Intrinsic Triangulations in Geometry Processing

Nicholas Sharp

CMU-CS-21-132
August 2021

Computer Science Department
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.

Copyright © 2021 Nicholas Sharp

This research was sponsored by National Science Foundation award IIS1943123; and by The David and Lucile Packard Foundation award 201868047. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1252522 and DGE 1745016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: geometry processing, discrete differential geometry, numerical computing

To Mariah, for supporting and inspiring me.
To my family, for love and patience.
To my friends, for endless joy.

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 decoupling 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 with ordinary vertex positions, but instead with only edge lengths.

The contributions of this thesis begin with new data structures for richly encoding intrinsic triangulations, which support new operations necessary for applications while remaining efficient and robust. Using these data structures, we describe a wide variety of mesh processing operations on intrinsic triangulations, including powerful retriangulation schemes with strong guarantees. Additionally, we demonstrate how intrinsic triangulations can be used for tasks beyond retriangulation, introducing a new flip-based algorithm for computing geodesic paths on surfaces. Finally, we present a generalization of intrinsic triangulations, to offer the same benefits for less-structured data such as nonmanifold meshes and point clouds. Throughout, we show applications to problems in geometry processing, where intrinsic triangulations lend much-needed automatic robustness to tasks including parameterization, vector field processing, spectral methods, and more.

Acknowledgments

This work would not have been possible without the support of so many mentors, colleagues, supporters, family members, and friends over the years. Words cannot express my gratitude, but I will try!

As my PhD advisor, Keenan taught me so much about geometry, research, academia, and life in general, for which I will always be grateful. Thank you for investing so much in me—I hope that one day I can pass it all on. The Geometry Collective has been a wonderful source of learning and friendship during my grad school, including Chris, Rohan, Katherine, Mark, Nicole, Etienne, Yousuf, Mina, Connor, Derek, Joel, and many others. In particular, I am so fortunate to have Mark and Yousuf as brilliant collaborators in our projects together. I am also grateful to the larger community at Carnegie Mellon: faculty including Kayvon, Ioannis, Jim, Jessica, Nancy, Gary, Danny, and Dejan, who I had the opportunity to learn from in and out of the classroom, as well as fellow students like Fait, Nic, Ravi, Vidya, Evan, Angela, David, Karima, Nico, and others, who were a pleasure to have by my side both as colleagues and as friends.

Before CMU, I got my start at Virginia Tech, a time which I look back on with great fondness. Professors Murali, Ross, and Zietsman were amazing supporters for an undergraduate wandering through research for the first time. My passion for computer science came from the competitive programming team more than anywhere else; I am indebted to my brilliant teammates Mike, Matt, Scott, Aziz, Brendan, Saurav, and most of all to Prof. Back for his endless support and inspiring intellectual curiosity. I am also grateful to everyone in Theta Tau, for helping me grow as an engineer and as a person. Even before college, I was incredibly lucky to have such great early mentors and role models, from Ms. Wingfield and Evans, to Coach Schmidt, to the endlessly patient Coach Summers.

Outside of the university, I had the good fortune to spend time at FRL with Yaser, Takaaki, Alex, Rob, Stephan, Martin, and many others, who collectively knew so much about computer vision and graphics that I could learn even by osmosis. I also had the pleasure of spending a few months in Paris with Maks, Adrien, Marie-Julie, and Souhaib, as well as an early visit with David in Aachen—thank you for being such wonderful hosts, and mind-expanding collaborators.

I am indebted to everyone who has worked tirelessly behind the scenes to make my research possible. In particular, Christina, Deb, and Tracy at CMU have my heartfelt thanks.

On the personal side, Mariah has been an incredible partner during this phase of my life, and I can't wait to embark on our next journey together. My parents Dawn and Joe deserve immense credit for fostering my love of learning—and stubborn competitiveness—from an early age, and I have always looked up to my brother Sam in more ways than he realizes. My friends kept me sane through all of this: Brendan, Andrew, Mike, Saurav, Samir, Chris, and all of VTqq—thank you so much for all the good times.

In more tangible terms, this work was made possible by generous financial support from an NSF Graduate Research Fellowship, and gifts from Autodesk, Adobe, and Facebook. And of course, I owe a huge thanks to my committee: Keenan, Ioannis, Anupam, and Maks, for their guidance of this thesis.

