Intrinsic Triangulations in Geometry Processing

Nicholas Sharp

July 2021

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Keenan Crane, Chair (CMU)
Ioannis Gkioulekas (CMU)
Anupam Gupta (CMU)
Maks Ovsjanikov (École Polytechnique)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: geometry processing, discrete differential geometry, numerical computing

Dedication will go here.

Abstract

This thesis treats the theory and practice of *intrinsic triangulations* and their use in 3D mesh processing algorithms. As geometric data becomes more ubiquitous in applications ranging from scientific computing to augmented reality to machine learning, there is a pressing need to develop algorithms that work reliably on low-quality data. Intrinsic triangulations provide a powerful framework for these problems, by de-coupling the mesh used to encode geometry from the one used for computation. The basic shift in perspective is to encode the geometry of a mesh not in terms of ordinary vertex positions, but instead only in terms of edge lengths.

We begin by outlining the long history of intrinsic triangulations in mathematics, as well as the basic algorithmic formulation and how it relates to problems in geometry processing. Then, we present recent data structures for richly encoding intrinsic triangulations (signposts, integer coordinates), which support additional functionality necessary for applications while remaining efficient and robust. Using these data structures, we describe a wide variety of mesh processing operations on intrinsic triangulations, which enable powerful retriangulation schemes with strong guarantees. We will also demonstrate how intrinsic triangulations can be used for tasks beyond retriangulation; including a new flip-based algorithm for computing geodesic paths on surfaces. Finally, we will present a generalization of intrinsic triangulations to handle less-structured data such as nonmanifold meshes and point clouds. Throughout, we will see applications of intrinsic triangulation to problems in geometry processing, where they lend much-needed automatic robustness to tasks including parameterization, vector field processing, spectral methods, and more.

Acknowledgments

Acknowledgements will go here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Triangular surface meshes are the cornerstone data structure in 3D geometry processing, typically represented by a collection of faces with a position for each vertex. What if instead of positions, geometry is encoded by a length associated with each edge? This small change yields an *intrinsic triangulation*, which has just the right structure to enable powerful new algorithms for a variety of tasks. This course will introduce the theory and practice of intrinsic triangulations, from their basic representation, to new algorithmic procedures, to applications in geometry processing.

Intrinsic triangulations bring together a multitude of concepts from topology and differential geometry in a discrete, computational setting. The name "intrinsic" itself arises from a core notion in differential geometry: many properties on a surface do not really depend on its embedding in space, but merely on the measurement of lengths and angles along the surface; such properties are termed *intrinsic*. The main utility of intrinsic triangulations comes from the ability to construct and manipulate triangulations which do not have any associated embedding in 3D space, yet still support a variety of useful computations Figure 1.2. We will see how working in this more general space of triangulations enables simple algorithms for hard problems in surface geometry.

One important practical reason to study intrinsic triangulations is their benefits for *robust geometry processing*. Geometric data plays an increasingly important role in tasks ranging from medical diagnosis to autonomous driving. Researchers have put enormous effort into developing sophisticated geometric algorithms, yet these algorithms are often not used in practice since preconditions on the input (such as manifoldness or the Delaunay condition) do not match up with the reality of actual data (*e.g.*, coarse, poorly-triangulated meshes generated for 3D printing or real-time visualization). Intrinsic triangulations offer a much-needed remedy, with powerful retriangulation schemes to improve the quality of a mesh. This shift in perspective provides a valuable bridge between existing algorithms and challenging geometric data, enabling for instance

- algorithms that were not originally designed to be numerically robust to be successfully run on extremely low-quality meshes,
- algorithms that were originally formulated only for the flat Euclidean plane to be applied to curved, irregularly-tessellated surfaces, and

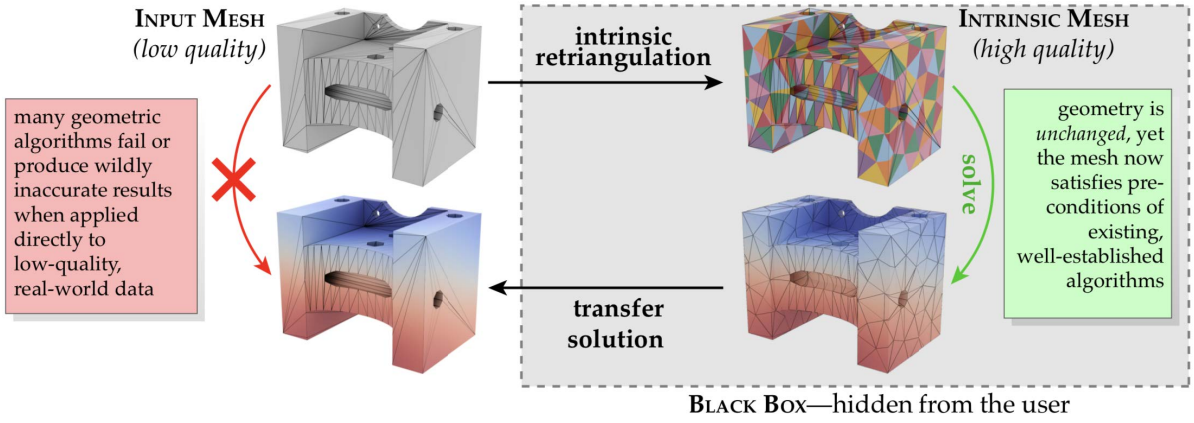


Figure 1.1: There is a major gap between the kind of data expected by geometric algorithms, and the quality of data encountered in real applications. The high-level goal of this course is to build a bridge that allow existing algorithms to be applied in much more challenging scenarios. Here, a high-quality *intrinsic triangulation* is overlaid on top of a low-quality input mesh, enabling an existing finite element solver to compute a more accurate solution. Since the underlying geometry is completely unchanged, this solution can easily be transferred back to the original mesh for further processing—without the end user ever having to know that a transformation was applied "under the hood."

- algorithms designed for homogeneous, isotropic problems to be applied directly to more general inhomogeneous, anisotropic settings.

The intrinsic approach also side-steps some fundamental, traditionally unavoidable challenges in geometric computing——such as the need to juggle the quality of geometric approximation with the quality of individual mesh elements.

Despite the great potential of intrinsic triangulations, they have not yet seen significant uptake amongst practitioners, remaining largely confined to the mathematical geometry community. There are several reasons for this, from the abstractness of the formulation to the apparent difficulty of basic computational operations. One key goal of this course is to "fill the gaps" by presenting the missing algorithms and data structures to make these tools useful in practice. Additionally, we describe many basic operations for the first time, and throughout release easily-usable software implementations.

1.1 Why this approach?

There are some good reasons for using intrinsic triangulations in practical algorithms:

- Many important problems are intrinsic. An increasingly large set of algorithms from geometric and scientific computing are expressed in terms of surface differential operators that are inherently intrinsic. A chief example is the discrete Laplace-Beltrami operator [Dzi88; PP93; War17], which (among many other things [SCV14]) provides the starting point for major sub-fields such as spectral geometry processing [LZ09]

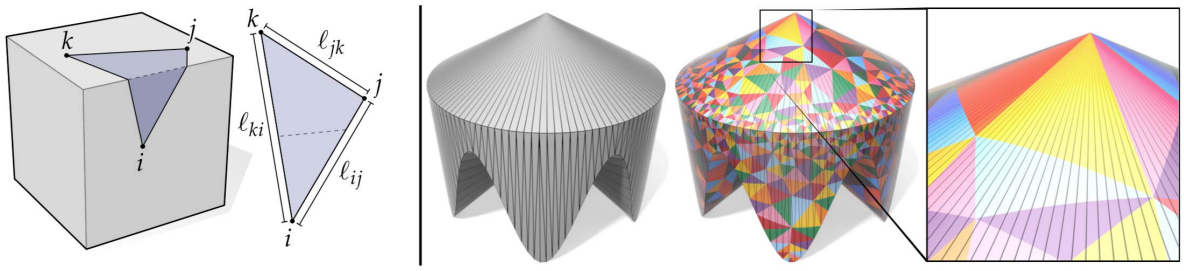


Figure 1.2: Conceptually, intrinsic triangles can "bend" across an underlying polyhedron, yet still flatten out into standard triangles described by three ordinary edge lengths (left). This flexibility enables things that are impossible with standard, extrinsic algorithms—here, a mesh with tiny input angles becomes a geometrically identical Delaunay triangulation with angles no smaller than 30° (right). Since the output is described by conventional data (connectivity + edge lengths) it can still be used directly by many standard simulation and mesh processing algorithms.

and functional maps [Ovs+16]. Beyond the Laplacian, other fundamental geometric quantities (curvatures, geodesic distances, the logarithmic map, and so on) can easily be computed from intrinsic data alone [SSC19b]. For such problems, working in the strictly larger space of intrinsic triangulations necessarily offers, *e.g.*, better accuracy with fewer degrees of freedom (as briefly explored in [SSC19a]).

- Intrinsic descriptions ignore features that don't matter. In shape analysis (e.g., classification or pairwise correspondence) extrinsic descriptions must somehow factor out features like rigid motions or isometric deformations (e.g., bending of an arm)—often at great computational expense [Hua+08; LSP08]. Intrinsic representations are oblivious to such transformations by construction.
- Traditional trade-offs can be avoided. Mesh processing frequently entails a "no free lunch" scenario where one must sacrifice either mesh quality, mesh size, or geometric approximation error. Intrinsic triangulations bypass this classic trade-off by decoupling the triangulation used to encode shape from the one used for computation.
- Volumetric data structures are not required. Techniques for surface meshing [CDS12, Chapter 13] and robust geometry processing [JKS13; Bar+18] often depend on 3D volumetric data structures which require significant storage, suffer from issues not encountered in 2D (e.g., "sliver" tetrahedra [KS16, Section 3]), and/or have trouble handling surfaces with boundary or self-intersection. The intrinsic approach provides some of the very first non-volumetric, surface-only versions of fundamental algorithms like Delaunay refinement (Section 5.3) and adaptive mesh refinement (Section 5.4).
- Some important problems do not even have an extrinsic formulation. Many geometry processing tasks can be framed as "squeezing" a curved manifold into a flat space—*e.g.*, surface parameterization [SSP08], shape recovery [IGG01; BI08], and structured meshing [Pai+15], to name a few. Some of these problems have convex formulations only

because they can pass through the larger space of intrinsic triangulations [Luo04; Spr19]; others simply have no meaningful formulation in the extrinsic context.

More broadly, building up general-purpose tools for working in the intrinsic setting not only improves solutions to existing problems, but enables one to ask entirely new questions. These questions push existing work on discrete differential geometry into the new territory of robust geometric computing, producing immediately valuable results in practice.

Chapter 2

Background

The goal of this course is to see how we can expand the standard view of meshes to enable more flexible algorithms in geometric computing. For this reason, we begin with some careful definitions. Though geometric computing often considers both surface and volume meshes, we will focus primarily on surfaces (see Section **??** for a discussion of possible extensions to the volumetric case).

The most important idea is that we will always work with two triangulations of the same surface:

- The **extrinsic mesh** is what one might ordinarily think of as a "triangle mesh:" a collection of points in $\mathbb{R}^3$, connected up into triangles using straight line segments in $\mathbb{R}^3$.
- The **intrinsic mesh** is most easily thought of as another triangulation that sits "on top of" the extrinsic mesh, whose edges are straight paths *along* the extrinsic mesh, rather than straight line segments in $\mathbb{R}^3$. As time goes on, we'll see that there is a much broader view of intrinsic triangulations, which does not require them to sit on top of an extrinsic surface.

2.1 Surface geometry

A surface mesh describes only the boundary of a solid region—or more generally a thin "shell" which need not be the boundary of any solid. Such meshes arise in a broad range of contexts. For instance, they might arise from scanning a real physical surface, they may be the output of a physical simulation algorithm, or they might be designed by an artist or engineer (see inset).

The framework of intrinsic triangulations makes two important assumptions. First, we imagine that the input geometry is an *exact*

Figure 2.1: An intrinsic triangulation exactly preserves the input geometry, while changing the mesh connectivity. Hence, if the input mesh gives an exact description of the geometry (as with the CAD model at right), it will not be corrupted; if the input exhibits noise or ap-

description of the shape of interest. One of the strengths of the intrinsic approach is that (unlike conventional remeshing) it exactly preserves the given shape, which means that one need not worry about, *e.g.*, corrupting small features, sharp edges, or surface detail while processing geometry. Of course, the intrinsic approach still applies even if the input only approximates the true geometry (as with, say, 3D scans)—we simply use the "exact input" hypothesis to guide decisions about data structures and algorithms. On the flip side, if there are defects in the input (noise, topological errors, *etc.*), these features will also be retained by the intrinsic mesh. In short: intrinsic meshes help to improve the quality of mesh elements, but do nothing to improve the quality of the underlying geometry.

Second, we assume that the geometry is given as a polyhedral surface with flat faces, and moreover, that some initial triangulation has been chosen for non-triangular faces. This assumption goes hand-in-hand with the first assumption: in order to exactly preserve the geometry, we must have a clear definition of what this geometry looks like. Nonplanar polygons (*i.e.*, polygons where all vertices do not sit in a common plane) do not provide a canonical definition—though some opportunities for processing nonplanar meshes are discussed in Section **??**. In contrast, planar polygons provide a well-defined geometry; assuming that such polygons have already been triangulated is merely a simplifying assumption that leads to concise descriptions of data structures and algorithms. Moreover, in several important cases the choice of triangulation will have no effect on the final result—such as when defining the intrinsic Laplacian Section **??**, or computing discrete conformal maps [GSC21b].

The description of a polyhedral surface can be divided into two basic pieces:

- A *topological complex* describes how mesh elements (vertices, edges, and faces) are connected, without any reference to the shape, size, or location of these elements. A good analogy would be an adjacency matrix for a graph, which indicates which nodes are connected by edges (and nothing more). Working with intrinsic triangulations will require us to expand our notion of connectivity beyond the usual "vertex-face" adjacency matrix common to many mesh data structures, as discussed in Section 2.2.
- Associated *geometric data* provides complementary information about shape. In particular, the geometry of an extrinsic mesh is given by ordinary vertex positions, whereas the geometry of an intrinsic mesh is described primarily by edge lengths. Sections **??** and **??** provide further details.

This division between topology and geometry also reflects the standard treatment differential geometry, where a surface is often thought of as an embedding of an abstract topological surface into $\mathbb{R}^n$. (For an introduction to this perspective, see Crane et al. [Cra+13, Chapter 3].)

2.2 Topological Triangulations

A *topological triangulation* T describes how a collection of vertices, edges, and faces should be connected up to form a mesh. Such triangulations describe only the *connectivity* of the mesh, and make no assumptions about geometry. For instance, triangles are not required to be flat, and edges are not required to be straight.

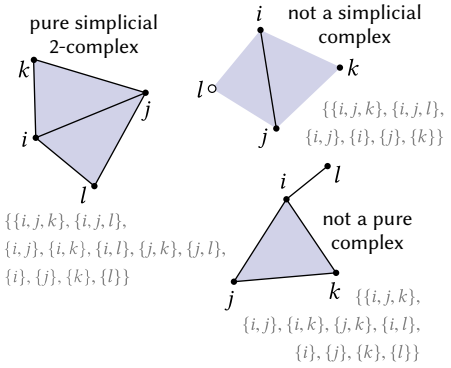
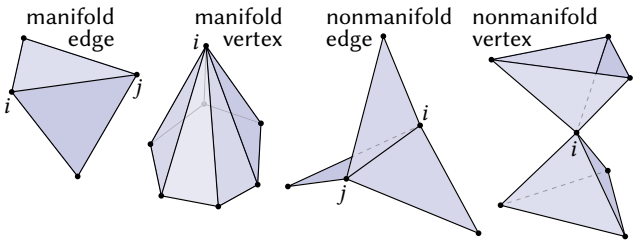


Figure 2.2: In a simplicial complex, an edge must have two distinct endpoints, and a triangle must have three distinct vertices.

Notation. We will refer to the vertices, edges, and faces of any topological triangulation T as V, E, and F, *resp.*, so that $T = (V, E, F)$. We use $\partial E \subset E$ to denote the set of boundary edges, *i.e.*, edges contained in exactly one triangle, and $\partial V \subset V$ to denote boundary vertices, *i.e.*, vertices contained in some boundary edge. Individual vertices will be denoted by indices $i \in V$. Likewise, edges and triangles will be written as pairs $ij \in E$ and triples $ijk \in F$ of vertices. A quantity u at corner i of triangle ijk will be denoted u_i^{jk}. Finally, sums and products appearing on the right-hand side of an expression are implicitly restricted to simplices appearing on the left-hand side—for instance, the expression $u_i = \sum_{ijk} v_{ijk}$ means *"to obtain u at vertex i, sum the quantity v over all triangles ijk containing vertex i."*

We will often (but not always) assume that T is *manifold* and *orientable*, as defined below. These assumptions simplify data structures and algorithms, and are often sufficient for working with real data—especially since one can sometimes build a "bridge" between nonmanifold meshes and algorithms that operate only on manifold data (see Section 7.1).

Manifold Triangulations. A topological triangulation T is *manifold* if we can find a small neighborhood around every point that can be flattened out into the plane. More concretely, an edge $ij \in E \setminus \partial E$ is manifold if it is contained in exactly one or two faces. A vertex $i \in V$ is manifold if (1) all edges incident on i are are manifold and (2) the faces incident on i form a single edge-connected component. A triangulation T is *edge-manifold* if all its edges are manifold.



Orientation. Orientability is a basic property of a surface—intuitively it says whether or not a surface has two distinct "sides.". For instance, a cylinder is orientable, but a Möbius strip is not (Figure 2.3). An edge between two vertices $i, j \in V$ can be given two different orientations: from i to j, and from j to i, which we denote by $\overrightarrow{ij}$ and $\overrightarrow{ji}$, *resp.*. Likewise, a triangle incident on three vertices $i, j, k \in V$ can be given a counter-clockwise orientation, denoted by $\overrightarrow{ijk}$ or any even permutation thereof, or a clockwise orientation, denoted by any odd permutation (*e.g.*,

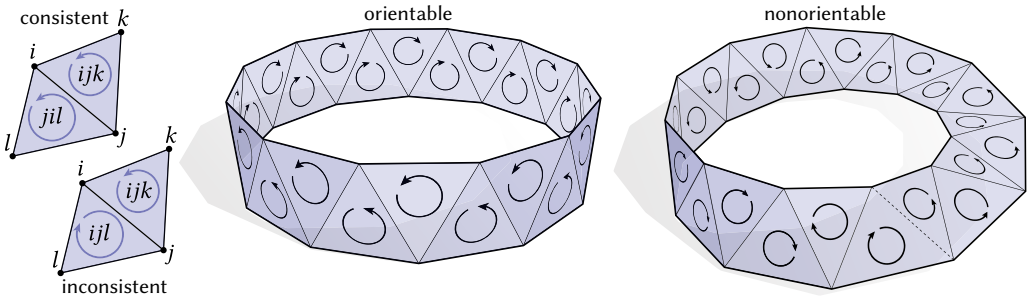


Figure 2.3: *Left:* a pair of triangles is consistently oriented if they *disagree* on the orientation of the shared edge. A triangulation is orientable if all triangles can be assigned consistent orientations *(center)* and nonorientable otherwise *(right)*.

$\overrightarrow{kji}$). Two oriented triangles that share an edge ij are consistently oriented if they *disagree* on the orientation of the shared edge, *e.g.*, $\overrightarrow{ijk}$ and $\overrightarrow{jil}$ are consistently oriented. A topological triangulation T is then *orientable* if all triangles can be given a consistent orientation.

