Exact Geodesics and Shortest Paths on Polyhedral Surfaces

Mukund Balasubramanian, *Member, IEEE,* Jonathan R. Polimeni, *Member, IEEE,* and Eric L. Schwartz

***Abstract*—We present two algorithms for computing distances along convex and non-convex polyhedral surfaces. The first algorithm computes exact minimal-geodesic distances and the second algorithm combines these distances to compute exact shortest-path distances along the surface. Both algorithms have been extended to compute the exact minimal-geodesic paths and shortest paths. These algorithms have been implemented and validated on surfaces for which the correct solutions are known, in order to verify the accuracy and to measure the run-time performance, which is cubic or less for each algorithm. The exact-distance computations carried out by these algorithms are feasible for large-scale surfaces containing tens of thousands of vertices, and are a necessary component of near-isometric surface flattening methods that accurately transform curved manifolds into flat representations.**

***Index Terms*—Differential geometry, flat maps, triangular meshes, surface-based analysis, computational geometry.**

I. Introduction

COMPUTING geodesics and shortest paths on surfaces is a challenging problem in computational and differential geometry, with several important areas of application such as computerized brain flattening [1], [2], texture mapping [3], surface partitioning [4], [5], terrain navigation [6], and path planning [7].

Most methods proposed for computing *exact* shortest paths on non-parametrized surfaces represented by polyhedral meshes are either prohibitively difficult to implement [8], [9], [10], [11], [12] or have impractical run times for meshes with many vertices [13]. As a result, several algorithms for computing *approximate* shortest paths have been proposed [6], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]. These methods are appropriate for applications such as interactive visualization and texture mapping, where extremely rapid solutions are desired and some loss of accuracy can be tolerated.

However, there is a need for exact solutions that can be computed in a reasonable amount of time, particularly in the context of quantitative surface-based analyses of brain structure and function [25], [26], where accuracy is at a premium. Furthermore, exact solutions can be used to provide an off-line characterization of the quality of faster, approximate techniques (e.g., Dijkstra's algorithm) on arbitrary surfaces.

Here we present two algorithms with cubic run-time performance for computing *distances* on (convex and non-convex) polyhedral surfaces: the first algorithm computes the exact *minimal-geodesic* distance between every pair of vertices and the second algorithm builds on the first to compute exact *shortest-path* distances (the difference between these two types of distance is described in the following sections). We also present two linear-time algorithms for constructing the *paths* corresponding to these distances. The first algorithm computes the exact minimal-geodesic path connecting any two vertices on the surface, given the output of the minimal-geodesic distance algorithm and, similarly, the second algorithm computes the exact shortest-path curve between any pair of vertices. These algorithms are simple to implement and are practical for meshes with tens of thousands of vertices.

The remainder of this paper is organized as follows: Section II provides the differential geometry background pertaining to geodesics on differentiable surfaces and Section III extends these concepts to geodesics on polyhedral surfaces. The LOS algorithm for computing minimal-geodesic distances is described in Section IV, the LOS-Floyd algorithm for computing shortest-path distances is described in Section V, and the LOS-Path and LOS-Floyd-Path algorithms for computing the corresponding paths are described in Section VI. Experimental results and validation tests are presented in Section VII. Section VIII discusses related work and applications, and Section IX provides a closing summary.

II. Geodesics on differentiable surfaces

A *geodesic* on a differentiable surface (2-manifold) is a differentiable curve that has zero geodesic curvature at each point on the curve [27]. The *geodesic curvature* at a point is given by $\kappa_g = \kappa \sin\theta$, where κ is the curvature of the curve at that point, and θ is the angle between the normal to the surface $\vec{\mathbf{N}}$ and the normal to the curve $\vec{\mathbf{n}}$. For a surface and curve embedded in 3-dimensional Euclidean space $\mathbb{R}^3$, this is equivalent to defining the geodesic curvature as the covariant derivative of the unit tangent vector to the curve [28].

The intuition behind this definition of geodesic curvature is as follows: consider a particle that travels at constant speed along a curve on a surface. The acceleration of this particle can be decomposed into one component normal to the surface—the *normal curvature*—and another component tangential to the surface—the geodesic curvature. In other words, the normal

M. Balasubramanian is with the Department of Cognitive and Neural Systems, Boston University, Boston, MA 02215, USA. E-mail: mukundb@cns.bu.edu.

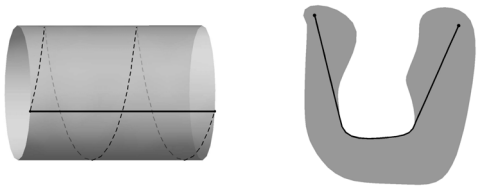
J. R. Polimeni was with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215, USA. He is now with the Department of Radiology, MGH, Athinoula A. Martinos Center for Biomedical Imaging, Harvard Medical School, Charlestown, MA 02129, USA. E-mail: jonp@nmr.mgh.harvard.edu.

E. L. Schwartz is with the Departments of Cognitive and Neural Systems, Electrical and Computer Engineering, and Anatomy and Neurobiology, Boston University, Boston, MA 02215, USA. E-mail: eric@bu.edu.

Manuscript received November 30, 2007. This work was supported in part by the National Institute for Biomedical Imaging and Bioengineering under grant R01 EB001550.

curvature is due to the bending of the surface, whereas the geodesic curvature is due to the bending of the curve within the surface [29]. Geodesic curvature and geodesics are therefore aspects of the *intrinsic* geometry of the surface—they are independent of the embedding of the surface in $\mathbb{R}^3$.

Geodesics are frequently identified with shortest paths on curved manifolds. Although this is often true, geodesics need not be shortest paths and shortest paths need not be geodesics, as demonstrated in Figure 1.



(a) (b)

Fig. 1. (a) A geodesic that is not a shortest path is shown as a dashed curve on the cylinder. The shortest path is shown as a solid curve. (b) A shortest path that is not a geodesic. This path is on a *closed* subset of $\mathbb{R}^2$, which is an example of a manifold with boundary.

The distinction between geodesics and shortest paths follows from an application of the calculus of variations [30] to the *arclength functional* L, which maps a curve (on a surface) to a real number corresponding to the arclength of the curve. The *variational derivative* δL of the functional L is analogous to the differential of a function of one variable: the critical points of L comprise the set of curves for which δL vanishes [31]. It can be proven that the curves on a surface that satisfy $\delta L = 0$ are precisely the geodesics of that surface [28].

However, there can be critical points where L does not take on its minimum value (as in Figure 1(a)), and the global minimum need not occur at a critical point, in which case it will lie on the boundary of the domain (as in Figure 1(b)).

III. Geodesics on triangular meshes

A *polyhedral surface* is a set of polygons that constitutes a piecewise-flat representation of a 2-dimensional surface embedded in $\mathbb{R}^3$. As it is straightforward to decompose simple polygons into triangles [32], [33], it shall henceforth be assumed that any polyhedral surface of interest has been converted into a *triangular mesh*, without loss of generality. For the triangular mesh to be a *manifold* (possibly with boundary), it must satisfy the following conditions [34], [35]:

- two triangles cannot intersect, except at a vertex or an edge shared by the two triangles;
- a triangle cannot share an edge with more than one other triangle;
- the m_i triangles sharing vertex i can be ordered into a sequence $\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_{m_i}$ such that adjacent triangles in this sequence share an edge; and
- a vertex on the boundary must be connected (via an edge) to exactly two other boundary vertices.