Contents

1 Introduction 1
- 1.1 Why This Approach? 3
  - 1.1.1 What Are Intrinsic Triangulations Not? 4
- 1.2 Historical Roots 5

2 Intrinsic Triangulations 7
- 2.1 Connectivity 9
- 2.2 Topological Data Structures 12
- 2.3 Geometry 13
  - 2.3.1 Extrinsic Geometry 14
  - 2.3.2 Barycentric Coordinates 14
  - 2.3.3 Intrinsic Geometry 15
  - 2.3.4 Edge Flips 16
  - 2.3.5 Cone Metric 18
  - 2.3.6 Length Based Formulas 19
  - 2.3.7 Local Coordinates 20
- 2.4 Tangent Vectors 23
  - 2.4.1 Geodesics 23
  - 2.4.2 Exponential Map 25
- 2.5 The Laplace Matrix 26
  - 2.5.1 The Mass Matrix 27

3 Representing Correspondence 29
- 3.1 Intrinsic Triangulations of Embedded Surfaces 29
  - 3.1.1 Common Subdivision 30
- 3.2 Correspondence Data Structures 30
- 3.3 Explicit Crossings 32
  - 3.3.1 Edge Flips 32
- 3.4 Signposts 33
  - 3.4.1 Tracing Through Triangulations 33
  - 3.4.2 Local Mesh Operations 34
  - 3.4.3 Queries 35
  - 3.4.4 Robustness 36
- 3.5 Integer Coordinates 36
  - 3.5.1 Normal Coordinates 37
  - 3.5.2 Roundabouts 38
  - 3.5.3 The Abstract Viewpoint 39

3.5.4 Local Mesh Operations . . . 39
3.5.5 Robustness . . . 40
3.6 Extracting the Common Subdivision . . . 40

4 Retriangulation **43**
4.1 Intrinsic Delaunay Triangulations . . . 43
4.1.1 Properties of Intrinsic Delaunay Triangulations . . . 45
4.1.1.1 Empty Triangle Circumcircles & Edge Disks . . . 45
4.1.1.2 Contains Nearest Neighbors . . . 45
4.1.1.3 Maximizes Angles . . . 46
4.1.1.4 Smoothest Piecewise-Linear Interpolation (Rippa's Theorem) . . . 46
4.1.1.5 Minimal Spectrum . . . 47
4.1.1.6 Minimal Minimum Spanning Tree . . . 47
4.1.1.7 Geometric Spanner . . . 47
4.2 Delaunay Flipping . . . 48
4.3 Delaunay Refinement . . . 50
4.4 Constrained Triangulation . . . 54
4.5 Optimal Delaunay Triangulation . . . 54
4.6 Adaptive Mesh Refinement . . . 55
4.7 Intrinsic Mollification . . . 55
4.8 Metric Scaling . . . 56
4.9 Comparison to Traditional Remeshing . . . 57
4.9.1 Other Notions of Delaunay . . . 57
4.9.2 Extrinsic Construction . . . 58
4.10 Robustifying Applications with Intrinsic Triangulations . . . 58
4.10.1 The Intrinsic Delaunay Laplacian . . . 59
4.10.2 Examples . . . 59
4.11 Transferring Solutions Between Triangulations . . . 61
4.11.1 Optimal Attribute Transfer . . . 61
4.11.2 Transferring Tangent Vectors . . . 62

5 Geodesics **63**
5.1 Geodesics from Intrinsic Edge Flips . . . 64
5.2 Geodesic Loops and Curve Networks . . . 67
5.3 Geodesic Bézier Curves . . . 67
5.4 Triangulated Geodesic Paths . . . 68
5.5 Single-Source Geodesics . . . 69

6 Generalized Domains **71**
6.1 Nonmanifold Intrinsic Triangulations . . . 71
6.2 Point Clouds . . . 74

7 Conclusion **77**
7.1 Open Questions . . . 78

A Evaluating Geometric Quantities **81**

References **83**

A note about content

The text and figures in this thesis have significant overlap with several papers published by myself and collaborators over the course of my PhD. These publications are the original source for such content, used with permission.