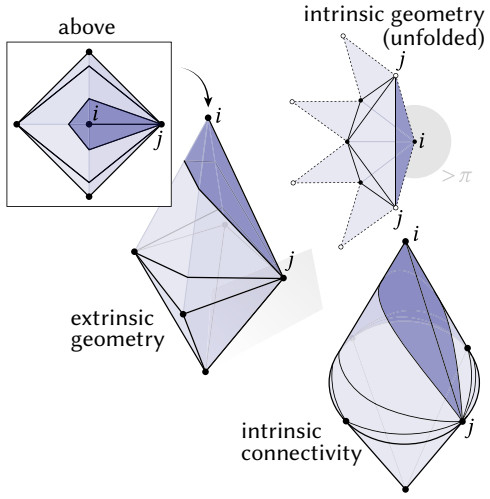


Figure 2.4: Intrinsic triangles can "wrap around" extrinsic polyhedra, allowing them to have unusual connectivity. Here, for instance, the dark blue triangle connects once to vertex i and twice to vertex j—effectively gluing two of its sides to each other along edge ij.

Simplicial complex A common way to describe a topological triangulation is via a *simplicial complex*, which describes all elements as subsets of the vertex set V. More abstractly, a simplicial complex is any collection of sets closed under the operation of taking subsets. The sets of size k are called *k-simplices*, corresponding to vertices ($k = 1$), edges ($k = 2$), triangles ($k = 3$). The subset relationships encode connectivity information—for instance, the edge $\{i, j\}$ is an edge of triangle $\{i, j, k\}$. A basic limitation of simplicial complexes is that they cannot describe elements with repeated vertices, since sets cannot have repeated elements. However, restricting our attention to the simplicial case will sometimes be useful for reasoning about algorithms, since we can make the simplifying assumption that every edge $ij \in E$ has two distinct vertices $i \neq j$, and every triangle $ijk \in F$ has three distinct vertices $i \neq j$, $i \neq k$, $j \neq k$. The simplicial complexes we consider will all be *pure 2-simplicial complexes*, meaning that every vertex $i \in V$ is contained in some triangle $ijk \in F$, and likewise, every edge $ij \in E$ is contained in some triangle $ijk \in F$.

Δ-complex When working with intrinsic triangulations we will inevitably need a more general Δ-complex[1]. The basic reason is that intrinsic triangles can wrap around the extrinsic surface in "unusual" ways. For instance, if the total angle around a vertex i of the extrinsic surface is less than π, then its neighborhood can be covered by a single intrinsic triangle glued to itself along an edge (as shown in Figure 2.4). In general, a Δ-complex can be viewed as a collection of disjoint triangles, along with information that describes how to glue the vertices and edges together. In particular, suppose we index the vertices of the disjoint triangles as $i_0 j_0 k_0, \ldots i_{|F|} j_{|F|} k_{|F|}$.. A Δ-complex can then be specified by giving a list of vertex gluings $a \sim b$ and edge gluings $(a, b) \sim (c, d)$, where a, b, c, d are vertices from the disjoint triangles. See Figure 2.5 for some examples. [Hat02, Section 2.1] gives a more precise definition of Δ-complexes; further intuition is given in Section 2.3, where we describe data structures for Δ-complexes. Note that every simplicial complex is also a Δ-complex. As in the simplicial case we will consider only *pure*, 2-dimensional Δ-complexes,

[1] pronounced "Delta complex"

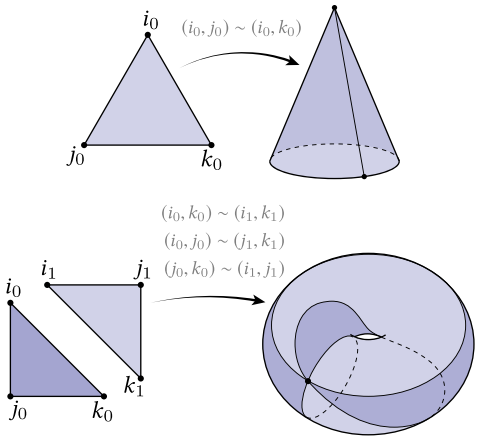


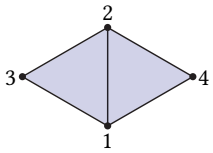
Figure 2.5: In a Δ-complex, the vertices of an edge or triangle are not required to be distinct. For instance, one can build a cone by gluing together two edges of the same triangle (top), or a torus out of two triangles and just a single vertex (bottom).

i.e., every vertex and edge is contained in some triangle (and triangles are the cells of greatest dimension).

Notation becomes more challenging when working with a Δ-complex, since edges and triangles are no longer uniquely determined by their vertices (see for instance Figure 2.5). One possibility is to write, say, $v_i(\sigma)$ to denote the ith vertex of a mesh element σ—for instance, $v_1(e)$ and $v_2(e)$ would then give the two endpoints of edge e. However, this notation quickly becomes tiresome. Instead, we will stick with the convention that a k-dimensional mesh element is specified by a juxtaposition of $k+1$ vertices—but importantly, one should not assume that these vertices are distinct, nor that they uniquely determine the identity of the mesh element. For instance, the symbol ijk simply denotes *some* triangle with vertices i, j, k, where these vertices need not be distinct vertices. The value of this notation is merely that it gives distinct names to the three corners of the triangle, which can be referenced in subsequent statements. The ambiguous identity of the element in question is typically not a problem, because we consider statements of the form *"for each triangle $ijk\ldots$"*, or sum a quantity over *all* triangles, *etc.*. Definitions, theorems, and algorithms will of course consider special cases (*e.g.*, elements with repeated vertices) as needed.

2.3 Topological Data Structures

Topological cell complexes can be encoded by a variety of *topological data structures*. One basic data structure is the *vertex-face adjacency list*, which simply describes each triangle as a list of three vertices. For instance, the mesh below could be encoded as a table



	v_1	v_2	v_3
f_1	1	2	3
f_2	2	1	4

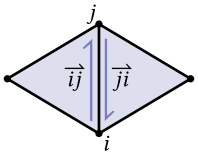
where each row describes a triangle, and the three columns give the indices of the three vertices. This representation is popular due to its conceptual simplicity, and ease of implementation (*e.g.*, it can be stored as just a $|F| \times 3$ dense array). However, it has one major shortcoming: a vertex-face adjacency list cannot, in general, be used to describe a Δ-complex. The basic reason is that it tells us only how to identify the *vertices* of different triangles in the adjacency list—but does not unambiguously determine how edges should be glued together. For example, Figure 2.6 shows an example of a vertex-face adjacency list where the edges can be glued together in many different ways. The reason this representation works for ordinary extrinsic triangle meshes is that the geometry canonically defines the gluings: the only way to connect two vertices in space is by the unique straight line segment between them. But when edges become geodesics on a polyhedral surface, there are often many different ways they can be glued together.

It is essential, therefore, that a data structure used to encode the connectivity of an intrinsic triangulation must describe how edges are glued together. In this section we consider several possibilities and their trade offs.

- **Halfedge mesh** — In a halfedge mesh, each edge is split into pairs of oppositely-oriented *halfedges*, which can be used to infer the rest of the connectivity information. Halfedge meshes makes it easy to circulate around vertices and faces in a consistent order—but as a consequence, they can only describe manifold, oriented surfaces.
- **Signed incidence matrices** — Rather than a single vertex-face adjacency list, signed incidence matrices separately encode vertex-edge and edge-face incidence relationships, via two sparse matrices. In contrast to a halfedge mesh, signed incidence matrices can encode spaces that are neither manifold nor orientable—but cannot easily circulate around vertices and faces.
- **Gluing map** — In addition to a standard vertex-face adjacency list, a gluing map explicitly specifies how the three sides of each triangle get glued to sides of other triangles in the mesh. In other words, it provides exactly the missing information about edge gluings. A gluing map is somewhere between a halfedge mesh and a signed incidence matrices: it can encode nonorientable meshes that are still edge-manifold, and can easily circulate around vertices and faces.

In general, it is important that the data structure used to encode connectivity is expressive enough to describe the type of complex needed for a given task. In the context of intrinsic triangulations we will need to work with so-called Δ-*complexes*[2], which can be encoded using standard data structures such as a halfedge mesh, or signed incidence matrices.

We will also consider the set of *halfedges* H, directed edges associated with each edge: for an edge ij the are two associated halfedges, one pointing from $i \to j$ and one from $j \to i$.



Halfedge meshes Oriented manifold triangulations can naturally be represented by a *halfedge mesh*. We split each edge ij into two oriented halfedges: $\overrightarrow{ij}$ and $\overrightarrow{ji}$. Then, we define two functions, *twin* and *next* on the set of halfedges. The twin function takes in a halfedge and returns the halfedge that points in the opposite direction along the same edge, while the next function returns the next halfedge along in the same face. Note that the twin function relies on the fact that our mesh is edge manifold, and the next function relies on the fact that our faces are oriented. These two functions make it easy to traverse neighborhoods in the mesh. Since these halfedges essentially represent a mesh as a collection of triangles which have been glued together, they support general Δ-complexes in addition to simplicial complexes, and are a natural candidate for representing intrinsic triangulations. However, one can also use face-vertex adjacency lists, augmented with a gluing map, as described by Sharp and Crane [SC20a, Section 4.1].

2.4 The Laplace matrix

[2] pronounced as "Delta-complexes"

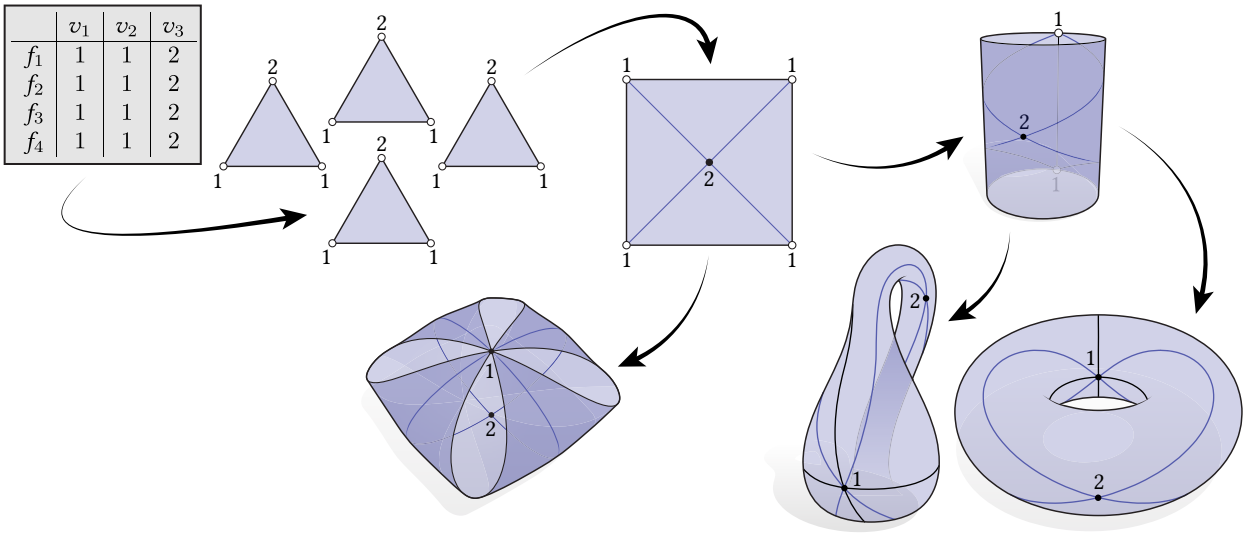
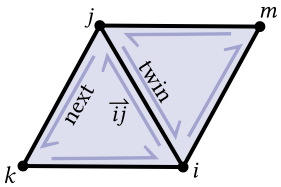


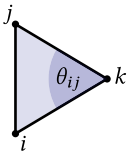
Figure 2.6: A basic vertex-face adjacency list (top left) provides an ambiguous encoding of connectivity. Here, for instance, it specifies that the mesh is comprised of four triangles and two distinct vertices. However, without additional information about how *edges* are glued together, there are many possible ways to glue these triangles together—three are shown at bottom.



The Laplace matrix for a triangle mesh is a fundamental quantity which appears widely in geometry processing algorithms, ranging from smoothing operations to conformal parameterization to spectral methods. It is the discrete equivalent of the continuous Laplacian Δ, or more formally the *Laplace-Beltrami* operator, when defined on curved surfaces. The Laplace matrix is a real sparse matrix $L \in \mathbb{R}^{|V|\times|V|}$, where each row corresponds to a vertex, and there is a nonzero entry L_{ij} corresponding to each edge ij in a mesh. We will discretize this matrix as the *cotan-Laplacian*, which can be derived in the context of electrical networks [Mac49; Duf59], minimal surfaces [PP93], finite elements, or discrete exterior calculus [Cra+13], among others. The entries of the cotan Laplacian are given by

$$w_{ij} = \sum_{ijk} \frac{1}{2} \cot \theta_{ij} \qquad L_{ij} = -w_{ij} \qquad L_{ii} = \sum_{j \in \mathcal{N}_i} w_{ij} \tag{2.1}$$

where w_{ij} is the *cotangent weight*, the sum of $\cot \theta_{ij}$ over all triangles in which an edge ij appears, with θ_{ij} as the angle of the triangle opposite the edge ij (inset). The off-diagonal entries L_{ij} are the negative cotangent weights for the pair of vertices L_{ij}, and the diagonal entries L_{ii} are the sum over all cotangent weights for vertex i. Note that self edges (where $i = j$) do not contribute to L. Past work varies in the sign given to the Laplace matrix; in this document we will always use the *positive (semi)-definite* Laplacian given above. Lastly, we note that the cotangent weights, and thus the Laplacian, can be constructed from just the edge lengths of each triangle if desired, which will be an important property when we consider intrinsic triangulations.



The Laplace matrix L actually represents the *weak Laplacian*, which means that $u^T L v$ approximates the smooth integral $\int_M u(x)\Delta v(x) dx$. For technical reasons, this means that if we wish to approximate the solution u to an equation of the form $\Delta u = f$, we actually need to solve the discrete equation $Lu = \mathsf{M}f$, where M is the *mass matrix*, a sparse symmetric matrix depending only on the area of faces of the mesh.

On "nice" triangulations (for instance, a triangulation with all acute angles), the cotangent weights will all be nonnegative $w_{ij} \geq 0$. However, on low-quality triangulations some of these weights may be negative. In the nice case where all weights are nonnegative, the Laplace matrix has a *maximum principle*—a desirable basic property of the continuous Laplacian, which guarantees that solutions to Laplace equations which have extrema only on the boundary. A key benefit of intrinsic triangulations will be the ability to construct a special high-quality Laplace matrix with all nonnegative cotangent weights, which not only guarantees the maximum principle but also generally improves accuracy (Section 5.8).

Edges with positive cotangent weights are *Delaunay* edges, corresponding to the widely-studied Delaunay property is from planar geometry.

Perfect Laplacians. **TODO: make the comments about Wardetzky et al's no free lunch paper and IDT here?**

2.5 Planar Delaunay triangulations

The *Delaunay triangulation* of a point set in the plane is a fundamental concept in computational geometry. **TODO: image of a planar Delaunay triangulation** These triangulations are widely studied for the geometric and algorithmic quantities; in many senses the Delaunay triangulation is the "best" triangulation of a point set, *e.g.* by *Rippa's Theorem* the Delaunay triangulation offers the smoothest linear interpolation of values at vertices [Rip90]. There are many equivalent characterizations of the Delaunay triangulation of a point set, including:

- The Delaunay triangulation is the dual of the *Voronoi diagram*.
- The Delaunay triangulation is the set of all triangles with empty circumcircles.
- The Delaunay triangulation is the unique triangulation where all cotangent weights are nonnegative.
- The connectivity of the Delaunay triangulation is that of the 3-dimensional lower convex hull of the point set, after lifting to a parabola according to $z := x^2 + y^2$.

The first three definitions are of particular interest in this text, because they can be easily generalized to intrinsic triangulations of curved surfaces. **TODO: stereographic projection condition?** **TODO: figure of these conditions**

Co-circular points The definitions give above are all very straightforward in general position, where a point set always has exactly one unique Delaunay triangulation. However, if the point set is not in general position, and contains four or more points on a common circle,

then there may be a small family of triangulations which are all Delaunay triangulations (or perhaps the Delaunay "triangulation" is really a polygonal complex with some > 3-sided faces, depending on the choice of definition). In this text we will take the former viewpoint and always work with triangulations, and thus must consider *a* Delaunay triangulation rather than *the* Delaunay triangulation. For example, in the **TODO: inset diagram** we show two different Delaunay triangulations of a point set, both of which have circumcircles with empty interiors, and all nonnegative cotangent weights, *etc.*.

Constructing planar Delaunay triangulations Many strategies have been developed to compute the Delaunay triangulation of a point set in the plane. A direct method is to construct a parabolic lifting of the point set in to $\mathbb{R}^3$, then compute convex hull of point set, however algorithms for constructing 3D convex hulls are nontrivial in and of themselves. A simpler approach is Lawson's Flipping procedure [Law72a], which takes any triangulation as input, then applies a greedy strategy to generate the Delaunay triangulation. This greedy strategy proceeds by "flipping edges", exchanging an adjacent pair of triangles ijk, jil for the opposite triangles ilk, jkl (see Section 3.3 for more about edge flips). In modern planar geometry, the edge-flipping approach is often avoided in favor of line-sweep approaches which are asymptotically more efficient **TODO: citation**. However, in our setting we are interested in the flip-based strategy, because it generalizes naturally to the setting of intrinsic triangulations, whereas linesweeps do not. Furthermore, we will show empirically that flipping exhibits essentially linear scaling on real data (Figure 5.2).

2.6 Tangent Vectors

TODO: Say what a tangent vector is, so that we can talk about tangent spaces and signposts and vector field processing later

2.6.1 The exponential map

The *exponential map* takes in a point x on a surface, as well as a tangent vector u, an returns the point $\exp_x(u)$ obtained by starting at x and walking along the surface in the direction of u for a distance $|u|$. This essentially boils down to laying out any triangles that the path passes through in the plane. Note that this layout depends only on the connectivity and edge lengths of our mesh, so the exponential map can be evaluated purely intrinsically.

TODO: figure

Chapter 3

Basic formulation

In this section, we introduce the basic classical formulation of intrinsic triangulations.

3.1 Extrinsic vs intrinsic information

Extrinsic properties fundamentally depend on how a shape sits in space, whereas *intrinsic* quantities do not——for instance, the length of a piece of string is intrinsic, whereas its bounding diameter is extrinsic. At present, we think about geometric computing mainly from an extrinsic viewpoint—for instance, the geometry of a mesh is typically given by explicit vertex positions $p_i \in \mathbb{R}^n$ in some global coordinate system. However, recent trends suggest that the intrinsic viewpoint can be quite powerful. For instance, tools from topological data analysis often treat data as an abstract distance metric with no a priori assumption on dimension or point locations; spectral geometry processing analyzes shape in terms of the modes and frequencies of the purely intrinsic Laplace-Beltrami operator. In this course, we take the intrinsic viewpoint to its natural conclusion for problems in basic polygon mesh processing. It provides tremendous flexibility, since it relaxes a basic assumption of classical geometric computing: a mesh no longer needs to be embedded as a collection of planar triangles in $\mathbb{R}^n$. Instead, one can work in the larger space of intrinsic triangulations (Section 3.2), providing a "relaxation" that helps overcome a wide variety of long-standing geometric challenges. Moreover, the conceptual difficulty of working with intrinsic triangulations can largely be encapsulated by a well-chosen software abstractions, as in numerical linear algebra. The goal of this course is to provide a deeper understanding of what new opportunities the intrinsic perspective can provide, and translate some of these insights into practical, broadly-accessible tools.

