Compressing geodesic information for fast point-to-point geodesic distance queries

Craig Gotsman · Kai Hormann

Abstract

Geodesic distances between pairs of points on a 3D mesh surface are a crucial ingredient of many geometry processing tasks, but are notoriously difficult to compute efficiently on demand. We propose a novel method for the *compact storage* of geodesic distance information, which enables answering point-to-point geodesic distance queries very efficiently. For a triangle mesh with n vertices, if computing the geodesic distance to all vertices from a single source vertex costs $O(f(n))$ time, then we generate a database of size $O(m\,n\log n)$ in $O\big((f(n)+m^3 n)\sqrt{n}\big)$ time in a preprocessing stage, where m is a constant that depends on the geometric complexity of the surface. We achieve this by computing a nested bisection of the mesh surface using separator curves and storing compactly-described functions approximating the distances between each mesh vertex and a small relevant subset of these curves. Using this database, the geodesic distance between two mesh vertices can then be approximated well by solving a small number of simple univariate minimization problems in $O(m\log n)$ worst case time and $O(m)$ average case time. Our method provides an excellent tradeoff between the size of the database, query runtime, and accuracy of the result. It can be used to compress exact or approximate geodesic distances, for example, those obtained by VTP (exact), fast DGG, fast marching, or the heat method (approximate) and is very efficient if $f(n)=n$, as for the fast DGG method.

Citation Info

Conference
SIGGRAPH Asia

Location
Daegu, December 2022

Publisher
ACM

Pages
Article 4, 9 pages

DOI
10.1145/3550469.3555412

1 Introduction

Computing geodesic paths and geodesic distances on 3D surfaces approximated as triangle meshes has been the subject of intense study over the past few decades. Since computing the exact (piecewise-linear) geodesic paths on the polyhedral surface is quite complicated [15, 18, 22], a variety of methods that *approximate* the geodesic paths and distances have been developed [13, 20, 24, 26], the most well-known being the *fast-marching* [11] and the *heat* method [6, 23]. While these methods are based on discretizing and solving a partial differential equation, the idea of *graph-based* methods [14, 16, 25, 28] is to augment the mesh graph with additional edges such that it is still sparse. This new graph is computed and stored in a preprocessing step, and geodesic distance queries can then be handled by finding shortest paths in this graph.

The main objective of all these methods is to compute the so-called *distance field* of a given *source* vertex, that is, the geodesic distance from the source to *all* other mesh vertices. For example, the graph-based, fast

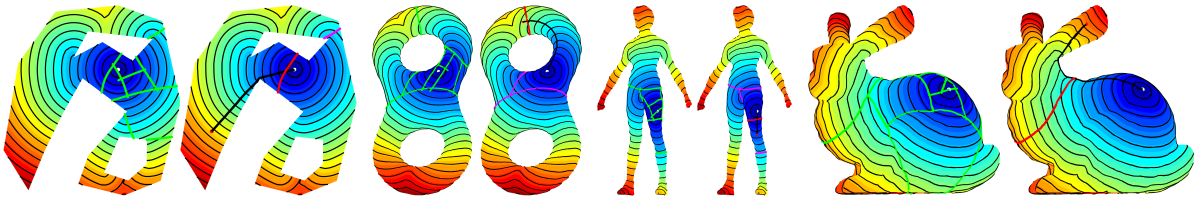


Figure 1: Our algorithm quickly computes the geodesic distance between arbitrary pairs of points on a 3D mesh. After recursively bisecting the mesh surface using separator curves, we precompute for each vertex (white) a distance field (using *any* state-of-the-art method) for all the sub-patches that contain the vertex, use them to determine the distance *functions* to a small subset (green) of these curves (left, for each of the four examples), and store their descriptions in a database. At query time (right, for each example), we efficiently compute the geodesic distance to any other vertex (red) by exploiting the fact that the geodesic path (black) between the vertices crosses at least one (red) of all the separator curves that the two vertices have in common (magenta). The high accuracy of our method can be observed by comparing the resulting distance field (right) with the distance field given by the method (here, VTP) used in the preprocessing step (left).

DGG algorithm [1] computes an approximate geodesic distance field for a triangle mesh with n vertices in $O(n)$ time, by employing a tailored version of Dijkstra's algorithm [7]. Given this field, it is then possible to generate geodesic paths from any vertex to the source by applying a gradient-descent-type tracing method to this distance field. For a comprehensive overview of geodesic path and distance algorithms, the interested reader is referred to the survey by Crane et al. [5].

The main problem with the geodesic field methods is that they are too slow to use in interactive applications where rapid computation of arbitrary point-to-point geodesic distances on very large meshes is required, since generating the entire distance field on these meshes could take seconds on state-of-the-art machines. To combat this, a number of methods have been developed that *compress* and *store* the geodesic distance information in a preprocessing step and then use this data to rapidly compute the approximate geodesic distance between any pair of vertices on demand. For these methods, there is typically a tradeoff between four performance measures: 1) preprocessing time, 2) storage size, 3) online query time, 4) result accuracy. Preprocessing time is usually of minor concern, since this is done only once when the data set is prepared. Storage size is of major concern, as it is undesirable to amplify the size of the original mesh dataset to the extent that it becomes unmanageable. However, larger storage sizes will usually result in faster and more accurate queries. As an extreme example, computing and storing the values of all $O(n^2)$ geodesic distances between all possible pairs of vertices in an n-vertex triangle mesh will result in prohibitive $O(n^2)$ storage, but perfectly accurate geodesic queries in $O(1)$ time (by simple lookup). Xin et al. [27] describe a sample-based method requiring $O(m^2+n)$ storage, with $O(1)$ time queries and $O(1/\sqrt{m})$ absolute accuracy, where m is the user-controlled number of samples. Burghard and Klein [3] describe a landmark-based method requiring $O(mn)$ storage with $O(m)$ time queries and $O(1/m)$ relative accuracy, where m is the number of landmarks.

1.1 Contribution

Our method falls in the category of *compression methods*. It can compress the geodesic distance information (exact or approximate) determined by *any* geodesic field method into a compact database and answer geodesic distance queries between any two mesh points in real-time. Our method has a user-controllable parameter m, requires $O(mn\log n)$ storage and results in $O(m\log n)$ time queries, with relative accuracy that seems, based on our experiments, to scale like $O(1/m^p)$, for some integer $p>1$. Our results show that an average relative error of less than 0.5%, with respect to the (exact or approximate) geodesic distances that our method compresses, can typically be achieved for $m=12$ and that the choice of m and the resulting accuracy do not depend on n, but rather on the geometric complexity of the surface that is approximated by the mesh.

