You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges

NICHOLAS SHARP and KEENAN CRANE, Carnegie Mellon University

This paper introduces a new approach to computing geodesics on polyhedral surfaces—the basic idea is to iteratively perform *edge flips*, in the same spirit as the classic Delaunay flip algorithm. This process also produces a triangulation conforming to the output geodesics, which is immediately useful for tasks in geometry processing and numerical simulation. More precisely, our FlipOut algorithm transforms a given sequence of edges into a locally shortest geodesic while avoiding self-crossings (formally: it finds a geodesic in the same *isotopy class*). The algorithm is guaranteed to terminate in a finite number of operations; practical runtimes are on the order of a few milliseconds, even for meshes with millions of triangles. The same approach is easily applied to curves beyond simple paths, including closed loops, curve networks, and multiply-covered curves. We explore how the method facilitates tasks such as straightening cuts and segmentation boundaries, computing geodesic Bézier curves, extending the notion of *constrained Delaunay triangulations (CDT)* to curved surfaces, and providing accurate boundary conditions for partial differential equations (PDEs). Evaluation on challenging datasets such as *Thingi10k* indicates that the method is both robust and efficient, even for low-quality triangulations.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: geodesic, edge flip, triangulation

ACM Reference Format:
Nicholas Sharp and Keenan Crane. 2020. You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges. *ACM Trans. Graph.* 39, 6, Article 249 (December 2020), 15 pages. https://doi.org/10.1145/3414685.3417839

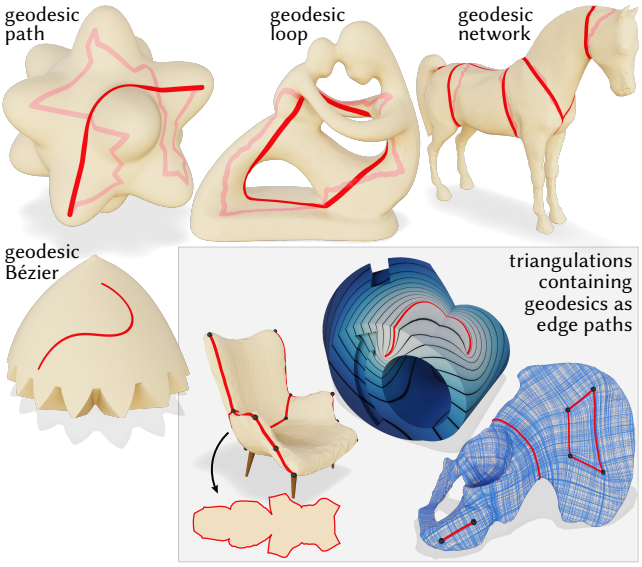


Fig. 1. We introduce an edge-flip based algorithm for computing geodesic paths, loops, and networks on triangle meshes. The algorithm also yields a triangulation containing these curves as edges, which can be used directly for subsequent geometry processing (*e.g.*, for cutting, or for solving PDEs).

1 INTRODUCTION

A *geodesic* is the natural generalization of a straight line to a curved surface: it is a trajectory of zero acceleration, or equivalently, a path of locally minimal length [Polthier and Schmies 2006]. Accurate geodesics are essential for tasks across science and engineering [Bose et al. 2011], and the ability to construct "straight lines" on polyhedral surfaces helps generalize classic algorithms from 2D computational geometry to curved surfaces (see for example Section 6.3). To date, there has been significant work on finding *minimal geodesics*, *i.e.*, globally shortest paths between two points (Section 2). In many applications, however, one desires a straight path different from the globally shortest one—consider for instance straightening a given cut graph or segmentation boundary (Section 6). Moreover, shortest path algorithms cannot find geodesics of more interesting topology, such as closed loops or curve networks (Figure 1, *top right*). The aim in this work is hence not to find geodesics of globally minimal length (which we cannot always guarantee—see Section 6.5), but rather to find locally shortest curves within the given isotopy class, *i.e.*, to "pull the given curves tight."

Importantly, geodesics are *intrinsic*: they do not depend at all at how a surface is embedded in space. For instance, a straight line drawn on a sheet of paper is still a geodesic if the paper is bent, folded, or rolled into a tube. Likewise, consider unfolding a pair of triangles ijk, jil into the plane. If we replace edge ij with the segment kl (performing an *intrinsic edge flip*), the intrinsic geometry of the surface—and hence its geodesics—remains unchanged, even though the mesh these triangles belong to may no longer admit a facewise linear embedding into $\mathbb{R}^3$. This perspective provides substantial flexibility in developing algorithms: it empowers us to "let go" of the triangulation initially used to encode the surface, and instead work with the much larger space of *intrinsic triangulations* (Section 3.1) which need not be realizable in $\mathbb{R}^3$. The basic idea of our algorithm, then, is to start with a path encoded as a sequence of edges. We repeatedly perform intrinsic edge flips to straighten this path (Figure 4, *inset*), until we ultimately obtain a geodesic.

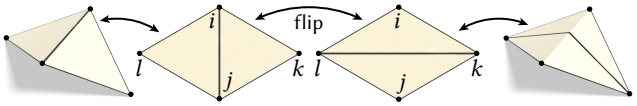


Fig. 2. An *intrinsic edge flip*, which is the basic operation in our algorithm. Note that we never need to actually embed the final, flipped triangulation.

Authors' address: Nicholas Sharp; Keenan Crane, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213.

0730-0301/2020/12-ART249
https://doi.org/10.1145/3414685.3417839

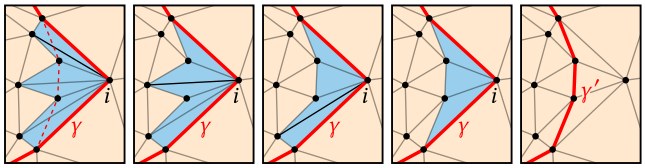


Fig. 3. FlipOut shortens a path through three consecutive vertices by pulling it as tight as possible (*left*, dashed line). To do so, it performs all valid edge flips in the "wedge" of triangles (in blue) incident on the middle vertex i. The curve γ' along the outside of the wedge *(far right)* is always shorter.

For an ordinary planar triangulation, our shortening operation is easy to describe. At any vertex i where the path is not yet straight, there will be a "wedge" of triangles of total angle less than 180° (Figure 3, *far left*). The following procedure always shortens the path, by locally pulling it tight:

- repeatedly flip any flippable edge out of the wedge (namely, any edge contained in a convex pair of triangles), and
- replace the initial path γ with the path γ' along the outside of the wedge.

Because this procedure literally flips edges out of a wedge, we call it the FlipOut algorithm. Geodesics are found by repeatedly applying FlipOut at non-straight vertices (Section 4). To extend this operation to curved surfaces we require additional data structures (Section 3), and must show that our local flipping procedure is guaranteed to reduce length in the general case (Appendix A).

2 RELATED WORK

Literature on geodesic algorithms is well-covered by several surveys [Peyré and Cohen 2005; Bose et al. 2011; Patané 2016; Crane et al. 2020]. Broadly, some algorithms compute *geodesic distance*, whereas ours finds *geodesic curves* which need not be minimal. Likewise, some algorithms interpret a mesh as an approximation of a smooth surface (in the spirit of scientific computing), whereas we view it as an exact description of the shape of interest (as often done in computational geometry). In short, we find exact geodesics that locally minimize length with respect to the given polyhedral geometry, and hence focus primarily on methods with similar capabilities.

2.1 Shortest Geodesics

Early algorithms for exact geodesics focused on the *shortest path problem*: find globally shortest geodesics from each vertex to a source. Such algorithms have roots in the strategy of Mitchell et al. [1987]: start at the source, and use a Dijkstra-like traversal to track "windows" of geodesic paths sharing a common history. This strategy does not apply to loops or networks, but has been generalized to polygonal sources [Bommes and Kobbelt 2007]. Kapoor [1999] gives an optimal algorithm for the minimal geodesic between two points, but it is prohibitively difficult to implement [Kirsanov 2008, Section 2]. Importantly, we do *not* advocate use of FlipOut when the goal is to compute the geodesic distance function—many other algorithms are specialized to this task [Kimmel and Sethian 1998; Surazhsky et al. 2005; Xin and Wang 2009; Crane et al. 2013; Ying et al. 2013; Xu et al. 2015; Ying et al. 2014; Qin et al. 2016; Wang et al. 2017; Ying et al. 2019; Adikusuma et al. 2020; Cao et al. 2020].

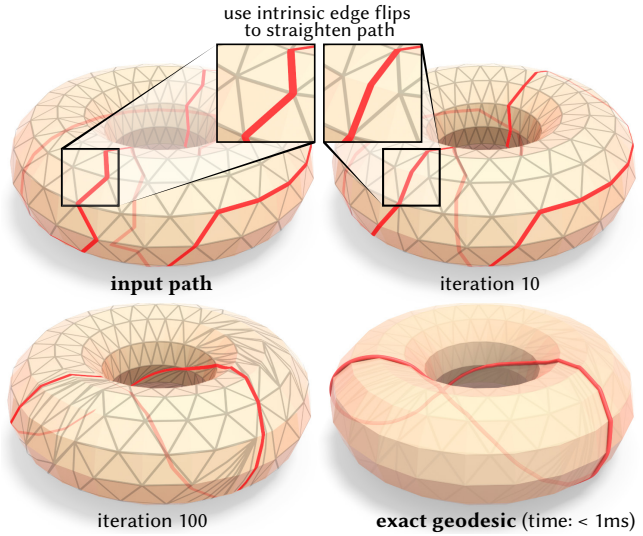
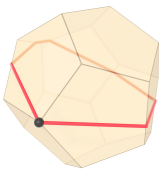


Fig. 4. Our algorithm straightens an input edge path (*top left*) to an exact polyhedral geodesic (*bottom right*) by systematically flipping edges of an intrinsic triangulation (inset).

2.2 Curve Shortening

Closer to our goal, several methods shorten a *given* curve—typically via an iterative process similar in spirit to the *curve shortening flow* from differential geometry [Gage 1990]. Such methods are critical in applications which do *not* seek shortest paths, as explored in Section 6. *Lagrangian* methods represent curves via vertices that can move freely along the surface or in $\mathbb{R}^3$ [Hass and Scott 1994; Martínez et al. 2005; Xin and Wang 2007; Appleboim et al. 2009; Xin et al. 2011; Han et al. 2017; Liu et al. 2017a; Remešíková et al. 2019; An 2019], whereas *Eulerian* methods encode curves as level sets of a scalar function [Sethian 1989; Wu and Tai 2010; Zhang et al. 2010]. Many of these methods seek to numerically approximate geodesics, whereas our method considers a discrete configuration space (the flip graph of a triangulation) that includes the exact solution. It hence avoids error-prone projection of approximate curves onto the surface, and terminates in finitely many operations (Theorem 4.2). From the viewpoint of discrete differential geometry [Crane and Wardetzky 2017] it faithfully captures key features of smooth curve-shortening flow: curves remain non-crossing and shrink to geodesics (or points) in finite time (*à la* [Hass and Scott 1994]). It can also be used to reliably find geodesics of mathematical interest, such as loops passing through a single vertex of a convex solid [Athreya et al. 2020] (see inset).



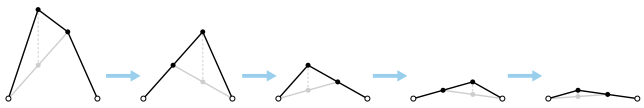


Fig. 5. Iterative averaging of continuous variables (here: coordinates in the plane) can converge very slowly to a limit solution. We consider a discrete state space, reaching the exact solution in finitely many steps.

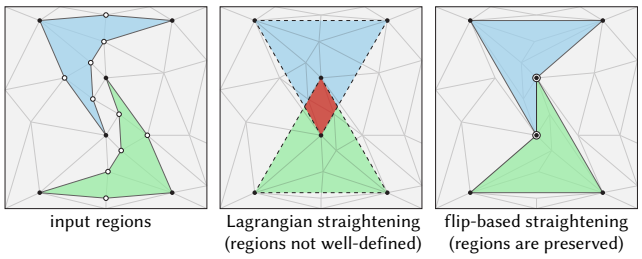


Fig. 6. Independently straightening each curve can introduce new crossings, which is problematic if the curves are meant to represent region boundaries. Our flip-based approach naturally preserves the given regions.