3.2 Intrinsic triangulations

Much like the embedded triangle mesh, an intrinsic triangulation $\mathcal{T} := (M, \ell)$ is given by connectivity $M = (V, E, F)$, a complex of vertices, edges, and triangular faces, and geometry ℓ. However, rather than vertex positions, the geometry is defined only by lengths $\ell_{ij} : E \to \mathbb{R}_{>0}$

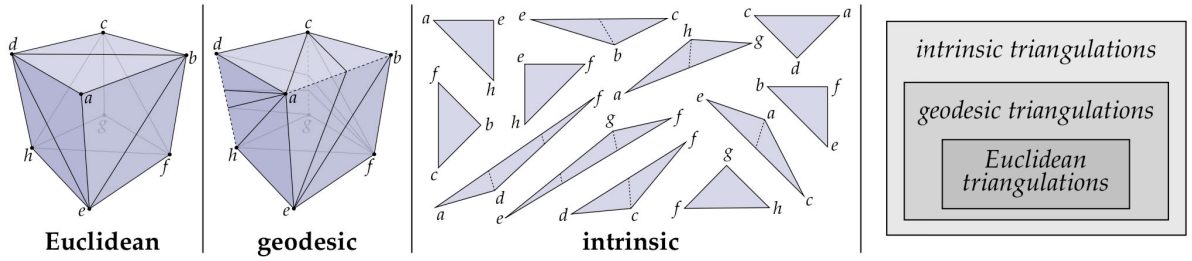
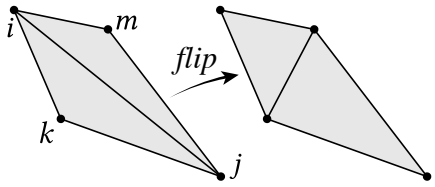


Figure 3.1: Traditional geometric computing considers triangulations where edges are straight line segments in Euclidean space (far left). The much larger spaces of *geodesic* and *intrinsic* triangulations provide tremendous additional flexibility by allowing edges to be straight paths along the surface (center left), or by just considering an abstract collection of triangles identified along shared edges (center right).

associated with each edge ij. We require that ℓ_{ij} satisfy the triangle inequality in each face; these edge lengths then determine a Euclidean metric on the interior of each triangle.

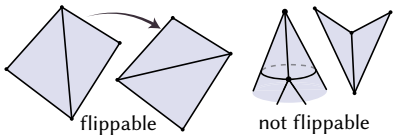
Topologically, we define the connectivity M to be a 2-manifold Δ-*complex* Hatcher [Hat02, Section 2.1], allowing *e.g.* self-edges which connect a vertex to itself (see Section **??** for an introduction and formal definition). Although polyhedral surfaces are most naturally modeled as a *simplicial* complex, formalizing intrinsic triangulations in the larger space Δ-complexes will be a necessary generalization for key results to hold. However, one must take care, as working in this more general space demands new nontrivial proofs even for seemingly intuitive properties—see *e.g.* Bobenko and Springborn [BS07] and Sharp and Crane [SC20b].

3.3 Edge flips



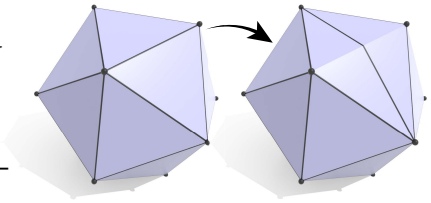
Intrinsic triangulations can be transformed via the *edge flip* operation, replacing some edge ij with the opposite diagonal km. In planar geometry, an edge flip would generally be implemented by measuring the distance between the endpoints k, m of the new edge (inset). However, one can observe that this operation is equivalently defined entirely from edge lengths, without any notion of vertex positions, by specifying the new edge length as the shortest path through the glued metric of the two triangles [Riv94]. Computationally, the new intrinsic edge length can be evaluated from the five already-known lengths in the diamond, via elementary geometry. This operation coincides with the ordinary planar notion of an edge flip for flat triangulations, but now generalizes directly to intrinsic triangulations which lack vertex positions, and may describe curved geometry. Traditionally, edge flips have been mainly used to achieve the Delaunay property (see Section 5.2), but in fact the edge flip is a general operation which can have other uses (*e.g.* Section 6.1).

Which edges can be flipped? An edge can be flipped if an only if it satisfies two conditions:



1. its endpoints both have degree at least two, and
2. its neighbors form a convex quad.

If condition 1 fails, then flipping the edge would lead to a degree-zero vertex, which is not allowed in a pure Δ-complex[1]. If condition 2 fails, then flipping the edge would lead to overlapping triangles, which no longer faithfully represent the geometry of the original mesh.

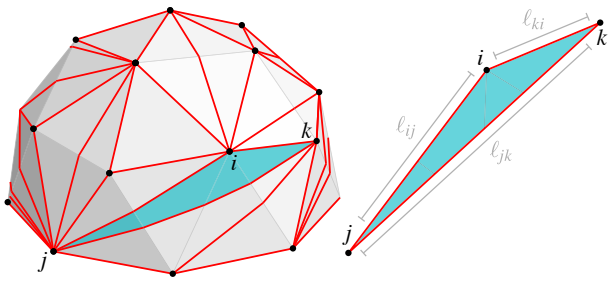


Intrinsic edge flips preserve geometry. The remarkable property of the these intrinsic edge flips is that they exactly preserve Euclidean metric of the surface described by the triangulation. As concrete examples, quantities like surface area, distance along the surface, and Gaussian curvature—intrinsic geometric quantities—are invariant under the edge flip operation. An alternate perspective which may elucidate this property is to view intrinsic triangulations as sitting along the surface of an abstract manifold equipped with a *cone metric*; edges flips simply transform between different triangulations of the manifold. We thus have the freedom to flip edges to generate desirable intrinsic triangulations, without any cost of approximation error in the representation of the underlying surface. Section 5 explores how this operation allows us to robustly construct triangulations with excellent numerical conditioning, and Section 6 uses a similar mechanism to introduce special geodesic edges into a triangulation.

Flip graph. The edge flip operation induces the *flip graph* on the set of intrinsic triangulations, where two triangulations are connected if they are related by a single edge flip. Analyzing the correctness and complexity of edge flip-based algorithms will amount to proving various properties of the flip graph. As a basic example, it is known that the flip graph is connected: all possible intrinsic triangulations of a given surface are related by edge flips [Law72b].

3.4 Intrinsic triangulations of embedded surfaces

Usually, we obtain an intrinsic triangulation by starting with an ordinary *extrinsic* mesh in $\mathbb{R}^3$, reading off the edge lengths $\ell_{ij} = |f_j - f_i|$ from the vertex positions to initialize an intrinsic triangulation, then transforming this triangulation via local intrinsic operations (edge flips, vertex insertions, *etc.*). Because these operations preserve the geometry of

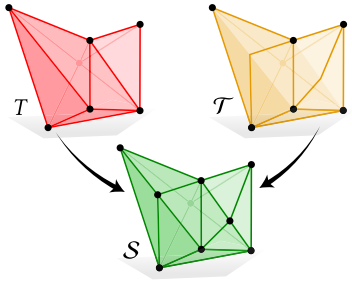


[1]Technically, we should also worry about degree-two vertices, since flipping a self edge can actually decrease vertex degree by two. However, one cannot have a degree-two vertex with a self-edge in a Δ-complex, as the neighboring faces could not be triangles.

the extrinsic surface, there is always a well-defined 1-to-1 map between points on the extrinsic and extrinsic meshes. The intrinsic triangulation can hence be drawn "on top of" the extrinsic mesh, where each intrinsic edge is drawn as a geodesic curve. A key computational challenge is to develop data structures which encode not just the edge lengths of the intrinsic triangulation, but furthermore encode the paths of these geodesic edge along the surface, and allow the bijection to be queried (Section 4).

From an algorithmic perspective, it may seem that this intrinsic triangulation is an abstract object, with little value for practical computation. Indeed, the intrinsic representation is by definition a less informative description of a surface, discarding geometric data like surface normals and dihedral angles. Furthermore, performing intrinsic edge flips implicitly constructs bent intrinsic triangles which lay across the surface, and whose edges are certainly not straight lines between vertices (inset). However, since the intrinsic metric is preserved, and thus intrinsic quantities like geodesic distances, tangent vectors, Laplacians, *etc.* remain perfectly well-defined, one can in fact evaluate many useful algorithms directly on an intrinsic triangulation, to great practical benefit (*e.g.* Section 5.3).

The common subdivision The common subdivision $\mathcal{S}$ of the original triangulation T and intrinsic triangulation $\mathcal{T}$ allows us to visualize $\mathcal{T}$ concretely. $\mathcal{S}$ is the polygon mesh obtained by "cutting up" T along the edges of $\mathcal{T}$ or vice versa. Any edge or face of T or $\mathcal{T}$ can be expressed as a union of edges or faces (*resp.*) in $\mathcal{S}$. Importantly, any piecewise-linear function on T or $\mathcal{T}$ can be represented exactly as a piecewise-linear function on $\mathcal{S}$. This allows us interpolate the vertex positions f_i from the original triangulation to obtain vertex positions on $\mathcal{S}$ which make each face planar and convex.



3.5 Intrinsic Delaunay triangulations

TODO: write this

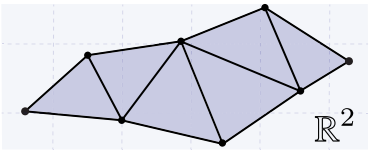
We will see in Section 5.2 that any intrinsic triangulation can be transformed to be Delaunay via a simple edge flipping algorithm, but for now we will just discuss the properties of an intrinsic Delaunay triangulation.

Cocircular quads In general position, the intrinsic Delaunay triangulation is unique; any given piecewise-flat polyhedron admits exactly one intrinsic triangulation among its vertex set which satisfies the Delaunay property. However, in non-generic configurations, there may be pairs of triangles which form co-circular quadrilaterals. These edges have cotangent weight exactly 0, and a triangulation will be intrinsic Delaunay before or after flipping such an edge. Accordingly, there is then some small family of intrinsic triangulations all of which are Delaunay, so in general must speak of *an* intrinsic Delaunay triangulation, rather than

the intrinsic Delaunay triangulation. Note that these edges with cotangent weight 0 do not contribute anything to the Laplace matrix, and thus the intrinsic Delaunay Laplace matrix really is unique for any polyhedron, even though the Delaunay triangulation is not—this is one of the core results of Bobenko and Springborn [BS07].

3.6 Properties of intrinsic delaunay triangulations

Here, we gather a collection of useful properties of intrinsic Delaunay triangulations. Properties of planar Delaunay triangulations usually extend to the intrinsic surface case, when defined in an appropriate manner.



There are two main proof techniques used to argue for these properties, and to generalize results from the planar setting. The first technique is to consider the change in the some quantity when a non-Delaunay edge is flipped. If the quantity can be shown to decrease whenever an edge is flipped, then it must be minimized on a Delaunay triangulation, though careful treatment of the non-general-position case where there are multiple Delaunay triangulations may be necessary. This strategy is useful for establishing minimal geometric quantities, bounds, and conserved quantities on the Delaunay triangulation. The second technique is to consider all possible triangle-strip unfoldings of a triangulation (inset). All of these unfoldings are themselves planar Delaunay triangulations, so this strategy is useful for generalizing properties from planar computational geometry to the intrinsic case.

The following properties all hold for intrinsic Delaunay triangulations of piecewise flat surfaces:

- **Empty Triangle Circumcircles.** If a triangle ijk appears in a Delaunay triangulation, then it has a geodesic circumcircle with empty interior [BS07, Proposition 10]. If any triangle ijk has a geodesic circumcircle with empty and interior and furthermore i,j,k are the only vertices on the boundary of the circle, then ijk necessarily appears in a Delaunay triangulation.
- **Empty Edge Disks.** Each edge ij in a Delaunay triangulation has a geodesic disk with i,j on its boundary and an empty interior. If i,j are the only vertices on the boundary of the disk, then the edge ij necessarily appears in a Delaunay triangulation. The disk need not be a *diametral* disk (that is, a disk for which edge ij is a diameter). The edges for which an empty diametral disk does exist form the *Gabriel graph* [GS69], which is a subgraph of a Delaunay triangulation.
- **Contains Nearest Neighbors.** Each vertex i always has an edge connecting to its nearest geodesic neighbor(s) j. This follows directly from the empty edge disks property.
- **Maximizes Angles.** A Delaunay triangulation maximizes the minimum corner angle in the triangulation. A stronger statement also holds in general position, when there are no cocircular quadrilaterals: the sequence of all corner angles sorted from smallest to largest is lexicographically minimized [Sib78].
- **Rippa's Theorem.** A Delaunay triangulation yields the smoothest interpolation of

piecewise-linear functions at vertices, in the sense of Dirichlet energy (Rippa's Theorem) [Rip90; BS07; Che+10].

- **Minimal Spectrum.** The eigenvalues of the Laplace matrix are minimized on a Delaunay triangulation, that is for the i'th eigenvalue λ_i, all other triangulations have $\lambda_i' \geq \lambda_i$ [Che+10].
- **Geometric Spanner.** In a Delaunay triangulation, the graph distance along edges between any two vertices is at most twice the geodesic distance between those vertices [Xia13].

Several of the proofs cited above are stated in the context of planar triangulations, but apply without modification to intrinsic triangulations. When the notion of a geodesic circumcircle or disk arises, it can be formalized as an isometric immersion of a Euclidean disk into the surface. Note that such an immersion is only well-defined if the immersed disk does not strictly contain any cone points, and that the immersed disk may overlap itself along the surface.

3.7 Other notions of Delaunay Triangulations

Cite all the planar stuff, incl. Shewchuk that discusses Delaunay, explain how it's different. Liu paper that does splits only so it isn't really intrisinc

Delaunay tetrahedralizations

3.8 Historical roots

Intrinsic triangulations have their origins in the work of Regge [Reg61], as a representation for the study of general relativity. Modern mathematical study defined the notion of the intrinsic Delaunay triangulation [Riv94], and established that it can be constructed by edge flipping [Ind+01; BS07]. This construction leads to the intrinsic Delaunay Laplacian, a canonical Laplace matrix associated with any polyhedron which is a function only of its geometry, not its tessellation [BS07]. Additional research has also studied deep connections between intrinsic triangulations and discrete uniformization [Luo04], as well as circle packings and conformal parameterization [KSS06].

Recovering an embedding Given any embedded triangulation, it is easy to construct an intrinsic triangulation by reading off the edge lengths. It is also natural to ask if one can go in the reverse direction: given an intrinsic triangulation, can we recover some embedding with the same edge lengths?

First, we can make a few observations. To begin with, there exist some valid intrinsic triangulations which satisfy the triangle inequality, but do not admit any embedding **TODO: tet with tiny t-rex arms**. Even when an embedding exists, it is generally not unique even beyond the obvious rigid transformations—any outward bump can be "popped in" as an inward bump to construct an alternate embedding with the same edge edge lengths **TODO: another image**.

Representations Traditionally, there has been relatively little study of practical aspects of intrinsic triangulations, such as appropriate data structures, robustness concerns, or integration with larger algorithmic pipelines in geometry processing. One notable exception is the work of Fisher et al. [Fis+07], which introduced a crossing-based data structure and demonstrated its use.

Chapter 4

Representations

The most obvious representation, or data structure, for an intrinsic triangulation is a list of edge lengths ℓ, along with some mesh data structure to encode the connectivity M. However, this representation leaves two open questions.

First, there are many mesh data structures, but because M may be a Δ-complex, not all will suffice (Section 4.1). Second, and more fundamentally, storing only M and ℓ is inherently abstract. In the common case where our intrinsic triangulation is defined atop an embedded mesh, it does not allow for reconstructing the trajectory of an intrinsic edge along the embedded surface. More generally, the abstract representation does not support computations which involve tangent-valued data, or require querying the bijection between the intrinsic triangulation and the underlying mesh. Accordingly, we also consider richer representations which support a full range of operations on the intrinsic triangulation (Sections 4.2-4.4). Table 4.1 summarizes the various operations permitted with the various representation.

4.1 Representing connectivity

The connectivity among $M = (V, E, F)$, the vertices, edges, and faces in an intrinsic triangulation, can be represented by a mesh data structure; these structures have been widely studied in applied geometry (see *e.g.* [Bot+10]). However, while most traditional triangle meshes are formalized as a simplicial complex, intrinsic triangulations demand a Δ-complex which may contain repeated elements (Section 3.2). Using, *e.g.*, a naive listing of the vertex indices for each face will not suffice, between it is impossible to distinguish faces with identical vertices.

Fortunately, many simple and standard mesh data structures represent general Δ-complexes without modification, such as edge-based winged-edge and halfedge structures [Bau75; Wei85; Ket99]. In fact, one can easily mimic these structures by augmenting a face-vertex list with an additional "gluing" array holding two values per face-side, as described in Sharp and Crane [SC20a, Section 4.1] in the context of intrinsic triangulations.

Operation	Meaning	Representation				Notes
		Lengths	Explicit	Signpost	Integer	
edge flip	replace ij with lm	✓	✓	✓	✓	
face split	insert a new vertex in intrinsic face ijk	✓		✓	✓	
edge split	insert a new vertex along intrinsic edge ij	✓		✓	✓	
vertex reposition	reposition vertex i along the surface	✓		✓	✓	must be an inserted vertex
remove vertex	remove vertex i and triangulate	✓		✓	✓	must be an inserted vertex
correspondence	map points between intrinsic and extrinsic mesh		✓	✓	✓	
transfer tangent data	transfer tangent vector data between the intrinsic and extrinsic mesh			✓		
extract edge	get the trajectory of intrinsic edge along the extrinsic mesh		✓	✓	✓	
extract common subdivision	get all faces of common subdivision		✓	✓	✓	signpost does not guarantee valid connectivity

Table 4.1: The operations supported by various representations of intrinsic triangulations. Only signposts and integer coordinates support a full range of remeshing operations, while still encoding the trajectory of intrinsic edges along the surface. In principle the Explicit representation could support insertion and removal operations, though these have not been described and may be prohibitively complex. Similarly, any data structure which can extract the common subdivision could in principle transfer tangent data, but it is easiest with the signpost representation.

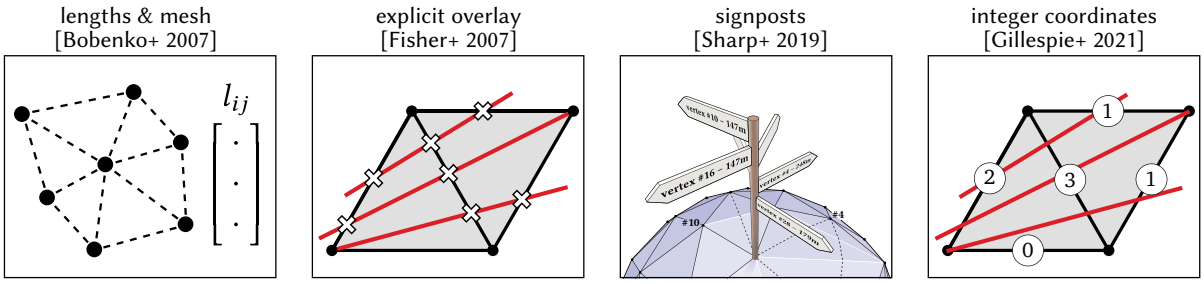


Figure 4.1: The representations for intrinsic triangulations considered in this course.