A surface given by a triangular mesh is not, in general, differentiable at triangle vertices or at points on the triangle edges. At these points, a curve on the surface is not differentiable, and therefore its curvature is undefined. Consequently, it might seem that the geodesic curvature (see Section II) must also be undefined on vertices and edges. However, this is not the case: triangle chains on meshes enable the definition of geodesic curvature on the edges of a triangular mesh.

A *triangle chain* is a sequence of triangles such that adjacent triangles in the sequence share an edge. Although it may not be possible to isometrically map (or "flatten") an entire triangle mesh into the plane, it is possible to isometrically flatten any triangle chain of a triangular mesh; that is, the triangle chain can be laid out in the plane without changing the lengths (and therefore the angles and area) of any of the triangles in the chain (see Figure 2). This can be seen by imagining each shared edge in the chain as a hinge; starting with the second triangle, each triangle can be rotated in turn until it is in the same plane as the previous triangle in the chain [36].

Consider a curve that is contained within a triangle chain, as shown in Figure 2(a). By isometrically flattening the triangle chain, we also map the curve to the plane isometrically (Figure 2(b)). Under such a mapping, the intrinsic geometry of the surface is preserved. Therefore, the problem of computing the geodesic curvature of a curve on a mesh can be reduced to the much simpler problem of computing the (geodesic) curvature of a planar curve.

In order to compute an unambiguous value for the geodesic curvature at all points of a curve, the curve must not pass through a triangle vertex, except at its endpoints. If a curve does pass through an intermediate vertex, it can be contained within more than one triangle chain, and can therefore have more than one flattened representation. There is no guarantee that these different planar curves will have the same curvature at the intermediate vertex, resulting in an ambiguous value for the geodesic curvature at that vertex.

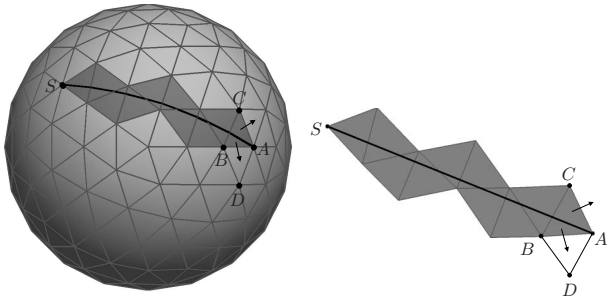
One possible solution to this problem is to define an intrinsically discrete measure of geodesic curvature that one can directly apply to the original curve [37], rather than invoking isometric flattening of triangle chains. Here we instead take the simpler approach of defining a geodesic on a triangular mesh as any path connecting two vertices that has the following properties: (i) the path is completely contained within some triangle chain and does not pass through any other vertices, and (ii) when this triangle chain and the path are laid out in the plane, the path is a straight line—a geodesic of the Euclidean plane. An important consequence of this definition is that some vertex pairs on a surface may have no geodesic connecting them.

IV. Computing minimal-geodesic distances

In this section, we present a simple, novel algorithm, the *LOS* algorithm, for computing the minimal (i.e., shortest) geodesic *distance* between every pair of vertices on a triangular mesh. If the mesh is the boundary of a convex region in Euclidean $\mathbb{R}^3$, then an important result is that the minimal geodesic between two vertices is guaranteed to be the shortest path between these points [36]. For a non-convex mesh, however, further computation is required to calculate shortest-path distances; in Section V, we show how graph-theoretic algorithms can be used to perform this computation.

A. Iterative growth of triangle chains

The LOS algorithm is based on the iterative growth of triangle chains on a triangular mesh and, simultaneously, on the plane. We begin by describing one iteration of this algorithm: consider a triangle chain that has already been flattened, for which the last triangle is $\triangle ABC$ and A is the last vertex (i.e., the vertex that is in the last triangle of the chain, but not in the next-to-last triangle). This triangle chain can be extended over the edge AB or over the edge AC, as shown in Figure 2.



(a) (b)

Fig. 2. (a) A triangle chain of a triangular-mesh approximation of the unit sphere, along with a curve contained within this triangle chain. The first vertex of the chain is S, the last triangle is $\triangle ABC$ and the last vertex is A. The triangle chain can be isometrically flattened: this is shown in (b) along with the flattening of the curve under this isometry. In this case, the flattened curve is a straight line, indicating that the curve shown in (a) is indeed a geodesic of the triangular mesh.

If we choose to grow the triangle chain over edge AB, then extending the flattened triangle chain requires the calculation of the coordinates of vertex D in the plane. This calculation is equivalent to finding the intersection of two circles given their centers and radii; the centers are the planar coordinates of vertices A and B, and the radii are the distances d_{AD} and d_{BD}, where d_{AD} is the 3-dimensional Euclidean distance between vertex A and vertex D, and d_{BD} is the 3-dimensional Euclidean distance between vertex B and vertex D. This is a straightforward problem that can be solved using the law of cosines.

Let us assume that a segment of the edge AB is "visible" from the first vertex S (i.e., the vertex that is in the first triangle of the chain, but not in the second triangle). By this we mean that a straight line can be drawn on the flattened triangle chain from S to any point on the segment, such that the straight line is completely contained within the triangle chain. The angle that any such line makes with the horizontal axis can be easily computed: let $\theta_{\min}$ be the smallest such angle and let $\theta_{\max}$ be the largest angle (see Figure 3). Let θ be the angle that the straight line from S to D on the flattened triangle chain makes with the horizontal axis. If $\theta \in [\theta_{\min}, \theta_{\max}]$, then a geodesic connecting S to D exists on this triangle chain; the length of the geodesic is simply the Euclidean distance between S and D in the plane.

If we continue to grow this triangle chain, we must again choose one of two edges over which to extend the triangle chain. If, every time we are confronted with such a choice, we first extend the chain over one edge and then return later to extend the chain over the other edge, we will have constructed every possible triangle chain of the triangular mesh, in a depth-first manner. For any such triangle chain, we can compute the length of the geodesic between the first vertex and the last vertex on the flattened chain (if a geodesic exists), storing this path if it is shorter than the geodesics on other triangle chains between these two vertices. This strategy of exhaustive search is feasible for meshes with a small number of vertices (≈ 1000) [13].

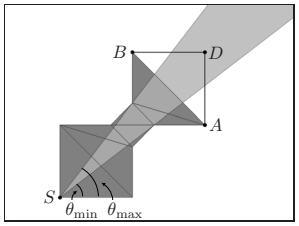
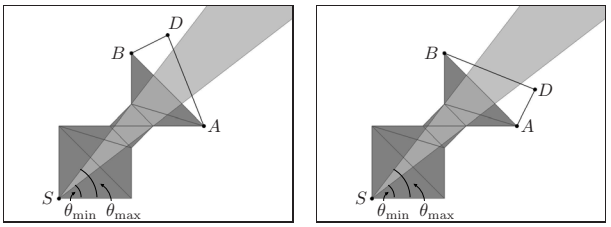


Fig. 3. If $\theta \in [\theta_{\min}, \theta_{\max}]$, where θ is the angle vertex D makes with the horizontal axis, then the straight line connecting vertex S to D will be contained within the flattened triangle chain. Therefore, a geodesic connecting S to D exists on this triangle chain. The angles $\theta_{\min}$ and $\theta_{\max}$ thus define a *visibility sector*—vertex D must lie within this sector to be "visible" from S.