In particular, see the following publications:

- Nicholas Sharp, Yousuf Soliman, and Keenan Crane. "The Vector Heat Method". *ACM Transactions on Graphics (TOG)* 38.3 (2019)
- Nicholas Sharp, Yousuf Soliman, and Keenan Crane. "Navigating Intrinsic Triangulations". *ACM Transactions on Graphics (TOG)* 38.4 (2019)
- Nicholas Sharp and Keenan Crane. "A Laplacian for Nonmanifold Triangle Meshes". *Computer Graphics Forum*. Vol. 39. 2020
- 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)
- Mark Gillespie, Nicholas Sharp, and Keenan Crane. "Integer Coordinates for Intrinsic Geometry Processing". *arXiv preprint arXiv:2106.00220* (2021)
- Nicholas Sharp, Mark Gillespie, and Keenan Crane. "Geometry Processing with Intrinsic Triangulations". *ACM SIGGRAPH 2021 Courses*. 2021

List of Figures

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. One of the goals of this thesis is to build a framework 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." . . 1

1.2 Traditionally, mesh 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). 2

1.3 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. 3

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 approximation error (as with the marching cubes approximation at left), these errors will get neither better nor worse. 7

2.2 In a simplicial complex, an edge must have two distinct endpoints, and a triangle must have three distinct vertices. 9

2.3 A pair of triangles is consistently oriented if they *disagree* on the orientation of the shared edge *(left)*. A triangulation is orientable if all triangles can be assigned consistent orientations *(center)* and nonorientable otherwise *(right)*. 10

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. 10

2.5 In a Δ-complex, the vertices of an edge or triangle need not be distinct. 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). 11

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. 13

2.7 Shapes that are difficult or impossible to embed in $\mathbb{R}^n$ are often much easier to describe intrinsically. For instance, the *flat torus (left)* can be thought of intrinsically as a square with opposite edges identified—whereas finding a flat embedding into $\mathbb{R}^3$ is a major challenge (image from [Bor+13]). Likewise intrinsic triangulations that describe a perfectly reasonable metric space, such as the tetrahedron depicted in the right, may be impossible to embed into $\mathbb{R}^3$ while preserving lengths. 14

2.8 *Left:* an edge flip modifies the connectivity of a triangle mesh, replacing triangles ijk, jim with triangles kmi, mkj. *Right:* whereas an ordinary extrinsic edge flip changes the mesh geometry, an intrinsic flip leaves the original geometry untouched. 16

2.9 *Left:* intrinsically, a small neighborhood around any vertex of a polyhedral surface is indistinguishable from a round circular cone. For instance, a vertex made from triangular pieces of paper can be easily smoothed out into a circular cone without stretching or ripping. *Center:* a good mental "cartoon" of an intrinsic triangulation is hence a surface where the edges are completely smooth, and only the vertices are visible. *Right:* from this perspective, no one triangulation is special—there are many intrinsic triangulations that describe the exact same geometry. 18

2.10 Given the three edge lengths of a triangle $\ell_{ij}, \ell_{jk}, \ell_{ki}$, one can always find a corresponding set of vertex positions $\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_k$ in either 2D or 3D. If the vertices of the 3D embedding are placed along the principal axes, then they are related to the barycentric coordinates by a simple diagonal scaling D. 20

2.11 Local coordinate system for tangent vectors at vertices. 23

2.12 A *geodesic* is often confused with a "shortest path" between points $\mathbf{p}$ and $\mathbf{q}$, but in general a geodesic can be any path that locally minimizes length, or equivalently, that exhibits no tangential acceleration. 24

2.13 There are two distinct ways to define geodesics on polyhedra: as "straightest" curves, with equal angle on both sides *(left)*, or as "locally shortest" curves that cannot be pulled tighter *(right)*. A curve through a positively-curved vertex ($\Omega_i > 0$) can always be made shorter, whereas there are many ways to continue a path through a negatively-curved vertex ($\Omega_i < 0$) while remaining locally shortest—for example, a path from $\mathbf{p}$ through i and then to either $\mathbf{q}_1$ or $\mathbf{q}_2$ is locally shortest. For intrinsic triangulations, we need only consider locally shortest geodesics. *(Leftmost figure adapted from [PS06])* 24