4.2 Explicit crossings

Even with a proper mesh data structure, additional data must be stored to encode how an intrinsic triangulation sits atop some initial embedded mesh (Section 3.4). The most direct approach is to explicitly store ordered lists of *crossings*, the locations where each intrinsic edge crosses an edge of the underlying mesh, as developed by Fisher et al. [Fis+07]. However, even just flipping an edge in this representation is a nontrivial operation which must traverse and update the crossing graph, as opposed to the formulaic constant-time updates offered by other representations. Additional operations beyond edge-flips have not been described for this representation, although in principle they might be implemented with some traversal of the crossing graph.

TODO: say more about explicit overlay so we don't short-change it...

4.3 Signposts

Rather than explicitly representing trajectory of each intrinsic edge, we can instead implicitly represent the trajectory by storing not just intrinsic edge lengths, but also the direction of each edge, as a tangent vector at the incident vertices; this strategy was developed by Sharp, Soliman, and Crane [SSC19a]. The benefit of this approach is that it retains the simplicity and efficiency of the lengths-only case, while any additional data about how the intrinsic triangulation sits atop the extrinsic triangulation can be lazily recovered from the signposts when needed. Furthermore, signposts are very useful for working with vector-valued quantities on a surface.

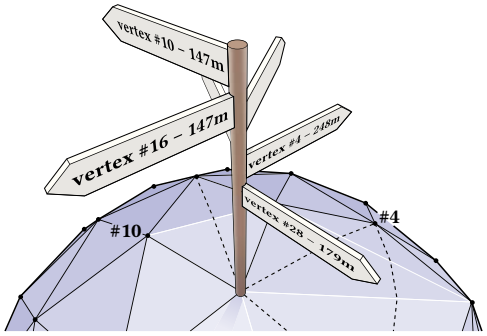
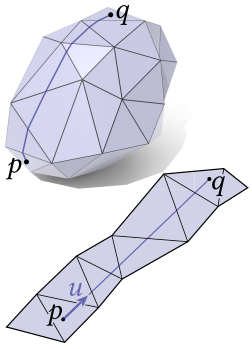


Figure 4.2: Signposts encode edges of an intrinsic by storing the length and direction of each edge in the tangent space each vertex.

More precisely, in addition to the connectivity and edge lengths always used to represent an intrinsic triangulation, the signpost data structure stores angles $\varphi : \mathcal{H} \to [0, 2\pi)$. Each angle φ_{ij} stores the direction of the halfedge from vertex i to vertex

j, in the local polar coordinate system at vertex i. These polar coordinate systems can be constructed with an arbitrary choice of reference direction at any vertex, edge, or face; see Sharp, Soliman, and Crane [SSC19a, Section 2.3] for a detailed construction. Additionally, as new points are inserted in to the intrinsic triangulation, we will store the location of each intrinsic vertex on the extrinsic mesh as barycentric coordinates in a face or edge.



Tracing through triangulations. The path of an intrinsic edge along the extrinsic mesh is recovered by "tracing" the edge along the surface, known also as evaluating the exponential map (Section 2.6.1). This is an easy and efficiently local geometric operation, which essentially amounts to evaluating many ray-line intersections while traversing the extrinsic mesh, to compute whether path exits each triangle and enters the next (inset).

In fact, this basic tracing operation enables many queries on the signpost intrinsic triangulation, such as evaluating the correspondence from some point on the intrinsic triangulation to the same point on the extrinsic triangulation, or vice-versa [SSC19a, Section 3.4]. Tracing out all of the edges in the intrinsic triangulation provides the necessary data to construct the common subdivision (Section 5.7).

Local mesh operations Sharp, Soliman, and Crane [SSC19a] define a wide variety of mesh-processing algorithms on the signpost representation of intrinsic triangulations. For the most basic edge flip, signposts can be updated via a simple local formula, just as edge lengths are updated [SSC19a, Section 3.3.1]. However, the key advancement of the signpost data structure is that it supports a wide variety of additional operations beyond flipping edges, such as inserting new vertices and repositioning vertices, while maintaining an encoding of the intrinsic triangulation sits atop the extrinsic mesh for queries like extracting the common subdivision or evaluating the bijection between the intrinsic triangulation and the underlying surface. These operations in turn enables higher-level retriangulation algorithms, like intrinsic Delaunay *refinement* and optimal Delaunay relaxation for the first time, discussed at length in Section 5. For a more detailed discussion of these operations, see Appendix **??**.

Robustness. The implicit encoding of intrinsic edge trajectories in a signpost triangulation is discretely exact: in proper real arithmetic, it would always perfectly reconstruct the paths and the corresponding crossing connectivity. Note that in practice, it is not necessary that tracing the edge ij "exactly" hit the vertex j, merely arriving in the correct vertex neighborhood is sufficient. However, even with this observation, recovering the trajectory in inexact floating-point arithmetic of an edge may fail in nearly-degenerate cases. Although this behavior is typical of geometry processing algorithms, it is particularly worrisome for intrinsic triangulations, for which a key application is robust computation on degenerate 3D models. The integer-based representation (below) supports a similar set of operations, while offering a guarantee of correctly recovering the connectivity of the common subdivision, at the cost of some increased algorithmic complexity.

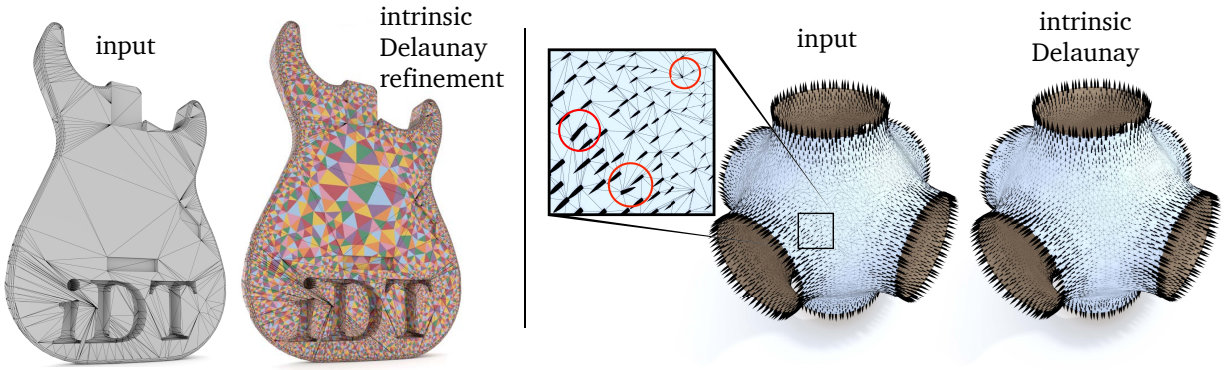
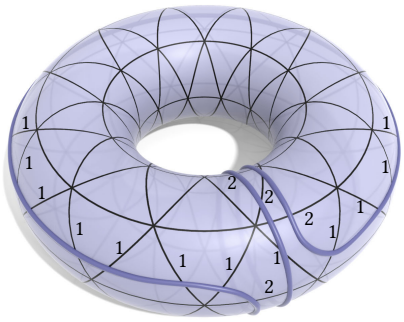


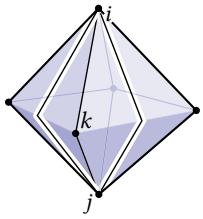
Figure 4.3: *Left*: The signpost data structure enables intrinsic Delaunay refinement for the first time, generating triangulations with good angle bounds. The black wireframe denotes the input mesh, while colored triangles give the intrinsic triangulation. *Right*: Signposts also enable vector field processing; the intrinsic Delaunay Laplacian offers a maximum principle for tangent vector fields, which here avoids unexpected flipped vectors when generating a smooth field.

4.4 Integer coordinates

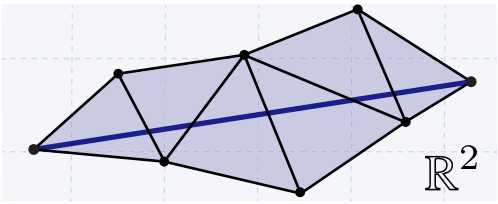
Rather than encoding the correspondence with floating-point signposts, one can instead use integer coordinates in the form of *normal coordinates* and *roundabouts* [GSC21a]. Like the signpost representation, these integer coordinates implicitly encode the correspondence: rather than maintaining an explicit representation of all crossings, one instead maintains some simpler data and lazily determines correspondence data as needed. More explicitly, in addition to the connectivity and edge lengths of the intrinsic mesh, this data structure stores normal coordinates $n : \mathcal{E} \to \mathbb{Z}$ and roundabouts $r : \mathcal{H} \to \mathbb{Z}_{\geq 0}$ on the intrinsic mesh. As with signposts, when new vertices are inserted into the intrinsic mesh one must store the corresponding location on the original mesh as barycentric coordinates in a face or edge.



Normal coordinates & roundabouts Normal coordinates have a long history in computational geometry and topology as a compressed representation of curves on a complex [Kne29; Hak61; SSŠ02]. The basic idea is to count how many times a curve crosses each edge of a triangulation (inset). These crossing counts are sufficient to recover the topological path of a curve on the surface. Moreover, when the curves are geodesic—as in the case of intrinsic triangulations—these counts actually determine the geometry of the curve. Although recovering the geometry of paths along the surface will still ultimately require imperfect floating point computation, normal coordinates offer important basic guarantees about properly representing connectivity, and will serve as a valuable tool for robust computing with intrinsic triangulations.



However, while normal coordinates determine the exact geometry of curves along a triangulation they fail to fully encode the correspondence between two triangulations, as they do not specify which logical edge each curve corresponds to. Since the triangulations are Δ-complexes, the endpoints of an edge may not uniquely identify it—see for example the inset, where a curve traced from i to j could correspond to either of the edges between them. Roundabouts, introduced by Gillespie, Springborn, and Crane [GSC21b], resolve this issue by explicitly tracking the cyclic ordering of edges from both triangulations around each vertex of T.



Tracing through triangulations These integer coordinates support tracing edges of the original triangulation T over the intrinsic triangulation $\mathcal{T}$. This tracing proceeds in two steps: first, one uses the normal coordinates to compute the triangle strip in $\mathcal{T}$ that the curve passes through. Then, using this one can determine the geometry of the curve by laying this triangle strip out in the plane and connecting the endpoints with a straight line (inset). This tracing procedure is sufficient to determine the correspondence between T and $\mathcal{T}$ and construct the common refinement, but it is not as flexible as the signpost tracing procedure, which can be used to directly query the correspondence between arbitrary points on either mesh.

Local mesh operations These integer coordinates support all of the local mesh operations that signposts do, enabling the same applications, *e.g.* intrinsic Delaunay refinement, to be performed while maintaining provably correct correspondence. For a more detailed discussion of these operations, see Appendix **??**.

Robustness **TODO: what to say here?**

Chapter 5

Intrinsic retriangulation

The intrinsic representation offers a large space of possible triangulations of a surface, all of which exactly encode the underlying intrinsic geometry. We traverse this space with retriangulation algorithms, perform edge flips, vertex insertions, and other operations, to construct triangulations with improved numerical conditioning and other desirable properties.

5.1 Comparison to traditional remeshing

Remeshing of surfaces meshes is widely studied in geometry processing [CDS12; CH11; All+08], but such methods must inevitably trade off between element quality and geometric approximation of the input surface. The intrinsic approach escapes this tradeoff, operating in a space of triangulations which all exactly represent the underlying geometry. In fact, intrinsic algorithms offer concrete algorithmic guarantees about the quality of the resulting meshes, which generally are not otherwise available for surface remeshing routines.

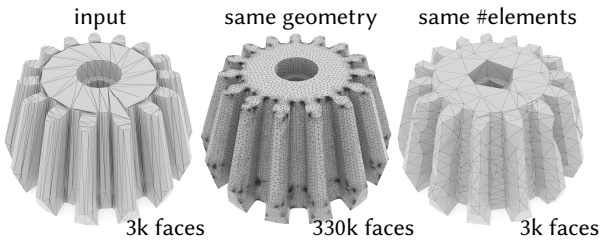


Figure 5.1: Traditional remeshing cannot improve element quality without increasing mesh size or disturbing the geometry; intrinsic triangulations escape this tradeoff.

Of course, the price for this algorithmic power is that the output of intrinsic retriangulation is an intrinsic object with only edge lengths, not a traditional mesh with vertex positions. Fortunately, it is straightforward to adapt subsequent computations to this paradigm; generally one simply needs to evaluate geometric quantities from edge lengths, rather than vertex positions. As one example, the GEOMETRY-CENTRAL C++ library contains a growing collection of routines which seamlessly support this paradigm [SC+19].

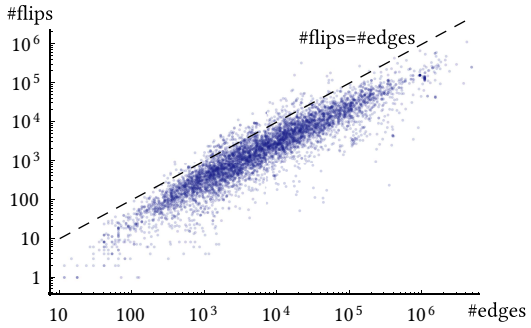
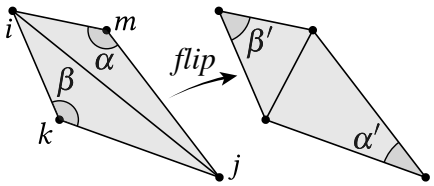


Figure 5.2: Number of edge flips to Delaunay on the Thingi10k dataset [ZJ16]; each point is a 3D model.

5.2 Delaunay flipping

In the plane, there are many equivalent characterizations of so-called Delaunay triangulations, such as the convex hull of a parabolic lifting, all triangles having empty circumcircles, or each edge having opposite corner angles $\alpha + \beta \leq \pi$. The last definition generalizes directly to intrinsic triangulations, computing corner angles via edge lengths, and defines the *intrinsic Delaunay triangulation* (IDT) [Riv94]. Just like the planar Delaunay triangulation associated with a point set, the intrinsic Delaunay triangulation is a fundamental discrete construction defined essentially uniquely[1] for any polyhedron. Much like in planar geometry, the intrinsic Delaunay triangulation has many benefits for numerical computation; in particular it generates the *intrinsic Delaunay Laplacian*. This Laplace matrix has many computational benefits: it uniquely offers a discrete maximum principle, and the corresponding basis functions maximize the minimum angle in any triangle (see *e.g.* Shewchuk [She97]).



The foundational result in the study of intrinsic triangulations is that any triangulation can be transformed to be the intrinsic Delaunay triangulation by repeatedly flipping any edge with $\alpha + \beta > \pi$ (inseet) [Ind+01; BS07]. This mirrors Lawson's edge flipping algorithm in the plane [Law77], though significantly more sophisticated machinery is needed to prove correctness in the intrinsic setting.

Delaunay edge flipping is then the most basic and essential intrinsic retriangulation scheme. Given any standard mesh with vertex positions, one can read off edge lengths to define an intrinsic triangulation, perform edge flips to generate the IDT, and construct the corresponding Laplace matrix for subsequent computation. Alternate approaches have constructed meshes satisfying the intrinsic Delaunay property as the dual of the geodesic voronoi diagram [Liu+17b], or by splitting edges [Liu+15], though these method are significantly more complicated and expensive than simply flipping edges. In terms of runtime, Delaunay edge flipping typically takes just milliseconds in practice for typical inputs, and an empirical study in Sharp, Soliman, and Crane [SSC19a] (Figure 5.2) showed linear scaling on a challenging

[1] In general position. Otherwise, there exists an equivalence class induced by cocircular triangles with $\alpha + \beta = \pi$.

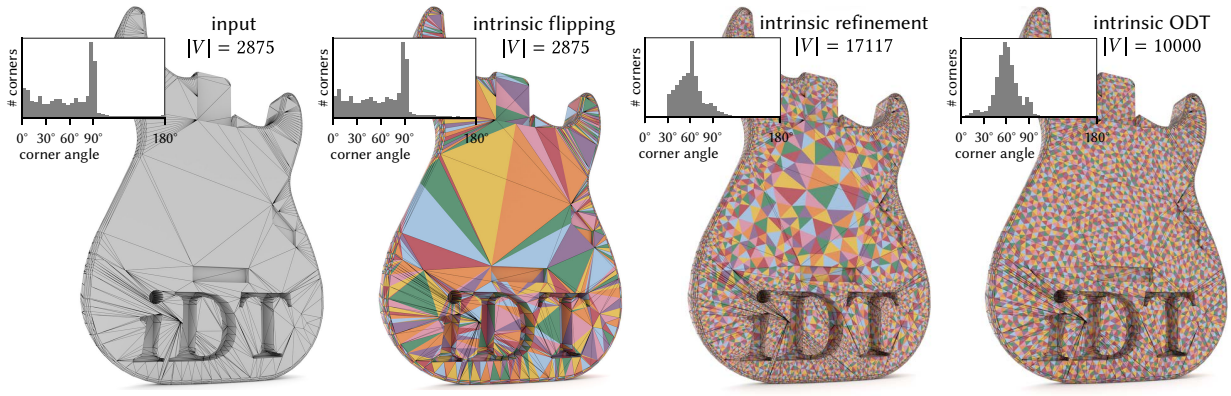


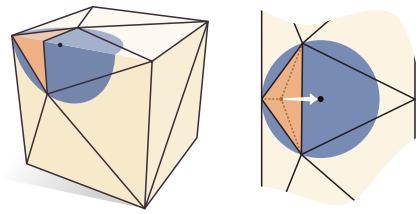
Figure 5.3: Intrinsic retriangulation schemes applied to a computed-aided design model with poor triangle quality. The black wireframe denotes the input mesh, colored triangles give the intrinsic triangulation. Delaunay flips achieve the Delaunay property for a fixed vertex set, while refinement and repositioning further improve triangle quality and vertex distribution.

dataset of real-world models [ZJ16]. However, there is a wide gap between this practical efficiency and asymptotic analysis: the only known runtime bound is exponential [Ind+01; BS07], although the most difficult known counterexample requires only $O(n^2)$ flips (consider triangulating points along the planar parabola $y = x^2$).

5.3 Delaunay refinement

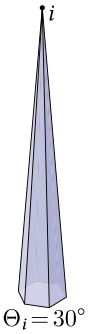
In practice one often seeks triangulations which satisfy criteria beyond the Delaunay property, such as bounds on angles or edge lengths. *Delaunay refinement* progressively inserts vertices to achieve a specified minimum-angle bound, while maintaining the Delaunay property. Sharp, Soliman, and Crane [SSC19a] describe *intrinsic* Delaunay refinement for the first time, making use of the insertion operations offered by the signpost data structure. The method uses an intrinsic variant of *Chew's 2nd algorithm* [Che93; She97], which essentially amounts to the following steps:

- Until a specified minimum angle bound is satisfied:
  - Flip to Delaunay
  - Find any triangle ijk that violates the angle bound
  - Insert the circumcenter p of ijk



Given that the signpost and integer coordinate data structures can flip edges and insert vertices, the only remaining difficulty with implementing this algorithm intrinsically is locating the circumcenter of ijk. But this is also fairly straightforward, thanks to the Delaunay property. Since the triangulation is Delaunay, ijk has an empty circumcircle—in particular, this circumcircle bounds an intrinsically-flat disk with a well-defined center which can be found tracing from the barycenter of ijk [GSC21a].

Note also that in the intrinsic setting, the underlying domain may have skinny needle vertices where the total angle is already smaller than the user-specified minimum angle bound (see inset). In this case, one should simply not perform refinement when the skinny angle is incident on a degree-1 vertex, and hence cannot possibly be improved.