(a) (b)

Fig. 4. (a) If $\theta > \theta_{\max}$, where θ is the angle vertex D makes with the horizontal axis, then there will not be a straight line connecting S to any point on the edge BD, such that the straight line is contained within the flattened triangle chain. Therefore, no geodesics will be found by extending the triangle chain over the edge BD. (b) Similarly, if $\theta < \theta_{\min}$, no geodesics will be found by extending the triangle chain over edge AD.

Instead of performing an exhaustive search, we can greatly reduce the number of triangle chains we need to construct by extending triangle chains only over edges that have a visible segment. To see this, consider the situation illustrated in Figure 4(a), where $\theta > \theta_{\max}$. No segment of edge BD can be connected to S with a straight line that is contained within the triangle chain; therefore, there is no point in extending the triangle chain over this edge—no geodesics will be found by growing the triangle chain in this direction. Similarly, if $\theta < \theta_{\min}$, then there is no point in extending the triangle chain over edge AD (see Figure 4(b)). **It is only when $\theta \in [\theta_{\min}, \theta_{\max}]$ that the triangle chain must be extended over both edges, AD and BD, in turn.** This realization can greatly reduce the number of triangle chains we need to construct in our search for geodesic paths, and is implemented in the LOS (for "line-of-sight") algorithm, which we present next.

B. The LOS algorithm: pseudocode

1) Choose a first vertex S.

2) Choose a triangle $\mathcal{T}$ that has S as a vertex, and let $A_{\mathcal{T}}$ and $B_{\mathcal{T}}$ be the other two vertices of $\mathcal{T}$. Compute the Euclidean distance between S and $A_{\mathcal{T}}$ and store this distance in the matrix $\boldsymbol{\delta}$ as the entry $\delta_{SA_{\mathcal{T}}}$. Similarly, compute and store the value for $\delta_{SB_{\mathcal{T}}}$. Map $\mathcal{T}$ to the plane, with S at the origin, $A_{\mathcal{T}}$ on the positive x-axis, and $B_{\mathcal{T}}$ in the upper half-plane $y > 0$. Let $\theta_{\min} = 0$ (i.e., the angle $SA_{\mathcal{T}}$ makes with the x-axis) and let $\theta_{\max}$ be the angle $SB_{\mathcal{T}}$ makes with the x-axis. At this point, we have a triangle chain that consists solely of the triangle $\mathcal{T}$.
3) Let $A = A_{\mathcal{T}}$, $B = B_{\mathcal{T}}$, and initialize the triangle-chain stack to be empty. We shall use this stack to implement the branching of the triangle chain, when we extend the triangle chain over one edge and later return to extend the chain over the other edge.
4) Find the triangle $\mathcal{U}$ that lies on the other side of AB. If no such triangle exists, or if $(\theta_{\max} - \theta_{\min})$ falls below machine precision, then there are two options:
   a) If the triangle-chain stack is not empty, pop the stack to retrieve new values for A, B, and their planar coordinates, along with the new values for $\theta_{\min}$ and $\theta_{\max}$.
   b) If the stack is empty, return to step 2, choosing a new triangle $\mathcal{T}$ that has S as a vertex. If every triangle that has S as a vertex has already been selected, return to step 1 and choose a new first vertex S. Once every vertex has been selected to be a first vertex in step 1, the algorithm terminates.
5) Add $\mathcal{U}$ to the flattened triangle chain. That is, calculate the planar coordinates of vertex D, which is the third vertex of triangle $\mathcal{U}$ (the other two vertices are A and B). Also, compute the angle θ that SD makes with the x-axis.
6) If $\theta < \theta_{\min}$, set $A = D$ and return to step 4.
7) If $\theta > \theta_{\max}$, set $B = D$ and return to step 4.
8) If $\theta \in [\theta_{\min}, \theta_{\max}]$, then perform the following steps:
   a) Compute the Euclidean distance between the planar representations of S and D. If this is the shortest distance from S to D that has been encountered thus far, then store this distance in the matrix $\boldsymbol{\delta}$ as the entry δ_{SD}. Also update the matrices $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\theta}$ such that $\mathcal{T}_{SD} = \mathcal{T}$, the first triangle on this triangle chain, and $\theta_{SD} = \theta$, which is equivalent to the angle (on the triangular mesh) that this geodesic from S to D makes with the straight edge from S to $A_{\mathcal{T}}$. (Note that the matrices $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\theta}$ are only required for the path construction algorithms that will be presented in Section VI—if it is just distances that are required and not paths, then these quantities need not be stored.)
   b) Push the information needed to extend the triangle chain over the edge AD onto the stack. That is, create a *stack item*: a data structure that has fields for A, B, and their planar coordinates, as well fields for $\theta_{\min}$ and $\theta_{\max}$. In the field for A, store the index of vertex A (with respect to the triangular mesh) along with its planar coordinates. In the field for B, store the index of vertex D and its planar coordinates. In the field for $\theta_{\min}$, store the current value of $\theta_{\min}$, and in the field for $\theta_{\max}$, store the current value of θ.
   c) Extend the triangle chain over the edge BD. That is, set $A = D$ and set $\theta_{\min} = \theta$.
   d) Return to step 4.

C. Termination of the LOS algorithm

Termination of the LOS algorithm follows from two conditions: (i) the length of any triangle chain constructed by the algorithm is finite and (ii) the number of triangle chains constructed is finite. The first condition follows directly from the LOS construction, which is characterized by the visibility sector shown in Figure 3. The angular extent of the visibility sector $(\theta_{\max} - \theta_{\min})$ must decrease as the triangle chain grows, resulting in termination via Step 4 of the LOS pseudocode—a non-decreasing angular extent would require unbounded triangle edge lengths, due to the expansion of the visibility sector "wavefront" with increasing triangle chain length. A more detailed description is provided in the supplementary materials.

The second condition follows from the first: let $\boldsymbol{\mathcal{C}}_{S,\mathcal{T}}$ be the longest triangle chain with first vertex S and initial triangle $\mathcal{T}$. Then every other triangle chain originating from S and $\mathcal{T}$ can be generated by branching off some triangle contained within $\boldsymbol{\mathcal{C}}_{S,\mathcal{T}}$. Since the number of triangles in $\boldsymbol{\mathcal{C}}_{S,\mathcal{T}}$ is finite (from the first condition), the number of triangle chains that can be constructed by branching off $\boldsymbol{\mathcal{C}}_{S,\mathcal{T}}$ is finite. Therefore, the total number of triangle chains constructed by the LOS algorithm is finite, as there are only a finite number of choices for S and $\mathcal{T}$.

V. Computing shortest-path distances

The minimal geodesic between two points on a surface is not necessarily the shortest path between these two points. Here we show how the shortest-path distance between every pair of vertices on a triangular mesh can be computed given the minimal-geodesic distance between every pair of vertices.

If the shortest path between two vertices does not pass through any intermediate vertices, then this path will be contained within some unique triangle chain. In this case, the shortest path is a minimal geodesic, and no further computation is required. However, in general, the shortest path β between two vertices, S and D, will pass through n intermediate vertices (in order): I_1, I_2, …, I_n. The path between two consecutive vertices in this sequence, I_j and I_{j+1}, will lie within some triangle chain and must be the shortest path (and minimal geodesic) between these two vertices; otherwise, the minimal geodesic between I_j and I_{j+1} could be used to create a shorter path than β (from S to D).

