: *Exponential map creates a geodesic polar coordinate system on the surface. (a) The polar coordinate system on the tangent plane $\mathcal{T}_\mathbf{p}$. (b) The polar coordinate system on the surface. (c) The exponential map naturally induces a texture mapping.*

geodesic path in linear time. Thus, it can be used to compute the discrete exponential map for real-world models. Figure 11 demonstrates the discrete exponential map based texture mapping.

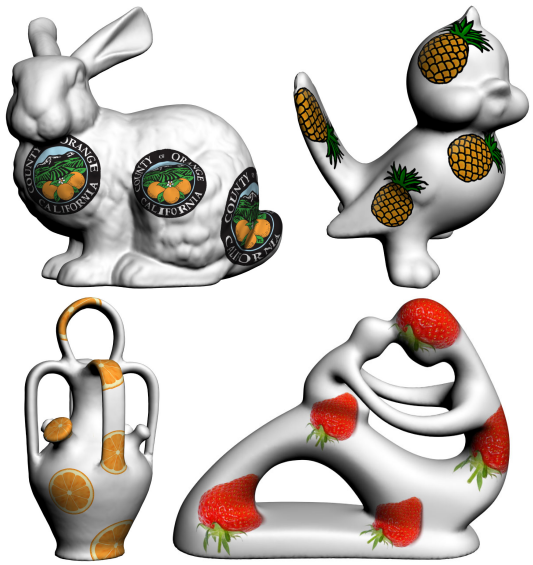


Figure 11: *Discrete exponential map based texture mapping.*

6.5 Limitations

The proposed framework has limitations.

First, Theorem 1 presents the upper bound of the approximation error for two query points located in *different* geodesic triangles. In fact, if two query points are close enough that they are in the *same* geodesic triangle, the approximation error could be higher. Figure 12 (a) illustrates an example, where a single geodesic triangle contains two highly curved features. Using the one-side unfolding, the two images $q_1^{'}$ and $q_2^{'}$ are close to each other, which leads to inaccurate result. In practice, such case can be improved by constructing a finer geodesic triangulation that separates the highly curved features, e.g., splitting $\triangle s_1 s_2 s_3$ such that the two features are separated in two geodesic triangles as shown in Figure 12(b). Then our two-side unfolding operation can generate much more accurate result. Another practical solution is to apply an exact geodesic algorithm, such as MMP [Mitchell et al. 1987], CH [Chen and Han 1990] and its variant [Xin and Wang 2009]. Note that these "single-source-any-destination" geodesic algorithms maintain a wave propagating from the source point. When the wavefront touches the other query point, we can terminate the algorithm immediately. As the two query points are close and in the same geodesic triangle, the computational cost is very low.

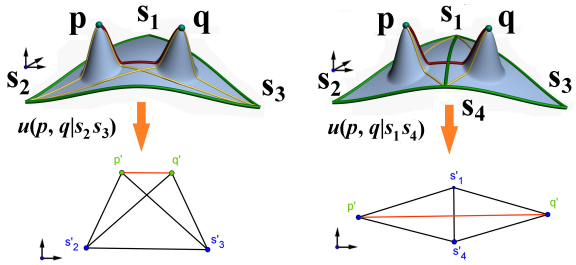


Figure 12: *Left: query points located in the same geodesic triangle may lead to large approximation error. Right: improved result can be obtained by constructing a finer geodesic triangulation that separates highly curved features.*

Second, the pre-processing step has $O(m^2 + n)$ space complexity. Although $m \ll n$, the number of sampling points m clearly depends on the mesh geometry and complexity. For a large-scale mesh with $n = 1000K$ vertices and $m = 20K$ sampling points, the memory cost is 3.2GB memory. This limits our method for no more than $20K$ sampling points on a 32-bit PC.

7 Conclusion

This paper proposes a practical and efficient algorithm to approximate the all-pairs geodesic on triangular meshes. The preprocessing step takes $O(mn^2 \log n)$ time, where n is the number of vertices of the input mesh and m is number of samples on the input surface, usually a few hundreds to several thousands. In the query step, our algorithm can answer an approximate geodesic distance between any two points (not necessarily mesh vertices) on the surface in constant time $O(1)$. Furthermore, the geodesic path and the geodesic distance field can be approximated in linear time. Both theoretical analysis and experimental results on real-world models show that our algorithm is efficient and accurate.

Future work. In our current implementation, the majority of preprocessing time is due to computing the "single-source-all-destination" exact geodesic distance using improved CH algorithm [Xin and Wang 2009] which takes $O(n^2 \log n)$ time. Schreiber and Sharir [Schreiber and Sharir 2006] developed an $O(n \log n)$ al-

gorithm for *convex* polytopes. This promising result leads to a popular belief in the research community that the lower bound for computing geodesic distances on *general* polytopes can be improved, which in turn will greatly reduce the preprocessing time in our framework.

We proved the error bound for the case that two query points are in *different* geodesic triangles. However, if two query points lie in the *same* geodesic triangles, the approximation error could be higher if the geodesic triangle is highly curved (see Figure 12). Although we did not present the error bound for this case, we believe that the approximation error is closely related to the maximal curvature of the geodesic triangle. We will further investigate this in the future.

Acknowledgement This work was supported by Singapore NRF Interactive Digital Media R&D Program, under research grant NRF2008IDM-IDM004-006. We would like to thank Ms. Shuang-Min Chen and Sun Qian for their help on videos and figures, Prof. Xianfeng Gu and Prof. Wei Luo for insightful discussions. Models courtesy of Stanford University and Aim@Shape.