Our method takes advantage of the fact that manifold 3D meshes have an edge structure similar to that of a planar graph, for which it is well known that compact separators exist, namely it is possible to partition a connected mesh into two unconnected balanced components by removing a compact set of *separating vertices* or *cut edges* [12]. By *balanced* we mean a ratio of no worse than $1:2$ in size, and by *compact* we mean that the number of separating vertices or cut edges is $O(\sqrt{n})$ for an n-vertex mesh. The key importance of such a separator in computing geodesic distances is the fact that the geodesic path between two vertices x and y on either side of the separator *must* pass through the separator, thus the geodesic distance $d(x,y)$ is the minimum of the distances $d(x,z)+d(z,y)$ for all z associated with the separator (Section 2.1). So, if we precompute $d(x,z)$ for all x and all z on the separator, it is relatively easy to compute $d(x,y)$ at query time. This principle may be generalized by further partitioning each component of the mesh in a recursive manner, constructing a so-called binary *nested bisection* tree [8]. Thus, geodesic distances between any pair of vertices may be computed by applying the same logic while traversing a path of the tree and examining all associated separators, whose number is $O(\log n)$ (Section 2.2). Nested bisection trees are widely used to implement recursive divide-and-conquer algorithmic strategies for graphs, for example, for efficiently solving linear systems in geometry processing [9]. On a 3D mesh, which represents an underlying continuous surface, the ideas are very similar, although care must be exercised to take into account that geodesic paths may cut through the graph edges and faces.

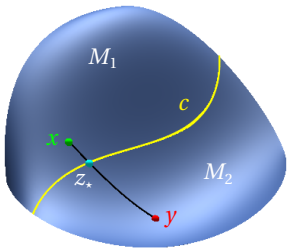
1.2 Overview

After deriving the theoretical background in the case of general manifold surfaces (Section 2) and proving the correctness of our method (Theorem 2.1), we discuss how this transfers to the case of discrete triangle meshes (Section 3), provide an error bound on our approximation (Lemma A.3), and explain the details of our implementation (Section 4). Finally, we present some experimental results for meshes with various geometric complexities (Section 5), before concluding and discussing future extensions of our method (Section 6).

2 Geodesic distances on surfaces

For any two points x and y on a manifold 3D surface M, the *geodesic distance* $d(x,y)$ is the length of a shortest path between x and y in M, where the length is measured with respect to the Riemannian metric of M. It is well known that the geodesic distance function satisfies the triangle inequality $d(x,y) \le d(x,z)+d(z,y)$, with equality if and only if z is a point on a shortest path between x and y.

2.1 Divide ...



Suppose that M is topologically equivalent to the disk and that c is a simple curve with endpoints on the boundary of M, such that c partitions M into two surfaces M_1 and M_2, which are themselves topologically equivalent to the disk (see inset). Given two points $x \in M_1$ and $y \in M_2$, clearly any path between x and y crosses c at least once. In particular, this is true for any shortest path, hence there exists a point $z_\star$ on c, such that $d(x,y) = d(x,z_\star)+d(z_\star,y)$. It then follows from the triangle inequality that the geodesic distance between x and y satisfies

$$d(x,y) = \min_{z \in c} \big(d(x,z) + d(z,y) \big).$$

Consequently, the geodesic distance between any points $x, y \in M$ can be found by solving a simple minimization problem, as long as

1. x and y are on different sides w.r.t. c, and
2. we know $d(x,z)$ for any $x \in M$ and any $z \in c$.

Note that this observation extends to the case where c is a loop (i.e., a closed curve), so that M_1 and M_2 are topologically equivalent to the disk and the cylinder, and even to the case where c is a *set* of curves, resulting in M_1 or M_2 consisting of more than one connected component. For example, if M is the surface of a human body, partitioned by two loops around the legs into M_1 consisting of the two (unconnected) legs, and M_2 consisting of the rest of the body. Multiple separating curves will also arise with surfaces having arbitrary topology, for example, partitioning the torus into two unconnected surfaces may require two separating curves (see Figure 2).

To turn this insight into an efficient strategy for answering point-to-point geodesic distance queries, it remains to deal with the two requirements mentioned above.

2.2 ... and conquer

The limitation that x and y must be on different sides of c, which covers at most 50% of the cases for randomly chosen points, can be easily overcome by partitioning M recursively into smaller and smaller surface patches using separators (see Figure 2).

A *surface patch* of M is a (set theoretically) closed subset $P \subseteq M$, possibly with multiple components. A *separator* S of a surface patch P is either a *path* (a simple curve connecting two boundary points of P), or a *loop* (a simple closed curve in P), or a finite collection of such curves, so that $P \setminus S$ consists of two unconnected components Q_1 and Q_2. Denoting the closures of these components by P_1 and P_2, we say that S *partitions* P into two disjoint (apart from common boundaries) sub-patches P_1 and P_2, such that $P_1 \cup P_2 = P$ and $P_1 \cap P_2 = S$. If P has multiple components, then it is possible to assign some components entirely to P_1 and all other components to P_2, so that S is empty.

A recursive partition or *nested bisection* of M can then be represented as a so-called *binary bisection tree* T, where each node N of T represents both a surface patch $P(N)$ and a separator $S(N)$ that partitions $P(N)$ into two sub-patches, with the latter corresponding to the children of N in T (see Figure 2). The root of T represents the entire surface M and an associated top-level separator. The recursive bisection terminates whenever a patch is small enough in some meaningful sense, so that the leaf nodes of T represent only such a small patch, but no separator. Clearly, for any surface point $x \in M$ there exists at least one corresponding leaf node N of T, such that $x \in P(N)$, and unless x is a point on at least one of the separators, this *leaf patch* $P(N)$ is unique.

Assume that all separator curves are parameterized, for example by arc length over the interval $I_c = [0, L_c]$, that is, $c : I_c \to M$, where L_c is the length of c. For any $x \in M$, let N_x be a corresponding leaf node in T, Γ_x

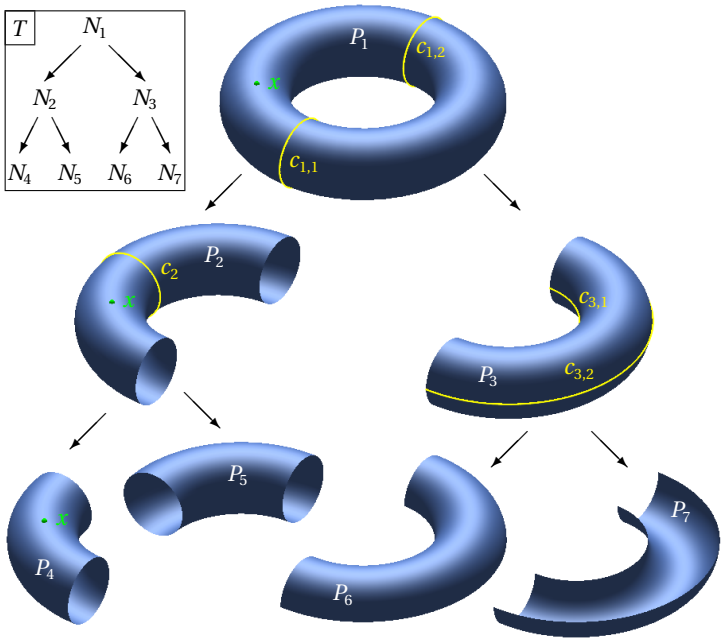


Figure 2: Recursive bisection of a surface $M = P_1$ into patches. The top-level separator consists of two loops, $S_1 = \{c_{1,1}, c_{1,2}\}$, and partitions the surface into the two patches P_2 and P_3. The patch P_2 is further partitioned into P_4 and P_5 by the single loop separator $S_2 = \{c_2\}$, and P_3 is partitioned into P_6 and P_7 by two paths $S_3 = \{c_{3,1}, c_{3,2}\}$. The partitions can be represented as a binary bisection tree T with nodes N_i (upper left), with each node representing a patch, $P_i = P(N_i)$, and, except in the case of leaf nodes, a separator $S_i = S(N_i)$. The leaf node associated with the indicated point $x \in M$ is $N_x = N_4$ and the path from N_x to the root of T is $\Gamma_x = (N_4, N_2, N_1)$.

be the path in T from N_x to the root of T, and $C_x = \{c \in S(N) : N \in \Gamma_x\}$ be the set of all curves belonging to separators associated with the nodes in Γ_x. For any $c \in C_x$, let

$$d_x^c(t) = d_{P(N)}(x, c(t)), \qquad t \in I_c \tag{1}$$

denote the geodesic distances from x to the points of c *within* the surface patch $P(N)$ associated with c, that is, where N is the node of T, such that $c \in S(N)$. We can then compute geodesic distances in M based on the following theorem.