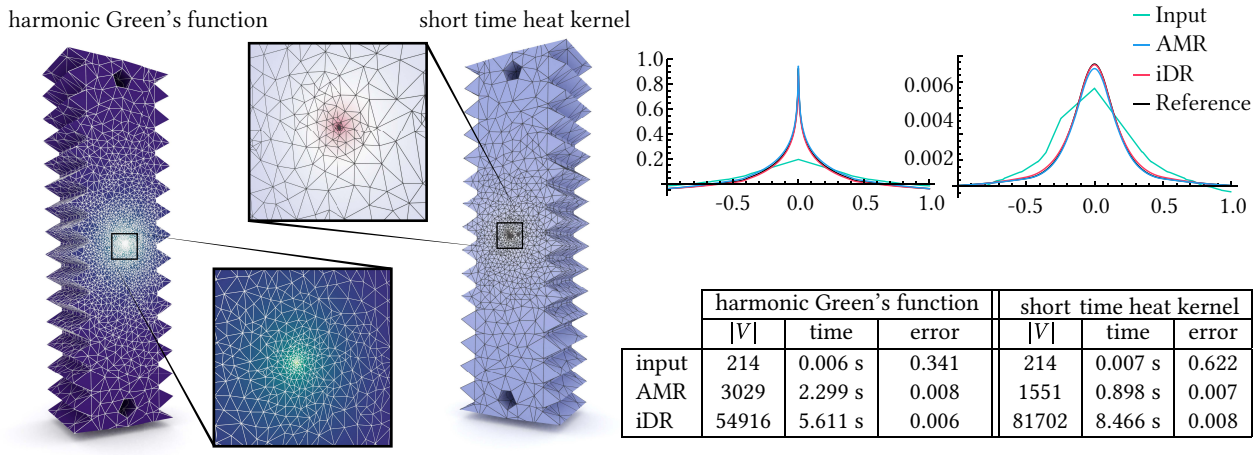


Meshes with boundary The algorithm gets slightly more complicated on meshes with boundary [GSC21a]. The problem is that when there is a boundary, a triangle's circumcenter may not lie within the mesh. In any case, we still trace towards the circumcenter from the barycenter (which is still guaranteed to lie within the mesh), but we may hit a boundary edge before arriving at the circumcenter. If we hit a boundary edge ij, we split it at its midpoint, flip to Delaunay, and remove all inserted interior vertices within a Dijkstra ball of radius ℓ_{ij} centered at the inserted point.

Robustness In practice intrinsic Delaunay refinement is quite effective, reproducing the behavior of the planar algorithm and consistently generating meshes with an interior angle bound of 30°. This matches the guarantee made in the planar case [Che87], though formally extending the analysis to the intrinsic setting on general meshes with boundary is an area of ongoing work. Gillespie, Sharp, and Crane [GSC21a] prove that the algorithm succeeds on meshes without boundaries or needle vertices, but the general case remains open. The practical utility of this approach should not be understated: it automatically generates a surface triangulation with guaranteed quality bounds, and furthermore it does not require the tuning of any numerical parameters. Such behavior is extremely valuable for robust PDE-based geometry processing in practice, generating high-quality meshes for downstream applications without any tradeoff of approximation error, and only modestly increasing element counts (Figure 5.5).

5.4 Optimal Delaunay triangulation

An *optimal Delaunay triangulation* seeks to improve quality not just by refining the triangulation, but also by adjusting the placement of vertices [CX04]. Sharp, Soliman, and Crane [SSC19a] apply this idea in the intrinsic setting by optimizing the location of inserted vertices—modifying the input vertices is of course undesirable, since it would change the surface geometry rather than just the triangulation. The basic strategy is to iteratively move all vertices toward the triangle area weighted sum of the circumcenters of incident triangles, again performing edge flips after each iteration to maintain the Delaunay property [CH11, Equation 4.13]. In the intrinsic setting we can locate circumcenters as in the previous section; rather than averaging these locations, we simply average the vectors *to* these locations, then use this average as our update direction. We insert new vertices on each iteration by splitting edges longer than a user-defined target length (*à la* Tournois et al. [Tou+09]). In general we observe the same behavior as in the Euclidean case: in contrast to Delaunay refinement, we get a better distribution of areas, at the cost of some skinnier angles (Figure 5.3). Crucially, in



	harmonic Green's function			short time heat kernel		
	$\|V\|$	time	error	$\|V\|$	time	error
input	214	0.006 s	0.341	214	0.007 s	0.622
AMR	3029	2.299 s	0.008	1551	0.898 s	0.007
iDR	54916	5.611 s	0.006	81702	8.466 s	0.008

Figure 5.4: Intrinsic AMR allows one to efficiently compute standard geometric kernels to high accuracy. Performing ordinary Delaunay refinement to the same accuracy requires 18x and 54x as many vertices on the harmonic Green's function and short time heat kernel *resp.* [SSC19a].

this process we do not reposition the initial vertices, only those which we previously inserted, and thus continue to exactly preserve the geometry.

5.5 Adaptive mesh refinement

Rather than globally improving an entire mesh, one could instead refine only in regions of interest. Since many PDEs used in geometry processing display interesting behavior only in a localized region of the mesh, this can greatly improve efficiency. Sharp, Soliman, and Crane [SSC19a, Section 5.4 & Supplemental] perform adaptive mesh refiniement (AMR), using *a posteriori* error estimates to guide Delaunay refinement to provide higher resolution near interesting features. They find that such adaptive refinement can produce 2-10x speedups when compared against global refinement when computing standard geometric kernels such as the *harmonic Green's function* or *short-time heat kernel* (Figure 5.4). As another application, Sharp *et al.* consider harmonic maps to the plane: intrinsic Delaunay triangulations guarantee that such parameterizations are injective thanks to the maximum principle, but still result in significant distortion near the boundary. AMR provides the resolution necessary to resolve the map in these boundary regions, while leaving the triangulation sparse in the smoother interior regions (Figure 5.6).

5.6 Intrinsic mollification

Meshes encountered "in the wild" may have near-degenerate geometry (*e.g.*, near-zero angles or areas) that can impair even basic floating point arithmetic [ZJ16]. Delaunay flips sometimes fix degenerate triangles, but are not guaranteed to do so, and even evaluating these flips may

be difficult on degenerate geometry. Extrinsically, it is difficult to repair degeneracies with any kind of guarantee, because a perturbation that improves one element might make another worse. However, in the intrinsic Sharp and Crane [SC20a] describe a simple *mollification* which provably resolves degeneracies, while making only a negligible change to the geometry.

The strategy is to increase the length of all edges by a small, constant amount until no input triangle is degenerate. More precisely, for each corner of each triangle we want

$$\ell_{ij} + \ell_{jk} > \ell_{ki} + \delta, \tag{5.1}$$

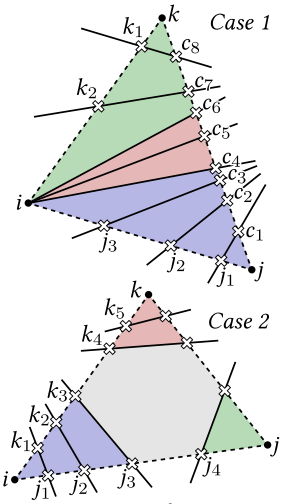
for some user-defined tolerance $\delta > 0$, *i.e.*, we want the triangle inequality to hold with significant inequality, so that triangles are nondegenerate. Then

$$\varepsilon := \max_{ijk} \max(0, \delta - \ell_{ki} - \ell_{ij} + \ell_{jk}) \tag{5.2}$$

is the smallest length we can add to *all* edge lengths to ensure that Equation 5.1 holds. Note that this strategy closely preserves the given geometry: at worst, ε can be just *slightly* larger than δ (due to floating point error); when the mesh is already nondegenerate, $\varepsilon = 0$. Applying intrinsic mollification as a pre-process allows us to apply intrinsic retriangulation even on inputs which are so degenerate that basic floating point arithmetic would otherwise file. We recommend $\delta = 1e^{-5}h$ as a reasonable default value for double precision arithmetic, where h is the mean edge length. Empirical studies [SC20a; GSC21a] have demonstrated that mollification enables intrinsic retriangulation of extremely poor quality meshes, successfully processing all models in the challenging Thingi10k dataset [ZJ16].

5.7 Extracting the common subdivision

One can construct the common subdivision $\mathcal{S}$ by cutting $\mathcal{T}$ along the edges of T, working independently in each face $ijk \in \mathcal{F}$. First, one must use tracing queries to determine where each edge of ijk crosses edges of T. Once these crossings have been computed, one can connect them up to obtain a small number of polygons, mostly triangles and quads. There are only two cases to consider : either some edge of ijk intersects each extrinsic edge, or not (see inset). Repeating this procedure for each face of $\mathcal{F}$ yields the entire common subdivision.



In case 1, all extrinsic edges intersect edge jk. Let $c_1, \ldots, c_r$ denote these crossings. For each crossing along ji, we emit a segment connecting j_a to c_a. Similarly, for each crossing along ki we emit a segment connecting k_a to c_{r-a+1}. Finally, we simply connect the remaining crossings along jk to vertex i.

In case 2, some extrinsic edges clip off each corner. At corner i, we emit segments between crossings j_p and k_p until the number of remaining crossings is equal to the number of crossings along edge jk. Repeating this procedure at the other two corners yields all of the necessary segments.

In principle, one could instead cut T along the edges of $\mathcal{T}$. However, this cutting procedure is in general more complicated, as $\mathcal{T}$ may have vertices within faces of T.

Warning: Even if both T and $\mathcal{T}$ consist of well-conditioned triangles, $\mathcal{S}$ may have low-quality or even near-degenerate faces. As such, it is not a suitable domain for computation, *e.g.* solving PDEs. Instead, it serves a complementary role, aiding visualization and transferring information between triangulations (Section 5.9).

5.8 Robustifying applications with intrinsic triangulations

A key application of intrinsic retriangulation is providing *robustness as a subroutine*: we can make classical algorithms dramatically more robust to low-quality inputs simply by running them on a high-quality intrinsic triangulation rather than directly on the input mesh. The basic pipeline is:

1. intrinsically retriangulate the input mesh,
2. solve your problem on the intrinsic triangulation,
3. copy the solution back to the input mesh.

TODO: Talk about choosing the right data structure.

Now, we discuss several examples.

The heat method for geodesic distance PDE-based methods abound in geometry processing, as they generally provide simple and inexpensive algorithms which benefit from decades of research into fast linear solvers. The heat method is a classic example, computing approximate geodesic distance on a mesh by solving a diffusion equation [CWW13]. Intrinsic retriangulation greatly improves the accuracy of the solution to this diffusion equation, which in turn leads to much more accurate distances on low quality meshes. (Figure 5.5)

The logarithmic map Here, we compute the *logarithmic map* (the inverse of the exponential map) using the vector heat method [SGW06; SSC19b]. This again boils down to solving a diffusion equation, but this time we diffuse tangent vectors rather than scalar functions. Although this algorithm uses tangent vectors during computation, the end result is simply a pair of functions, so this could be done on the pure edge length data structure, with no additional correspondence tracking. However, rendering the intrinsic logarithmic map, as we do in Figure 5.7, requires the common subdivision and hence a more sophisticated data structure.

Globally-optimal direction fields We can also use intrinsic triangulations to compute tangent vector fields. For instance, here we compute smooth vector fields by minimizing a vector Dirichlet energy [Knö+13]. In this case, the solution is a tangent vector field, rather than a scalar function, so even copying values at vertices requires some sort of correpondence data structure. (Figure 5.8)

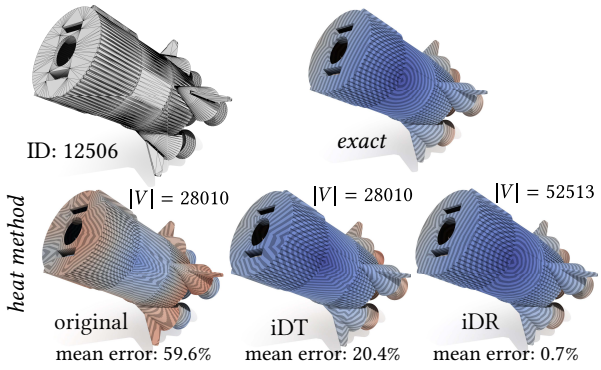


Figure 5.5: Intrinsic triangulations dramatically improve the quality of solutions from PDE-based geometry processing algorithms such as the heat method when run on low-quality geometry. [SSC19a]

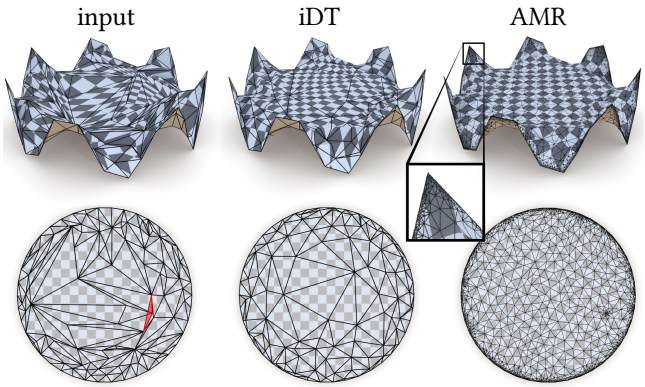


Figure 5.6: Using an intrinsic Delaunay triangulation ensures that a harmonic parameterization is flip-free, while adaptive mesh refinement provides high-resolution in the interesting regions of the mesh. [SSC19a]

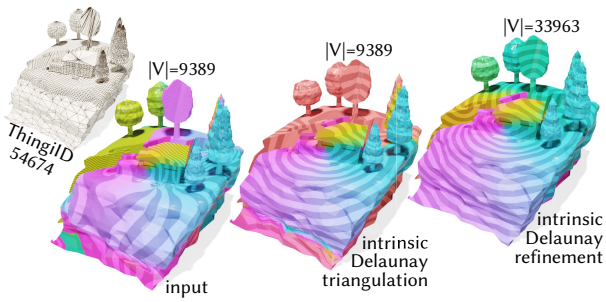


Figure 5.7: Here we visualize a local parameterization, the *logarithmic map*, computed via the vector heat method. Although the vector heat method internally uses tangent vector diffusion, the final logarithmic map is a scalar function, and can hence be visualized using the integer coordinate representation. [GSC21a]

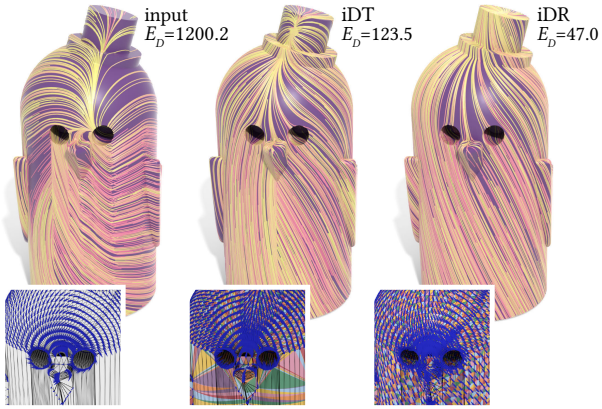


Figure 5.8: The signpost data structure also enables processing tangent data. Here, smoothest vector fields computed on an intrinsic triangulation have much lower Dirichlet energy than one computed on a low-quality mesh. [SSC19a]

5.9 Transferring solutions between triangulations

After computing the solution to a problem on the intrinsic triangulation, one often wants to obtain a solution on the original mesh. The simplest strategy is to simply copy values at vertices back to the original mesh. Since intrinsic retriangulation never removes original vertices, this operation is always well-defined, and can even be performed when using the plain edge length representation. However, this elementary strategy is not optimal, and does not apply to more general tangent vector data.

Optimal attribute transfer Instead of simply copying values back at vertices, we can find the function on the original mesh which is closest to the intrinsic solution. If we desire the L^2-closest function, then this amounts to a sparse linear least-squares system constructed over the common refinement [GSC21a]. Other notions of closeness, and other basis functions could easily be treated similarly. Note that this operation is impossible using only intrinsic edge lengths, as we require the common refinement; one must instead use one of the richer data structures.

Formally, suppose we are given a piecewise-linear function f on the intrinsic triangulation and seek the piecewise-linear function $\hat{f}$ on the extrinsic mesh which minimizes the squared L^2 distance:

$$\|f - \hat{f}\|_{L^2}^2 := \int_M \left| f(x) - \hat{f}(x) \right|^2 \, dx. \tag{5.3}$$

If f and $\hat{f}$ were defined over a *single* mesh, then we could evaluate this distance exactly using the Galerkin mass matrix M [SF08]:

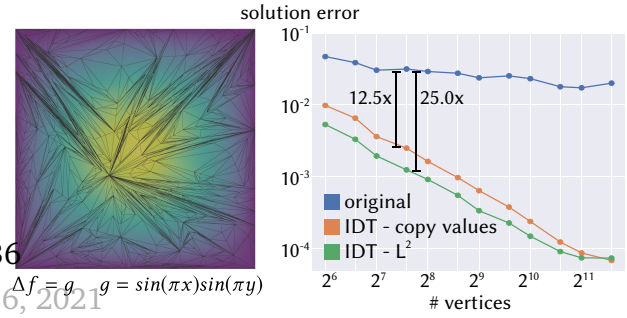
$$\|f - \hat{f}\|_{L^2}^2 = (f - \hat{f})^T \mathsf{M} (f - \hat{f}), \tag{5.4}$$

where we implicitly identify the piecewise-linear functions f and $\hat{f}$ with finite-dimensional vectors in $\mathbb{R}^{|V|}$ given by their values at mesh vertices. Although f and $\hat{f}$ are not both piecewise-linear over the original or intrinsic triangulation, they are both piecewise-linear over the common subdivision $\mathcal{S}$. Thus, we can evaluate the L^2 distance as

$$\|f - \hat{f}\|_{L^2}^2 = (\mathcal{P}_i f - P_o \hat{f})^T \mathsf{M}_{\mathcal{S}} (\mathcal{P}_i f - P_o \hat{f}), \tag{5.5}$$

where $\mathsf{M}_{\mathcal{S}}$ is the Galerkin mass matrix of the common subdivision, and $P_o, \mathcal{P}_i$ are interpolation matrices mapping functions on the original and intrinsic triangulations to $\mathcal{S}$ *resp.*. In particular, P_o is a $|V| \times |V^{\mathcal{S}}|$ matrix where each row corresponds to a vertex of $\mathcal{S}$ and has that vertex's barycentric coordinates on the original mesh as entries, and likewise for $\mathcal{P}_i$. We can now find the function $\hat{f}$ which is L^2-closest to the intrinsic function f simply by minimizing Equation 5.5, which amounts to solving a linear least-squares system.

The following example illustrates the benefits of this technique [GSC21a]. Consider a low-quality mesh of the unit square generated via random edge splits. We solve the Poisson equation $\Delta f = \sin(\pi x)\sin(\pi y)$ on



the input mesh, and its intrinsic Delaunay triangulation, and then transfer the intrinsic solution back to the original mesh by copying values, and using the L^2 projection described above. This yields 3 different solutions on the original mesh. We then compute the error of each solution, represented in the basis of the original mesh, against an analytic ground truth and plot the average error after 100 trials. Copying values back provides much more accuracy than solving directly on the original mesh, and picking the L^2-closest function results in even better solution.

Transferring tangent vectors Sometimes, our quantity of interest is a tangent vector field rather than a mere scalar function. In this case, even copying the solution back at vertices becomes complicated, as tangent vectors on the intrinsic and original meshes are represented in different bases **TODO: have we talked about tangent spaces before?**. Copying tangent vectors is easiest to do using the signpost data structure, since signposts directly encode the mapping between these bases **TODO: Explain how to do this?**. However, one could in principle compute this mapping using the explicit overlay or integer encodings.