Like FlipOut, several methods transform a given curve into an exact, locally shortest geodesic. Martínez et al. [2005] perform iterative local relaxation of continuous Lagrangian variables (Figure 5), which can be very slow to converge (Figure 17). Xin and Wang [2007] and Xin et al. [2011] accelerate relaxation by finding straight lines through unfolded triangle strips, yielding exact paths and loops (*resp.*) after finitely many steps, and real-world performance very similar to ours (Section 5.2.1). In contrast to these methods, we encode curves via integer data on an evolving triangulation, rather than floating-point coordinates on a fixed mesh. Moreover, Lagrangian- and flip-based methods provide complementary functionality (Figure 6): Lagrangian methods allow curves to cross (important when the input has crossings), whereas flipping prevents crossing by construction (critical when curves represent, *e.g.*, cuts or region boundaries—see Section 6). Moreover, the flip-based approach produces a significant asset that previous methods do not: a triangulation conforming to the straightened curves. These curves can hence be immediately used as cut or boundary curves for subsequent mesh processing (Section 6).

2.3 Edge Flipping

Our method is also connected to Lawson's greedy flip-based algorithm for *Delaunay triangulation* [1977], which was generalized by Rivin [1994, Section 10] to *intrinsic* Delaunay triangulations of polyhedral surfaces [Kharevych et al. 2006; Dyer et al. 2007; Liu et al. 2015, 2017b]. Indermitte et al. [2001] and Bobenko and Springborn [2007] show that an intrinsic Delaunay triangulation can always be obtained via a finite number of flips; empirically, the number of flips tends to be roughly linear in mesh size [Sharp et al. 2019b, Figure 10]. FlipOut exhibits analogous behavior: it always yields a geodesic after finitely many flips (Theorem 4.2), and appears to take about $O(m^{1.5})$ flips in practice, where m is the input path length (Figure 16). Similar to Delaunay flipping, the crux of our termination proof is that there is a function which is reduced by each local operation (namely, the path length), and that the set of geodesics with bounded length is finite (Theorem 4.2). More broadly, flips have proven to be a powerful tool in geometry processing [Fisher et al. 2007; Sharp et al. 2019b], computational topology [Weeks 1993; Bern et al. 2002; Erickson 2014; Bell 2016] and discrete differential geometry [Rivin 1994; Springborn 2019], but to date have not been used to compute geodesics.

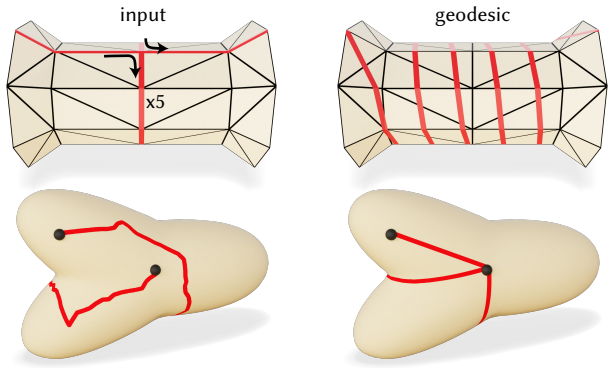


Fig. 8. Our path representation allows us to straighten paths that overlap many times (*top*), or those that get pulled tight around endpoints of the path itself (*bottom*).

3 PRELIMINARIES

3.1 Discrete Surfaces

We represent a polyhedral surface as an *intrinsic triangulation* $\mathcal{T} = (\mathsf{M}, \ell)$, where M gives the connectivity and edge lengths ℓ describe the geometry, as detailed below. See Sharp et al. [2019b] for a more thorough introduction to intrinsic triangulations.

3.1.1 Connectivity. We use $\mathsf{M} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ to denote the combinatorics of a triangulation with vertices V, edges E, and faces F. Formally, by a *triangulation* we mean a manifold two-dimensional Δ-*complex* in the sense of Hatcher [2002, Section 2.1]. The vertices of an edge or triangle in a Δ-complex are not required to be distinct—for instance, it can contain a *loop-edge* connecting a vertex to itself (see inset). Though such configurations do not ordinarily appear in the input, they can arise due to subsequent edge flips. A list of $k+1$ indices denotes a k-simplex—for instance, i is a vertex, ij is an edge, and ijk is a triangle (though in general, distinct indices may refer to the same vertices). We use $\overset{jk}{i}$ to denote a *triangle corner* at vertex i of triangle ijk; two triangle corners are *edge connected* if they share an edge, *e.g.*, $\overset{jk}{i}$ and $\overset{lj}{i}$ share edge ij. Finally, an expression of the form $a_i = \sum_{ijk} b_{ijk}$ means that a value a_i at vertices is obtained by summing values b_{ijk} over all triangles ijk containing i (and similarly for edges, *etc.*).

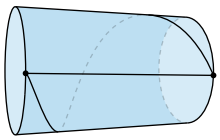
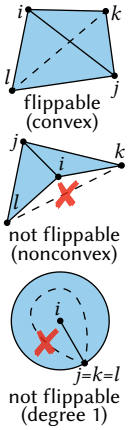


Fig. 7. A Δ-complex with two vertices, four edges, and two triangles.

3.1.2 Geometry. Though the geometry of a polyhedron is often given by its vertex positions $f : V \to \mathbb{R}^3$, we will need only the edge lengths $\ell_{ij} := |f_j - f_i|$. In general, any collection of edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$ that satisfy the usual triangle inequalities allow us to locally construct each face as a triangle in the Euclidean plane. From here, we can easily read off quantities such as the interior angles θ_i^{jk} at triangle corners, the angle sums $\Theta_i := \sum_{ijk \in \mathsf{F}} \theta_i^{jk}$, and the *angle defects* $\Omega_i := 2\pi - \Theta_i$, which provide a discrete analog of Gaussian curvature. This intrinsic data is all we will need to compute geodesics, though see Section 5.1 for comments about extracting the final path.

3.1.3 Intrinsic Edge Flips. Using an intrinsic triangulation enables us to modify connectivity while exactly preserving the original intrinsic geometry. The only operation we need is an *intrinsic edge flip* FlipEdge($\mathcal{T}$, ij), which replaces two triangles ijk, jil with the triangles ilk, klj, and reads off the new length ℓ_{kl} from the planar layout (Figure 2). Extrinsically, this new edge looks "bent", but intrinsically it is a perfectly straight geodesic along the input surface.

Most past work focuses on flipping non-Delaunay edges; serendipitously, such flips are always valid [Bobenko and Springborn 2007, Proposition 11]). We flip edges for a completely different reason (to compute geodesics), and must therefore carefully consider the conditions under which an edge can be flipped.



Definition. An edge ij is *flippable* if i and j have degree > 1, and the triangles containing ij form a convex quadrilateral when laid out in the plane.

These conditions ensure that we both maintain a valid combinatorial triangulation (*i.e.*, a valid Δ-complex) and preserve the intrinsic geometry of the original surface. Note that an ordinary, simplicial input mesh cannot have degree-1 vertices. However, such vertices can still arise from a sequence of edge flips, and in general it may not be possible to construct geodesics without passing through a non-simplicial triangulation. We therefore consider the more general case of a Δ-complex throughout.

3.2 Discrete Curves

3.2.1 Edge Paths. We represent curves as connected sequences of directed edges, which we refer to as *edge paths*. We call the edges of an edge path *segments* (reserving the word *edge* for mesh edges), and the vertices of an edge path *nodes* (reserving the word *vertex* for mesh vertices). *Interior nodes* are nodes not at path endpoints; *e.g.*, a closed loop has only interior nodes. A *joint* γ_{abc} is a sequence of two distinct segments (ab, bc). Joints are hence *oriented*, *i.e.*, $\gamma_{cba} \neq \gamma_{abc}$.

Note that since the triangulation itself is mutable (via edge flips), the set of curves that can be represented via edge paths is far larger than the set of paths along edges of the input mesh. Most importantly, we can always represent a geodesic in the same isotopy class as the given curve. To process input curves which might not be aligned with mesh edges, one can either cut the mesh along the curve (Figure 9), or simply snap the curve to nearby edges—yielding a much smaller output triangulation.

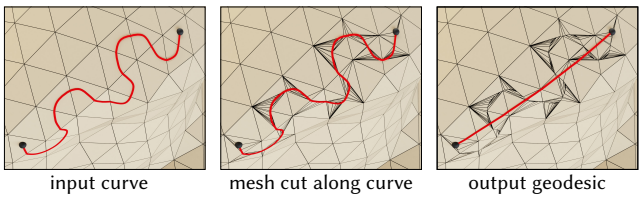


Fig. 9. Input curves not aligned with edges can be handled by either cutting the mesh (as shown here) or just snapping to a nearby sequence of edges.

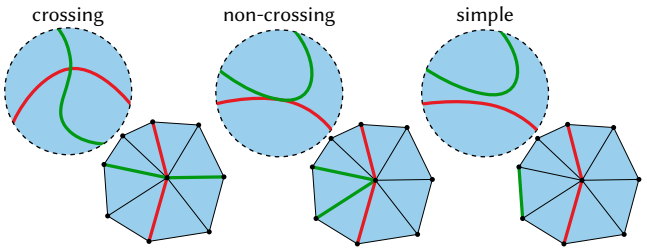


Fig. 10. We consider any path of edges that is *non-crossing*, *i.e.*, that can be locally turned into a simple curve by a small perturbation.

3.2.2 Non-Crossing Curves. We also require that edge paths are *non-crossing*, which is weaker than saying that they are *simple* (*i.e.*, injective). In particular, a curve γ is non-crossing if in an arbitrarily small open neighborhood around any point we can make a continuous motion (formally: a homotopy) such that the restriction of the new curve to this neighborhood is simple (Figure 10). In other words: if all intersections are *non-transversal*.

Non-crossing curves include multiply-covered curves, *i.e.*, those that wind over the same edge several times (Figure 8). To encode such paths, we store (i) the sequence of segments, and, (ii) for each edge in E, an ordered list of segments crossing over this edge (Figure 11). This data does not play a role in the description or analysis of our algorithm, and primarily serves to clarify implementation. A key strength of our approach is that we will guarantee that non-crossing curves remain non-crossing while a curve is being shortened; this property is crucial for applications in geometry processing (Section 6.1).

3.2.3 Wedges. A joint γ_{abc} partitions the triangles around vertex b into two sets (one of which may be empty), which we call *wedges*. The *degree* $\deg(\mathcal{W})$ is the number of triangles in wedge $\mathcal{W}$, and any edges on the wedge interior are *incident* on the wedge (for instance, a degree-1 wedge has no incident edges). The associated *wedge angle* $\alpha(\mathcal{W})$ is just the sum of interior angles at the common vertex i:

$$\alpha(\mathcal{W}) := \sum_{\substack{jk \\ i} \in \mathcal{W}} \theta_i^{jk}. \tag{1}$$

The *outer arc* of a wedge $\mathcal{W}$ is the collection of edges opposite any corner in $\mathcal{W}$. A *boundary wedge* $\mathcal{W}$ is any wedge that includes edges on the boundary of the surface; here we define the wedge angle to be $\alpha(\mathcal{W}) := \infty$, which helps simplify our algorithm.

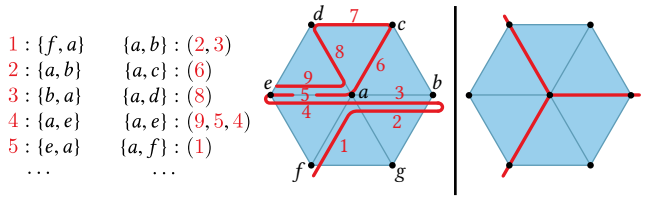


Fig. 11. *Left:* A curve may include the same edge multiple times. To specify its isotopy class, we store an ordered list of segments at each edge. *Right:* We represent more general curve networks by just storing several curves.