Theorem 2.1. *Suppose that $x, y \in M$ are two surface points with $N_x \neq N_y$. Let $\Gamma = \Gamma_x \cap \Gamma_y$ be the common tail of the paths Γ_x and Γ_y and $C = C_x \cap C_y$ be the set of all curves belonging to separators associated with the nodes in Γ. Then the geodesic distance between x and y is*

$$d(x, y) = \min_{c \in C} d^c(x, y), \tag{2}$$

where

$$d^c(x, y) = \min_{t \in I_c} \big(d_x^c(t) + d_y^c(t) \big) \tag{3}$$

is the length of the shortest path from x to y via *c and* within *the surface patch $P(N)$ associated with c.*

Proof. By the triangle inequality, $d(x, y) \leq d^c(x, y)$ for all $c \in C$, with equality if and only if c is crossed by a shortest path between x and y and if this shortest path is contained in the surface patch $P(N)$ associated with c. It remains to show that this is true for some $c_\star \in C$, which we call the *determining separator curve*. To this end, let γ be a shortest path from x to y and consider the root node N of T, which is the last node of the path Γ. Obviously, γ is contained in $P(N) = M$. If γ intersects the top-level separator $S(N)$, then $c_\star$ belongs to $S(N)$, and we are done. Otherwise, γ is contained entirely in one of the sub-patches generated by the separator $S(N)$. By construction, this sub-patch is represented by the node N' that precedes N in Γ. We can now repeat the previous argument for $P(N')$ and $S(N')$, until we find $c_\star$. This happens at latest when we arrive at the first node of Γ, because x and y are, by construction, in different sub-patches of the corresponding surface patch. □

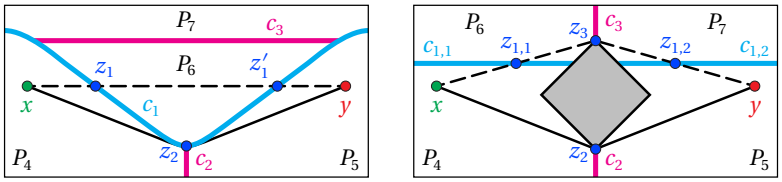


Figure 3: Two examples in the plane, where the determining separator curve (cyan) is not associated with a lowest-level separator (magenta). In both examples, the bisection tree T is as in Figure 2 (upper left), $x \in P_4$, $y \in P_5$, $\Gamma = (N_2, N_1)$, and the shortest path (solid) between x and y within $P_2 = P_4 \cup P_5$ crosses the lowest-level separator curve c_2 at z_2. This path is longer than the globally shortest path γ (dashed), which is not contained in P_2 and determined by a curve from the top-level separator. In the left example, γ crosses c_1 at z_1 and at z_1', indicating that the function $d_x^{c_1}(t) + d_y^{c_1}(t)$ has two local minima, both with magnitude $d(x,y)$. In the right example, which contains a hole (gray), γ crosses the top-level separator curves $c_{1,1}$ and $c_{1,2}$ at $z_{1,1}$ and $z_{1,2}$, respectively. Consequently, the geodesic distance $d(x,y)$ can be found by minimizing either of the two functions $d_x^{c_{1,1}}(t) + d_y^{c_{1,1}}(t)$ and $d_x^{c_{1,2}}(t) + d_y^{c_{1,2}}(t)$. Note that c_3 is not a determining curve, even though γ crosses c_3 at z_3, because $c_3 \notin C$, that is, neither of x or y are aware of c_3.

Remark 2.1. In most cases, the determining curve $c_\star$ belongs to the lowest-level separator $S(N_\star)$ associated with the first node $N_\star$ of Γ, because this is the only separator with respect to which x and y belong to different sub-patches, so that any path between x and y within $P(N_\star)$ must cross $S(N_\star)$. For any other node $N \in \Gamma$, the separator $S(N)$ does not actually separate x and y, because both vertices belong to the same sub-patch with respect to $S(N)$. Still, it may happen that the shortest path between x and y within $P(N)$ crosses $S(N)$, either because some $c \in S(N)$ is not a shortest path itself (see Figure 3, left), or because $S(N)$ consists of more than one curve, for example if $P(N)$ contains (geometric or topological) holes (see Figure 3, right). In this case, the globally shortest path between x and y is not contained in $P(N_\star)$ and $c_\star$ is associated with some other node of Γ. For that reason, we need to consider *all* $c \in C$ in (2).

If T is a perfect binary tree with height h and all the 2^h leaf patches are of equal size, then the probability that $N_x = N_y$ for two randomly chosen points x and y is $1/2^h$. Hence, the geodesic distance $d(x,y)$ can be determined as in Theorem 2.1 in more than 99% of the cases, provided that $h \geq 7$.

3 Geodesic distances on 3D meshes

Computing geodesic distances as proposed in Theorem 2.1 requires to know $d_x^c(t)$ in (1) for all $t \in I_c$ and all $x \in M$. Since both I_c and M are uncountable sets, it is in general impossible to compute and store this data in a preprocessing step. However, if M is a (discrete) manifold 3D mesh and we are interested only in *approximate* geodesic distances between the *vertices* of M, then we can turn the idea of Theorem 2.1 into an efficient algorithm for point-to-point queries with excellent accuracy at moderate storage cost.

3.1 Preprocessing

Given an n-vertex mesh M, the first task is to build the binary bisection tree T, that is, to find separators that recursively partition M into smaller and smaller patches until all patches have at most $V_{\max}$ vertices. These separators consist of (piecewise-linear) curves on the mesh surface, cutting through arbitrary mesh faces and edges and preferably *not* through vertices. During this process we make sure that all separator curves are assigned a unique index and likewise for all leaf patches. For each vertex $x \in M$, we then store:

1. The index ℓ_x of the leaf patch that contains x.
2. The set J_x of indices of all separator curves in C_x. These are the curves *associated* with x.
3. For any curve c with index in J_x, the m parameters of a function $\hat{d}_x^c : I_c \to \mathbb{R}$ that approximates the function $d_x^c : I_c \to \mathbb{R}$.