2.14 *Left:* the exponential map simply walks along the surface in a given direction $\mathbf{x}$, starting at a given point $\mathbf{p}$. *Right:* on a triangle mesh, the exponential map can be computed by tracing out a straight line through a strip of triangles. Each step amounts to performing ray-edge intersections, then transporting the ray direction vector to the next triangle. . 25

3.1 The representations for intrinsic triangulations considered in this thesis. 32

3.2 Flipping an edge in the explicit crossing representation [Fis+07]. 32

3.3 Signposts encode edges of an intrinsic by storing the length and direction of each edge in the tangent space each vertex. 33

4.1 Intrinsic retriangulation schemes applied to a computed-aided design model with poor triangle quality. The black wireframe denotes the extrinsic 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. 43
4.2 There are many equivalent characterizations of the Delaunay triangulation. Here we illustrate a few examples. 44
4.3 An empirical study of the number of edge flips to produce an intrinsic Delaunay triangulation [SSC19a]. Each point is a 3D model from the Thingi10k dataset [ZJ16]. 48
4.4 *Left*: Rich data structures enable intrinsic Delaunay refinement, generating triangulations with good angle bounds. The black wireframe denotes the extrinsic mesh, while colored triangles give the intrinsic triangulation. *Right*: Signposts further enable vector field processing; the Laplacian of the intrinsic Delaunay triangulation offers a maximum principle for tangent vector fields, which here avoids unexpected flipped vectors when generating a smooth field. 50
4.5 *Top*. Planar constrained Delaunay triangulations are used to produce high-quality triangulations which conform to a collection of specified lines. *Bottom*. Constrained intrinsic triangulations play a similar role on surfaces, preserving a set of intrinsic edges in the triangulation. 54
4.6 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]. 55
4.7 Traditional remeshing cannot improve element quality without increasing mesh size or disturbing the geometry; intrinsic triangulations escape this tradeoff. 57
4.8 *Left*: Extrinsic schemes may need to insert many vertices and create skinny triangles to produce a Delaunay triangulation while preserving the shape, while the intrinsic approach preserves the vertex set and improves triangles quality. *Right*: Small corner angles in an extrinsic triangulation must remain if extrinsic shape is to be preserved, but instead preserving only the intrinsic shape allows these angles to be improved. . . 58
4.9 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] 59
4.10 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] 59
4.11 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] 60
4.12 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] 60

5.1 FLIPOUT shortens the curve γ by repeatedly flipping edges to introduce a shorter path. 65
5.2 FLIPOUT notation. 65

5.3 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. 66
5.4 Careful treatment of noncrossing curves enables finding geodesics that overlap many times (*top*), or those that get pulled tight around endpoints of the path itself (*bottom*). 66
5.5 Algorithm 7 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. 68
5.6 Geodesic loops generated by with edge flips in an intrinsic triangulations. Inset values give the runtimes. 68
5.7 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*). 68
5.8 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. 69

6.1 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 extrinsic mesh, which is nonmanifold and has many low-quality triangles, yields only numerical noise. Substituting the nonmanifold IDT Laplacian constructed on the tufted cover generates the expected smooth deformation. 73
6.2 The tufted intrinsic Delaunay point cloud Laplacian, demonstrated here for spectral conformal parameterization of a point cloud [Mul+08]. 75
6.3 Angular parameterizations of point clouds, computed with the tufted intrinsic Delaunay point cloud Laplacian via the vector heat method [SSC19b]. 75