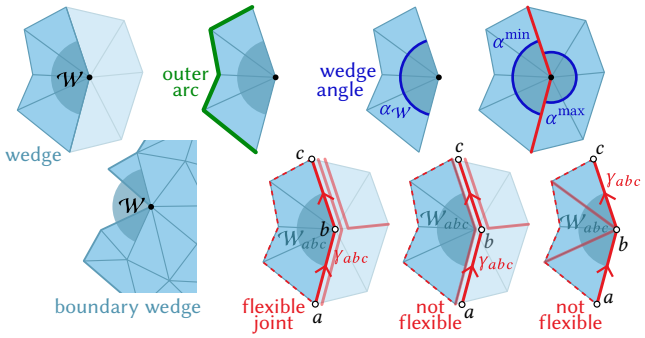


Fig. 12. Anatomy of a wedge. Flexible joints γ_{ijk} can be shortened, since they are not "blocked" by other parts of the path.

We use $\alpha_b^{\min}$ and $\alpha_b^{\max}$ to denote the smaller and larger wedge angles (*resp.*); if the two angles are equal, the designation of min/max is arbitrary. We use $\mathcal{W}_{abc}$ to denote the wedge to the left of γ_{abc}, and α_{abc} to denote the angle of $\mathcal{W}_{abc}$. Finally, a joint γ_{abc} in path γ is *flexible* if it is isotopic to the outer arc of $\mathcal{W}_{abc}$ (Figure 12, *bottom right*). More explicitly, γ_{abc} is flexible if (i) no other segments from γ are incident on $\mathcal{W}_{abc}$, and (ii) both ab and bc are the leftmost segments on their respective edges. Intuitively, a flexible wedge is not "blocked" from moving by any other segments of the curve.

3.2.4 Geodesics. A *geodesic* on a polyhedral surface is a curve that cannot be made shorter by any local perturbation. Away from vertices, a polyhedral geodesic just look likes a straight line when the triangles it passes through are unfolded into the plane. A curve passing through a positively curved vertex ($\Omega_i > 0$) cannot be a geodesic, since we can always make it shorter by going around the cone tip. However, there are many locally shortest paths passing through any negatively curved vertex ($\Omega_i \leq 0$). In general, then, a polyhedral geodesic is comprised of straight lines through triangle strips, possibly meeting at vertices of nonpositive curvature. See [Crane et al. 2020, Section 2.2.1] for further discussion.

3.2.5 Geodesic Edge Paths. To see if an edge path is a geodesic, we need only examine its behavior at vertices. In particular, a flexible joint γ_{abc} can always be made shorter if $\alpha_{abc} < \pi$ (by pushing it into the wedge). Hence, we say that γ_{abc} is *locally shortest* if

$$\alpha_b^{\min} \geq \pi. \tag{2}$$

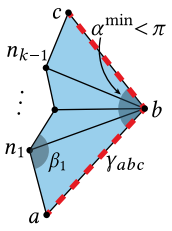
As expected, this definition implies that a locally shortest joint cannot pass through a positively-curved vertex: we have

$$\Theta_b = \alpha_b^{\min} + \alpha_b^{\max}, \tag{3}$$

and since $\alpha_b^{\max} \geq \alpha^{\min} \geq \pi$, we have $\Theta_b \geq 2\pi$, *i.e.*, $\Omega_b \leq 0$. (Note that a locally shortest node at a flat vertex is somewhat non-generic: there would have to be a partition into two wedges of angle exactly equal to π.) An edge path is then geodesic if every interior node is locally shortest. This definition agrees with the usual notion of locally shortest polyhedral geodesics considered in other exact algorithms [Mitchell et al. 1987, Lemma 3.5], rather than the *straightest* definition studied by Polthier and Schmies [2006].

4 ALGORITHMS

We now give the general version of the FlipOut algorithm, which provably reduces length at any node that is not already locally shortest. We can then find geodesic paths (Section 4.2), loops (Section 4.3) and curve networks (Section 6.1) by greedily applying FlipOut until Equation 2 holds at every interior node. Although an intrinsic triangulation is used for computation, the final path is easily extracted either as a sequence of edge crossings on the input mesh, or as points in $\mathbb{R}^3$ (Section 5.1).



Throughout we consider a local isometric unfolding of any wedge $\mathcal{W}$ into the plane, as shown in the inset. However, this unfolding never needs to be constructed explicitly, since all quantities of interest can be obtained directly from the intrinsic edge lengths ℓ. Let $k := \deg(\mathcal{W})$, and let $a = n_0, n_1, \ldots, n_k = c$ be the vertices along the outer arc. For $i = 1, \ldots, k-1$, let β_i be the angle between edge $n_{i-1}n_i$ and edge $n_i n_{i+1}$ in $\mathcal{W}$. Whenever an edge is flipped, these quantities are re-labeled as just described. Note that in a general Δ-complex the elements in this diagram may not all be distinct—for instance, one of the n_i may be equal to b. Appendix A examines such cases in detail.

4.1 Local Shortening by Edge Flips

Consider a path consisting of a single joint γ_{abc}. If the angle α_{abc} is less than π, then we can always shorten the path by the simple edge flipping procedure in Algorithm 1. This algorithm iterates through edges incident on the smaller wedge $\mathcal{W}$, "flipping out" any edges with $\beta_i < \pi$ to remove them from the wedge, until $\beta_i \geq \pi$ at all vertices $n_1, \ldots, n_{k-1}$. The process is visualized in Figure 13.

Algorithm 1 FlipOut($\mathcal{T}, \gamma_{abc}$)

Input: A triangulation $\mathcal{T}$, and a flexible joint γ_{abc} where $\alpha_{abc} < \pi$ and segments ab, bc are distinct.
Output: A shorter edge path γ_{shorter} connecting a to c in an updated triangulation $\mathcal{T}$.

```
while any β_i < π do
    j ← min i s.t. β_i < π        ▷ choose the first edge with β_i < π
    FlipEdge(T, bn_j)
end while
γ_shorter ← (a, n_1, ..., n_{k-1}, c)        ▷ path along the outer arc
return T, γ_shorter
```

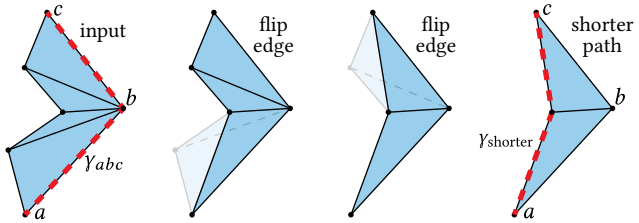


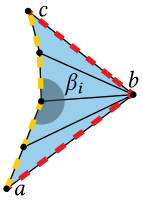
Fig. 13. An edge path that is not yet geodesic can always be shortened by flipping edges incident on a non-straight vertex, as described in Algorithm 1, and replacing the old path γ_{abc} with the new path γ_{shorter}.

Implementation is simplified by observing that the smallest index j is at most one smaller than its value on the previous iteration, because all earlier angles are unchanged. If $\deg(\mathcal{W}_{abc}) = 1$, the wedge contains just a single corner $\overset{ac}{b}$ and the new path is just the edge ac. If $\deg(\mathcal{W}_{abc}) = 0$, the path contracts along an edge. (In a practical implementation we must also update the segment lists for all edges on the wedge boundary.)

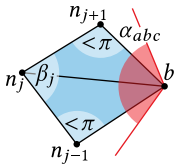
Importantly, FlipOut always reduces the path length:

Theorem 4.1. *When Algorithm 1 terminates,* $|\gamma_{\text{shorter}}| < |\gamma_{abc}|$, i.e., *the new path is shorter than the input path.*

Proof. Upon termination the angles β_i are all greater than or equal to π. Hence, in the planar layout, γ_{shorter} is a convex curve contained in the initial curve γ_{abc} (see inset). A corollary of *Crofton's formula* [1868] is that for two nested convex curves sharing endpoints, the inner one is shorter. Since the planar layout is isometric, the lengths of curves in this planar diagram exactly match the lengths of edge paths on the surface, and thus $|\gamma_{\text{shorter}}| < |\gamma_{abc}|$. □



Of course one must also argue that Algorithm 1 terminates in a finite number of steps, and that each step is valid (*i.e.*, edges that need to be flipped are actually flippable). For a simplicial complex, these facts are easy to establish: the algorithm terminates because each flip decreases the degree of $\mathcal{W}_{abc}$; all flips are valid since $\beta_i < \pi$ and $\alpha_{abc} < \pi$, hence the edge diamond is convex (and there are no degree-1 vertices in a simplicial complex). Additional machinery is needed to prove termination and validity in the general case, as described in Appendix A.



4.2 Geodesic Paths

Given an edge path $\gamma_{x\leftrightarrow y}$ between vertices x and y, we can iteratively apply FlipOut to interior nodes to obtain an exact polyhedral geodesic in finitely many steps (Theorem 4.2). Here, the subroutine UpdatePath$(\gamma_{x\leftrightarrow y}, \gamma, \gamma')$ simply replaces the old subpath γ with the new (shorter) subpath γ'. In practice we maintain a priority queue of flexible joints, sorted by smallest α.

Algorithm 2 MakePathGeodesic$(\mathcal{T}, \gamma_{x\leftrightarrow y})$

Input: A triangulation $\mathcal{T}$ and an edge path $\gamma_{x\leftrightarrow y}$, connecting vertices x and y.

Output: A geodesic edge path $\gamma_{x\leftrightarrow y}$ in an updated triangulation $\mathcal{T}$.

1: **while** $\gamma_{x\leftrightarrow y}$ is not geodesic **do**
2: $\quad\gamma_{abc} \leftarrow$ flexible joint in $\gamma_{x\leftrightarrow y}$ with smallest angle α_{abc}
3: $\quad\mathcal{T}, \gamma_{\text{shorter}} \leftarrow$ FlipOut$(\mathcal{T}, \gamma_{abc})$ ▷ *locally shorten*
4: $\quad\gamma_{x\leftrightarrow y} \leftarrow$ UpdatePath$(\gamma_{x\leftrightarrow y}, \gamma_{abc}, \gamma_{\text{shorter}})$ ▷ *update*
5: **end while**
6: **return** $\mathcal{T}, \gamma_{x\leftrightarrow y}$

Processing flexible joints means we flip only edges that are not part of a path, and hence need not consider how flips should modify the path. It also ensures that we preserve the isotopy class (and thus the non-crossing property): since the joint γ_{abc} is flexible, it can (by definition) be moved to the outer arc of its wedge $\mathcal{W}_{abc}$ via an isotopy. One might worry, however, that we run out of flexible joints before the path is geodesic. Fortunately this is not the case: if a joint is not flexible, then either (a) it is blocked by some other joint (or sequence of joints), in which case the outermost joint can be shortened and removed, or (b) it is blocked by an endpoint, in which case it is already locally shortest within its isotopy class. Finally, the algorithm always finishes in finite time (though note that this proof does not apply to closed loops—see Section 4.3):

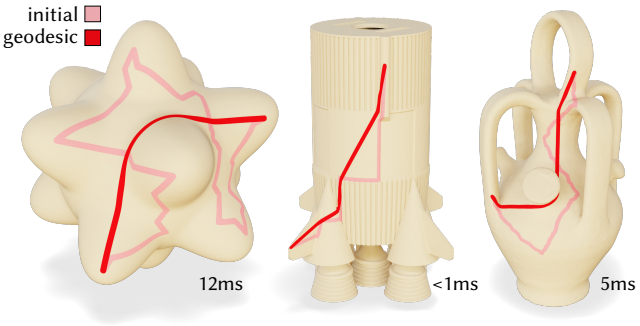


Fig. 14. Basic results from our method, where an initial path is shortened to a geodesic by flipping edges. Inset values give the runtimes. All of the resulting curves shown are exact polyhedral geodesics.