Chapter 6

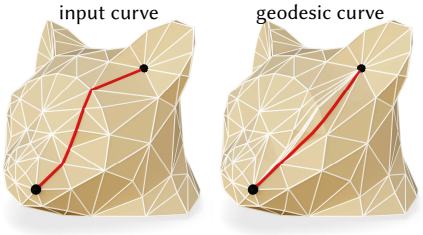
Geodesics

Geodesic curves generalize the notion of a straight line to curved surfaces; they can be defined formally as locally-shortest paths, or curves of zero tangential acceleration. Fast and accurate geodesics are essential for tasks across science and engineering [Bos+11], and the ability to construct "straight lines" on polyhedral surfaces helps generalize classic algorithms from 2D computational geometry to curved surfaces. Here, we will consider *exact, polyhedral* geodesics, which are exactly geodesic along the piecewise-flat geometry, as opposed to some approximate or smoothed notion of geodesic.

Computationally, geodesics are most widely studied in the context of finding *globally shortest* geodesics from a source point; such paths are useful for defining distance along a surface—the *geodesic distance* between two points is the length of the shortest path along the surface between. Algorithms in this vein have roots in the strategy of Mitchell, Mount, and Papadimitriou [MMP87]: start at the source, and use a Dijkstra-like traversal to track "windows" of geodesic paths sharing a common history. Many improvements, generalizations, and approximations have since been developed along this line of research [KS98; Sur+05; BK07; XW09; CWW13; YWH13; Xu+15; YXH14; Qin+16; Wan+17; Yin+19; AFH20; Cao+20].

However, in geometry processing we will often also need to consider geodesic paths which are not necessarily globally-shortest geodesics. For instance, every edge of an intrinsic triangulation is a geodesic path, but not necessarily the *shortest* geodesic path between the endpoints. Though less widely-studied than the geodesic distance problem, there is also a wide variety of existing algorithms for constructing and manipulating these more general general geodesic curves. One important approach is to shorten a *given* curve to be a geodesic, akin to a curve shortening flow on the surface [Gag90]. Such procedures can construct a larger space of geodesics beyond merely shortest geodesics including even closed loops, and can preserve the topological class of curves—both of which are critical for tasks like modeling and fabrication. *Lagrangian* approaches to curve shortening represent curves via vertices that can move freely along the surface or in $\mathbb{R}^3$ [HS94; MVC05; XW07; ASS09; XHF11; Han+17; Liu+17a; RŠN19], whereas *Eulerian* methods encode curves as level sets of a scalar function [Set89; WT09; Zha+10].

Given the variety of complicated algorithms which have been developed to construct geodesic curves, it is somewhat remarkable that they emerge automatically as the

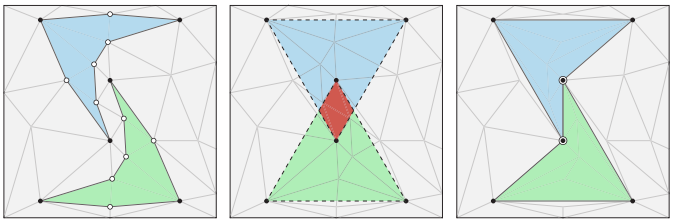


edges of an intrinsic triangulation, and can be manipulated by simple edge flips. In fact, Sharp and Crane [SC20b] shows that a greedy edge flipping strategy can be used to intentionally introduce particular edges in an intrinsic triangulation; providing a simple and efficient scheme for constructing geodesic paths (see inset). The resulting strategy is quite simple, and more interestingly represents a totally different approach to finding geodesics, compared to the window-based and unfolding-based approaches prevalent in past work. By building a method for computing geodesics within the framework of intrinsic triangulations, we benefit from all of the machinery in the preceding sections, such as the highly robust integer-based representation (Section 4.4). The remainder of this section will outline the intrinsic flip-based geodesic algorithm of Sharp and Crane [SC20b], and show how it can be applied to problems in geometry processing.

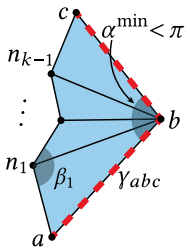
6.1 Geodesics from intrinsic edge flips

The edges in an intrinsic triangulation are always geodesics along the surface (Section 3); the basic strategy is then to construct desired geodesic paths with a simple greedy edge flipping policy which intentionally introduce edges of interest in an intrinsic triangulation. In particular, we will take as input some path γ along the edges of an intrinsic triangulation $\mathcal{T}$ (or more generally a loop or even a network of paths and loops). As output we will produce a geodesic path γ' which is isotopic to the input, along the edges of an updated intrinsic triangulation $\mathcal{T}'$. As the algorithm proceeds, the curve will always be represented as a sequence of edges in an intrinsic triangulation, akin to a path in the graph theory sense. Finally, these paths can be extracted as explicit polylines along the surface as a post-process, using the data structures described in Section 4.

Noncrossing curves. This algorithm operates in the space of noncrossing curves: the input curves must not cross transversely, and it is guaranteed that the output will not contain any new crossings. Noncrossing curves arise frequently in geometry processing algorithms, such as cuts or seams made to parameterize a shape with low distortion [CZ18; LDB17; SC18], and as the boundaries of regions or segmentations along a shape (see *e.g.* [CGF09]). Straightening such curves to be geodesic is important to enable manufacturing in computational fabrication, or satisfy smoothness objectives in optimization. Flip-based geodesic computation is particularly suited for straighting such curves precisely because it guarantees to preserve the noncrossing property—otherwise, if one were to create a crossing while making the curves geodesic (inset), it would render the curves useless for the corresponding applications. In our algorithms, the noncrossing property is captured by the notion of a *flexible joint*, essentially a local region



of the path which can be straightened without introducing a crossing; see Sharp and Crane [SC20b, Section 3.2] for a formal definition.



Constructing geodesics with edge flips. The key algorithmic building block is the FLIPOUT subroutine, so-named because it flips edges out of a neighborhood to provably introduce a shorter path (see inset diagram for notation). At a vertex i where the path is not yet a geodesic, FLIPOUT simply repeatedly flips any edge outgoing from i which can be flipped (see Section 3.3, essentially any edge contained in a convex diamond). In Sharp and Crane [SC20b, Theorem 4.1], this subroutine is proven to always terminate with a shorter path along the perimeter. The essence of the proof is that each flip removes an edge from the neighborhood of i, until the only edges left form a convex curve along the perimeter. The convex perimeter curve is guaranteed to be shorter than the initial path as a corollary of *Crofton's formula* [Cro68]: for two nested convex curves sharing endpoints, the inner one is shorter. The formal proof for the general case of Δ-complex (Sharp and Crane [SC20b, Appendix A]) is nontrivial, but necessary because the triangulation may be reduced to a Δ-complex at intermediate stages of the algorithm.

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 $\mathcal{T}$.

1: **while** any $\beta_i < \pi$ **do**
2: $\quad j \leftarrow \min i$ s.t. $\beta_i < \pi$ ▷*choose the first edge with* $\beta_i < \pi$
3: $\quad$ FLIPEDGE$(\mathcal{T}, bn_j)$
4: **end while**
5: $\gamma_{\text{shorter}} \leftarrow (a, n_1, \ldots, n_{k-1}, c)$ ▷*path along the outer arc*
6: **return** $\mathcal{T}, \gamma_{\text{shorter}}$

This procedure is easiest to conceptualize in a planar triangulation (as shown in Figure 6.1). However, the algorithm (as well as its proof of correctness) depends only on measuring triangle corner angles and flipping edges—both of which are perfectly well-defined on intrinsic triangulations. Thus we can use this procedure to shorten paths, and ultimately construct

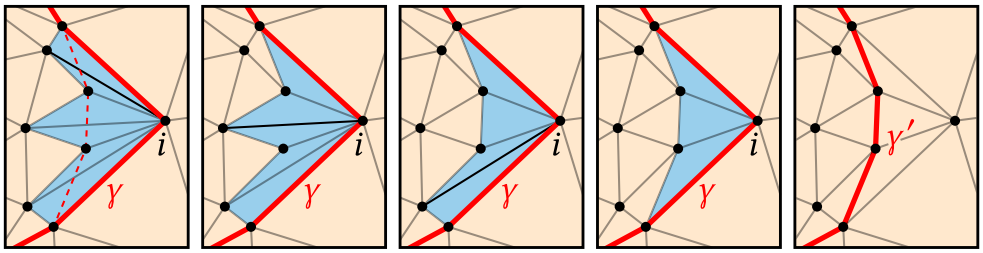


Figure 6.1: FLIPOUT shortens the curve γ by repeatedly flipping edges to introduce a shorter path.

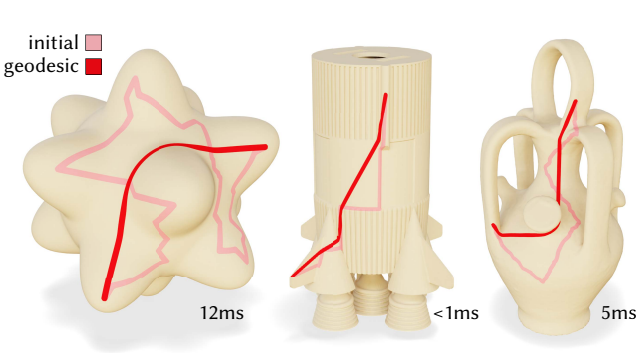


Figure 6.2: Basic results shortening an initial path to a geodesic by flipping edges. Inset values give the runtimes. All of the resulting curves shown are exact polyhedral geodesics.

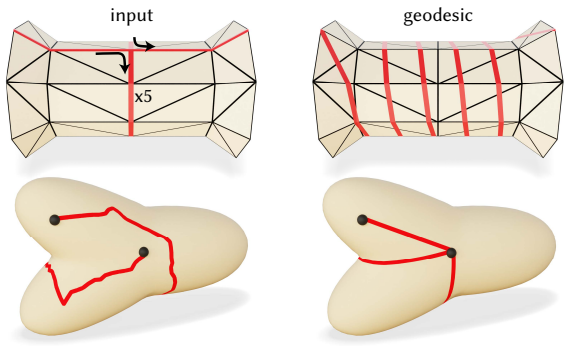


Figure 6.3: A careful representation of paths along handles enables finding geodesics that overlap many times (*top*), or those that get pulled tight around endpoints of the path itself (*bottom*).

geodesics, along curved surfaces.

Geodesic paths. Given an edge path $\gamma_{x\leftrightarrow y}$ between vertices x and y, we can iteratively apply FLIPOUT, shortening the path on each iteration, to obtain an exact polyhedral geodesic in finitely many steps (Figure 6.2). Algorithm 2 describes the procedure, while Theorem 1 argues that it must terminate after a finite number of steps. 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 curve corner angle 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 $\mathcal{T}$.

```
while γ_{x↔y} is not geodesic do
    γ_{abc} ← flexible joint in γ_{x↔y} with smallest angle α_{abc}
    T, γ_shorter ← FLIPOUT(T, γ_{abc})          ▷locally shorten
    γ_{x↔y} ← UPDATEPATH(γ_{x↔y}, γ_{abc}, γ_shorter)   ▷update
end while
return T, γ_{x↔y}
```

Theorem 1. *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 $|\gamma_{t+1}| < |\gamma_t|$, 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 [Ind+01, 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. □

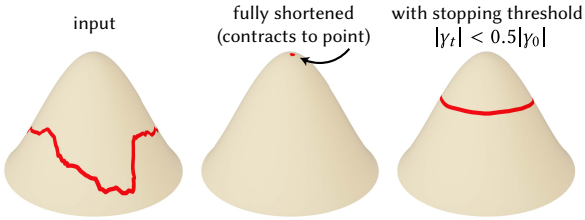


Figure 6.4: Algorithm 2 acts as discrete curve-shortening flow; stopping the procedure early via a length or angle threshold generates straighter curves, without drifting too far from the initialization or contracting to a point.

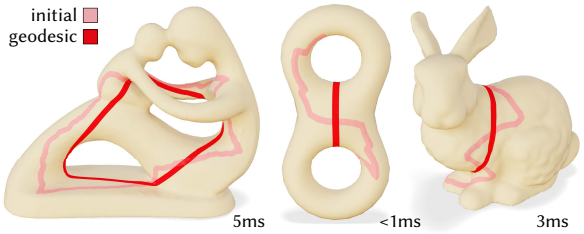


Figure 6.5: Geodesic loops generated by with edge flips in an intrinsic triangulations. Inset values give the runtimes.

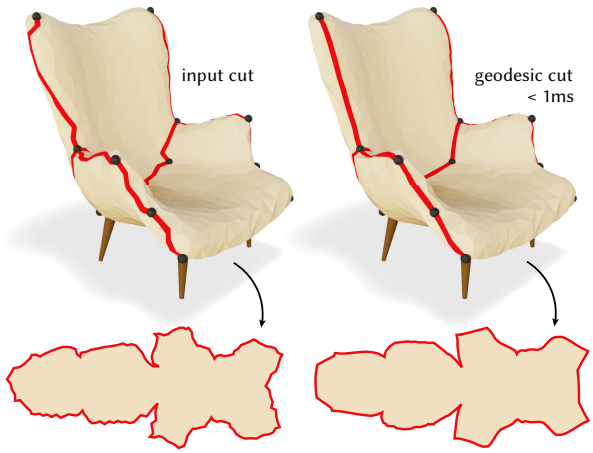


Figure 6.6: Curve networks arise when cutting and flattening a shape for computational fabrication. Our method is perfectly suited to straighten an initial cut network along edges (*left*) to a geodesic network (*right*), yielding a much more natural pattern for fabrication (*bottom*).

Rather than running to completion, one can also terminate Algorithm 2 when there is a sufficient length decrease. Doing so generates curves which are shorter/straighter but not yet geodesic, like a curve-shortening flow (Figure 6.4).

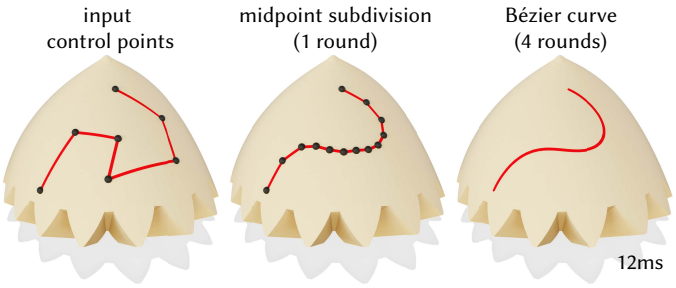
The only other machinery needed for this algorithm is a data structure to encode paths along the edges of a mesh. For simple curve between two points, it is generally sufficient to flag edges which make up the path, but to support any more complicated configurations which may arise (Figure 6.3), we describe a data structure which encodes an ordered stack of path segments along each edge of a mesh—see Sharp and Crane [SC20b, Section 3.2] for full details.

6.2 Geodesic loops and curve networks

The iterative shortening procedure in Algorithm 2 can also be applied to closed loops, or even networks of paths and loops along the surface. For the closed loops, a small extra step of the algorithm is needed for cases where the loop consists of a single edge, and it should be noted that the proof of termination does not handle this case, but we always observe termination in practice—see Sharp and Crane [SC20b, Appendix B] for details. Constructing geodesic

loops (Figure 6.5) is particularly interesting, because algorithms for finding shortest geodesics are fundamentally unable to construct them, as no initial point the loop is known *a priori*. Loops and more general curve networks arise frequently in geometry processing, *e.g.* as cut graphs for parameterization and fabrication (Figure 6.6), or as the boundaries of regions on a surface.

6.3 Geodesic Bézier curves



Morera, Carvalho, and Velho [MCV08] 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 the inset figure, we inherit all the benefits of the flip-based scheme, and can construct intrinsic triangulations with edge paths approximating Bézier curves.

6.4 Constrained intrinsic retriangulation

A key benefit of constructing geodesic curves via edge flips is that the procedure yields not just the curves, but also a triangulation of the domain which conforms to those curves. This allows the scheme to serve a much-needed role for generating constrained intrinsic triangulation of surfaces, akin to constrained Delaunay triangulation in the plane [Che89]. Such triangulations are a necessary ingredient for many applications in PDE-based geometry processing, where solutions must conform to some boundary conditions defined along the surface. Figure 6.8 demonstrates two practical applications of this technique, showcasing a cross field conforming to curves on a 3D scan of a pelvis [Knö+13], and a Poisson equation taking boundary conditions from a Bézier curve on a mechanical part.

6.5 Single-source geodesics

The methods described above are a kind of curve-shortening flow: they take some particular curve as a input, and shorten it to be a geodesic. It is natural to wonder whether the same techniques can be applied to the widely-studied single source all destination (SSAD) problem, where one seeks a geodesic path to all other vertices in a mesh.

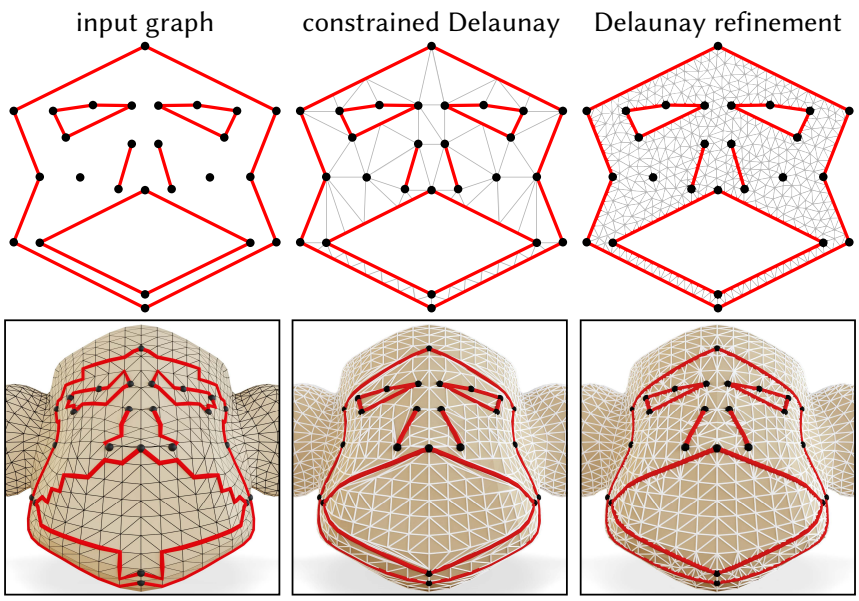


Figure 6.7: In the plane, a *constrained Delaunay triangulation (CDT) (top center)* contains a given set of input segments *(top left)*; CDTs with good angle bounds *(top right)* are critical for, *e.g.*, numerical simulation. The intrinsic triangulations produced by our geodesic straightening procedure extend CDTs to surface meshes *(bottom row)*.