References

AGARWAL, P. K., ARONOV, B., O'ROURKE, J., AND SCHEVON, C. A. 1997. Star unfolding of a polytope with applications. *SIAM J. Comput. 26*, 6, 1689–1713.

BARBEHENN, M. 1998. A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Trans. Comput. 47*, 2, 263.

BASWANA, S., HARIHARAN, R., AND SEN, S. 2003. Maintaining all-pairs approximate shortest paths under deletion of edges. In *SODA '03*, 394–403.

CHEN, J., AND HAN, Y. 1990. Shortest paths on a polyhedron. In *SCG '90*, 360–369.

CHOI, J., SELLEN, J., AND YAP, C.-K. 1994. Approximate euclidean shortest path in 3-space. In *SCG '94*, 41–48.

CLARKSON, K. 1987. Approximation algorithms for shortest path motion planning. In *STOC '87*, 56–65.

COOK IV, A. F., AND WENK, C. 2009. Shortest path problems on a polyhedral surface. In *WADS '09*, 156–167.

DIJKSTRA, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1, 269–271.

DOR, D., HALPERIN, S., AND ZWICK, U. 2000. All-pairs almost shortest paths. *SIAM J. Comput. 29*, 1740–1759.

HUANG, Q.-X., ADAMS, B., WICKE, M., AND GUIBAS, L. J. 2008. Non-rigid registration under isometric deformations. In *SGP '08*, 1449–1457.

KANAI, T., AND SUZUKI, H. 2000. Approximate shortest path on polyhedral surface based on selective refinement of the discrete graph and its applications. In *GMP '00*, 241–250.

KIMMEL, R., AND SETHIAN, J. A. 1998. Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, 8431–8435.

LEE, H., TONG, Y., AND DESBRUN, M. 2005. Geodesics-based one-to-one parameterization of 3d triangle meshes. *IEEE MultiMedia 12*, 27–33.

MITCHELL, J. S. B., MOUNT, D. M., AND PAPADIMITRIOU, C. H. 1987. The discrete geodesic problem. *SIAM J. Comput. 16*, 4, 647–668.

MITCHELL, J. S. B. 2004. New results on shortest paths in three dimensions. In *SCG '04*, 124–133.

OSADA, R., FUNKHOUSER, T., CHAZELLE, B., AND DOBKIN, D. 2002. Shape distributions. *ACM Trans. Graph. 21*, 807–832.

PAPADIMITRIOU, C. H. 1985. An algorithm for shortest-path motion in three dimensions. *IPL 20*, 259–263.

PEYRÉ, G., AND COHEN, L. D. 2006. Geodesic remeshing using front propagation. *Int. J. Comput. Vision 69*, 1, 145–156.

POLTHIER, K., AND SCHMIES, M. 1999. Geodesic flow on polyhedral surfaces. In *Proceedings of Eurographics-IEEE Symposium on Scientific Visualization '99*, 179–188.

SANDER, P. V., WOOD, Z. J., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2003. Multi-chart geometry images. In *SGP '03*.

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph. 25* (July), 605–613.

SCHREIBER, Y., AND SHARIR, M. 2006. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. In *SCG '06*, 30–39.

SHARIR, M., AND SCHORR, A. 1986. On shortest paths in polyhedral spaces. *SIAM J. Comput. 15*, 1, 193–215.

SPIRA, A., AND KIMMEL, R. 2002. Geodesic curvature flow on parametric surfaces. In *Curve and Surface Design*, 365–373.

SURAZHSKY, V., SURAZHSKY, T., KIRSANOV, D., GORTLER, S. J., AND HOPPE, H. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3, 553–560.

WEBER, O., DEVIR, Y. S., BRONSTEIN, A. M., BRONSTEIN, M. M., AND KIMMEL, R. 2008. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph. 27*, 4.

XIA, J., GARCIA, I., HE, Y., XIN, S.-Q., AND PATOW, G. 2011. Editable polycube map for gpu-based subdivision surfaces. In *Symposium on Interactive 3D Graphics and Games*, 151–158.

XIN, S.-Q., AND WANG, G.-J. 2009. Improving Chen and Han's algorithm on the discrete geodesic problem. *ACM Trans. Graph. 28*, 4, 1–8.

XIN, S.-Q., AND WANG, G.-J. 2010. Applying the improved Chen and Han's algorithm to different versions of shortest path problems on a polyhedral surface. *CAD 42*, 942–951.

XIN, S.-Q., YING, X., AND HE, Y. 2011. Efficiently computing geodesic offsets on triangle meshes by the extended xin-wang algorithm. *Computer Aided Design 43*, 1468–1476.

XIN, S.-Q., HE, Y., AND FU, C.-W. 2012. Efficiently computing exact geodesic loops within finite steps. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, to appear.

ZELINKA, S., AND GARLAND, M. 2004. Similarity-based surface modelling using geodesic fans. In *SGP '04*, 204–213.

ZHOU, K., SYNDER, J., GUO, B., AND SHUM, H.-Y. 2004. Isocharts: stretch-driven mesh parameterization using spectral analysis. In *SGP '04*, 45–54.

ZWICK, U. 1998. All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms. In *FOCS '98*, 310–319.

ZWICK, U. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM 49*, 289–317.