Theorem 4.2. *Algorithm 2 terminates in finitely many iterations.*

Proof. The path γ_t at iteration t is a collection of segments which are geodesic curves between vertices. We have $|\gamma_{t+1}| < |\gamma_t|$ by Theorem 4.1, and will denote the initial (maximum) length by $L = |\gamma_0|$. To show termination, we will argue that the set of possible paths γ_t is finite. Consider $\mathcal{G}_L$, the set of all geodesic curves which connect pairs of vertices and have length $\leq L$; this set is finite [Indermitte et al. 2001, Prop. 1]. Let $l_{\min}$ be the shortest curve in $\mathcal{G}_L$, and observe that all γ_t have at most $n_{\max} := \lfloor L/l_{\min} \rfloor$ segments. Thus every possible path γ_t is a collection of at most $n_{\max}$ geodesics from the finite set $\mathcal{G}_L$, and there are finitely many such collections. □

Rather than running to completion, one can also terminate Algorithm 2 when there is a sufficient length decrease, or a sufficiently large $\alpha^{\min}$. Doing so generates curves which are shorter/straighter but not yet geodesic, akin to curve-shortening flow (Section 6.1).

4.3 Closed Geodesic Loops

Algorithm 2 can be used to find geodesic loops (Figure 15) with one small modification. Near termination, a loop γ might be comprised of a single node i connected to itself via a loop-edge—but still have an angle $\alpha_i^{\min} < \pi$. This node violates the precondition of Algorithm 1, since the incident segments are not distinct. Here we simply replace the loop-edge with the two opposite edges of the triangle in the wedge at i. This step makes γ longer, but allows Algorithm 2 to continue making progress (see Appendix B). The modified procedure cannot get stuck: flexible joints can always be processed. However, since the curve can now get longer, our proof of termination no longer applies. In practice, we have always observed termination, obtaining geodesic loops after a fairly small number of iterations.

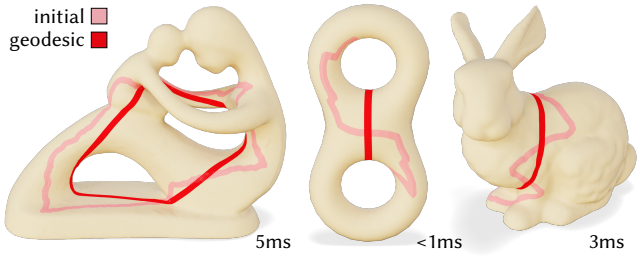


Fig. 15. Our method also applies to loops—here initials loop are shortened to geodesics via edge flips.

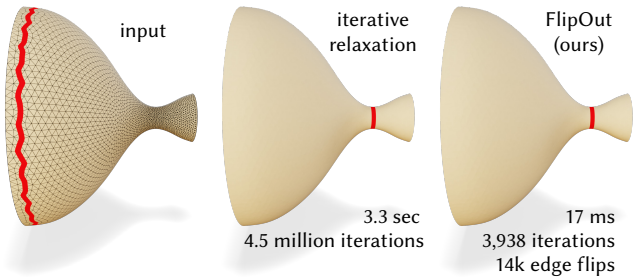


Fig. 17. For a near-straight curve far from the solution, iterative local relaxation à la [Martínez et al. 2005] converges very slowly. Our flip-based scheme gives the exact result in finitely many iterations, and is 200x faster.

5 EVALUATION

5.1 Experimental Setup

Implementation. Our implementation uses the `geometry-central` (C++) mesh processing library [Sharp et al. 2019a]; timings reported in figures are measured using a single core of an i7-4790K CPU on a machine with 16GB RAM. To determine where the final geodesic path crosses edges of the input triangulation we use the signpost data structure from Sharp et al. [2019b], though one could also use the data structure of Fisher et al. [2007]. In all experiments we include the cost of extracting the path in the timing, and verify that the resulting path is indeed an exact geodesic, as detailed below.

Datasets. We ran experiments on the manifold, non-degenerate models in the *Princeton Shape Benchmark* [Shilane et al. 2004] (~600 models), the *MPZ* dataset [Myles et al. 2014] (~ 100 models), and the *Thingi10k* dataset [Zhou and Jacobson 2016] (~ 6200 models).

Task. We compute many geodesics by shortening randomly generated initial paths. For each mesh we perform 10 trials in which we sample two random (connected, distinct) vertices on the mesh, compute an initial edge path between them using Dijkstra's algorithm, shorten the path to a geodesic using Algorithm 2, and extract the resulting path as a polyline along the input surface. Trials where the initial path was already geodesic are omitted from timing plots.

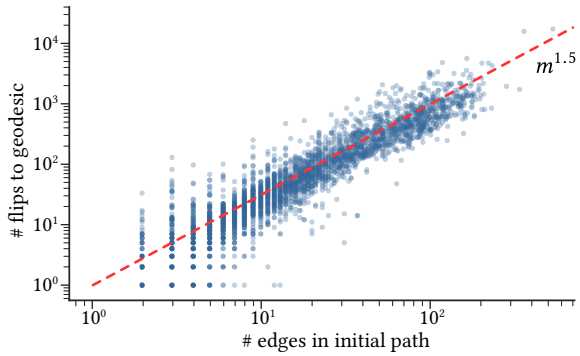
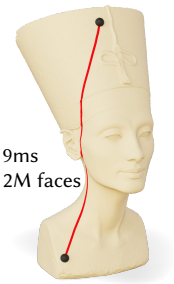


Fig. 16. We do not derive any worst-case bound for the number of edge flips needed to straighten a particular path, but in practice our algorithm scales well. On the MPZ dataset, the number of flips needed to make a path a geodesic goes as $m^{1.5}$, where m is the initial path length.

5.2 Performance



Our method is extremely fast in practice: typical examples take a few milliseconds, large meshes take only 10s of milliseconds (see inset), and our most expensive experiment took just 0.5 seconds to shorten a 1500 segment path down to a single-segment geodesic. As a point of reference, straightening a shortest path to a geodesic adds negligible overhead to initializing the path via Dijkstra's algorithm (about 1/10th the cost—see Figure 18). Empirically, the number of edge flips to straighten a path scales as $O(m^{1.5})$, where m is the number of initial path segments (Figure 16). Since a mesh with V vertices has shortest edge paths of average length $m \approx \sqrt{V}$, this translates to a roughly $O(V^{0.75})$ cost for straightening a shortest path between two vertices.

5.2.1 Performance Comparisons. We also compared performance with other exact path straightening methods. Importantly, all recent path straightening methods (with the exception of Martínez et al. [2005]) cost significantly less than computing an initial guess via Dijkstra's algorithm. Hence, the relative cost of these methods makes little difference for real-world use; other features such as accuracy, robustness, and the type of inputs/outputs provided are likely more important in practice. Note that we did not perform direct comparisons with window-based methods, which are quite fast but solve a fundamentally different problem (see Section 2.1).

Comparison to iterative relaxation. In our experiments, FlipOut was about 100× faster on average than the iterative relaxation scheme of Martínez et al. [2005] (Figure 18); we used the recommended stopping tolerance of $\epsilon = 0.005$ radians, but initialized via Dijkstra rather than fast marching since this choice had no significant effect on relative timings. The slow performance of iterative relaxation might be attributed to the fact that the amount of progress made on each iteration is not bounded away from zero (Figure 5), whereas each FlipOut iteration decreases the size of a discrete set of possible states (Theorem 4.2). Acceleration via L-BFGS can help, but yields only approximate geodesics [Liu et al. 2017a, Figure 1] and very similar performance to FlipOut (compare speedup relative to Martínez et al. [2005] in Figure 18 with Liu et al. [2017a, Table 2]). On the other hand, optimization-based schemes like these nicely support user-defined penalties like anisotropic metrics.

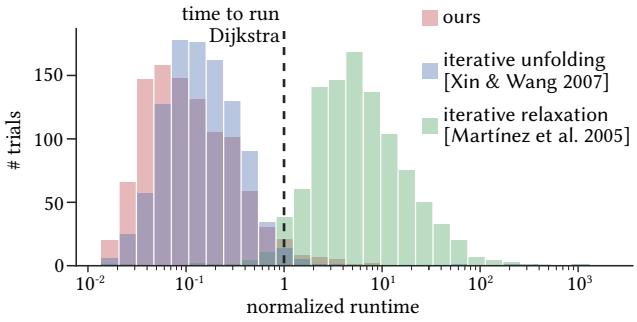
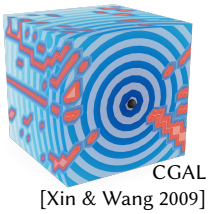
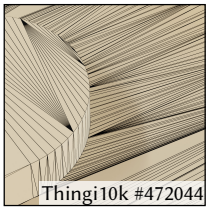


Fig. 18. On average, our method is about 100× faster than basic iterative relaxation, and comparable to iterative unfolding. These histograms show runtimes on the MPZ dataset for shortening an initial path to a geodesic, normalized by the time to run Dijkstra's algorithm between the endpoints. Our method adds little additional cost on top of computing an initial Dijkstra path; here only about 10% more.

Comparison to iterative unfolding. FlipOut also achieves comparable performance to the iterative unfolding scheme of Xin and Wang [2007] (Figure 18), which computes exact shortest paths in unfolded triangle strips; Xin et al. [2011] apply a similar technique for loops. A possible explanation for the similar performance is that these schemes construct similar sequences of geodesic polylines between vertices. Yet in contrast to iterative unfolding, FlipOut guarantees that curves remain non-crossing, and provides a conforming triangulation suitable for subsequent processing.

5.3 Floating Point Robustness





In exact arithmetic, FlipOut is guaranteed to yield an exact geodesic after finitely many iterations. Yet algorithms that are provably correct in exact arithmetic can still have trouble when implemented in finite-precision floating point. For instance, even CGAL's implementation of Xin and Wang [2009] can yield bogus results due to floating point issues, as shown in the inset where some vertices are erroneously marked as unreachable [Kiazyk et al. 2015]. Likewise, if meshes are bad enough (as in the inset, *bottom*), floating-point implementations of exact curve shortening algorithms can suffer from numerical issues (*e.g.*, iterating forever). Our implementation of FlipOut yields a geodesic polyline to within 10^{-4} radians on 99.4% of trials from Section 5.1; problem cases can all be attributed to extremely skinny triangles in the input mesh. For comparison, our implementations of Xin and Wang [2007] and Martínez et al. [2005] achieve success rates of 96.2% and 84.4% respectively, for the same task. In the case of FlipOut, challenging models might easily be dealt with by applying the simple intrinsic mollification strategy recently proposed in [Sharp and Crane 2020, Section 4.5]. On meshes without near-degenerate triangles our floating point implementation of FlipOut works perfectly—achieving, for instance, a 100% success rate on the MPZ dataset.

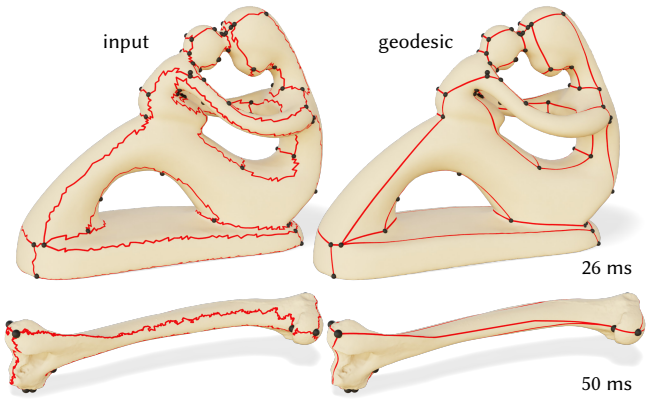


Fig. 19. Curve networks arise naturally as texture seams—here for instance seams on 3D scans of a statue (*top*) and a panda humerus (*bottom*) are straightened by our method while preserving the seam topology.

6 EXTENSIONS AND APPLICATIONS

We briefly explore applications enabled by our approach, many of which require geodesics beyond shortest paths: straightening curves that are far from minimal, working with closed loops or general networks, preserving partitions or cut regions, or using a conforming triangulation for subsequent mesh processing. Overall, a unified framework for straightening curves inside a triangulation provides a natural analog for constraining segments in a 2D triangulation—a critical tool in 2D computational geometry (Section 6.3).