List of Tables

3.1 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. 31

Chapter 1

Introduction

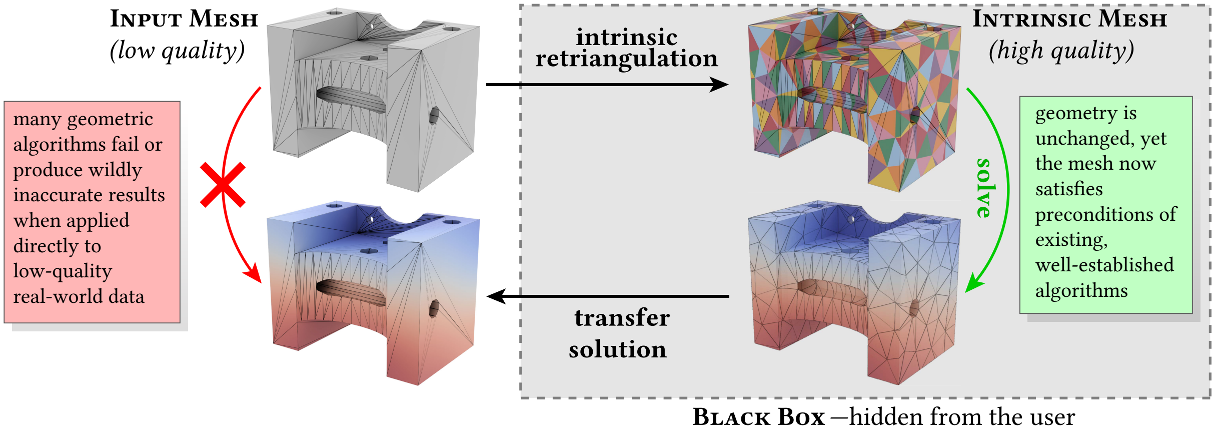

INPUT MESH
(low quality)
many geometric algorithms fail or produce wildly inaccurate results when applied directly to low-quality real-world data
intrinsic retriangulation
INTRINSIC MESH
(high quality)
geometry is unchanged, yet the mesh now satisfies preconditions of existing, well-established algorithms
solve
transfer solution
BLACK BOX —hidden from the user

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. One of the goals of this thesis is to build a framework 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.”

Geometric data plays an increasingly vital role in tasks ranging from computational fabrication to augmented reality to autonomous driving. Triangle meshes are a basic representation for 3D geometry, playing the same central role as pixel arrays in image processing. Hence, even seemingly small shifts in the way we think about triangle meshes can have major consequences for a wide variety of applications. In this thesis, we will we explore what happens if we replace the ordinary *extrinsic* encoding of mesh geometry, via vertex positions in $\mathbb{R}^n$, with an alternative *intrinsic* description, via lengths associated with edges. The resulting *intrinsic triangulations* are far more flexible than their traditional extrinsic counterparts, yet still provide the geometric information needed to execute many fundamental geometry processing tasks. This thesis introduces the theory and practice of intrinsic triangulations, from their basic representation, to new data structures and algorithms, to applications in geometry processing.

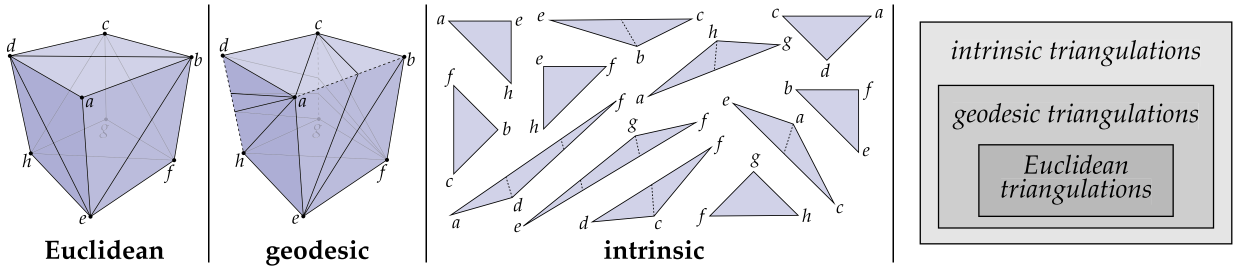


Figure 1.2: Traditionally, mesh edges are straight line segments in Euclidean space (far left). The much larger spaces of *geodesic* and *intrinsic* triangulations provide tremendous additional flexibility by allowing edges to be straight paths along the surface (center left), or by just considering an abstract collection of triangles identified along shared edges (center right).

Intrinsic triangulations bring many ideas across topology and differential geometry together in a discrete, computational setting. In particular, the name “intrinsic” arises from a central concept in modern differential geometry: many properties of a surface do not depend on how the surface is embedded in space, but only on local measurements of quantities like angles and distances *along* the surface. For instance, the shortest path along the surface between two points on a sheet of paper is unchanged if the paper is rolled up into a tube (intrinsic), whereas the surface normal at a point may be very different (extrinsic). A very good mental model is to think about maps of the Earth: although no individual map depicts the whole planet as a round ball floating in outer space, each map conveys very useful information about some local neighborhood. Likewise, intrinsic triangulations enable one to inspect and manipulate local pieces of a mesh, without needing to know how (or whether) it floats in space.