Under the assumptions that T is balanced and that all separators consist of $O(1)$ separator curves, we end up with $O(m \log n)$ data that needs to be stored for each vertex, that is, $O(mn \log n)$ overall.

We may further store the exact geodesic distance $d(x,y)$ for every pair of vertices x, y with $\ell_x = \ell_y$ in a sparse symmetric matrix $D \in \mathbb{R}^{n \times n}$. For each of the approximately $n/V_{\max}$ leaves of T, the matrix D contains at most ${V_{\max}}^2$ distances (for all vertex pairs within the leaf patch), so the total number of non-zero values in D is about $n V_{\max}$. Taking $V_{\max} = O(m \log n)$, the complexity of storing D is $O(mn \log n)$, hence does not dominate the storage complexity of the separator distance data.

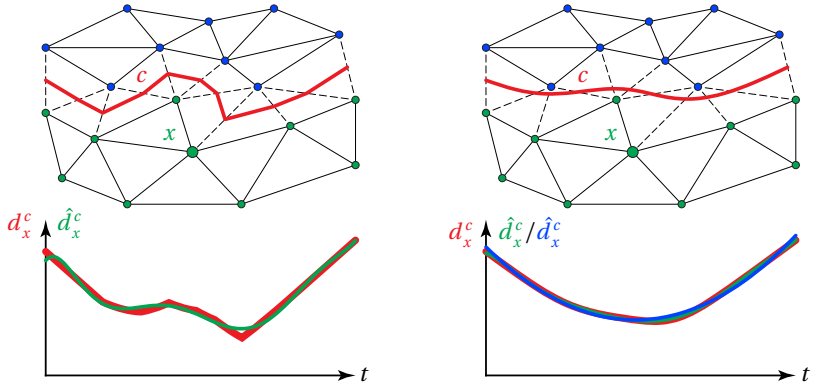


Figure 4: For the recursive bisection of a given (planar) mesh, we find a set of cut edges (dashed) to partition a patch into two sub-patches of roughly the same size (top). For the piecewise-linear separator curve c that connects the midpoints of the cut edges (top left), the separator distance functions d_x^c (red) are only piecewise smooth (bottom left). The best fitting polynomial of degree 7 (green) has a relative error of up to 24%. After smoothing the separator curve (top right), the distance function can be approximated by a degree 7 polynomial with a relative error of less than 1.5%, and even the best-fitting quadratic polynomial (blue) has a relative error of at most 7% (bottom right).

3.2 Online point-to-point queries

Equipped with this data, the algorithm for computing the geodesic distance between arbitrary mesh vertices x and y now becomes very simple. If $\ell_x = \ell_y$, then we either look up $d(x,y)$ in D or use the Euclidean distance $\|x-y\|$ between both vertices as an approximation of $d(x,y)$. Otherwise, we follow Theorem 2.1 and first compute for every $j \in J = J_x \cap J_y$ the minimum of $\hat{d}_x^c(t) + \hat{d}_y^c(t)$, where c is the curve with index j, that is,

$$\hat{d}_j = \min_{t \in I_c} \big(\hat{d}_x^c(t) + \hat{d}_y^c(t) \big), \tag{4}$$

and then find the minimum of these values to get the approximation

$$\hat{d}(x,y) = \min_{j \in J} \hat{d}_j \tag{5}$$

of the geodesic distance $d(x,y)$. We can further improve this algorithm, if we store, in addition to the data mentioned in Section 3.1, for every $x \in M$ and any c with index in J_x the minimum

$$m_x^c = \min_{t \in I_c} \hat{d}_x^c(t)$$

of $\hat{d}_x^c$. Then, while processing a curve c with index in J, we first check if $m_x^c + m_y^c$ is greater than the smallest $\hat{d}_j$ that we have found with other curves so far, and ignore the current curve in this case. This *quick reject* strategy is most effective if we process the separator curves bottom-up, starting with the lowest-level separator curves associated with the first node of Γ (see Remark 2.1).

If we assume that the minimum in (4) can be found in $O(m)$ time, then the time complexity for this query is $O(m \log n)$ in the worst case and $O(m)$ on average (see Lemma A.2). As for the accuracy of the query result, we can prove that it is controlled entirely by the approximation error ε of the functions $\hat{d}_x^c$ (see Lemma A.3), which in turn is inversely proportional to the parameter m. In practice, the user specifies the desired ε and the algorithm determines $m(\varepsilon)$.

4 Implementation details

We have implemented our method in MATLAB 2021b using standard routines and packages. For the computation of geodesic distance fields during preprocessing, we used the available VTP [18] and fast DGG [1] software in C++ for exact and approximate geodesic distances, respectively, as well as Gabriel Peyré's MATLAB code [17] for the *fast-marching* (FM) method [11] and the *geodesics in heat* (HEAT) method [6].

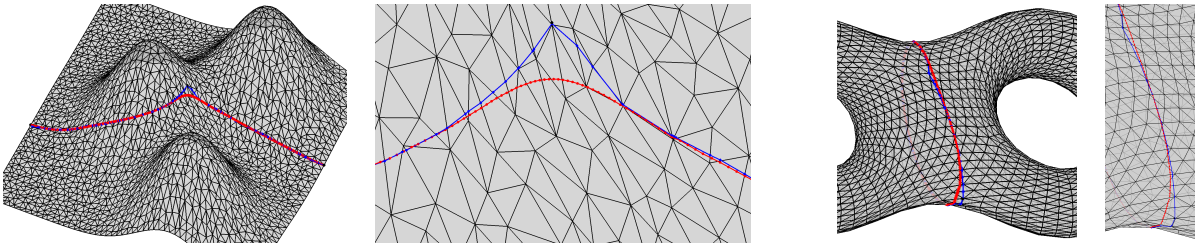


Figure 5: Piecewise-geodesic separator curves consisting of two (for paths; left) or three geodesics (for loops; right) before (blue) and after (red) refinement and smoothing.

4.1 Computing the binary bisection tree

Recursive bisection is performed by successive *graph cut* computations using a simple Fiedler-vector-based method [21], although any other graph cut algorithm could be used (see the survey by Buluç et al. [2]), some implemented quite efficiently in the METIS software package [10]. Given a triangle mesh patch P with n vertices, this method finds a compact set of *cut edges*, such that removing these edges from P partitions P into two (relatively balanced) sub-patches P_1 and P_2.

At this point, one could take the piecewise linear curves that connect the midpoints of the cut edges in the correct order as separator curves, but the resulting separator distance functions d_x^c will not be approximated well by low-degree polynomials (see Figure 4, left). Much better approximations, both in terms of smaller error and lower degree, can be achieved if the separator curves are smooth (see Figure 4, right). We therefore select three evenly spaced points along the edge cut (including the endpoints in the case of open curves) and connect them by geodesic paths, which are also piecewise linear curves, defined by a sequence of edge intersection points. To smooth out the interior points where these geodesics meet, we simply sub-sample by a factor of three and repeatedly filter and project the samples back to the mesh a few times (see Figure 5), but other techniques, like tangential smoothing, could be used, too.