6.1 Curve Networks

A *curve network* is a collection of non-crossing paths and loops, with endpoints meeting at fixed, marked vertices. Such networks arise naturally when, *e.g.*, segmenting a surface into regions, or cutting a surface to flatten it—in such applications, generating intersecting curves during straightening can render the output meaningless (Figure 6). Algorithm 2 can be applied to networks with essentially no modification: we simply choose a joint from *any* curve to shorten on each iteration. Since all curves are contained in the same triangulation, the topology of the network is automatically preserved. Figure 19 shows an example where jagged texture seams in a parameterization are naturally smoothed out by the shortening process. Similarly, segmenting mesh faces yields unnatural, irregular boundaries; shortening these curves yields more plausible smooth part boundaries (Figure 21). For segmentations, we stop the flow when curves reach 90% of their original length, preventing regions from drifting or contracting to a point.

Low-curvature seams are especially valuable for computational fabrication, where designs are cut out of flat materials such as thin plywood or sheet metal [Callens and Zadpoor 2018]—here jagged seams may be difficult to manufacture. Accordingly, recent research has sought to design smooth cuts [Lucquin et al. 2017; Sharp and Crane 2018]. One approach is to connect special *darts* or *cone points* via shortest paths [Kharevych et al. 2006]; we further improve these cuts via straightening (Figure 22), and generate a conforming triangulation which facilitates subsequent analysis (Section 6.4).

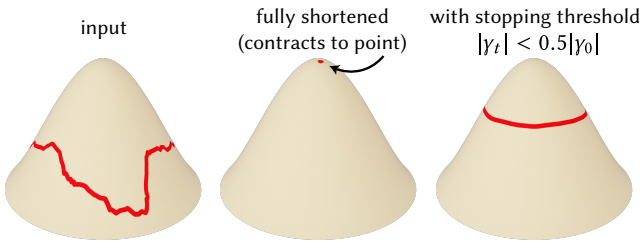


Fig. 20. Our Algorithm 2 acts as curve-shortening flow; stopping the procedure early via a length or angle threshold generates straighter curves, without drifting too far from the initialization or contracting to a point.

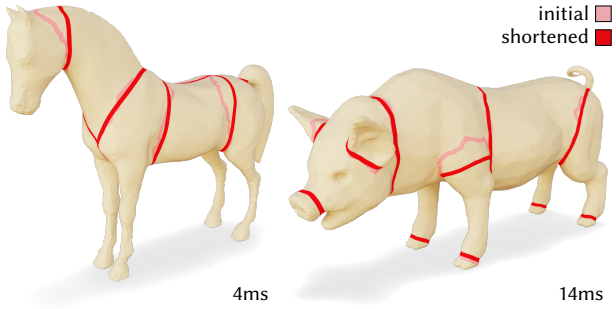


Fig. 21. Face-based segmentations have jagged region boundaries—straightening boundaries yields a more natural segmentation while preserving patch topology. A length threshold prevents excessive drift. Segmentations from Chen et al. [2009].

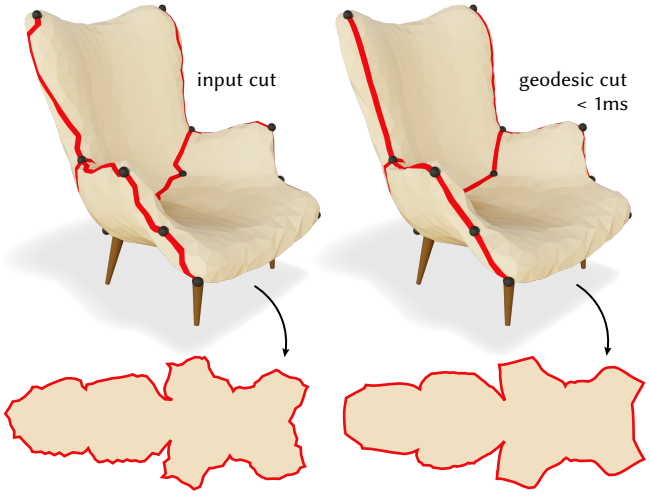


Fig. 22. A common operation in computational fabrication is to cut and flatten a shape to be manufactured from sheets of material. Our method is perfectly suited to straighten an initial cut network along edges (*left*) to a geodesic network (*right*), yielding a much more natural pattern for fabrication (*bottom*).

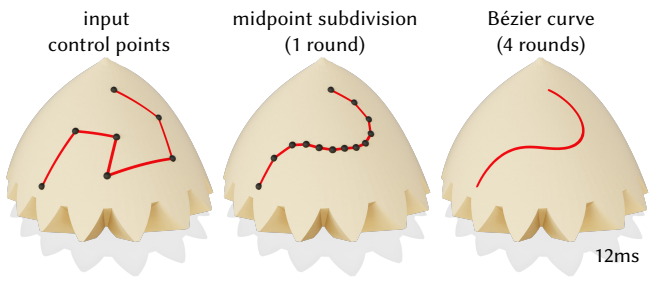


Fig. 23. Bézier curves are widely used to represent smooth curves in space. Geodesic shortening provides the necessary operation to construct Bézier curves on surfaces, via midpoint subdivision with de Casteljau's algorithm.

6.2 Geodesic Bézier Curves

Morera et al. [2008] extend Bézier curves to polygonal surfaces—such curves are useful for, *e.g.*, annotation and geometric modeling. Given a set of control points (connected by Dijkstra paths), a geodesic version of de Casteljau's algorithm transforms the control polygon to a smooth Bézier curve by repeated application of the following procedure:

(1) Shorten all curves between control points to geodesics
(2) Insert a new control point vertex at the midpoint between each pair of old control points
(3) Un-mark all old control points except the first and last
(4) If there are >2 points left, return to (1), and shrink the working set to exclude the first and last control points.

By using Algorithm 2 for step (1) as shown in Figure 23, we inherit all the benefits of our flip-based scheme, such as the construction of a conforming triangulation (Figure 26, *bottom*).

6.3 Intrinsic Constrained Delaunay Triangulation (iCDT)

In the plane, a *constrained Delaunay triangulation (CDT)* is a maximally Delaunay triangulation of a fixed set of points $p_1 \ldots, p_k \in \mathbb{R}^2$ and a prescribed set of segments, known as a *planar straight line graph*—see Chew [1989] for a precise definition. The ability to include segments is critical for, *e.g.*, meshing a simulation domain for finite element analysis.

We generalize this idea to surface meshes, where the mesh vertices play the role of the points p_i (Figure 25), and straight segments are replaced by geodesics. We call such input a *polyhedral geodesic curve graph (PGCG)*. Specifying this graph is slightly trickier than in 2D, since (i) there is not a unique geodesic between two endpoints, and (ii) it is difficult for a user to provide segments that are already geodesic. We hence take as input any collection of edge paths in the same isotopy class as the desired PGCG (Figure 25, *bottom left*) and apply our straightening procedure. We then apply the usual Delaunay flip algorithm (Lawson's algorithm), but do not flip any edges belonging to the PGCG. The result is an *intrinsic constrained Delaunay triangulation (iCDT)* (Figure 25, *bottom center*). As in the plane, no new vertices need to be inserted to construct this triangulation, and all prescribed (geodesic) segments are included.

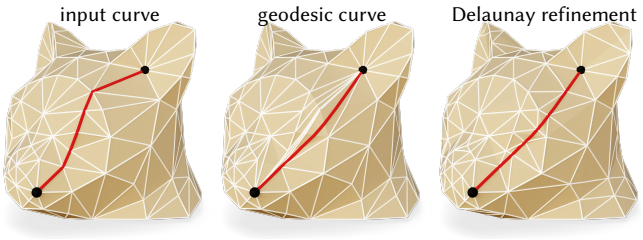


Fig. 24. Straightening a geodesic may introduce long skinny triangles (*center*); applying intrinsic Delaunay refinement as a post-process yields a high quality triangulation well-suited for subsequent computation (*right*).

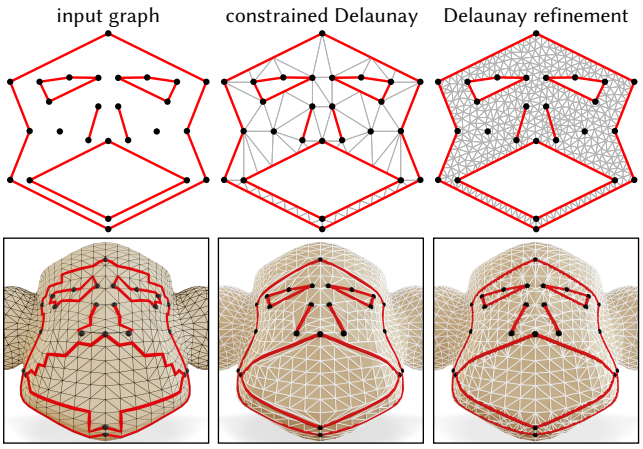


Fig. 25. In the plane, a *constrained Delaunay triangulation (CDT) (top center)* contains a given set of input segments *(top left)*; CDTs with good angle bounds *(top right)* are critical for, *e.g.*, numerical simulation. The intrinsic triangulations produced by our geodesic straightening procedure extend CDTs to surface meshes *(bottom row)*.

6.4 Boundary Conditions for PDEs

Even after Delaunay flips, a geodesic CDT may contain poor-quality triangles, making it ill-suited for calculations like solving partial differential equations (PDEs). If greater element equality is desired we can run intrinsic Delaunay refinement (Figure 25, *bottom right*), as described by Sharp et al. [2019b]. Geodesic curves are split as though they are boundary edges [Chew 1993; Shewchuk 1997]. This process yields a triangulation where elements have good aspect ratios, satisfy the intrinsic Delaunay property, and exactly conform to the requested geodesic paths. In contrast to, say, slicing along geodesics and retriangulating (*à la* Figure 29, *left*) or weakly enforcing boundary conditions on the input mesh, the intrinsic triangulation can be used directly for simulation and PDE-based geometry processing via standard algorithms. For instance, Figure 22, *bottom right*, shows a parameterization computed using the method of Springborn et al. [2008]; Figure 26 shows examples of solving a Poisson equation subject to Dirichlet boundary conditions, and generating a feature-aligned cross field (*à la* [Knöppel et al. 2013]).

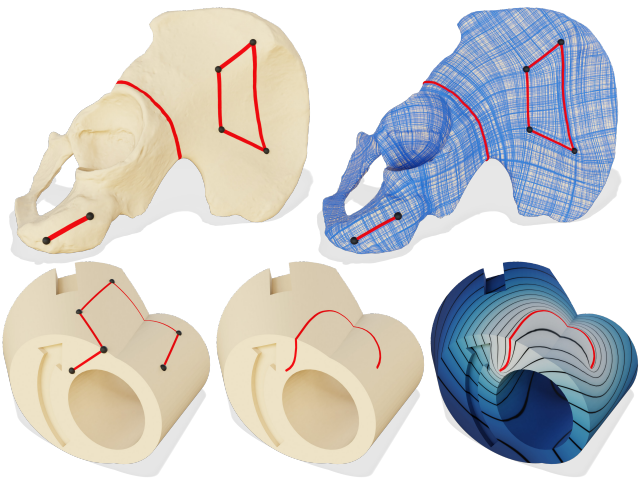


Fig. 26. An intrinsic triangulation conforming to geodesic or Bézier curves can be used to directly enforce boundary conditions for PDE-based algorithms. Here we compute the smoothest cross field aligned to a set of geodesic paths *(top)*, and solve a Poisson equation with Dirichlet boundary conditions enforced along a geodesic Bézier curve *(bottom)*.

6.5 Single Source Geodesic Paths

Our flip-based strategy can also be used to construct a tree of geodesics from all vertices to a given source, all of which are either minimal or very close to minimal. Remarkably, all of these geodesics can be simultaneously realized by edges of a single intrinsic triangulation (Figure 27). The basic idea is to repeatedly run Dijkstra's algorithm from the source vertex, use FlipOut to shorten a single flexible wedge, then recompute the shortest path tree until convergence. However, we do not recompute the entire tree each time—instead we incrementally straighten paths at the frontier of the Dijkstra search, accepting a vertex only once it is connected to the source by a geodesic. This procedure is guaranteed to yield a geodesic tree in finite time by the same arguments as in Theorem 4.2.