Thus the shortest path between two vertices of a triangular mesh must be a union of minimal geodesics. To find this shortest union, we first construct the *minimal-geodesic distance graph*: the nodes of this graph are the vertices of the mesh, two nodes will be connected by an arc (i.e., graph

edge) if a minimal geodesic exists between them, and the weight of an arc is the minimal-geodesic distance between the corresponding vertices.

Standard graph-theoretic algorithms for computing shortest-path distances on weighted graphs can then be used to construct the union of minimal geodesics that yields the shortest distance between pairs of vertices on a triangular mesh. Two canonical algorithms are Dijkstra's algorithm for finding the shortest distance from one node (the "source") to all of the other nodes on a graph [38] and Floyd's algorithm for finding the shortest distance between all pairs of nodes in a graph [39].

As we are interested in the shortest-path distance between *every* pair of vertices, Floyd's algorithm is the natural choice to apply to the minimal-geodesic distance graph that is computed by the LOS algorithm, resulting in the *LOS-Floyd* algorithm: for every pair of vertices S and D, each vertex in turn is considered as an intermediate vertex, and if the distance between S and D is reduced by passing through an intermediate vertex I, the updated distance $\delta_{SD} = \delta_{SI} + \delta_{ID}$ is stored in the matrix $\boldsymbol{\delta}$.

As with the LOS algorithm (Section IV), some additional book-keeping is required in order to use the path construction algorithms that will be presented in Section VI; for Floyd's algorithm, this takes the form of a *successor matrix* $\boldsymbol{R}$ that is initialized such that $R_{SD} = D$ for each vertex pair (S, D) for which a minimal geodesic exists. When the distance between S and D is lowered by going through an intermediate vertex I, $\boldsymbol{R}$ is updated by setting $R_{SD} = R_{SI}$.

Unlike the LOS algorithm, the LOS-Floyd algorithm will always return a finite value for the distance between each vertex pair, under the conditions that the vertex coordinates are themselves finite and that the triangular mesh constitutes a single connected component—although there may not always be a geodesic between two vertices, there will always be a shortest path between them if the two conditions above are met. Unlike geodesics, shortest paths also have the property that they cannot pass through a triangle more than once. Therefore, when computing shortest paths, we can use a modified version of the LOS algorithm that does not allow a triangle to appear more than once in a triangle chain. This results in an improvement in the efficiency of the computation, as shown in the supplemental material.

VI. Constructing paths

Although the distances computed by the LOS and LOS-Floyd algorithms provide the *lengths* of minimal geodesics and shortest paths, respectively, these algorithms do not explicitly construct the paths themselves—in the LOS algorithm, the minimal-geodesic distance between a pair of vertices results from computing a single Euclidean distance in the plane (via isometric flattening of the triangle chain containing this geodesic), as opposed to the more costly strategy of first constructing this piecewise-linear path explicitly, and then computing its length by summing up the lengths of its constituent straight-line segments.

In this section, we present the *LOS-Path* algorithm, which rapidly constructs the minimal-geodesic *path* in $\mathbb{R}^3$ connecting any two given vertices, given the output of the LOS algorithm, and show that a slight modification of this algorithm can be used to construct non-minimal geodesics. We also present the *LOS-Floyd-Path* algorithm, which rapidly constructs shortest paths on triangular meshes.

The output of these algorithms is a list of 3-dimensional coordinates $\{\vec{X}_k\}$, where $k = 1, \ldots, K$ such that $\vec{X}_1$ contains the coordinates of the first point on the path, $\vec{X}_K$ contains the coordinates of the last point, and $\vec{X}_k$ and $\vec{X}_{k+1}$ are the endpoints of the k-th straight-line segment.

A. The LOS-Path algorithm: pseudocode

1) Given a source vertex S and a target vertex P, use the matrices $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\theta}$ returned by the LOS algorithm (see step 8a of Section IV-B) to set triangle $\mathcal{T} = \mathcal{T}_{SP}$ and angle $\theta_0 = \theta_{SP}$. Also, set $\vec{X}_1$ to the 3-dimensional coordinates of vertex S and let $k = 1$.
2) Let A and B be the other two vertices of $\mathcal{T}$. Place S at the origin, A on the (horizontal) x-axis, and B in the upper half-plane.
3) Find the intersection of the planar representation of AB and the ray from the origin that makes an angle θ_0 with the x-axis. Find the corresponding point on the 3-dimensional line segment AB of the triangular mesh. Increment k by 1 and store the 3-dimensional coordinates of this point as $\vec{X}_k$.
4) Find the triangle $\mathcal{U}$ that lies on the other side of AB.
5) Let D be the third vertex of triangle $\mathcal{U}$ (other than A or B). Calculate the planar coordinates of D and the angle θ that SD makes with the x-axis.
6) If $\theta < \theta_0$, extend the triangle chain over the edge BD. That is, set $A = D$ and return to step 3.
7) If $\theta > \theta_0$, extend the triangle chain over the edge AD. That is, set $B = D$ and return to step 3.
8) If $\theta = \theta_0$ (which occurs when $D = P$), let $K = k + 1$, store the 3-dimensional coordinates of D as $\vec{X}_K$ and let the algorithm terminate.

B. Constructing non-minimal geodesics

The LOS-Path algorithm can be easily modified to construct non-minimal geodesics, such as the solenoidal winding on the cylinder shown in Figure 1(a): in step 1, instead of choosing $\mathcal{T}$ and θ_0 based on the output of the LOS algorithm, $\mathcal{T}$ is now chosen to be any triangle that has the source S as a vertex and θ_0 is chosen to be any angle less than or equal to the angle between the two edges of $\mathcal{T}$ that meet at S. Together, $\mathcal{T}$ and θ_0 specify the initial direction of heading of a geodesic originating from vertex S. (Note that the target vertex P is now left unspecified.)

This modification of the LOS-Path algorithm solves an *initial value problem*, whereas the LOS-Path algorithm (which uses the output of the LOS algorithm) solves a *boundary value problem* for constructing geodesics on triangular meshes.

The termination criteria for the initial value problem must also be considered—step 4 can be modified to result in algorithm termination if no triangle $\mathcal{U}$ lies on the other side of AB, i.e., if the geodesic intersects a boundary of the surface.

The algorithm should also terminate when the pathlength (or the number of line segments in the path) exceeds some predetermined cutoff.

C. The LOS-Floyd-Path algorithm

The successor matrix $\boldsymbol{R}$ returned by the LOS-Floyd algorithm can be used to rapidly identify the intermediate vertices along the shortest path between a source vertex S and a target vertex P: the first intermediate vertex is given by $I_1 = R_{SP}$, the second intermediate vertex is given by $I_2 = R_{I_1 P}$, and so on.

As the shortest path between any two consecutive intermediate vertices I_j and I_{j+1} must be a minimal geodesic (see Section V), we can construct the shortest path from S to P by concatenating the minimal geodesics (as constructed by the LOS-Path algorithm) from S to I_1, I_1 to I_2, ..., and I_n to P—this procedure shall henceforth be referred to as the LOS-Floyd-Path algorithm for constructing shortest paths on triangular meshes.