The result are ordered sets of regularly-spaced *separator samples* along smooth curves that partition P into two sub-patches, which may be a little different than the two sub-patches defined by the original edge cut, since the separator curves may have moved somewhat on the mesh surface as a result of the smoothing process. Since partitioning P exactly along the separator curves would require splitting all triangles traversed by the separator into an excessive amount of sub-triangles, we rather keep these triangles intact and add them to both sub-patches for further processing, marking the vertices that do not belong to the patch as "ghost" vertices.

Figure 6 shows some examples of the resulting bisections. In all our examples, we used a threshold of $V_{\max} = 30$ vertices for terminating the recursive bisection. This consistently gave leaf patches that were sufficiently close to planar, so that the Euclidean distance between vertex pairs from the same leaf patch was within our accuracy tolerance, and we could thus avoid storing the matrix D.

4.2 Approximating the separator distance functions

While building the bisection tree, we determine for every patch P the set $V(P)$ of vertices of all triangles that contain a separator sample, and we compute for every $v \in V(P)$ the geodesic distances $d(v,x)$ to all vertices $x \in P$, using either VTP to obtain exact distances or one of the geodesic field methods (DGG, FM, HEAT) to get approximate distances. If the computation of a single geodesic distance field in a mesh with n vertices costs $O(f(n))$ time, then the overall time of this preprocessing stage is $O(f(n)\sqrt{n})$ for compact separators (see Lemma A.1). With this data at hand, we then approximate the separator distance functions d_x^c for every mesh vertex $x \in M$ and all separator curves c with index in J_x as follows.

Following Rustamov et al. [19], we first approximate the geodesic distances $d(x,s)$ from x to all separator samples $s \in c$ as

$$d(x,s)^2 \approx \hat{d}(x,s)^2 = \sum_{i=1}^{3} \lambda_i d(x,v_i)^2 - \sum_{i=1}^{3} \sum_{j=i+1}^{3} \lambda_i \lambda_j \|v_i - v_j\|^2,$$

where $[v_1, v_2, v_3]$ is the triangle containing s and $\lambda_1, \lambda_2, \lambda_3$ are the barycentric coordinates of s, that is, $s = \sum_{i=1}^{3} \lambda_i v_i$ and $\sum_{i=1}^{3} \lambda_i = 1$. This approximation is exact in the plane and very accurate in general.

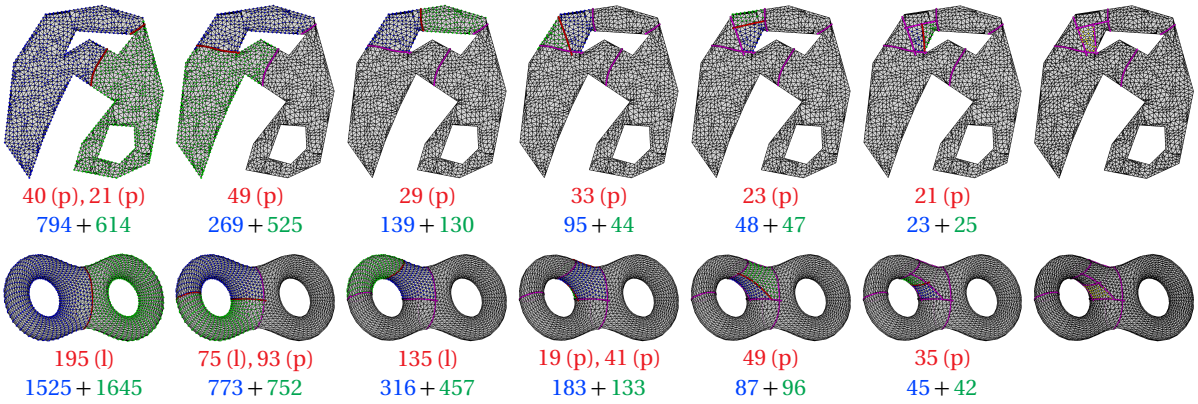


Figure 6: Visualization of the nodes along the leftmost branch of the binary bisection tree T for the *Holes* (top) and the *Eight* (bottom) meshes (cf. Figure 7), from the root node (left) to the leaf node (right). Each node N represents a patch P and (except for leaf nodes) a separator S consisting of one or more separator curves, represented by separator samples (red), which partition P into a left (blue) and a right (green) sub-patch. The numbers below each image state the number of samples per separator (red) and its type (**p**ath or **l**oop), as well as the number of vertices in the sub-patches (blue, green). During the construction of T, we compute the approximate separator distance functions $\hat{d}_x^c$ for all vertices x in P (blue and green) and all separator curves $c \in S$ (red). At the end of this process, we will have collected and will store for all vertices x (yellow) in the leaf patch, the coefficients of the functions $\hat{d}_x^c$ for all c with index in J_x, that is, for the curves belonging to all parent separators (magenta). These are the curves associated with x.

An obvious option, to which we refer to as "SEPsam" below, is to now store *all* the values $\hat{d}(x,s)$ along with the (parametric) distances of the samples along c and take the piecewise linear interpolant of these values as $\hat{d}_x^c$ for answering the online queries. However, this leads to unnecessarily large databases.

A much better alternative (which we refer to as "SEPfun") is to approximate d_x^c by an analytic function which can be described compactly with few parameters. Since the separator curves are regular and smooth, the separator distance functions tend to be smooth and relatively well-behaved, thus not too difficult to approximate. For example, it is well-known that a function that is p times continuously differentiable can be approximated by a degree-m polynomial with $O(1/m^p)$ error [4]. We choose the best between the polynomial approximating the distances $\hat{d}(x,s)$, and the polynomial approximating the squared distances $\hat{d}(x,s)^2$. The reason for the latter is that this yields a better approximation when the vertex is very close to the separator. The quality of the approximation is measured as the average relative error, thus the least-squares polynomial fit is done using weighted least squares, where the weight of a sample point s is $1/\hat{d}(x,s)$. Starting from a quadratic fit, we increase the degree until either the approximation error falls below a threshold (we used 0.5%) or the maximum degree allowed (we used $m_{\max} = 12$) is reached. For the resulting function $\hat{d}_x^c$ we then store the vector of $m+1$ polynomial coefficients and one bit to distinguish between approximation of distance vs. squared distance in the database. In the case of loop separators, since the polynomial fit is on an interval, we found it is best to shift the samples (and distances), so that the maximal distance is at the boundary of the interval, and to add the shift offset to the database entry. The overall time required for this preprocessing stage is $O(m^3 n\sqrt{n})$, where $m = m_{\max}$ (see Lemma A.1). Clearly, since the complexity of the separator distance functions d_x^c (in terms of number of local extrema) tends to increase with the geometric complexity of the shape that the mesh represents, larger values of $m_{\max}$ may be required to approximate them with the desired accuracy in those cases, but the interplay between $m_{\max}$ and the accuracy is *independent* of n.

4.3 Computing approximate geodesic distances

At query time, we need to compute the minimum of $\hat{d}_x^c + \hat{d}_y^c$ for all c associated with both x and y. For SEPsam, this is done by finding the minimum of sums of sample distances. For SEPfun, our implementation uses the "brute force" method of sampling the parameter interval at a regular spacing of 0.005 (after scaling the 3D mesh to fit within the unit cube), and exhaustively searching for the minimum among evaluations of the function at these parameter values, each costing $O(m)$ time. It would be straightforward, and more efficient, to employ a more sophisticated gradient descent or Newton method, since the analytic description of the functions and their derivatives are available, but this might require a few random starts to avoid getting stuck in local minima.