Experimentally, this technique provides exact distances to most vertices: for instance, in Figure 27 more than 95% of geodesics are minimal (up to a relative error in length of less than 10^{-6}); of the remaining 5%, the maximum error was no more than 1.04%. However, we did not compare against existing shortest geodesic algorithms (Section 2.1), which are specifically tailored to this problem and are likely more accurate/efficient. The intriguing observation here is simply that the solution can be obtained via edge flips, and represented in a triangulation of the same size as the input.

This same technique also provides a *logarithmic* map from the source, which gives the direction and distance one needs to walk from the source vertex i to reach any given target vertex j (Figure 28). Such maps are valuable in geometry processing for tasks from surface statistics to decaling [Schmidt et al. 2006; Sharp et al. 2019c; Herholz and Alexa 2019]. For each target vertex we locate the ancestor edge in the geodesic tree which emerges from the source—the direction of this edge in the tangent space of the source gives the angular component of the logarithmic map, and the geodesic distance gives the magnitude.

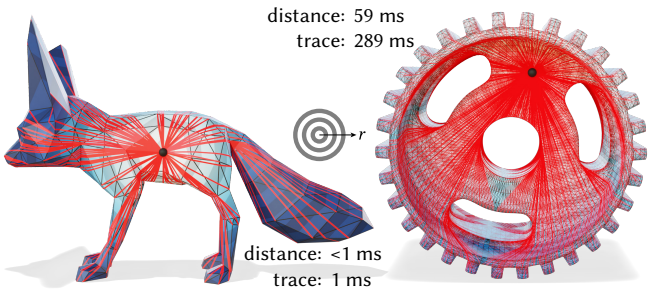


Fig. 27. Our approach can also be used to find geodesics to all vertices from a single source. Perhaps surprisingly, all of these geodesic edges exist simultaneously in a single intrinsic triangulation.

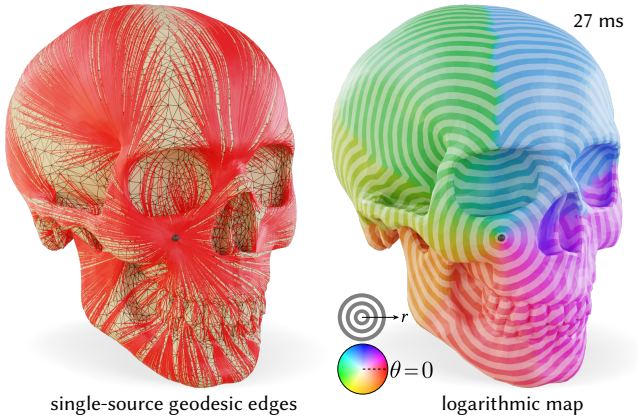


Fig. 28. Our FlipOut procedure can be used to efficiently introduce a geodesic edge path from a single source to all other vertices (*left*). Reading off distance to each vertex and the direction of the edges emanating from the source yields a discretization of the *logarithmic map* (*right*). The cyclic color palette encodes the angular component of the logarithmic map, while stripes denote isocontours of its magnitude.

6.6 Comparison to Lagrangian Approaches

One could also try implementing the applications from this section using a Lagrangian method, such as Martínez et al. [2005], Xin and Wang [2007], or Xin et al. [2011]. Like FlipOut, these methods yield exact locally shortest geodesic paths (Figure 30) but with a fundamental difference: they do not produce a triangulation conforming to the straightened paths. Moreover, inserting a Lagrangian curve by "slicing" through triangles introduces additional vertices, whereas FlipOut preserves the input vertex set (which can be important for downstream processing). This property in turn yields smaller and higher-quality triangulations for finite element problems (Figure 29).

Other differences between FlipOut and Lagrangian methods depend on the application. For instance, when straightening segmentation boundaries or cut curves, Xin and Wang [2007] yields identical results for examples in Figures 19, 21 and 22, but in Figure 6, *center* yields overlapping curves that fail to define a valid segmentation. Here one would have to implement a collision detection scheme to preserve the isotopy class; FlipOut preserves

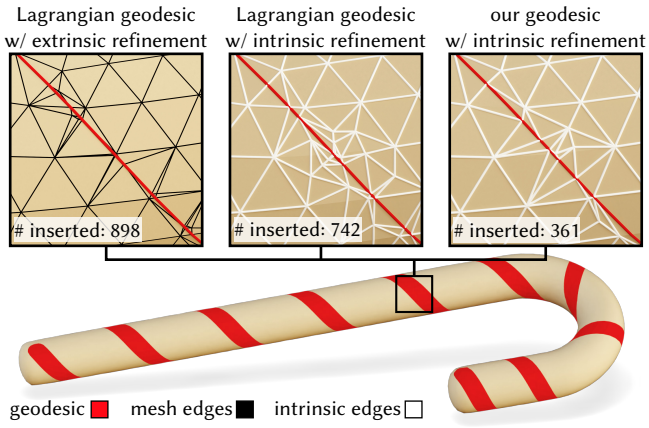


Fig. 29. To get a high-quality simulation mesh, one could straighten curves using a Lagrangian method, "slice" through triangles, then perform either extrinsic *(left)* or intrinsic *(center)* Delaunay refinement. However, FlipOut produces meshes with fewer vertices and higher-quality triangles *(right)* since it need not insert vertices where geodesics cross extrinsic edges.

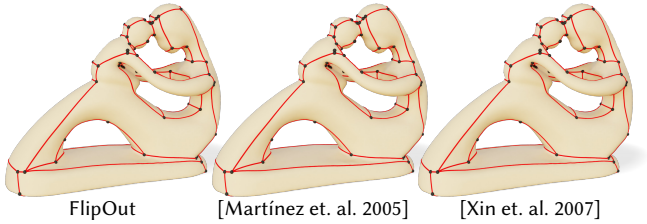


Fig. 30. For non-crossing curves, the geodesics from FlipOut are generally indistinguishable from past Lagrangian approaches. However, FlipOut uses a very different representation, enabling it to guarantee non-crossing output and further generate a complete triangulation of the domain.

this class by construction. As noted in Morera et al. [2008, Section 4.2], any exact method yields identical geodesic Bézier curves (*à la* Figure 23), though FlipOut provides a smaller mesh that omits extrinsic edge crossings. Notably, Lagrangian methods such as Xin and Wang [2007] cannot be used to construct geodesic CDTs (*à la* Figure 24), since they must insert new vertices. Like a standard 2D CDT, FlipOut triangulates the given points and segments without inserting any new vertices (Section 6.3). The PDE examples in Section 6.4 yield nearly identical results when starting with Lagrangian geodesics, though again yield larger meshes with poorer-quality elements (Figure 29) that reduce accuracy and increase solve time.

Finally, solving the single-source problem via a Lagrangian method is far more expensive than our simultaneous Dijkstra approach (Section 6.5), since each path must be straightened independently. Moreover, though neither strategy guarantees minimal geodesics, our approach tends to find shorter paths. *E.g.*, for the gear in Figure 27 straightening each curve via Xin and Wang [2007] takes 6240 ms, and yields minimal geodesics to only 84.9% vertices, versus 348 ms and 99.6% minimal geodesics for FlipOut. Our flip-based approach also preserves an important invariant of the minimal solution, namely that paths cannot cross transversely (since they are supported on the edges of a common triangulation).

7 LIMITATIONS AND FUTURE WORK

We have not established worst-case bounds on the time complexity of our method. Flip-based algorithms on surfaces have resisted worst-case analysis in part because (unlike in 2D) there is not a finite number of intrinsic triangulations on a fixed vertex set [Bobenko and Springborn 2007]. Yet just as intrinsic Delaunay flipping exhibits good empirical scaling behavior [Sharp et al. 2019b, Fig. 10], FlipOut exhibits good scaling in practice across a variety of models (Section 5.2). Also, our proof of termination does not apply to closed loops—though we always observe termination in practice.

Since we encode curves as edge sequences, our approach is fundamentally limited to non-crossing curves. But for the same reason, it preserves a curve's isotopy class—complementing existing Lagrangian methods that do not, for instance, respect region boundaries. In the context of geometry processing, preserving such arrangements is a key strength of our approach (Section 6.1).

Like all other exact methods, trouble can still arise due to floating-point arithmetic—though in practice we succeed on all but a handful of pathological examples (Section 5.3). Mollification *à la* Sharp and Crane [2020, Section 4.5] may eliminate even these cases.

Overall, formulating geodesic problems in the setting of intrinsic triangulations opens up rich new possibilities. For instance, anisotropic metrics, which we did not explore here, might be addressed by simply adjusting the input edge lengths (in the spirit of Campen et al. [2013]). More broadly, our investigation makes it clear that intrinsic triangulations have utility far beyond intrinsic *Delaunay* triangulations, which have been the main focus of prior work. We are optimistic that continued investigation of the intrinsic picture will lead to creative new approaches to geometry processing.

ACKNOWLEDGMENTS

This work is supported by a Packard Fellowship, NSF CAREER Award 1943123, an NSF Graduate Research Fellowsip, and gifts from Autodesk, Activision Blizzard, Adobe, Disney, and Facebook.

REFERENCES

Y. Adikusuma, Z. Fang, and Y. He. 2020. Fast Construction of Discrete Geodesic Graphs. *ACM Trans. Graph.* 39, 2 (2020).

P. An. 2019. Finding Shortest Paths in a Sequence of Triangles in 3D by the Planar Unfolding. *Numerical Functional Analysis and Optimization* 40, 8 (2019), 944–952.

E. Appleboim, E. Saucan, and J. Stern. 2009. *Normal Approximations of Geodesics on Smooth Triangulated Surfaces.* Technical Report. CCIT Report.

J. Athreya, D. Aulicino, and W. Hooper. 2020. Platonic Solids and High Genus Covers of Lattice Surfaces. *Experimental Mathematics* (2020).

M. Bell. 2016. Simplifying Triangulations. *arXiv:1604.04314* (2016).

M. Bern, D. Eppstein, and J. Erickson. 2002. Flipping Cubical Meshes. *Engineering with Computers* 18, 3 (2002), 173–187.

A. Bobenko and B. Springborn. 2007. A Discrete Laplace–Beltrami Operator for Simplicial Surfaces. *Discrete & Computational Geometry* 38, 4 (2007).

D. Bommes and L. Kobbelt. 2007. Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes. In *VMV*, Vol. 7. 151–160.

P. Bose, A. Maheshwari, C. Shu, and S. Wuhrer. 2011. A Survey of Geodesic Paths on 3D Surfaces. *Comput. Geom. Theory Appl.* 44, 9 (Nov. 2011), 13.

S. Callens and A. Zadpoor. 2018. From Flat Sheets to Curved Geometries: Origami and Kirigami Approaches. *Materials Today* 21, 3 (2018), 241 – 264.

M. Campen, M. Heistermann, and L. Kobbelt. 2013. Practical Anisotropic Geodesy. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 63–71.

L. Cao, J. Zhao, J. Xu, S. Chen, G. Liu, S. Xin, Y. Zhou, and Y. He. 2020. Computing Smooth Quasi-geodesic Distance Field (QGDF) with Quadratic Programming. *Computer-Aided Design* (2020).

X. Chen, A. Golovinskiy, and T. Funkhouser. 2009. A Benchmark for 3D Mesh Segmentation. *ACM Trans. Graph. (Proc. SIGGRAPH)* 28, 3 (Aug. 2009).

L. Chew. 1989. Constrained Delaunay Triangulations. *Algorithmica* 4, 1-4 (1989).

L. Chew. 1993. Guaranteed-quality Mesh Generation for Curved Surfaces. In *Proceedings of the Ninth Annual Symposium on Computational Geometry (SCG '93)*.

K. Crane, M. Livesu, E. Puppo, and Y. Qin. 2020. A Survey of Algorithms for Geodesic Paths and Distances. *arXiv:2007.10430* (2020).

K. Crane and M. Wardetzky. 2017. A Glimpse Into Discrete Differential Geometry. *Notices of the American Mathematical Society* 64, 10 (November 2017), 1153–1159.

K. Crane, C. Weischedel, and M. Wardetzky. 2013. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. *ACM Trans. Graph.* 32, 5 (Oct. 2013).

M. Crofton. 1868. On the Theory of Local Probability, Applied to Straight Lines Drawn at Random in a Plane; the Methods Used Being Also Extended to the Proof of Certain New Theorems in the Integral Calculus. *Philosophical Transactions of the Royal Society of London* 158 (1868), 181–199.