VII. Results

A. Minimal geodesics versus shortest paths

The difference between minimal geodesics and shortest paths on triangular meshes is illustrated in Figure 5, which shows the output of the LOS-Path and LOS-Floyd-Path algorithms.

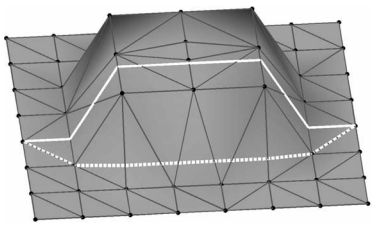
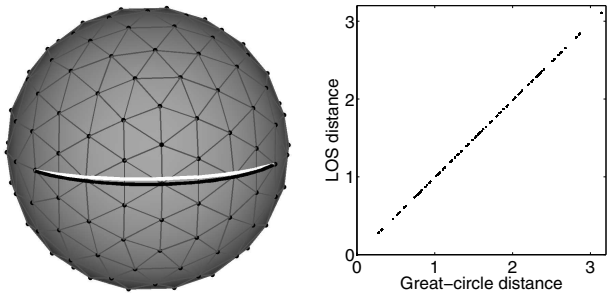


Fig. 5. The minimal geodesic between two vertices, as given by the LOS-Path algorithm, is shown in solid white. The shortest path, as given by the LOS-Floyd-Path algorithm, is shown in dashed white. Note that the shortest path passes through two intermediate vertices, and is composed of the union of three minimal geodesics.

B. Validating the LOS and LOS-Path algorithms

Minimal geodesics on the sphere are known to be segments of great circles (and are also shortest paths), and closed-form expressions for the corresponding distances are easily derived. Therefore, a piecewise-flat approximation of the unit sphere was used to test the accuracy of the LOS and LOS-Path algorithms.

Figure 6(a) shows the shortest great-circle segment connecting a pair of vertices and the output of the LOS-Path algorithm. Figure 6(b) shows the agreement between the great-circle distances and the LOS output for every pair of vertices on the mesh. Note that the minor discrepancy seen here is due to the fact that the piecewise-flat surface does not constitute a perfect representation of the sphere (which is curved at all points).

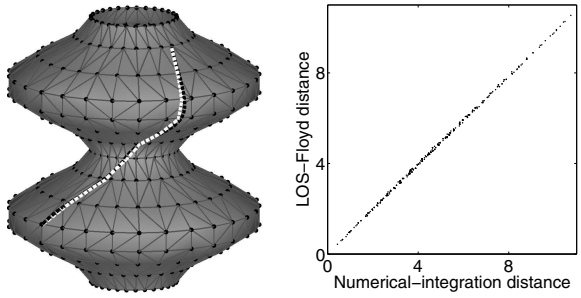


(a) (b)

Fig. 6. (a) The shortest great-circle segment connecting a pair of vertices is shown in black, and the path returned by the LOS-Path algorithm is shown in white. (b) For every pair of vertices in the mesh, the length of the shortest great-circle segment is plotted against the distance computed by the LOS algorithm. These points lie almost exactly on the line $y = x$ ($R^2 > 0.999$).

C. Validating the LOS-Floyd and LOS-Floyd-Path algorithms

On convex polyhedral surfaces, minimal geodesics (given by the LOS-Path algorithm) and shortest paths (given by the LOS-Floyd-Path algorithm) are identical. Therefore, a validation of the LOS-Floyd and LOS-Floyd-Path algorithms (that does not simply reduce to a validation of the LOS and LOS-Path algorithms) requires knowledge of shortest paths on non-convex surfaces.



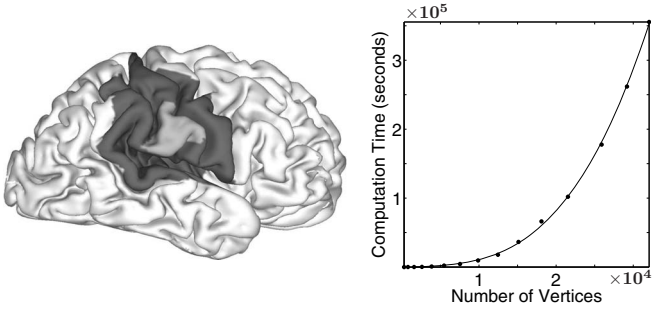
(a) (b)

Fig. 7. (a) The shortest path connecting a pair of points, as given by numerical integration on a surface of revolution with generating curve specified by $p(u_1) = 2-\cos(2u_1)$ and $q(u_1) = u_1$ (see supplemental material), is shown in dashed black. The `ode45` function in MATLAB (The MathWorks, Natick, MA) was used for numerical integration. A triangular mesh approximating this surface is shown in gray, and the path returned by running the LOS-Floyd-Path algorithm on this mesh is shown in dashed white. (b) For every pair of vertices in the mesh, the distance given by numerical integration is plotted against the distance computed by the LOS-Floyd algorithm. These points lie almost exactly on the line $y = x$ ($R^2 > 0.999$).

For the special case of differentiable surfaces of revolution, shortest paths between pairs of points can be found via the numerical integration of ordinary differential equations (see supplemental material for details). We can therefore compare these paths (and their lengths) on a non-convex, differentiable surface of revolution with the output of the LOS-Floyd and LOS-Floyd-Path algorithms on a piecewise-flat approximation of this surface, as shown in Figure 7. This figure demonstrates excellent agreement between the paths, thus validating the LOS-Floyd and LOS-Floyd-Path algorithms. Similar agreement was found for other surfaces of revolution, both convex, such as spheroids and paraboloids, and non-convex, such as catenoids and hyperboloids of one sheet.

D. Run-time performance and complexity

To evaluate the run-time performance of the LOS-Floyd algorithm, several subsets of a large triangular mesh (with over $50,000$ vertices) were extracted and the time taken by the algorithm to compute all shortest-path distances was measured for each subset (see Figure 8). These results show that the LOS-Floyd run time increases as a cubic function of the number of vertices N, a reflection of the $\mathcal{O}(N^3)$ run time of Floyd's algorithm and of the LOS component alone (see supplemental material).



(a) (b)

Fig. 8. (a) A triangular mesh with $52,360$ vertices, representing the right hemisphere of human cerebral cortex. A small subset of this surface, with 670 vertices, is indicated by the middle shade of gray. A second, larger subset, with 8441 vertices, is created by taking the union of the small subset with the region shaded dark gray. (b) The LOS-Floyd algorithm was used to compute the shortest-path distance between every pair of vertices on several such subsets extracted from the full cortical surface, and the number of vertices in each subset is plotted against the time taken for the algorithm to execute. A third degree polynomial fits the data points extremely well ($R^2 > 0.999$), as shown by the black curve, indicating that the LOS-Floyd algorithm has cubic run-time performance. All times were measured on a workstation with a 2.4 GHz AMD Opteron processor and 16 GB of memory.

It is possible to construct a mesh for which the LOS strategy (i.e., only extending triangle chains over edges with a visible segment, described in Section IV-A) is no more efficient than exhaustive search, as shown in Figure 9. For this mesh, the vertices were intentionally arranged such that any triangle chain with first vertex S must always be extended over both possible edges during the LOS search. Therefore, meshes constructed in this manner will exhibit a run time that is exponential with the number of vertices [13], demonstrating that the (worst-case) time complexity of the LOS algorithm is exponential. However, aside from such pathological meshes deliberately designed to defeat the LOS strategy, we have never observed anything other than cubic run-time performance for the LOS-Floyd algorithm on any "real-world" meshes, including a number of different brain surface meshes.