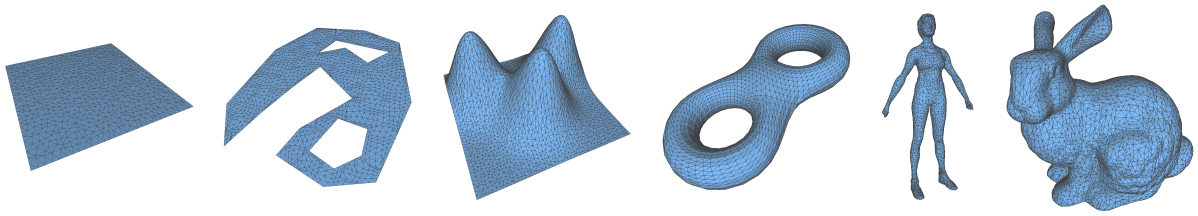


Figure 7: Meshes used for the evaluation and comparison in Table 1.

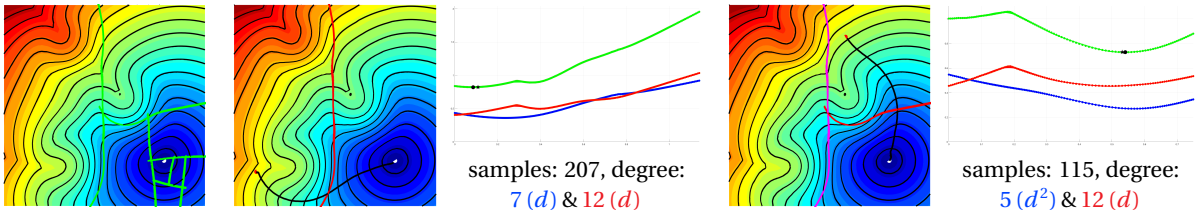


Figure 8: Exact geodesic distance field (left) for a vertex x (white) of the *Peaks* mesh and its associated separators (green), compared to the approximate geodesic distance field computed with our SEP method, as well as two examples of the approximate separator distance functions $\hat{d}_x^c$ (blue) and $\hat{d}_y^c$ (red) and their sum (green) for the separator curve c (red) that determines the geodesic distance from x to y (red), with the number of samples used by SEPsam and the degrees and types of polynomial fits (distance vs. squared distance) used by SEPfun. SEPsam finds the minimum at the black dot and SEPfun the one at the black star.

5 Experimental results

In our first experiment, we used our separator-based method (SEP) on a number of 3D meshes (see Figure 7) for compressing exact geodesic distances, computed with VTP. We used only modest-size meshes, because our unoptimized preprocessing code requires many calls to the external VTP code, and is therefore relatively slow on large 3D meshes. Although SEP is designed specifically for fast geodesic distance queries on vertex pairs, we evaluated the performance by computing distance fields, simply by calling SEP $n-1$ times. Some examples of the resulting distance fields are given in Figure 1 as well as in Figure 8, which also shows some of the approximating separator distance functions that are stored in the database. To quantify the accuracy of a distance field of a source vertex x we used the L_1 norm (ME) and the L_2 norm (RMSE) of the relative error per vertex,

$$\mathrm{ME}(x) = \frac{1}{n-1}\sum_{y\in M} e(x,y), \qquad \mathrm{RMSE}(x) = \sqrt{\frac{1}{n-1}\sum_{y\in M} e(x,y)^2}, \qquad e(x,y) = \frac{|d(x,y)-\hat{d}(x,y)|}{d(x,y)},$$

since both were used in previous works. The reported accuracy was computed as the average of these errors over 100 randomly chosen source vertices x. We compare the accuracy of SEP to the one obtained by our implementation of the simple landmark-based algorithm (LAND) of Burghard and Klein [3], which the authors claim is superior to the previous method by Xin et al. [27], using a number of landmarks m equal to the cost per vertex of SEPfun. As summarized in Table 1, the storage cost of our SEPfun method is more than an order of magnitude smaller than with SEPsam, with almost the same accuracy, which in turn is more than ten times better than the accuracy obtained by LAND. We further note that the binary bisection trees are nicely balanced and that the error and storage cost for SEP tend to increase with the geometric complexity of the shape, as expected.

In a second experiment, we used SEPfun with $m_{\max} = 12$ and $V_{\max} = 30$ to compress approximate geodesic distances obtained by fast DGG and FM for different resolutions of the *Bunny* mesh. In all cases, the mean relative errors (with respect to the DGG or FM distances) were on the order of 0.5% (ME) and 1.1% (RMSE), and our unoptimized MATLAB code was able to answer several thousand queries per second on a Windows 11 laptop with 3.0 GHz Intel Core i7-1185G7 processor and 32 GB RAM. The plots in Figure 9 confirm the theoretical asymptotic time and space complexities. Building the binary bisection tree T, computing the sample distances $\hat{d}(x,s)$, and fitting the polynomial distance functions $\hat{d}_x^c$ account for roughly 10%, 40%, and 50% of the preprocessing time, respectively.

mesh	size	SEP		SEPsam			SEPfun			LAND		
M	n	$V_{\max}$	av/max h	ME	RMSE	cost	ME	RMSE	cost	ME	RMSE	m
Square	900	30	5.5/6	0.0	0.2	289.8	0.1	0.2	16.8	14.3	40.8	17
Holes	1,408	30	6.2/8	0.1	0.5	254.7	0.1	0.6	26.0	8.5	34.2	26
Peaks	2,500	30	7.1/9	0.1	0.2	544.4	0.2	0.3	37.5	8.7	32.3	37
Eight	3,070	30	8.0/12	0.1	0.5	761.8	0.3	0.7	65.3	5.3	21.2	65
Female	4,040	30	9.3/13	0.2	0.9	789.9	0.4	0.9	86.1	3.7	19.5	86
Bunny	5,051	30	8.8/12	0.3	1.1	920.9	0.4	1.3	72.2	4.0	18.9	72

Table 1: Comparison of the accuracy of our separator-based method (SEP) using separator distance samples (SEPsam) or polynomial separator distance functions with $m_{\max} = 12$ (SEPfun) with the landmark-based (LAND) algorithm for computing geodesic distance fields using a database of exact geodesic distances on several triangle meshes M (see Figure 7). The mean L_1 relative error (ME) and the mean L_2 relative error (RMSE) are given in %. For SEP, we report the average and maximum depth h of the binary bisection tree T (with up to $V_{\max}$ vertices in each leaf patch) and the number of (double) values stored per vertex (cost), implying that the size of the database is $(8n \cdot \text{cost})$ bytes. For LAND, we state the number m of landmarks used, chosen such that the SEPfun and LAND databases have similar size.