R. Dyer, H. Zhang, and T. Möller. 2007. Delaunay Mesh Construction. *Proceedings of the 5th Eurographics Symposium on Geometry Processing*.

J. Erickson. 2014. Efficiently Hex-meshing Things with Topology. *Discrete & Computational Geometry* 52, 3 (2014), 427–449.

M. Fisher, B. Springborn, P. Schröder, and A. Bobenko. 2007. An Algorithm for the Construction of Intrinsic Delaunay Triangulations with Applications to Digital Geometry Processing. *Computing* 81, 2 (Nov 2007).

M. Gage. 1990. Curve Shortening on Surfaces. *Annales scientifiques de l'École Normale Supérieure* Ser. 4, 23, 2 (1990), 229–256.

X. Han, H. Yu, Y. Yu, and J. Zhang. 2017. A Fast Propagation Scheme for Approximate Geodesic Paths. *Graphical Models* 91 (2017), 22 – 29.

J. Hass and P. Scott. 1994. Shortening Curves on Surfaces. *Topology* 33, 1 (1994).

A. Hatcher. 2002. *Algebraic Topology.* Cambridge University Press.

P. Herholz and M. Alexa. 2019. Efficient Computation of Smoothed Exponential Maps. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 79–90.

C. Indermitte, T. Liebling, M. Troyanov, and H. Clémençon. 2001. Voronoi Diagrams on Piecewise Flat Surfaces and an Application to Biological Growth. *Theoretical Computer Science* 263 (2001).

S. Kapoor. 1999. Efficient Computation of Geodesic Shortest Paths. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*.

L. Kharevych, B. Springborn, and P. Schröder. 2006. Discrete Conformal Mappings via Circle Patterns. *ACM Trans. Graph.* 25, 2 (2006).

S. Kiazyk, S. Loriot, and E. Colin de Verdière. 2015. CGAL 5.0.2—Triangulated Surface Mesh Shortest Paths. http://www.cgal.org.

R. Kimmel and J. Sethian. 1998. Fast Marching Methods on Triangulated Domains. *Proc. Nat. Acad. Sci.* 95 (1998).

D. Kirsanov. 2008. Implementation of Exact Geodesics on Triangular Meshes. code.google.com/archive/p/geodesic/.

F. Knöppel, K. Crane, U. Pinkall, and P. Schröder. 2013. Globally Optimal Direction Fields. *ACM Trans. Graph.* 32, 4 (2013).

C. Lawson. 1977. Software for C1 Surface Interpolation. In *Mathematical software*. Elsevier, 161–194.

B. Liu, S. Chen, S. Xin, Y. He, Z. Liu, and J. Zhao. 2017a. An Optimization-driven Approach for Computing Geodesic Paths on Triangle Meshes. *Computer-Aided Design* 90 (2017).

Y. Liu, D. Fan, C. Xu, and Y. He. 2017b. Constructing Intrinsic Delaunay Triangulations from the Dual of Geodesic Voronoi Diagrams. *ACM Trans. Graph.* 36, 2 (2017).

Y. Liu, C. Xu, D. Fan, and Y. He. 2015. Efficient Construction and Simplification of Delaunay Meshes. *ACM Trans. Graph.* 34, 6 (2015).

V. Lucquin, S. Deguy, and T. Boubekeur. 2017. SeamCut: Interactive Mesh Segmentation for Parameterization. In *ACM SIGGRAPH 2017 Technical Briefs*.

D. Martínez, L. Velho, and P. Carvalho. 2005. Computing Geodesics on Triangular Meshes. *Computers & Graphics* 29, 5 (2005).

J. Mitchell, D. Mount, and C. Papadimitriou. 1987. The Discrete Geodesic Problem. *SIAM J. Comput.* 16, 4 (Aug. 1987), 22.

D. Morera, P. Carvalho, and L. Velho. 2008. Modeling on Triangulations with Geodesic Curves. *The Visual Computer* 24, 12 (Dec 2008).

A. Myles, N. Pietroni, and D. Zorin. 2014. Robust Field-aligned Global Parametrization. *ACM Trans. Graph.* 33, 4 (2014).

G. Patané. 2016. STAR—Laplacian Spectral Kernels and Distances for Geometry Processing and Shape Analysis. In *Computer Graphics Forum*.

G. Peyré and L. Cohen. 2005. Heuristically Driven Front Propagation for Geodesic Paths Extraction. In *International Workshop on Variational, Geometric, and Level Set Methods in Computer Vision*. Springer, 173–185.

K. Polthier and M. Schmies. 2006. Straightest Geodesics on Polyhedral Surfaces. In *ACM SIGGRAPH 2006 Courses*.

Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang. 2016. Fast and Exact Discrete Geodesic Computation based on Triangle-oriented Wavefront Propagation. *ACM Trans. Graph.* 35, 4 (2016).

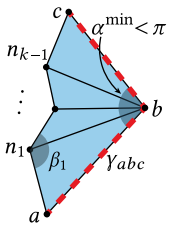
M. Remešíková, M. Šagát, and P. Novysedlák. 2019. Discrete Lagrangian Algorithm for Finding Geodesics on Triangular Meshes. *Applied Mathematical Modelling* (2019).

I. Rivin. 1994. Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume. *Annals of mathematics* 139, 3 (1994).

R. Schmidt, C. Grimm, and B. Wyvill. 2006. Interactive Decal Compositing with Discrete Exponential Maps. In *ACM SIGGRAPH 2006 Papers*.
J. Sethian. 1989. A Review of Recent Numerical Algorithms for Hypersurfaces Moving with Curvature Dependent Speed. *J. Differential Geometry* 31 (1989), 131–161.
N. Sharp and K. Crane. 2018. Variational Surface Cutting. *ACM Trans. Graph.* 37, 4 (2018).
N. Sharp and K. Crane. 2020. A Laplacian for Nonmanifold Triangle Meshes. *Computer Graphics Forum (SGP)* 39, 5 (2020).
N. Sharp, K. Crane, et al. 2019a. geometry-central. www.geometry-central.net.
N. Sharp, Y. Soliman, and K. Crane. 2019b. Navigating Intrinsic Triangulations. *ACM Trans. Graph.* 38, 4 (2019).
N. Sharp, Y. Soliman, and K. Crane. 2019c. The Vector Heat Method. *ACM Trans. Graph.* 38, 3 (2019).
J. Shewchuk. 1997. *Delaunay Refinement Mesh Generation.* Ph.D. Dissertation. Carnegie Mellon University. Tech Report CMU-CS-97-137.
P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. 2004. The Princeton Shape Benchmark. In *Proceedings Shape Modeling Applications, 2004.* IEEE.
B. Springborn. 2019. Ideal Hyperbolic Polyhedra and Discrete Uniformization. *Discrete & Computational Geometry* (Sep 2019).
B. Springborn, P. Schröder, and U. Pinkall. 2008. Conformal Equivalence of Triangle Meshes. *ACM Trans. Graph.* 27, 3 (2008).
V. Surazhsky, T. Surazhsky, D. Kirsanov, S. Gortler, and H. Hoppe. 2005. Fast Exact and Approximate Geodesics on Meshes. *ACM Trans. Graph.* 24, 3 (2005).
X. Wang, Z. Fang, J. Wu, S. Xin, and Y. He. 2017. Discrete Geodesic Graph for Computing Geodesic Distances on Polyhedral Surfaces. *Computer Aided Geometric Design* 52 (2017).
J. Weeks. 1993. Convex Hulls and Isometries of Cusped Hyperbolic 3-manifolds. *Topology and its Applications* 52, 2 (1993).
C. Wu and X. Tai. 2010. A Level Set Formulation of Geodesic Curvature Flow on Simplicial Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 16, 4 (July 2010).
S. Xin, Y. He, and C. Fu. 2011. Efficiently Computing Exact Geodesic Loops within Finite Steps. *IEEE Transactions on Visualization and Computer Graphics* 18, 6 (2011).
S. Xin and G. Wang. 2007. Efficiently Determining a Locally Exact Shortest Path on Polyhedral Surfaces. *Computer-Aided Design* 39, 12 (2007).
S. Xin and G. Wang. 2009. Improving Chen and Han's Algorithm on the Discrete Geodesic Problem. *ACM Trans. Graph.* 28, 4 (2009).
C. Xu, T. Wang, Y. Liu, L. Liu, and Y. He. 2015. Fast Wavefront Propagation for Computing Exact Geodesic Distances on Meshes. *IEEE transactions on visualization and computer graphics* 21, 7 (2015).
X. Ying, C. Huang, X. Fu, Y. He, R. Yu, J. Wang, and M. Yu. 2019. Parallelizing Discrete Geodesic Algorithms with Perfect Efficiency. *Computer-Aided Design* 115 (2019).
X. Ying, X. Wang, and Y. He. 2013. Saddle Vertex Graph: a Novel Solution to the Discrete Geodesic Problem. *ACM Trans. Graph.* 32, 6 (2013).
X. Ying, S. Xin, and Y. He. 2014. Parallel Chen-Han (PCH) Algorithm for Discrete Geodesics. *ACM Trans. Graph.* 33, 1 (2014).
J. Zhang, C. Wu, J. Cai, J. Zheng, and X. Tai. 2010. Mesh snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow. In *Computer graphics forum*, Vol. 29. Wiley Online Library.
Q. Zhou and A. Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv:1605.04797* (2016).

A LOCAL SHORTENING IN A Δ-COMPLEX

In this appendix we prove the validity and termination of our local shortening routine in the general case of a Δ-complex. The difficulty arises from the counter-intuitive existence of *twice-incident edges* (defined below). We first provide some general comments, then prove that the procedure works as stated in Algorithm 1, even in the general setting. Throughout we use the notation of the unfolded wedge shown inset.



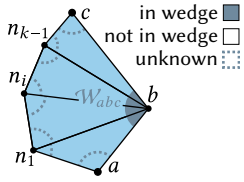
Flipping the first edge. Algorithm 1 flips the edge bn_j, where j is the minimum index i such that $\beta_i < \pi$. Choosing this edge to flip, as opposed to *any* such edge, may seem incidental, but is in fact necessary for the correctness of the algorithm, and is used twice in our argument. First, there may be an edge incident on a degree-1 vertex n_i where $\beta_i < \pi$; these edges cannot be flipped topologically. However, letting j be the minimum i ensures that we will select some other edge to be flipped, as will be shown in Theorem A.5. Second, one could get stuck in an infinite cycle flipping an edge back and forth within the wedge; selecting the minimum i avoids this possibility, as shown in Theorem A.8.

Alternatives. Many other possible algorithms leverage the same basic idea as Algorithm 1. For example, we considered a variant which first flips all twice-incident edges, then flips the remaining edges in any order—or even a variant which randomly flips any flippable edge. One can argue that these strategies achieve the same effect. We chose the variant given in Algorithm 1 because its implementation is straightforward, and does not involve any special consideration of Δ-complexes (although the proof does).

A.1 Preliminaries

All statements in the remainder of this Appendix assume the preconditions of Algorithm 1. Recall that ${}^{xz}_{y}$ refers to the corner at vertex y, formed by the counter-clockwise triple xyz, and that an edge is *incident* on a wedge if it is contained in its interior.

Twice-incident edges. The complexity in the case of a Δ-complex arises because the elements of the unfolded wedge diagram are not necessarily distinct. In particular, there may exist *twice-incident* edges, where for some incident edge e, *both* endpoints of e meet b within the wedge. Such an edge appears twice in the unfolded diagram, and some n_i will be equal to b. Twice-incident edges are necessarily loop-edges, so this situation does not arise in a simplicial complex—we must treat this general case nonetheless, because edge flips may create a Δ-complex at intermediate steps of our algorithm.