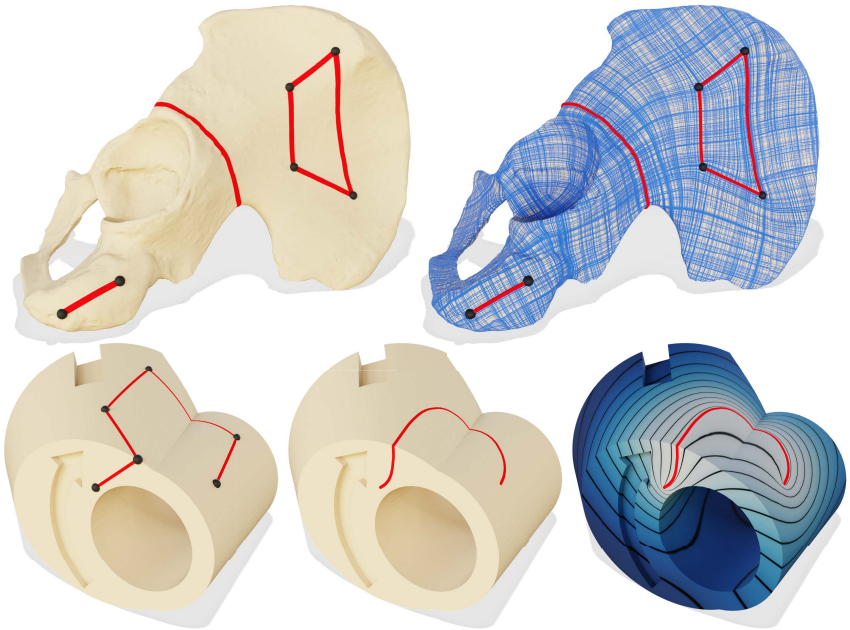
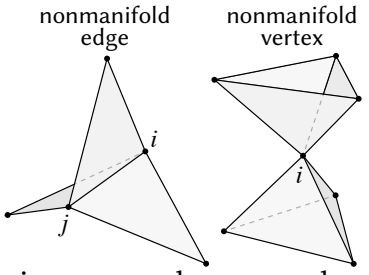


Figure 6.8: PDEs taking boundary conditions from constrained intrinsic triangulations. *Top*, a cross field conforming to curves on a 3D scan of a pelvis [Knö+13], and *bottom*, a Poisson equation with boundary conditions defined along a Bézier curve on a mechanical part.

Sharp and Crane [SC20b] also present simple algorithm for this task based on the FLIPOUT subroutine, the basic idea of which is to run a Dijkstra search outward from the source vertex, while constantly applying the FLIPOUT subroutine at the frontier to ensure all accepted paths are geodesic. Of course, this procedure is only guaranteed to yield geodesics, not necessarily globally-shortest geodesics, but in practice it is quite efficient and very often does find shortest geodesics (see inset). Perhaps the most interesting consequence of this algorithm is that it constructively implies the existence of a single triangulation of a surface, which contains among its edge set geodesics from a particular source vertex to every other vertex in the mesh—it is remarkable that such a triangulation exists at all.

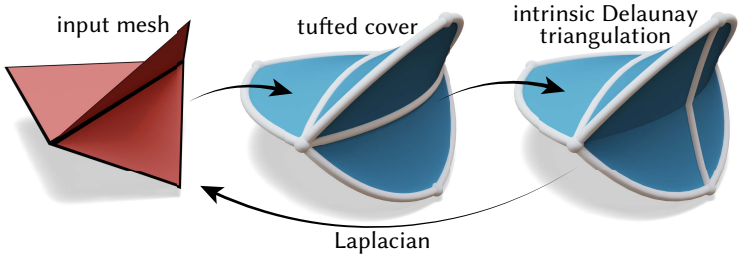
Chapter 7

Generalized domains



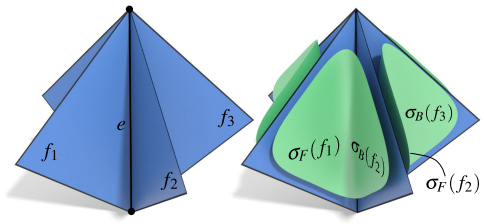
Traditionally, the formulation of intrinsic triangulations begins with a strong assumption about the connectivity of the input domain: a manifold, oriented triangle mesh (Section **??**). *Manifold* connectivity requires local patches of the surface be homeomorphic to $\mathbb{R}^2$, that is, any neighborhood of the surface looks like the plane; examples of nonmanifold edges and vertices are shown inset. However, geometric data encountered in practice often has nonmanifold features, either intentionally or due to noise, or may have even less structure in the form of a point cloud—handling such inputs is crucial to leverage intrinsic triangulations for robust geometry processing. This section introduces a simple technique by Sharp and Crane [SC20a], which constructs a covering space that enables the construction of an intrinsic Delaunay triangulation all triangle meshes, even those which may be nonmanifold and nonorientable. With this same technique, we can also generate intrinsic Delaunay triangulations of point clouds which have no connectivity at all (Section 7.2).

7.1 Nonmanifold intrinsic triangulations



The tufted cover. To construct intrinsic Delaunay triangulations of nonmanifold meshes, we will first build a covering of the triangulation which we call the *tufted cover*; it has the same vertex set as the input triangulation but twice as many faces, glued together along their edges according simple strategy. Then, we can construct the intrinsic Delaunay triangulation of the tufted cover, and *e.g.* use its Laplace matrix as a high-quality Laplacian for the original mesh. The key observation behind this strategy is that only nonmanifold *edges* obstruct the usage of intrinsic triangulations, because there is no notion of an edge flip at a nonmanifold edge; nonmanifold *vertices* are a non-issue. For this reason, the tufted cover is built such that all edges are manifold. The intrinsic triangulation

of this covering space then supports edge flip operations, which can be used *e.g.* to construct the high-quality intrinsic Delaunay Laplacian, and use it as the Laplace matrix for the original nonmanifold mesh.



The tufted cover. In particular, the tufted cover is defined by splitting each face f in to two copies $\sigma_F(f)$ and $\sigma_B(f)$, then cyclically gluing together consecutive faces around each original edge (see inset). By construction, the tufted cover will always be an edge manifold, closed, and oriented triangulation. The resulting triangulation will have twice as many faces, but the exact same vertex set as the original mesh, which makes it trivial to transfer functions and operators defined at vertices between the tufted cover and the original mesh. The name "tufted cover" arises because the nonmanifold vertices are reminiscent of the buttons on tufted upholstery. Note that, purely for visualization, we "inflate" the cover outward to clearly distinguish front and back faces, but the actual geometry of each triangle remains flat.

More precisely, the tufted cover of an input mesh $M = (V, E, F)$ is a triangle mesh $\tilde{M} = (\tilde{V}, \tilde{E}, \tilde{F})$ with the same vertices ($\tilde{V} = V$), together with a *gluing map* $\tilde{G}$. Recall from Section **??** that a simple face-vertex list is often not sufficient to specify the connectivity of a triangulation; in this section we will additionally make use of the gluing map $\tilde{G}$ to precisely specify connectivity while building the tufted cover. The gluing map specifies, for each side of each triangle, the side of some other triangle to which it is glued. A side is encoded as a pair (f, s) of a face f and a side within that face s.

For each face $f \in F$, $\tilde{T}$ has two oppositely oriented copies $\sigma_F(f), \sigma_B(f) \in \tilde{F}$ which one can think of as the "front" and the "back" of f, respectively. Nonmanifold edges are resolved by the way we define the gluing map $\tilde{G}$. We first list the faces around each edge $e \in E$ in a circular order $\rho^e := (f_1, \ldots, f_k)$; this ordering can be chosen as the angular order of the faces around the edge, though any choice of ordering will suffice ([SC20a, Section 5.4]). If we imagine that these faces are consistently orientated relative to e, then we just glue them "front to back" along the shared edge, *i.e.*, we glue $\sigma_F(f_i)$ to $\sigma_B(f_{i+1 \bmod k})$ for $i = 1, \ldots, k$ (the inset figure gives an example). A more precise description of the gluing procedure which takes orientation into account is given in Algorithm 3; here SIDE(e, f) just gives the side index of e within face f (1, 2, or 3).

Algorithm 3 CONSTRUCTTUFTEDCOVER(M, ρ)

Input: A (possibly nonmanifold) triangle mesh M and an ordering ρ of faces around each edge.

Output: The tufted cover mesh $\tilde{M}$ and edge glue map $\tilde{G}$

1: $\tilde{F} \leftarrow \bigcup_{ijk \in F} \{ijk, jik\}$ ▷*two copies of each face*
2: $\tilde{G} \leftarrow \{\}$ ▷*assemble an edge glue map*
3: **for each** edge $e \in E$ **do**
4: **if** e and $\sigma_F(\rho^e_1)$ have the same orientation **then**
5: $f \leftarrow \sigma_F(\rho^e_1)$

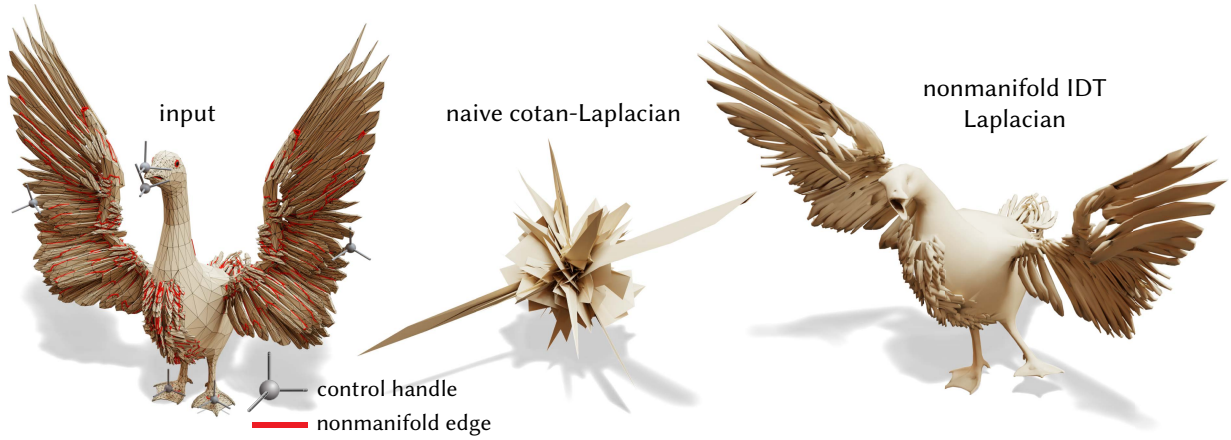


Figure 7.1: Robustness in the context of differential surface editing [Yu+04; Lip+04; Sor+04], where a system of equations involving a Laplacian is solved to deform a 3D model. Applying these techniques naively in the input mesh, which is nonmanifold and has many low-quality triangles, yields only numerical noise. Substituting the nonmanifold IDT Laplacian generates the expected smooth deformation.

```
    else f ← σ_B(ρ_1^e)
    for i = 1, ..., k do                                   ▷letting k := |ρ_e|
        g_1 ← σ_F(ρ^e_{i+1 mod k})
        g_2 ← σ_B(ρ^e_{i+1 mod k})
        if f and g_1 have different orientation along e then
            SWAP(g_1, g_2)
        G̃(f, SIDE(e, f)) ← (g_1, SIDE(e, g_1))
        G̃(g_1, SIDE(e, g_1)) ← (f, SIDE(e, f))
        f ← g_2
return M̃, G̃
```

It should be emphasized that the tufted cover still lacks vertex-manifold connectivity; in fact the it is nonmanifold at nearly every vertex. We have constructed a triangulation that has exactly the right structure to apply intrinsic edge flips, but have not otherwise rectified the nonmanifoldness.

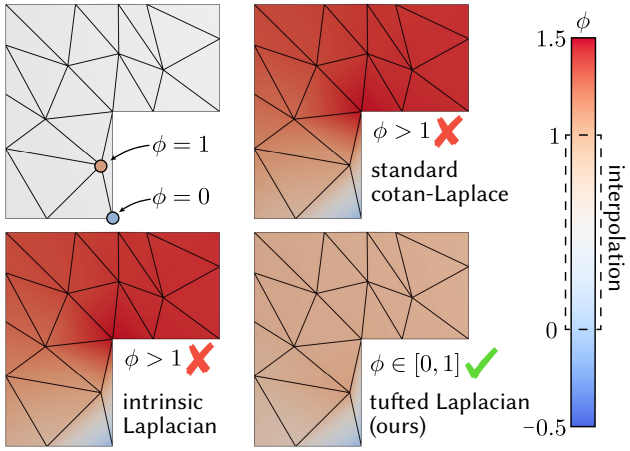
We construct the intrinsic Delaunay Laplacian on the tufted cover exactly as described in Section 5.2, dividing the matrix by a factor of 2, because the tufted cover is a double cover of the original mesh. This construction offers a high-quality cotan-Laplace matrix for nonmanifold meshes for the first time, extending the benefits to *all* triangle meshes without restrictions on connectivity. Substituting this operator in to existing algorithms immediately improves there performance and robustness on low-quality input meshes (Figure 7.1).

Beyond intrinsic Delaunay. The work of Sharp and Crane [SC20a] focuses entirely on building the nonmanifold intrinsic Delaunay Laplace matrix, which is the most basic use of intrinsic triangulations; for instance it does not demand any of the higher-order data structures

like signposts and normal coordinates described in Section 5. It is reasonable to wonder whether the other techniques in this text can be likewise generalized to the nonmanfiold intrinsic setting. The answer is generally yes, though care must be taken about their meaning on the tufted cover, which is not vertex manifold and furthermore is a double cover of the original surface.

For instance, the integer-based representation of Section 4.4 can be used without modification, but signposts (Section 4.3) require vertex tangent spaces, which are not well-defined without vertex manifoldness—a remedy is to split each vertex of the tufted cover in to multiple copies, corresponding to each tangent neighborhood. Intrinsic Delaunay refinement (Section 5.3) is well-defined on the tufted cover, with the caveat that new vertices should be inserted as new "tufted" vertices, rather than separately on each sheet of the cover. Edge flip geodesics (Section 6) can likewise be applied to paths along a sheet of the tufted cover, though as the algorithm progresses the path will stay along the initial sheet of the cover, which may or may not be the expected interpretation.

Domains with boundary. The tufted intrinsic Delaunay Laplacian has useful properties not only for nonmanifold meshes, but for any mesh with boundary. Both cotan-Laplace and the intrinsic Laplacian will have negative edge weights for a boundary edge ij opposite an obtuse angle θ_k^{ij}. Delaunay flips are no help here, since boundary edges cannot be flipped. Such weights pose no problem when Dirichlet boundary conditions are enforced along the entire domain boundary (since boundary weights do not enter into the equation), but can be problematic for interpolating other sets of pinned values. In contrast, since our tufted cover is always closed and Delaunay, *all* weights of our Laplacian are nonnegative—even for edges incident on the boundary. Hence, harmonically interpolating *any* set of pinned values will yield a function bounded within the range of these values. In the inset figure we show an example where two known values at vertices are harmonically interpolated across a shape; only the tufted intrinsic Delaunay Laplacian gives a guarantee that the interpolated values stay within the range of the inputs. (We note that harmonic interpolation by pinned values at vertices is not a well-posed convergent discretization, but nonetheless it is common and pragmatic.) This property provides additional robustness for algorithms built on top of interpolated weights, Green's functions, *etc.*



7.2 Point clouds

Point clouds, simply a collection of points $p \in \mathbb{R}^3$, are a common alternative to meshes for representing surfaces in geometry processing, though the absence of connectivity makes many

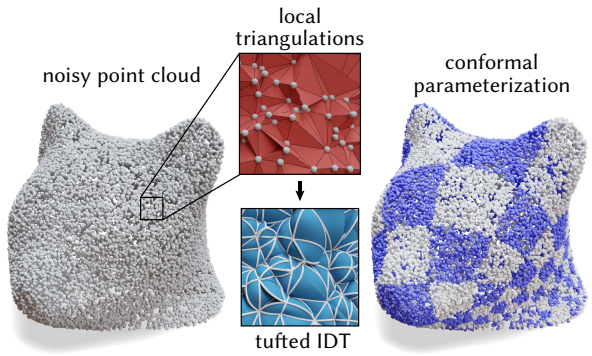


Figure 7.2: The tufted intrinsic Delaunay point cloud Laplacian, demonstrated here for spectral conformal parameterization of a point cloud [Mul+08].

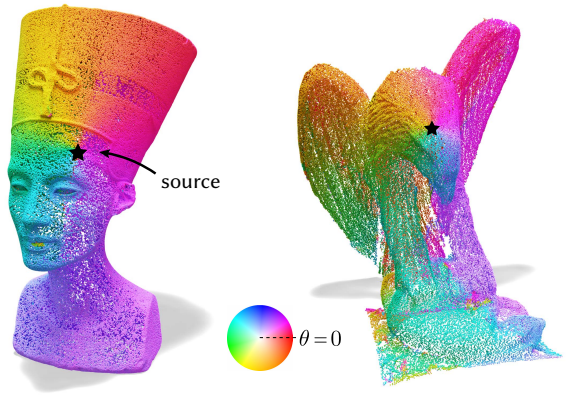
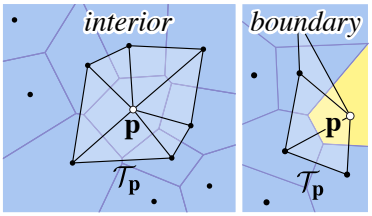


Figure 7.3: Angular parameterizations of point clouds, computed with the tufted intrinsic Delaunay point cloud Laplacian via the vector heat method [SSC19b].

computations more difficult. We can use the tufted cover to construct a high-quality Laplace matrix for points clouds, which inherits all of the benefits of intrinsic triangulations. The basic idea is to union together many local triangulations among each point and its nearest neighbors, which represents the surface well but creates many nonmanifold edges, repeated triangles, *etc.*, then build the intrinsic Delaunay triangulation of the tufted cover of the union, which works because the tufted cover places no constraints on connectivity.

More precisely, a common strategy for building a Laplacian on a point cloud $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^3$ is to (i) identify the k nearest neighbors of each point p, (ii) project these neighbors onto an estimated tangent plane, and (iii) construct the planar Delaunay triangulation $\mathcal{T}_p$ of the projected points These local triangulations are then used to build a Laplacian in a variety of ways (see Sharp and Crane [SC20a]). For instance, both Belkin *et al.* [BSW08] and Liu *et al.* [LPG12] use the triangulations purely to determine the mass matrix M; edge weights are determined via a Gaussian function of the distance in $\mathbb{R}^3$ (and are hence always positive). Such schemes are accurate and have nice theoretical properties (*e.g.*, pointwise convergence for fairly uniform point distributions) but involve numerical parameters which are difficult to estimate, and matrices that are quite dense. Other schemes use the local triangulations only to determine connectivity; the original point locations are still used to accumulate cotan weights [CRT04; Cao+10]. A significant benefit of this approach is that (like mesh-based Laplacians) it accurately handles *nonuniform* point distributions, while still retaining a high degree of sparsity. However, since edges may not satisfy the local Delaunay property, the resulting Laplacian can have negative edge weights.

Sharp and Crane [SC20a] observe that nonmanifold intrinsic Delaunay triangulations can be directly applied in this setting. Like past approaches, the point cloud Laplacian is constructed from from local triangulations, but rather than accumulating weights independently, we form the union $\mathcal{T} = \bigcup_{p \in P} \mathcal{T}_p$ of triangles containing p in all local triangulations $\mathcal{T}_p$; points contained



in a noncompact cell of the local Voronoi diagram can be tagged
as boundary vertices (see inset). The resulting global triangulation $\mathcal{T}$ has highly irregular connectivity, is nonmanifold almost everywhere, and has duplicate copies of many faces. However, we can simply proceed as before: build the tufted cover, flip to an intrinsic Delaunay triangulation, and read off the corresponding Laplace matrix L—which can then be used directly on the original point cloud P. (We also multiply L by $1/3$, since triangles triply-cover the local neighborhoods.) This Laplacian exhibits all the desired properties (symmetry, positive-definiteness, nonnegative edge weights, *etc.*) while remaining very sparse; unlike schemes based on Gaussian weights, there are no parameters to estimate or tune.