In fact, the practical utility of intrinsic triangulations comes from the ability to work with a much larger space of meshes than can be represented via the ordinary (extrinsic) approach (Figure 1.2). In this sense, they provide a “relaxation” of the standard picture, which in turn provides new capabilities for geometry processing (Figure 1.3). Yet since the final set of intrinsic mesh operations looks much like those available on an ordinary mesh, the operations can be encapsulated in a “black box” interface that hides much of the complexity of working in the intrinsic setting. Hence, algorithms written for intrinsic triangulations often end up looking very similar to ordinary (extrinsic) mesh code.

The framework of intrinsic triangulations is particularly useful for improving the *robustness* of existing algorithms. Researchers and engineers have put tremendous effort into developing sophisticated algorithms for surface mesh processing, yet these algorithms often cannot be used in practice since preconditions on the input do not match up with the reality of actual data (*e.g.*, coarse or poorly-triangulated meshes from 3D printing or real-time visualization). In a perfect world, geometric software would automatically build a triangulation “under the hood” that satisfies the preconditions of a given algorithm, run the algorithm, then return the solution in a format usable within the original context. Modern numerical linear algebra packages like MATLAB provide an excellent analogy: to solve a linear system $Ax = b$ users can just type `x = A\b`; the matrix `A` is then intelligently re-ordered and factorized to improve stability, accuracy, and efficiency. As a result, non-expert users trivially benefit from sophisticated, performance-tuned solvers—which has led to rapid growth of fields like image processing, computer vision, and machine learning. Geometric computing has not yet achieved this same level of simplicity—but the intrinsic framework described in these notes provides some important steps in that direction.

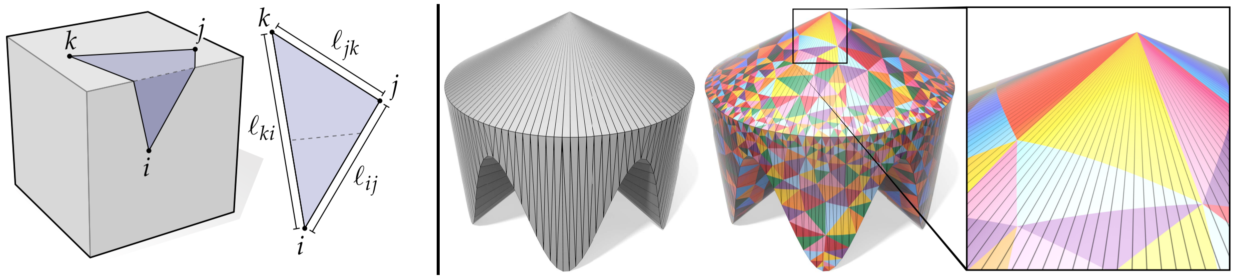


Figure 1.3: Conceptually, intrinsic triangles can "bend" across an underlying polyhedron, yet still flatten out into standard triangles described by three ordinary edge lengths (left). This flexibility enables things that are impossible with standard, extrinsic algorithms—here, a mesh with tiny input 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.

In particular, intrinsic triangulations provide a valuable bridge between low-quality data, and algorithms that assume high-quality input (Figure 1.1), enabling:

- 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
- algorithms designed for manifold, orientable data to be applied directly to arbitrary triangle meshes.

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

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 is the starting point for most PDE-based geometry processing, as well as sub-fields such as spectral geometry processing [LZ09], functional maps [Ovs+16], and so on [SCV14]. Beyond the Laplacian, other fundamental geometric quantities (curvatures, geodesic distances, the logarithmic map, and so on) can easily be computed from intrinsic data alone. For such problems, working in the strictly larger space of intrinsic triangulations offers, *e.g.*, better accuracy with fewer degrees of freedom.
- **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 generation frequently encounters a "no free lunch" scenario where one must compromise on 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 difficulties not encountered in 2D (*e.g.*, "sliver" tetrahedra), 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 4.3) and adaptive mesh refinement (Section 4.5).
- **Some important problems do not even have an extrinsic formulation.** Many geometry processing tasks are formulated by mapping a curved manifold into a flat space—*e.g.*, surface parameterization [SSP08a], 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 definition 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 prompts one to ask entirely new questions, or take completely different approaches.