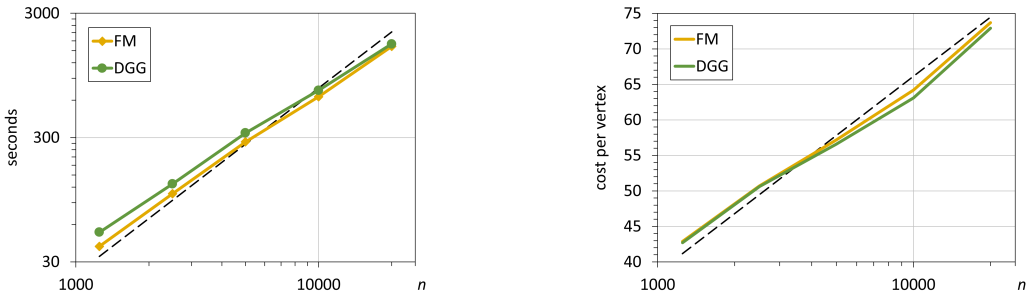


Figure 9: Preprocessing time in seconds (left) and cost per vertex (right) of our SEP method for compressing geodesic distances on the *Bunny* mesh for different values of n, compared to $O(n\sqrt{n})$ and $O(\log n)$, respectively (dashed).

Since our code uses file I/O to obtain geodesic distance field data from an external "black box", the preprocessing is quite slow and requires about 30 minutes for a mesh with $n = 20,000$ vertices using DGG or FM, even though both methods approximate a single distance field in $O(n)$ time. For VTP, we can handle only $n = 5,000$ vertices in the same time, because it takes $O(n^2)$ time to compute an exact distance field with VTP. However, we expect an optimized and integrated implementation to be much faster, because all unnecessary I/O would be eliminated, and some of the black box internal computations (e.g. as done by DGG) can be parallelized. Furthermore, our preprocessing can also benefit from parallelization, since it may perform its many calls to the distance field "black box" in parallel.

6 Conclusion and discussion

We have presented a novel method to rapidly answer pairwise geodesic distance queries on 3D mesh surfaces. The most complex step of the preprocessing stage, where the database required to support the online queries is constructed, is the computation of geodesic distances between mesh vertices and separator samples, which can be done using any existing algorithm for computing exact or approximate geodesic distance fields. The error introduced by our compression and query method is very small.

We use polynomials to approximate the distance functions. This was the simplest choice, but it is certainly possible to use any other set of approximating functions, such as rational functions or trigonometric polynomials, which may have better approximation power. We also tacitly ignored the fact that the distance functions are not smooth at points where the separator curve intersects the cut locus. A better approach would be to detect these points and to approximate the resulting pieces individually.

We have stated that for SEP the storage cost *per vertex* scales like $O(m \log n)$. This is what happens when the resolution n of a mesh representing a given shape increases. In practice, for a given target relative error, the cost per vertex will probably be much less than that as n increases, since the nested bisection may be terminated much earlier, resulting in leaf patches whose sizes are much larger than $O(m \log n)$, as they will still be quite planar (and easily approximated by flat patches).

Our implementation may be made a little more robust by simplifying the graph cut bisection and subsequent generation of smooth separator samples. For example, it is possible to partition the mesh using axis-parallel planes to cut through the bounding box of the mesh (similar to a 3D-tree construction). Although the resulting separators will not be as compact and balanced as those generated by the more sophisticated method we use, they are much faster to generate and will be as smooth as the mesh surface, by construction.

References

[1] Y. Y. Adikusuma, Z. Fang, and Y. He. Fast construction of discrete geodesic graphs. *ACM Transactions on Graphics*, 39(2):Article 14, 14 pages, Apr. 2020. The FastDGG source code is available at https://github.com/GeodesicGraph/DGG-SVG.

[2] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In L. Kliemann and P. Sanders, editors, *Algorithm Engineering*, volume 9220 of *Lecture Notes in Computer Science*, pages 117–158. Springer, Cham, 2016.

[3] O. Burghard and R. Klein. Simple, robust, constant-time bounds on surface geodesic distances using point landmarks. In *Proceedings of the 20th International Symposium on Vision, Modeling and Visualization*, VMV '15, pages 17–24, Aachen, Germany, 2015. Eurographics Association, Goslar.

[4] E. W. Cheney. *Introduction to Approximation Theory*. Chelsea Publishing Company, New York, 2nd edition, 1982. ISBN 978-0-82840-317-7.

[5] K. Crane, M. Livesu, E. Puppo, and Y. Qin. A survey of algorithms for geodesic paths and distances, 2020, arXiv:2007.10430.

[6] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 32(5):Article 152, 11 pages, Sept. 2013.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, Dec. 1959.

[8] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[9] P. Herholz, T. A. Davis, and M. Alexa. Localized solutions of sparse linear systems for geometry processing. *ACM Transactions on Graphics*, 36(6):Article 183, 8 pages, Dec. 2017.

[10] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. The METIS source code is available at http://glaros.dtc.umn.edu/gkhome/views/metis.

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

[12] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, Apr. 1979.

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

[14] W. Meng, S. Xin, C. Tu, S. Chen, Y. He, and W. Wang. Geodesic tracks: computing discrete geodesics with track-based Steiner point propagation. *IEEE Transactions on Visualization and Computer Graphics*, 28(12):4887–4901, Dec. 2022.

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

[16] G. Nazzaro, E. Puppo, and F. Pellacini. *geoTangle*: interactive design of geodesic tangle patterns on surfaces. *ACM Transactions on Graphics*, 41(2):Article 12, 17 pages, Apr. 2022.

[17] G. Peyré. Numerical Tours in Matlab, 2022. http://www.numerical-tours.com/matlab/.

[18] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Transactions on Graphics*, 35(4):Article 125, 13 pages, July 2016. The VTP source code is available at https://github.com/YipengQin/VTP_source_code.

[19] R. M. Rustamov, Y. Lipman, and T. Funkhouser. Interior distance using barycentric coordinates. *Computer Graphics Forum*, 28(5):1279–1288, July 2009.

[20] N. Sharp and K. Crane. You can find geodesic paths in triangle meshes by just flipping edges. *ACM Transactions on Graphics*, 39(6):Article 249, 15 pages, Nov. 2020.

[21] D. A. Spielman and S.-H. Teng. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra and its Applications*, 421(2–3):284–305, Mar. 2007.

[22] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics*, 24(3):553–560, July 2005.

[23] 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*, 43(2):579–594, Feb. 2021.

[24] P. Trettner, D. Bommes, and L. Kobbelt. Geodesic distance computation via virtual source propagation. *Computer Graphics Forum*, 40(5):247–260, Aug. 2021.

[25] X. Wang, Z. Fang, J. Wu, S.-Q. Xin, and Y. He. Discrete geodesic graph (DGG) for computing geodesic distances on polyhedral surfaces. *Computer Aided Geometric Design*, 52–53:262–284, Mar.–Apr. 2017.

[26] S.-Q. Xin and G.-J. Wang. Efficiently determining a locally exact shortest path on polyhedral surfaces. *Computer-Aided Design*, 39(12):1081–1090, Dec. 2007.

[27] 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*, I3D '12, pages 31–38, Costa Mesa, CA, USA, 2012. Association for Computing Machinery, New York.

[28] X. Ying, X. Wang, and Y. He. Saddle vertex graph (SVG): a novel solution to the discrete geodesic problem. *ACM Transactions on Graphics*, 32(6):Article 170, 12 pages, Nov. 2013.