The time taken by the LOS-Path algorithm to construct a minimal geodesic between two vertices is proportional to the number of triangles in the triangle chain containing the geodesic, as no branching is involved. As the number of triangles in a chain is $\mathcal{O}(N)$, the LOS-Path algorithm has linear time complexity. The additional time taken by the LOS-Floyd-Path algorithm is the time required to traverse the chain of intermediate vertices from the source vertex to the target vertex, via the successor matrix $\boldsymbol{R}$. Again, no branching is involved, and so the time taken is proportional to the number of intermediate vertices, which is $\mathcal{O}(N)$. Therefore, the LOS-Floyd-Path algorithm also has linear time complexity.

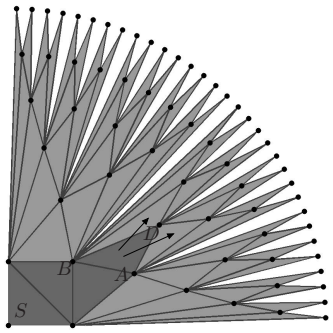


Fig. 9. A mesh on which the LOS strategy does not result in an advantage over exhaustive search. A triangle chain with first vertex S and last vertex D is shown in dark gray. As D was positioned during the construction of this planar mesh to be visible from S (i.e., the coordinates of D were chosen such that $\theta_{SD} = \frac{1}{2}[\theta_{SA} + \theta_{SB}]$), this triangle chain must be extended over both edge AD and edge BD in the search for geodesics, as indicated by the arrows. As this is true (by construction) for every triangle chain starting from S, the LOS strategy reduces to exhaustive search on meshes constructed in this manner.

VIII. Discussion

A. Comparison with other exact shortest-path algorithms

Several algorithms for computing exact shortest paths and distances on polyhedral surfaces have previously been proposed [8], [9], [10], [11], [13], [36], [40]. However, it is not clear how some of these methods could be implemented. To the best of our knowledge, the only exact algorithms that have actually been implemented, other than the algorithms presented here, are the WS algorithm [13], the MMP algorithm [36] (implemented by Surazhsky et al. [41]), and the CH algorithm [40] (implemented by Kaneva and O'Rourke [42] and Lanthier et al. [6]).

Unlike these other methods, the work presented here makes an important distinction between *minimal geodesics* and *shortest paths*—although these are equivalent on convex polyhedral surfaces, they will (in general) differ on non-convex surfaces. While shortest paths are preferable for most applications, minimal geodesics may be more suitable for certain operations such as parallel transport [37], surface partitioning [4], [5], and path planning [7], by virtue of being totally straight (i.e., having zero geodesic curvature at all points), unlike shortest paths that pass through intermediate vertices.

For computing shortest paths on non-convex polyhedral surfaces, the main novelty in the current approach lies in the observation that following a minimal-geodesic computation (e.g., LOS/LOS-Path) with the application of a shortest-path *graph* algorithm (e.g., Floyd's or Dijkstra's algorithm) results in the computation of shortest paths on the polyhedral surface. Note that whereas the extension of the shortest-path computation from the convex to the non-convex case involves nothing more than the application of a standard graph-theoretic algorithm here, this extension is non-trivial for the WS, CH, and MMP algorithms. In particular, Kaneva and O'Rourke [42] report several problems with the CH algorithm on non-convex surfaces, including problems with collinear and

boundary vertices. Our approach, on the other hand, does not suffer from these issues.

The CH and MMP algorithms compute the shortest-path distance from one vertex to all others (the *single-source* problem), whereas the WS and LOS-Floyd algorithms compute the shortest-path distance between every pair of vertices (the *all-pairs* problem). To facilitate comparison of the time and space complexity of these four algorithms, we will assume that the CH and MMP algorithms have been used to solve the all-pairs problem by taking each vertex in turn as the source. Then, the (worst-case) time complexity is exponential for the WS and LOS-Floyd algorithms (as shown in [13] and in Section VII-D above, respectively), $\mathcal{O}(N^3)$ for the CH algorithm [40], and $\mathcal{O}(N^3 \log N)$ for the MMP algorithm [36]. However, Surazhsky et al. [41] report that their implementation of the MMP algorithm exhibits sub-cubic run-time performance on their test surfaces, and similarly we have never observed anything other than cubic run-time performance when executing our implementation of the LOS-Floyd algorithm on many different brain surface meshes. In order to directly compare the run time of our implementation of the LOS-Floyd algorithm with that of the Surazhsky MMP implementation, we used a spherical surface with 2048 triangles as input, and measured the LOS-Floyd run time to be about 30 seconds on a 2.4 GHz AMD Opteron processor. Based on the run times reported in Figure 6 of Surazhsky et al. [41], we estimate that, given the same spherical surface as input, it would take approximately 60 seconds (on a 2.4 GHz PC) for their MMP implementation to compute the distance between every vertex pair.

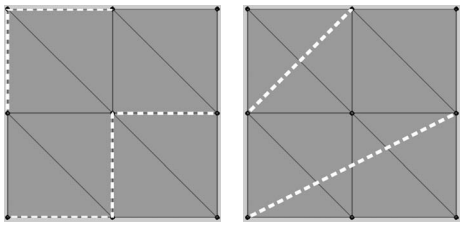
As for space complexity, the storage of distances requires $\mathcal{O}(N^2)$ space for any all-pairs algorithm and, unlike the situation for single-source algorithms, this will typically dominate any additional space requirements of the algorithm. For the LOS algorithm, these additional space requirements are governed by the maximum triangle chain length and the maximum stack size, each of which is $\mathcal{O}(N)$, and therefore the space complexity of the LOS and LOS-Floyd algorithms is $\mathcal{O}(N^2)$, being dominated by the storage of the distances.

The LOS-Floyd algorithm has also been validated more extensively than any of the existing exact algorithms. Aside from qualitative inspection of the computed paths, quantitative validation of the WS, CH and MMP algorithms extends no further than comparing the algorithm output on polyhedral approximations of the sphere to the known analytic distances on the sphere (as in Section VII-B). As shown in Section VII-C, by numerically integrating the geodesic differential equations for surfaces of revolution, we can quantitatively validate shortest-path computations on a variety of non-trivial, non-convex surfaces. Solving the geodesic differential equations for more general differentiable manifolds will facilitate an even greater level of quantitative validation.

B. Accuracy of distance computations

The construction of a polyhedral mesh that adequately represents and approximates some surface of interest can be a challenging problem in computational geometry [43], [44], [45]. As the quality of the surface approximation varies, depending on the problem domain and the method of mesh construction, so will the accuracy of shortest paths computed on the polyhedral mesh, with respect to the true shortest paths on the surface of interest.

However, the results shown in Figure 6 and Figure 7 indicate that the methods presented here are highly accurate even when rather coarse meshes are used as piecewise-flat approximations of smooth surfaces. These results also indicate that, for the LOS and LOS-Floyd algorithms, numerical errors due to finite machine precision are not significant.