Equipped with a high-quality point cloud Laplacian, we can easily translate many algorithms designed for meshes to the point cloud setting. In Figure 7.2 we use this Laplacian to generate a conformal parameterization of a noisy point cloud, adapting the mesh-based method of [Mul+08], and in Figure 7.3 shows a more sophisticated algorithm run directly on point clouds: parameterization by the logarithmic map, computed via the *vector heat method* [SSC19b, Section 8.2].

Chapter 8

Conclusion

References

[AFH20] Yohanes Yudhi Adikusuma, Zheng Fang, and Ying He. "Fast Construction of Discrete Geodesic Graphs". *ACM Transactions on Graphics (TOG)* 39.2 (2020).

[All+08] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. "Recent Advances in Remeshing of Surfaces". *Shape analysis and structuring* (2008).

[ASS09] Eli Appleboim, Emil Saucan, and Jonathan Stern. *Normal Approximations of Geodesics on Smooth Triangulated Surfaces*. Tech. rep. CCIT Report, 2009.

[Bar+18] Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. "Fast Winding Numbers for Soups and Clouds". *ACM Transactions on Graphics (TOG)* 37.4 (2018).

[Bau75] Bruce G Baumgart. "A Polyhedron Representation for Computer Vision". *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*. 1975.

[BI08] Alexander I Bobenko and Ivan Izmestiev. "Alexandrov's Theorem, Weighted Delaunay Triangulations, and Mixed Volumes". *Annales de l'institut Fourier*. Vol. 58. 2008.

[BK07] David Bommes and Leif Kobbelt. "Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes". *VMV*. Vol. 7. 2007.

[Bos+11] Prosenjit Bose, Anil Maheshwari, Chang Shu, and Stefanie Wuhrer. "A Survey of Geodesic Paths on 3D Surfaces". *Computational Geometry* 44.9 (2011).

[Bot+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. 2010.

[BS07] Alexander Bobenko and Boris Springborn. "A Discrete Laplace–Beltrami Operator for Simplicial Surfaces". *Discrete & Computational Geometry* 38.4 (2007).

[BSW08] Mikhail Belkin, Jian Sun, and Yusu Wang. "Discrete Laplace Operator on Meshed Surfaces". *Proceedings of the Twenty-Fourth Annual Symposium on Computational Geometry*. 2008.

[Cao+10] Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. "Point Cloud Skeletons via Laplacian Based Contraction". *Shape Modeling International Conference*. 2010.

[Cao+20] Luming Cao, Junhao Zhao, Jian Xu, Shuangmin Chen, Guozhu Liu, Shiqing Xin, Yuanfeng Zhou, and Ying He. "Computing Smooth Quasi-Geodesic Distance Field (QGDF) with Quadratic Programming". *Computer-Aided Design* 127 (2020).

[CDS12] Siu-Wing Cheng, Tamal K Dey, and Jonathan Shewchuk. *Delaunay Mesh Generation*. 2012.

[CGF09] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. "A Benchmark for 3D Mesh Segmentation". *ACM Transactions on Graphics (TOG)* 28.3 (2009).

[CH11] Long Chen and Michael Holst. "Efficient Mesh Optimization Schemes Based on Optimal Delaunay Triangulations". *Computer Methods in Applied Mechanics and Engineering* 200.9-12 (2011).

[Che+10] Renjie Chen, Yin Xu, Craig Gotsman, and Ligang Liu. "A Spectral Characterization of the Delaunay Triangulation". *Computer aided geometric design* 27.4 (2010).

[Che87] L. P. Chew. "Constrained Delaunay Triangulations". *Proceedings of the Third Annual Symposium on Computational Geometry*. SCG '87. 1987. ISBN: 0-89791-231-4.

[Che89] L. Paul Chew. "Constrained Delaunay Triangulations". *Algorithmica* 4.1-4 (1989).

[Che93] Chew Chew L. Paul. "Guaranteed-Quality Mesh Generation for Curved Surfaces". *Proceedings of the Ninth Annual Symposium on Computational Geometry*. SCG '93. 1993. ISBN: 0-89791-582-8.

[Cra+13] Keenan Crane, Fernando de Goes, Mathieu Desbrun, and Peter Schröder. "Digital Geometry Processing with Discrete Exterior Calculus". *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. New York, NY, USA, 2013.

[Cro68] Morgan William Crofton. "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).

[CRT04] Ulrich Clarenz, Martin Rumpf, and Alexandru Telea. "Finite Elements on Point Based Surfaces". *SPBG*. 2004.

[CWW13] Krane Crane, Clarisse Weischedel, and Max Wardetzky. "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow". *ACM Transactions on Graphics (TOG)* 32.5 (2013).

[CX04] Long Chen and Jin-chao Xu. "Optimal Delaunay Triangulations". *Journal of Computational Mathematics* (2004).

[CZ18] Sebastien J. P. Callens and Amir A. Zadpoor. "From Flat Sheets to Curved Geometries: Origami and Kirigami Approaches". *Materials Today* 21.3 (2018).

[Duf59] Richard J Duffin. "Distributed and Lumped Networks". *J. Math. Mech.* (1959).

[Dzi88] Gerhard Dziuk. "Finite Elements for the Beltrami Operator on Arbitrary Surfaces". *Partial Differential Equations and Calculus of Variations*. 1988.

[Fis+07] Matthew Fisher, Boris Springborn, Peter Schröder, and Alexander Bobenko. "An Algorithm for the Construction of Intrinsic Delaunay Triangulations with Applications to Digital Geometry Processing". *Computing* 81.2 (2007).

[Gag90] Michael E Gage. "Curve Shortening on Surfaces". en. *Annales scientifiques de l'École Normale Supérieure* Ser. 4, 23.2 (1990).

[GS69] K Ruben Gabriel and Robert R Sokal. "A New Statistical Approach to Geographic Variation Analysis". *Systematic zoology* 18.3 (1969).

[GSC21a] Mark Gillespie, Nicholas Sharp, and Keenan Crane. “Integer Coordinates for Intrinsic Geometry Processing”. *arXiv preprint arXiv:2106.00220* (2021). arXiv: 2106.00220.

[GSC21b] Mark Gillespie, Boris Springborn, and Keenan Crane. “Discrete Conformal Equivalence of Polyhedral Surfaces”. *ACM Transactions on Graphics (TOG)* 40.4 (2021).

[Hak61] Wolfgang Haken. “Theorie Der Normalflächen”. *Acta Mathematica* 105.3-4 (1961).

[Han+17] Xiaoguang Han, Hongchuan Yu, Yizhou Yu, and Jianjun Zhang. “A Fast Propagation Scheme for Approximate Geodesic Paths”. *Graphical Models* 91 (2017).

[Hat02] Allen Hatcher. *Algebraic Topology*. Algebraic Topology. 2002. ISBN: 978-0-521-79540-1.

[HS94] Joel Hass and Peter Scott. “Shortening Curves on Surfaces”. *Topology* 33.1 (1994).

[Hua+08] Qi-Xing Huang, Bart Adams, Martin Wicke, and Leonidas J Guibas. “Non-Rigid Registration under Isometric Deformations”. *Computer Graphics Forum*. Vol. 27. 2008.

[IGG01] Martin Isenburg, Stefan Gumhold, and Craig Gotsman. “Connectivity Shapes”. *Proceedings Visualization, 2001. VIS’01.* 2001.

[Ind+01] Claude Indermitte, Thomas M. Liebling, Marc Troyanov, and Heinz Clémençon. “Voronoi Diagrams on Piecewise Flat Surfaces and an Application to Biological Growth”. *Theoretical Computer Science* 263 (2001).

[JKS13] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. “Robust Inside-Outside Segmentation Using Generalized Winding Numbers”. *ACM Transactions on Graphics (TOG)* 32.4 (2013).

[Ket99] Lutz Kettner. “Using Generic Programming for Designing a Data Structure for Polyhedral Surfaces”. *Computational Geometry* 13.1 (1999).

[Kne29] Hellmuth Kneser. “Geschlossene Flächen in Dreidimensionalen Mannigfaltigkeiten.” *Jahresbericht der Deutschen Mathematiker-Vereinigung* 38 (1929).

[Knö+13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. “Globally Optimal Direction Fields”. *ACM Transactions on Graphics (TOG)* 32.4 (2013).

[KS16] Marc Khoury and Jonathan Richard Shewchuk. “Fixed Points of the Restricted Delaunay Triangulation Operator”. *32nd International Symposium on Computational Geometry (SoCG 2016)*. 2016.

[KS98] Ron Kimmel and James A Sethian. “Fast Marching Methods on Triangulated Domains”. *Proceedings of the National Academy of Sciences* 95 (1998).

[KSS06] Liliya Kharevych, Boris Springborn, and Peter Schröder. “Discrete Conformal Mappings via Circle Patterns”. *ACM Transactions on Graphics (TOG)* 25.2 (2006).

[Law72a] Charles L Lawson. “Generation of a Triangular Grid with Applications to Contour Plotting”. *Jet Propulsion Laboratory Technical Memo* 299 (1972).

[Law72b] Charles L Lawson. “Transforming Triangulations”. *Discrete mathematics* 3.4 (1972).

[Law77] Charles L Lawson. “Software for C1 Surface Interpolation”. *Mathematical Software*. 1977.

[LDB17] Victor Lucquin, Sébastien Deguy, and Tamy Boubekeur. “SeamCut: Interactive Mesh Segmentation for Parameterization”. *ACM SIGGRAPH 2017 Technical Briefs*. 2017.

[Lip+04] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rossi, and Hans-Peter Seidel. “Differential Coordinates for Interactive Mesh Editing”. *Proceedings Shape Modeling Applications*. 2004.

[Liu+15] Yong-Jin Liu, Chun-Xu Xu, Dian Fan, and Ying He. “Efficient Construction and Simplification of Delaunay Meshes”. *ACM Transactions on Graphics (TOG)* 34.6 (2015).

[Liu+17a] Bangquan Liu, Shuangmin Chen, Shi-Qing Xin, Ying He, Zhen Liu, and Jieyu Zhao. “An Optimization-Driven Approach for Computing Geodesic Paths on Triangle Meshes”. *Computer-Aided Design* 90 (2017).

[Liu+17b] Yong-Jin Liu, Dian Fan, Chun-Xu Xu, and Ying He. “Constructing Intrinsic Delaunay Triangulations from the Dual of Geodesic Voronoi Diagrams”. *ACM Transactions on Graphics (TOG)* 36.2 (2017).

[LPG12] Yang Liu, Balakrishnan Prabhakaran, and Xiaohu Guo. “Point-Based Manifold Harmonics”. *IEEE Transactions on Visualization and Computer Graphics* 18.10 (2012).

[LSP08] Hao Li, Robert W Sumner, and Mark Pauly. “Global Correspondence Optimization for Non-Rigid Registration of Depth Scans”. *Computer Graphics Forum*. Vol. 27. 2008.

[Luo04] Feng Luo. “Combinatorial Yamabe Flow on Surfaces”. *Communications in Contemporary Mathematics* 6.05 (2004).

[LZ09] Bruno Levy and Richard Hao Zhang. “Spectral Geometry Processing” (2009).

[Mac49] Richard MacNeal. “The Solution of Partial Differential Equations by Means of Electrical Networks”. PhD Thesis. Caltech, 1949.

[MCV08] Dimas Martínez Morera, Paulo Cezar Carvalho, and Luiz Velho. “Modeling on Triangulations with Geodesic Curves”. *The Visual Computer* 24.12 (2008).

[MMP87] Joseph SB Mitchell, David M. Mount, and Christos H. Papadimitriou. “The Discrete Geodesic Problem”. *SIAMComp* 16.4 (1987).

[Mul+08] Patrick Mullen, Yiying Tong, Pierre Alliez, and Mathieu Desbrun. “Spectral Conformal Parameterization”. *Computer Graphics Forum*. Vol. 27. 2008.

[MVC05] Dimas Martínez, Luiz Velho, and Paulo C Carvalho. “Computing Geodesics on Triangular Meshes”. *Computers & Graphics* 29.5 (2005).

[Ovs+16] Maks Ovsjanikov, Etienne Corman, Michael Bronstein, Emanuele Rodolà, Mirela Ben-Chen, Leonidas Guibas, Frederic Chazal, and Alex Bronstein. “Computing and Processing Correspondences with Functional Maps”. *SIGGRAPH ASIA 2016 Courses*. 2016.

[Pai+15] Gilles-Philippe Paillé, Nicolas Ray, Pierre Poulin, Alla Sheffer, and Bruno Lévy. “Dihedral Angle-Based Maps of Tetrahedral Meshes”. *ACM Transactions on Graphics (TOG)* 34.4 (2015).

[PP93] Ulrich Pinkall and Konrad Polthier. “Computing Discrete Minimal Surfaces and Their Conjugates”. *Experimental mathematics* 2.1 (1993).

[Qin+16] Yipeng Qin, Xiaoguang Han, Hongchuan Yu, Yizhou Yu, and Jianjun Zhang. “Fast and Exact Discrete Geodesic Computation Based on Triangle-Oriented Wavefront Propagation”. *ACM Transactions on Graphics (TOG)* 35.4 (2016).

[Reg61] Tullio Regge. “General Relativity without Coordinates”. *Il Nuovo Cimento (1955-1965)* 19.3 (1961).

[Rip90] Samuel Rippa. “Minimal Roughness Property of the Delaunay Triangulation”. *Computer Aided Geometric Design* 7.6 (1990).

[Riv94] Igor Rivin. “Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume”. *Annals of mathematics* 139.3 (1994).

[RŠN19] Mariana Remešíková, Marián Šagát, and Peter Novysedlák. “Discrete Lagrangian Algorithm for Finding Geodesics on Triangular Meshes”. *Applied Mathematical Modelling* 76 (2019).

[SC+19] Nicholas Sharp, Keenan Crane, et al. *Geometry-Central*. 2019.

[SC18] Nicholas Sharp and Keenan Crane. “Variational Surface Cutting”. *ACM Transactions on Graphics (TOG)* 37.4 (2018).

[SC20a] Nicholas Sharp and Keenan Crane. “A Laplacian for Nonmanifold Triangle Meshes”. *Computer Graphics Forum*. Vol. 39. 2020.

[SC20b] Nicholas Sharp and Keenan Crane. “You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges”. *ACM Transactions on Graphics (TOG)* 39.6 (2020).

[SCV14] Justin Solomon, Keegan Crane, and Etienne Vouga. “Laplace-Beltrami: The Swiss Army Knife of Geometry Processing”. *Symposium on Geometry Processing Graduate School (Cardiff, UK, 2014)*. Vol. 2. 2014.

[Set89] James A Sethian. “A Review of Recent Numerical Algorithms for Hypersurfaces Moving with Curvature Dependent Speed”. *Journal of Differential Geometry* 31 (1989).

[SF08] Gilbert Strang and George J Fix. “An Analysis of the Finite Element Method (2 Ed.)” 212 (2008).

[SGW06] Ryan Schmidt, Cindy Grimm, and Brian Wyvill. “Interactive Decal Compositing with Discrete Exponential Maps”. *ACM SIGGRAPH 2006 Papers*. 2006.

[She97] Jonathan Richard Shewchuk. “Delaunay Refinement Mesh Generation”. PhD Thesis. Carnegie Mellon University, 1997.

[Sib78] Robin Sibson. “Locally Equiangular Triangulations”. *The computer journal* 21.3 (1978).

[Sor+04] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. “Laplacian Surface Editing”. *Symposium on Geometry Processing*. 2004.

[Spr19] Boris Springborn. “Ideal Hyperbolic Polyhedra and Discrete Uniformization”. *Discrete & Computational Geometry* (2019).

[SSC19a] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. “Navigating Intrinsic Triangulations”. *ACM Transactions on Graphics (TOG)* 38.4 (2019).

[SSC19b] Nicholas Sharp, Yousuf Soliman, and Keenan Crane. “The Vector Heat Method”. *ACM Transactions on Graphics (TOG)* 38.3 (2019).

[SSP08] Boris Springborn, Peter Schröder, and Ulrich Pinkall. “Conformal Equivalence of Triangle Meshes”. *ACM Transactions on Graphics (TOG)* 27.3 (2008).

[SSŠ02] Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. “Algorithms for Normal Curves and Surfaces”. *International Computing and Combinatorics Conference*. 2002.

[Sur+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J Gortler, and Hugues Hoppe. “Fast Exact and Approximate Geodesics on Meshes”. *ACM transactions on graphics (TOG)* 24.3 (2005).

[Tou+09] Jane Tournois, Camille Wormser, Pierre Alliez, and Mathieu Desbrun. “Interleaving Delaunay Refinement and Optimization for Practical Isotropic Tetrahedron Mesh Generation”. *ACM Transactions on Graphics (TOG)* 28.3 (2009).

[Wan+17] Xiaoning Wang, Zheng Fang, Jiajun Wu, Shi-Qing Xin, and Ying He. “Discrete Geodesic Graph (DGG) for Computing Geodesic Distances on Polyhedral Surfaces”. *Computer Aided Geometric Design* 52 (2017).

[War17] Max Wardetzky. “A Primer on Laplacians” (2017).

[Wei85] Kevin Weiler. “Edge-Based Data Structures for Solid Modeling in Curved-Surface Environments”. *IEEE Computer graphics and applications* 5.1 (1985).

[WT09] Chunlin Wu and Xuecheng Tai. “A Level Set Formulation of Geodesic Curvature Flow on Simplicial Surfaces”. *IEEE Transactions on Visualization and Computer Graphics* 16.4 (2009).

[XHF11] Shi-Qing Xin, Ying He, and Chi-Wing Fu. “Efficiently Computing Exact Geodesic Loops within Finite Steps”. *IEEE transactions on visualization and computer graphics* 18.6 (2011).

[Xia13] Ge Xia. “The Stretch Factor of the Delaunay Triangulation Is Less than 1.998”. *SIAM Journal on Computing* 42.4 (2013).

[Xu+15] Chunxu Xu, Tuanfeng Y Wang, Yong-Jin Liu, Ligang Liu, and Ying He. “Fast Wavefront Propagation (FWP) for Computing Exact Geodesic Distances on Meshes”. *IEEE transactions on visualization and computer graphics* 21.7 (2015).

[XW07] Shi-Qing Xin and Guo-Jin Wang. “Efficiently Determining a Locally Exact Shortest Path on Polyhedral Surfaces”. *Computer-Aided Design* 39.12 (2007).

[XW09] Shi-Qing Xin and Guo-Jin Wang. “Improving Chen and Han’s Algorithm on the Discrete Geodesic Problem”. *ACM Transactions on Graphics (TOG)* 28.4 (2009).

[Yin+19] Xiang Ying, Caibao Huang, Xuzhou Fu, Ying He, Ruiguo Yu, Jianrong Wang, and Mei Yu. “Parallelizing Discrete Geodesic Algorithms with Perfect Efficiency”. *Computer-Aided Design* 115 (2019).

[Yu+04] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. “Mesh Editing with Poisson-Based Gradient Field Manipulation”. *SIGGRAPH*. 2004.

[YWH13] Xiang Ying, Xiaoning Wang, and Ying He. “Saddle Vertex Graph: A Novel Solution to the Discrete Geodesic Problem”. *ACM Transactions on Graphics (TOG)* 32.6 (2013).

[YXH14] Xiang Ying, Shi-Qing Xin, and Ying He. “Parallel Chen-Han (PCH) Algorithm for Discrete Geodesics”. *ACM Transactions on Graphics (TOG)* 33.1 (2014).

[Zha+10] Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng, and Xue-cheng Tai. “Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow”. *Computer Graphics Forum*. Vol. 29. 2010.

[ZJ16] Qingnan Zhou and Alec Jacobson. “Thingi10K: A Dataset of 10,000 3D-Printing Models”. *arXiv:1605.04797* (2016). arXiv: 1605.04797.