A.1 Preprocessing time

Lemma A.1. *Suppose that the binary bisection tree T is constructed using balanced and compact separators and that we use a geodesic field method that determines a single geodesic distance field in an n-vertex mesh in $O(f(n))$ time. Then computing the relevant local geodesic distance fields within $P(N)$ for all nodes $N \in T$ (see first paragraph of Section 3.1) costs $O(f(n)\sqrt{n})$ time, and all best-fitting polynomials $\hat{d}_x^c$ of degree at most m can be found in $O(m^3 n\sqrt{n})$ time.*

Proof. We first recall that a balanced and compact separator splits an n-vertex patch into two sub-patches with at least $n/3$ and at most $2n/3$ vertices each and such that the number of separating vertices is $O(\sqrt{n})$. Hence, a node $N \in T$ at level $k \geq 0$ represents a patch $P = P(N)$ with at most $n_k = (2/3)^k n$ vertices and a separator $S(N)$ with at most $O(\sqrt{n_k})$ separator samples, and the number of vertices in $V(P)$ is also $O(\sqrt{n_k})$. As the number of vertices in all patches at level k adds to n and $f(n)$ is at least $O(n)$, computing *one* local distance field in *each* patch at level k costs at most $O(f(n))$ time in total. Since we have to compute at most $O(\sqrt{n_k})$ local distance fields in each patch, the overall runtime is $O(f(n)\sqrt{n})$, because

$$\sum_{k=0}^{\infty} \sqrt{n_k} = \sum_{k=0}^{\infty} \sqrt{(2/3)^k n} = \left(3+\sqrt{6}\right)\sqrt{n}. \tag{6}$$

For each mesh vertex $x \in M$, we then approximate the distances to the separator samples $s \in c$ for all separator curves c with index in J_x. Since there are $O(1)$ such curves at every level k with $O(\sqrt{n_k})$ samples, this amounts, by (6), to approximating $O(\sqrt{n})$ distances, each in $O(1)$ time. For each c at level k, constructing and solving the normal equations for finding the least-squares polynomial fit of degree m to the $O(\sqrt{n_k})$ distance values $\hat{d}(x,s)$ takes $O(m^2\sqrt{n_k})$ time, and we do this for all degrees from $m = 2$ up to $m = m_{\max}$. By (6), and since

$$\sum_{m=2}^{m_{\max}} m^2 = O(m_{\max}^3),$$

the overall time for computing all polynomials is $O(m_{\max}^3\sqrt{n})$. □

A.2 Query time

Lemma A.2. *Suppose that the binary bisection tree T is constructed using balanced separators and that the minimum of the sum of two approximate separator distance functions $\hat{d}_x^c$ and $\hat{d}_y^c$ in (4) can be found in $O(g(n))$ time. Then the overall time complexity for querying the distance between x and y (see Section 3.2) is $O(g(n)\log n)$ in the worst case and $O(g(n))$ on average.*

Proof. Without loss of generality, we assume that $\ell_x \neq \ell_y$. Answering the distance query then requires to compute the minimum in (4) for all separator curves with index $j \in J = J_x \cap J_y$. Since there are $O(1)$ separator curves for every node in T, the size of J is $O(\eta)$, where η is the length of $\Gamma = \Gamma_x \cap \Gamma_y$. The worst case occurs if x and y are separated by a lowest-level separator, so that η equals the height of T, that is, $\eta = O(\log n)$.

On average, however, since all separators are balanced, the probability that x and y are separated by the top-level separator, so that $\eta = 1$, is at least $m_0 = 1 - \left(\frac{1}{3}\right)^2 - \left(\frac{2}{3}\right)^2 = \frac{4}{9}$. More generally, the probability for x and y to be separated by a separator at level k, so that $\eta = k+1$, is at least $m_k = \left(\frac{5}{9}\right)^k \frac{4}{9}$. Since

$$\sum_{k=0}^{\infty} m_k(k+1) = \frac{4}{9}\sum_{k=0}^{\infty}\left(\frac{5}{9}\right)^k (k+1) = \frac{9}{4}.$$

the average value of η is $O(1)$. □

A.3 Relative approximation error

Lemma A.3. *Suppose that the relative approximation error of $\hat{d}_x^c$ is bounded by $\varepsilon < 1$, that is,*

$$\frac{\left|\hat{d}_x^c(t) - d_x^c(t)\right|}{d_x^c(t)} \le \varepsilon, \qquad t \in I_c, \tag{7}$$

for all vertices $x \in M$ and any $c \in C_x$. Then the same error bound holds for the relative approximation error of the geodesic distance,

$$\frac{\left|\hat{d}(x,y) - d(x,y)\right|}{d(x,y)} \le \varepsilon \tag{8}$$

for all vertices $x, y \in M$.

Proof. Since $d_x^c(t) > 0$, it follows from (7) that

$$-\varepsilon d_x^c(t) \le \hat{d}_x^c(t) - d_x^c(t) \le \varepsilon d_x^c(t),$$

hence

$$(1-\varepsilon) d_x^c(t) \le \hat{d}_x^c(t) \le (1+\varepsilon) d_x^c(t), \tag{9}$$

which implies

$$d_x^c(t) \le \frac{1}{1-\varepsilon}\hat{d}_x^c(t). \tag{10}$$

To prove the bound in (8), let $c_\star$ be the *determining curve* and $t_\star \in I_{c_\star}$ be the *determining parameter* of $d(x,y)$, in the sense that

$$d(x,y) = d^{c_\star}(x,y) = d_x^{c_\star}(t_\star) + d_y^{c_\star}(t_\star),$$

and $\hat{c}_\star$ and $\hat{t}_\star \in I_{\hat{c}_\star}$ be the determining curve and parameter for $\hat{d}$,

$$\hat{d}(x,y) = \hat{d}_x^{\hat{c}_\star}(\hat{t}_\star) + \hat{d}_y^{\hat{c}_\star}(\hat{t}_\star).$$

On the one hand, it then follows from (2), (3), and (10) that

$$d(x,y) \le d_x^c(t) + d_y^c(t) \le \frac{1}{1-\varepsilon}\hat{d}_x^c(t) + \frac{1}{1-\varepsilon}\hat{d}_y^c(t)$$

for any $c \in C$ and any $t \in I_c$, including $\hat{c}_\star$ and $\hat{t}_\star$, so that

$$d(x,y) \le \frac{1}{1-\varepsilon}\hat{d}(x,y)$$

and thus

$$-\varepsilon d(x,y) \le \hat{d}(x,y) - d(x,y). \tag{11}$$

On the other hand, we conclude from (4), (5), and (9) that

$$\hat{d}(x,y) \le \hat{d}_x^c(t) + \hat{d}_x^c(t) \le (1+\varepsilon) d_x^c(t) + (1+\varepsilon) d_y^c(t)$$

for any $c \in C$ and any $t \in I_c$, including $c_\star$ and $t_\star$. Hence,

$$\hat{d}(x,y) \le (1+\varepsilon) d(x,y)$$

and

$$\hat{d}(x,y) - d(x,y) \le \varepsilon d(x,y),$$

which, together with (11), implies (8). □