Our proof carefully considers which triangle corners in the unfolding belong to the wedge $\mathcal{W}_{abc}$ formed by a joint γ_{abc} as flipping proceeds. This wedge is comprised of an edge connected set of *inner corners* ${}^{n_i n_{i+1}}_{b}$ incident on b (for $0 \le i < k$). The inner corners are all distinct, since they arise from splitting the neighborhood of vertex b along γ_{abc}. However, not all elements of the unfolded diagram are necessarily distinct: triangles may appear twice in the unfolding, and thus some *outer corners* ${}^{n_{i+1} b}_{n_i}$ and ${}^{b n_{i-1}}_{n_i}$ may coincide with inner corners, *i.e.*, they also belong to the wedge $\mathcal{W}_{abc}$. Our arguments will follow from reasoning about which of the outer corners are necessarily not included in the wedge.

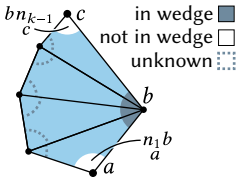
Lemma A.1. *No triangle has all three corners in $\mathcal{W}_{abc}$.*

Proof. The interior angles of a triangle sum to π, but Algorithm 1 considers only wedges of total angle $\alpha < \pi$. □

Lemma A.2. *The bounding edges ab and bc are not incident on the wedge,* i.e. *they are distinct from every edge bn_i for $0 < i < k$.*

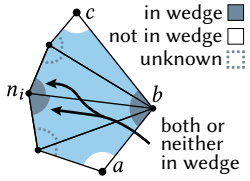
Proof. If these edges were incident, then the joint would not be flexible, because a segment of the path would be in the interior of the wedge. However, Algorithm 1 assumes the joint γ_{abc} is flexible as a precondition. □

Lemma A.3. *The first outer corner ${}^{n_1}_{a}b$ and last outer corner ${}^{bn_{k-1}}_{c}$ are not in the wedge $\mathcal{W}_{abc}$.*



Proof. Observe that ${}^{n_1}_{a}b$ is bordered on the right by edge ab, thus for ${}^{n_1}_{a}b$ to be in the wedge, ab would need to appear as some bn_i for $i \geq 1$. However, by Lemma A.2 edge ab is not an interior edge, so the first corner can only be in the wedge if $ab = bc$. But this is not possible, since then the joint γ_{abc} would not be flexible: both segments of the joint (which must be distinct, by precondition) would run along the same edge, blocking an isotopy to the outer arc. A nearly identical argument applies to the last corner ${}^{bn_{k-1}}_{c}$. □

Lemma A.4. *For $0 < i < k$, the outer corner ${}^{bn_{i-1}}_{n_i}$ is in the wedge if and only if the adjacent outer corner ${}^{n_{i+1}b}_{n_i}$ is also in the wedge.*



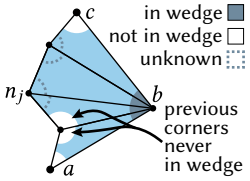
Proof. There are only two places in the wedge where corners that share an edge endpoint have different wedge memberships: at the bounding edges ab and bc. However, Lemma A.2 establishes that none of the edges bn_i are the bounding edges. Thus the corners ${}^{bn_{i-1}}_{n_i}$ and ${}^{n_{i+1}b}_{n_i}$ must either both be in the wedge, or both not be in the wedge. □

Note that when the two outer corners in Lemma A.4 are in the wedge, then the edge bn_i is a twice-incident edge.

A.2 Validity

To show that Algorithm 1 is valid, we must argue that every edge flip is valid and can actually be performed. As noted in Section 3.1.3, an edge is flippable as long as it satisfies a geometric condition (it is contained in a convex quadrilateral), and a topological one (both endpoints have degree > 1). Theorem A.5 will show that the algorithm preserves a key invariant: none of the outer corners in the already-processed region are contained in the wedge. Theorem A.6 will then use this property to show that all flips are valid.

Theorem A.5. *When flipping edge bn_j in Algorithm 1, none of the previous outer corners ${}^{n_{i+1}b}_{n_i}$ and ${}^{bn_{i-1}}_{n_i}$ for $i < j$ are in the wedge.*

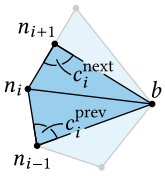


Proof. The first corner ${}^{n_1}_{a}b$ is not in the wedge by Lemma A.3. All other previous outer corners come in pairs, incident on some edge bn_i with $0 < i < j$. From Lemma A.4, we know that if one of the corners in the pair were in wedge, then both would be, and bn_i would be a twice-incident edge. Twice-incident edges always have $\beta < \pi$ since both endpoints are contained in the wedge with $\alpha < \pi$. Thus we would have $\beta_i < \pi$, for some $i < j$, yet this is impossible, since we chose j as $j = \min_i \beta_i < \pi$. Therefore none of the previous outer corners are in the wedge. □

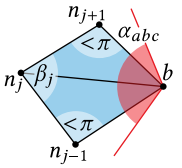
We now prove the key result needed to ensure validity of our algorithm, namely that we only perform valid edge flips. Henceforth we will use

$$c_i^{\text{prev}} := {}^{n_i b}_{n_{i-1}} \qquad \text{and} \qquad c_i^{\text{next}} := {}^{bn_i}_{n_{i+1}}$$

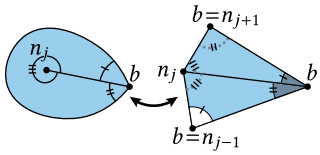
to refer to the outside corners before and after the interior edge bn_i, *resp.*



Theorem A.6. *In Algorithm 1, the edge bn_j can always be flipped.*



Proof. To see that the edge flip is geometrically valid (*i.e.*, the quadrilateral is convex), the argument is the same as in the simplicial case. The two angles at c_j^{prev} and c_j^{next} are less than π, since they are corners of Euclidean triangles. The angle at the outer corner is $\beta_j < \pi$, since the algorithm only flips the edge when $\beta_j < \pi$. Finally, the angle at the inner corner ${}^{n_{j-1}}_{b}n_{j+1} < \pi$ because it is contained in the wedge angle which is $\alpha < \pi$ by the precondition of the algorithm.



We must also argue that bn_j can be flipped topologically. Vertex b always has degree > 1 because bn_j is distinct from the bounding edges by Lemma A.2. If $n_j = b$, this same reasoning applies to the other endpoint (*e.g.* when bn_j is a twice-incident edge). If $n_j \neq b$, then n_j cannot be degree 1—if it were, then the incident corner ${}^{n_j n_{j+1}}_{b}$ is the same as the previous outer corner c_j^{prev} (identified with single tick marks in the inset figure). However, by Theorem A.5 the previous outer corner is never incident, so this situation cannot occur. □

A.3 Termination

We next argue that Algorithm 1 terminates, *i.e.* we do not flip edges indefinitely. In the simplicial case it is easy to see that the algorithm terminates, since each edge flip removes an edge from the wedge and decreases the wedge degree k. In a general Δ-complex, the argument is somewhat more complicated, because flipping an edge adjacent to a twice-incident edge does not remove it from the wedge, but merely moves the flipped edge to a different location in the wedge and leaves k unchanged.

We will argue that on each iteration *either* the wedge degree shrinks, *or* the first twice-incident edge gets closer to the beginning of the wedge. More precisely, let ρ be the smallest index i such that edge bn_i is a twice-incident edge (ρ need not be defined if there are no such edges). In Theorem A.7 we will argue that k never increases, and that on every iteration either (a) k decreases, or (b) ρ decreases. These decreases will then imply termination of the algorithm (Theorem A.8).

Theorem A.7. *On each iteration, either (a) k decreases or (b) ρ decreases and k does not change.*

Proof. Each iteration flips some edge bn_j. In the common case where c_j^{next} is *not* in the wedge then k will decrease, and otherwise if it *is* in the wedge then ρ will decrease. Recall that in both cases, we know by Theorem A.5 that c_j^{prev} is not in the wedge.

Suppose c_j^{next} is *not* in the wedge. Then, neither endpoint of the newly created edge will be incident on the wedge, and thus the wedge degree k decreases by at least 1.

Now, suppose c_j^{next} *is* in the wedge. The edge bn_j cannot be a twice-incident edge, because if it were then all three corners of the triangle $bn_{j+1}n_j$ would be incident on the wedge, violating Lemma A.1. Moreover, c_j^{prev} is not in the wedge, so the flip trades this once-incident edge for a once-incident edge at corner c_j^{next}, and k does not change. However, in this case the edge bn_{j+1} is necessarily a twice-incident edge (Lemma A.4) and $\rho = j + 1$. Furthermore, none of the outer corners prior to j are incident (Theorem A.5), so the newly created edge will not occupy any of the corners ${}^{n_l}_{\bar{b}^1}n_l$ with $l \leq j$. Therefore the newly created edge will appear at some index $j' > j + 1$, and will come *after* the twice-incident edge bn_{j+1}. Thus ρ, the index of the first twice-incident edge, decreases. □

Of course, whenever we flip a twice-incident edge, ρ may increase as we uncover a new nearest twice-incident edge—this does not affect the next argument. Similarly, when we remove the last twice-incident edge ρ is undefined, but this does not matter since all subsequent flips will decrease k. The combined decrease of k and ρ implies the algorithm must terminate.

Theorem A.8. *Algorithm 1 terminates.*

Proof. Let k_0 denote the initial wedge degree, and note that on every iteration we have $k \leq k_0$ and $\rho < k_0$, and both are bounded below by 0. By Theorem A.7, k will never increase, and furthermore k must decrease at most every k_0 iterations because ρ cannot decrease more than k_0 times in row. Therefore k must decrease at most every k_0 iterations, and is bounded from below, so the algorithm terminates. □

B PERTURBING SINGLE-SEGMENT LOOPS

In Section 4.3, we consider applying our Algorithm 2 to closed loops, rather than just open paths. This generalization comes almost for free, except one edge case: a loop composed of a single segment. This case violates the preconditions of our Algorithm 1. We suggest an additional update rule which covers the case of a single-segment loop; this appendix gives additional explanation and justification for this rule.

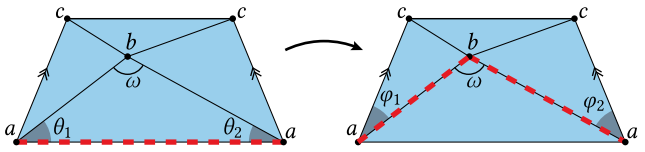


Fig. 31. We resolve a single-segment loop by perturbing the path to run along the opposite edges of its triangle, enabling the algorithm to keep making progress. In general there may be zero or more than one triangle in the regions swept out by the angles φ_1 and φ_2 (here exactly one triangle is drawn for both)—nonetheless, the argument made by measuring the labeled angles still applies.

Whenever a loop consists of a single segment and has $\alpha < \pi$ at a flexible joint, one can identify the triangle aab which borders the segment and is on the side of the joint to be shortened (Figure 31). We replace the loop segment aa with the new segments ab and ba, running along the other two edges of this triangle. This update is necessarily an isotopy, since it is a motion within a single triangle face. After this update, Algorithm 2 proceeds as usual.

At first, this update may seem unreasonable in that it would be immediately reverted as Algorithm 2 proceeds. However, we can argue that (at least locally) the algorithm always continues making progress after the update. Recall that

$$\varphi_1 + \theta_1 + \theta_2 + \varphi_2 = \alpha < \pi, \tag{4}$$

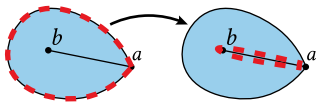
because the path is not yet locally shortest, and

$$\theta_1 + \theta_2 + \omega = \pi, \tag{5}$$

because they are the corners of a triangle. These can be rearranged to see

$$\varphi_1 + \varphi_2 < \omega. \tag{6}$$

Thus when Algorithm 2 proceeds by straightening the next smallest angle, it will not undo the move by shortening the joint at ω, it will instead straighten at the other joint spanned by φ_1 and φ_2, constructing some new shorter curve with b as a node. In the case of a single-segment loop which encircles a degree-1 vertex, this update will adjust the path to run out and back along the vertex's one edge, just before the loop collapses to a point (see inset diagram).



As noted in Section 4.3, our proof of Theorem 4.2 that Algorithm 2 terminates no longer applies, because this update makes the path longer. However, we conjecture that the algorithm nonetheless always terminates, and have always observed it to do so in our experiments. One might be able to generalize the termination proof by showing that subsequent iterations of Algorithm 2 necessarily find a loop which is shorter than the initial single-segment loop.