(a) (b)

Fig. 10. One of the simplest schemes for computing approximate shortest paths on triangular meshes involves the application of Dijkstra's algorithm to the graph whose nodes are the triangle vertices and whose arcs are the triangle edges. Two such paths on a flat surface are shown in dashed white in (a). The corresponding exact shortest paths, computed by the LOS-Floyd-Path algorithm, are shown in (b). Note that the Dijkstra paths are constrained to lie along triangle edges, whereas the exact shortest paths cut across triangle faces. Therefore, the Dijkstra method always overestimates distance: the lengths of the two paths shown here are overestimated by a factor of $\sqrt{2}$ for the upper path and $3/\sqrt{5}$ for the lower path.

Unlike exact algorithms, methods that compute *approximate* shortest paths on polyhedral surfaces will introduce errors even when the polyhedral mesh perfectly represents the surface of interest and arbitrary-precision (exact) arithmetic is used. A number of these approximate algorithms have been proposed [6], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24]; one such scheme, involving a simple application of Dijkstra's algorithm, is illustrated in Figure 10. The accuracy of these approximate algorithms is typically unknown for arbitrary surfaces, and therefore exact algorithms such as LOS-Floyd can be used to establish the accuracy of these approximate techniques on piecewise-flat surfaces (e.g., see [6]), or to tune the parameters of an approximate method in order to maximize accuracy. (We emphasize that we use "accuracy" to mean the agreement between values computed by an algorithm and values known to be true, not the *order of accuracy* based on a Taylor series analysis, which is often used to characterize numerical approximation schemes [46].) As approximate algorithms are usually faster than exact algorithms, they may be the method of choice in situations where their accuracy has been established and is known to be adequate for the problem at hand.

C. Flattening polyhedral surfaces

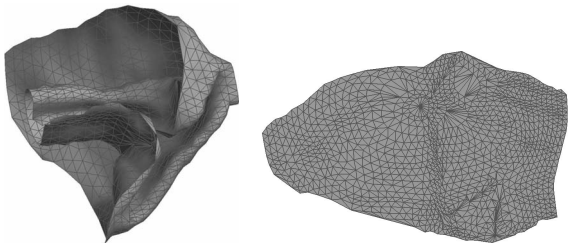
An important reason for computing the full set of exact shortest-path distances is to be able to construct and evaluate "flat maps" of brain surfaces [1], [2], [47], [48]. These flat maps have proven to be useful not just for visualization, but also for mathematically modeling and characterizing the

intrinsic shape and topographic structure of visual cortex, both in humans and macaques [25], [26].

It is impossible to map a surface with non-zero Gaussian (intrinsic) curvature into the plane without introducing some amount of geometric distortion [27], [28], [29]. The set of shortest-path distances between all vertex pairs of a polyhedral surface characterizes the *global metric structure* of the surface and can be used to quantify the metric distortion that results from flattening. This distortion (i.e., "flattening error") can be calculated by comparing the shortest-path distance between each pair of vertices on the polyhedral surface with the Euclidean distance between the corresponding points in the plane.

Exact shortest-path distances can be used not only to *measure* distortions in flat maps, but also to *construct* flat maps by adjusting the positions of vertices in the plane until the measured distortion is minimized. This scheme for flattening polyhedral surfaces was introduced by Schwartz et al. [2], using the WS algorithm to compute exact shortest-path distances.

As the WS algorithm was considered to be too computationally expensive for meshes with a large number of vertices ($\gg 1000$), modifications of this flattening scheme were proposed [3], [49] that incorporate approximate, rather than exact, shortest-path distances. However, by using the LOS-Floyd algorithm presented in this report, it is now feasible to compute exact distances for flattening surfaces with tens of thousands of vertices.



(a) (b)

Fig. 11. (a) A triangular mesh representation of macaque primary visual cortex, reconstructed from serial sections. (b) The result of flattening the surface shown in (a) with the "DMflatten" algorithm [47]. The overall (root mean square) error for this flat map is 4.2%, calculated directly from the shortest-path distances returned by the LOS-Floyd algorithm.

Figure 11 shows the result of flattening macaque primary visual cortex with the "DMflatten" algorithm [47], which incorporates the exact distances computed by the LOS-Floyd algorithm, along with several other improvements over the original Schwartz flattening algorithm [2]. The overall error associated with flattening this surface is 4.2%, which is sufficiently low to justify working with the flattened representation rather than the original, folded surface. This allows the well-understood mathematical techniques of planar Euclidean geometry and complex analysis to be used for the modeling and analysis of spatial patterns of activity within brain surfaces [50].

IX. Summary

Two algorithms have been introduced for computing distances along a manifold polyhedral surface based on the notion of geodesic curvature defined on a triangular mesh. The LOS algorithm computes exact minimal-geodesic distances along triangle chains, and the LOS-Floyd algorithm combines these distances to compute exact shortest-path distances along the surface. Each algorithm has been implemented and validated on surfaces on which distances are known, in order to verify the accuracy and to measure the run-time performance, which is cubic or less for both algorithms. Also, both algorithms have been extended to compute the exact minimal-geodesic paths and shortest paths. While several algorithms exist for computing approximate distances, typically their accuracy is unknown for arbitrary surfaces, and they will therefore be unsuitable for problems that require quantitative measures of the approximation error. The exact-distance computations carried out by the LOS or LOS-Floyd algorithms are feasible for large-scale surfaces containing tens of thousands of vertices, and are a necessary component of our near-isometric surface flattening algorithm that acts to minimize the metric distortions inherent in the transformation of curved manifolds into flat representations.

Acknowledgments

The authors would like to thank Dan Cruthirds and Oliver P. Hinds for their helpful insights and comments on this project, as well as Dr. Guillermo Sapiro and the anonymous reviewers, whose feedback led to a significant improvement of the manuscript.

References

[1] E. Schwartz and B. Merker, "Flattening cortex: an optimal computer algorithm and comparisons with physical flattening of the opercular surface of striate cortex," *Society for Neuroscience Abstracts*, 1985.

[2] E. L. Schwartz, A. Shaw, and E. Wolfson, "A numerical solution to the generalized mapmaker's problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1005–1008, 1989.

[3] G. Zigelman, R. Kimmel, and N. Kiryati, "Texture mapping using surface flattening via multidimensional scaling," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 2, pp. 198–207, 2002.

[4] S. Katz and A. Tal, "Hierarchical mesh decomposition using fuzzy clustering and cuts," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*. New York, NY: ACM, 2003, pp. 954–961.

[5] V. Krishnamurthy and M. Levoy, "Fitting smooth surfaces to dense polygon meshes," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. New York, NY: ACM, 1996, pp. 313–324.

[6] M. Lanthier, A. Maheshwari, and J. R. Sack, "Approximating shortest paths on weighted polyhedral surfaces," *Algorithmica*, vol. 30, no. 4, pp. 527–562, 2001.

[7] F. A. Jolesz, W. E. Lorensen, H. Shinmoto, H. Atsumi, S. Nakajima, P. Kavanaugh, P. Saiviroonporn, S. Seltzer, S. Silverman, M. Phillips, and R. Kikinis, "Interactive virtual endoscopy," *American Journal of Radiology*, vol. 169, pp. 1229–1237, 1997.

[8] M. Sharir and A. Schorr, "On shortest paths in polyhedral surfaces," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 193–215, 1986.

[9] D. M. Mount, "On finding shortest paths on convex polyhedra," Department of Computer Science, University of Maryland, Baltimore, MD, Tech. Rep. 1495, 1984.

[10] J. O'Rourke, S. Suri, and H. Booth, "Shortest paths on polyhedral surfaces," in *Lecture Notes in Computer Science*. Berlin: Springer, 1985, vol. 182, pp. 243–254.

[11] S. Kapoor, "Efficient computation of geodesic shortest paths," in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. New York, NY: ACM Press, 1999, pp. 770–779.

[12] J. O'Rourke, "Computational geometry column 35," *International Journal of Computational Geometry and Applications*, vol. 9, no. 4–5, pp. 513–515, 1999.

[13] E. Wolfson and E. L. Schwartz, "Computing minimal distances on arbitrary polyhedral surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 1001–1005, 1989.

[14] J. Hershberger and S. Suri, "Practical methods for approximating shortest paths on a convex polytope in $\mathbb{R}^3$," *Computational Geometry*, vol. 10, no. 1, pp. 31–46, 1998.

[15] P. K. Agarwal, S. Har-Peled, M. Sharir, and K. R. Varadarajan, "Approximating shortest paths on a convex polytope in three dimensions," *Journal of the ACM*, vol. 44, no. 4, pp. 567–584, 1997.

[16] K. R. Varadarajan and P. K. Agarwal, "Approximating shortest paths on a nonconvex polyhedron," *SIAM Journal on Computing*, vol. 30, no. 4, pp. 1321–1340, 2000.

[17] S. Har-Peled, "Constructing approximate shortest path maps in three dimensions," *SIAM Journal on Computing*, vol. 28, no. 4, pp. 1182–1197, 1999.

[18] T. Kanai and H. Suzuki, "Approximate shortest path on a polyhedral surface and its applications," *Computer-Aided Design*, vol. 33, no. 11, pp. 801–811, 2001.

[19] M. Kageura and K. Shimada, "Finding the shortest path for quality assurance of electric components," *Journal of Mechanical Design*, vol. 126, pp. 1017–1026, 2004.

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

[21] J. A. Sethian and A. Vladimirsky, "Fast methods for the eikonal and related hamilton-jacobi equations on unstructured meshes," *Proceedings of the National Academy of Science (USA)*, vol. 97, no. 11, pp. 5699–5703, 2000.

[22] M. Novotni and R. Klein, "Computing geodesic paths on triangular meshes," in *Proceedings of the 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2002, pp. 341–347.

[23] A. Bartesaghi and G. Sapiro, "A system for the generation of curves on 3d brain images," *Human Brain Mapping*, vol. 14, no. 1, pp. 1–15, 2001.

[24] N. Khaneja, M. Miller, and U. Grenander, "Dynamic programming generation of curves on brain surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1260–1265, 1998.

[25] O. P. Hinds, J. R. Polimeni, M. L. Blackwell, C. J. Wiggins, G. C. Wiggins, A. van der Kouwe, L. L. Wald, E. L. Schwartz, and B. Fischl, "Reconstruction and analysis of human V1 by imaging the stria of Gennari using MRI at 7T," *Society for Neuroscience Abstracts*, 2005.

[26] J. R. Polimeni, O. P. Hinds, M. Balasubramanian, A. van der Kouwe, L. L. Wald, A. M. Dale, B. Fischl, and E. L. Schwartz, "The human V1-V2-V3 visuotopic map complex measured via fMRI at 3 and 7 Tesla," *Society for Neuroscience Abstracts*, 2005.

[27] D. J. Struik, *Lectures on Classical Differential Geometry*, 2nd ed. Reading, MA: Addison-Wesley, 1961.

[28] M. P. do Carmo, *Differential Geometry of Curves and Surfaces*. Englewood Cliffs, NJ: Prentice-Hall, 1976.

[29] J. McCleary, *Geometry from a Differentiable Viewpoint*. Cambridge, UK: Cambridge University Press, 1994.

[30] D. R. Smith, *Variational Methods in Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[31] I. M. Gelfand and S. V. Fomin, *Calculus of Variations*. Englewood Cliffs, NJ: Prentice-Hall, 1963.

[32] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, "Triangulating a simple polygon," *Information Processing Letters*, vol. 7, no. 4, pp. 175–179, 1978.

[33] R. E. Tarjan and C. J. V. Wyk, "An $O(n \log \log n)$-time algorithm for triangulating a simple polygon," *SIAM Journal on Computing*, vol. 17, no. 1, pp. 143–178, 1988.

[34] L. C. Kinsey, *Topology of Surfaces*. New York, NY: Springer-Verlag, 1993.

[35] H. Edelsbrunner, *Geometry and Topology of Mesh Generation*. Cambridge, UK: Cambridge University Press, 2001.

[36] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM Journal on Computing*, vol. 16, no. 4, pp. 647–668, 1987.

[37] K. Polthier and M. Schmies, "Straightest geodesics on polyhedral surfaces," in *Mathematical Visualization: Algorithms, Applications, and Numerics*, H.-C. Hege and K. Polthier, Eds. New York, NY: Springer-Verlag, 1998, pp. 135–150.

[38] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[39] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, p. 345, 1962.

[40] J. Chen and Y. Han, "Shortest paths on a polyhedron," *International Journal of Computational Geometry and Applications*, vol. 6, pp. 127–144, 1996.

[41] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe, "Fast exact and approximate geodesics on meshes," *ACM Transactions on Graphics*, vol. 24, no. 3 (SIGGRAPH 2005), pp. 553–560, 2005.

[42] B. Kaneva and J. O'Rourke, "An implementation of Chen and Han's shortest paths algorithm," in *Proceedings of the 12th Canadian Conference on Computational Geometry*, 2000, pp. 139–146.

[43] H. Fuchs, Z. M. Kedem, and S. P. Uselton, "Optimal surface reconstruction from planar contours," *Communications of the ACM*, vol. 20, no. 10, pp. 693–702, 1977.

[44] O. P. Hinds, J. R. Polimeni, and E. L. Schwartz, "Brain surface reconstruction from slice contours," *NeuroImage*, vol. 31, no. 1, p. S445, 2006.

[45] B. Fischl, A. Liu, and A. M. Dale, "Automated manifold surgery: constructing geometrically accurate and topologically correct models of the human cerebral cortex," *IEEE Transactions on Medical Imaging*, vol. 20, no. 1, pp. 70–80, 2001.

[46] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*. New York, NY: Cambridge University Press, 1988.

[47] M. Balasubramanian, J. R. Polimeni, and E. L. Schwartz, "Quasi-isometric flattening of large-scale cortical surfaces," *Society for Neuroscience Abstracts*, 2005.

[48] ——, "Quantitative evaluation and comparison of cortical flattening algorithms," *Society for Neuroscience Abstracts*, 2006.

[49] B. Fischl, M. I. Sereno, and A. M. Dale, "Cortical surface-based analysis II: Inflation, flattening and a surface-based coordinate system," *NeuroImage*, vol. 9, no. 2, pp. 195–207, 1999.

[50] E. L. Schwartz, "Computational studies of the spatial architecture of primate visual cortex: Columns, maps, and protomaps," in *Primary Visual Cortex in Primates*, ser. Cerebral Cortex, A. Peters and K. Rockland, Eds. New York, NY: Plenum Press, 1994, vol. 10, pp. 359–411.