1.1.1 What Are Intrinsic Triangulations Not?

Intrinsic triangulations are not, of course, a remedy for all difficulties encountered in geometry processing:

- **Intrinsic triangulations do not improve geometric approximation quality.** As discussed in Chapter 2, a basic assumption of the intrinsic triangulation framework is that the input mesh provides an exact description of the geometry of interest. Atomic operations are designed to preserve the geometry exactly, and can neither degrade nor improve the approximation quality of the input geometry—though they can *significantly* improve the quality of individual triangular elements.
- **Intrinsic triangulations do not repair topological defects.** Likewise, the intrinsic framework will not fill holes in the data, nor fix spurious topological features (like small handles) that result from "upstream" algorithms like surface reconstruction. The operating assumption is that the given topology is the correct topology. (Of course, nothing prevents one from running standard mesh repair algorithms prior to intrinsic processing.)
- **Intrinsic triangulations do not yet provide solutions for volumetric problems.** Algorithms and data structures for intrinsic processing of 2D (surface) data are at this point fairly mature; those for working with 3D (volumetric) meshes are largely unexplored—this is a significant opportunity for future work.
- **An intrinsic triangulation is not a standard triangle mesh.** Most importantly, the basic premise of intrinsic geometry processing is to work with a larger space of triangle meshes that *cannot* be expressed via ordinary flat triangles in 3D space (Figure 1.2). Of course, many "downstream" algorithms may still require an ordinary mesh (or other standard data) as input. Section 4.11 explores the many ways computation on intrinsic meshes can be used to facilitate

improved computation downstream. The most basic observation is that one can typically subdivide an intrinsic mesh into an ordinary extrinsic one (though many other options are available).

1.2 Historical Roots

Though combinatorial triangulations have long played a role in topology, graph theory, *etc.*, the framework discussed in this thesis owes its greatest debt to work that considers polyhedral surfaces from the intrinsic *geometric* perspective pioneered by Gauss [Gau25] and Riemann [Rie54]. One of the first major results about the intrinsic geometry of polyhedra was Alexandrov's uniqueness theorem for embeddings of convex polyhedra published in the 1940s. Subsequently, Regge [Reg61] explored the use of intrinsic triangulations to approximate the equations of general relativity. Already in these early works we find statements that strongly resemble the modern perspective on intrinsic triangulations. For instance, Regge writes:

> *"It is interesting to notice that the intrinsic geometry of M is completely fixed by the connection matrix and the length of all edges. The connection matrix is essentially a list of all faces, edges, and vertices of M and a list of their mutual relationship,* i.e.*, by reading it one can decide which vertices, edges, belong to a given face, etc. The connection matrix supplies us with all the topological information needed in the construction of M."*

The "connection matrix" described by Regge is what we call in these notes a *topological data structure* (Section 2.2), which encodes the connectivity of the mesh; he also makes the critical observation that the edge lengths alone are sufficient to describe some of the most basic geometric quantities:

> *"The metric tensor on the other hand is replaced by the lengths of the edges ... the knowledge of the lengths of all edges of M implies the knowledge of all angles and therefore of the deficiencies."*

In other words, one can read off the curvature of the discrete surface from the edge lengths, as discussed in Section 2.3.6. Most importantly, Regge recognizes that the triangulation itself is superficial, and merely serves as a "scaffolding" to define the underlying space:

> *"Since we are chiefly interested in the intrinsic geometry of manyfolds, we are not particularly interested in the edges of M and we regard them as a rather immaterial convention for dividing M into triangles, any other convention being just as good."*

The fact that the initial triangulation is not "special" is what allows us to move through a much larger space of meshes than we can in the extrinsic setting—this perspective is discussed further in Section 2.3.5.

In more recent years, work by Rivin [Riv94a] and others laid the foundation for algorithms by introducing the notion of *intrinsic Delaunay triangulations*, which have many attractive features for geometry processing (Section 4.1); work by Indermitte et al. [Ind+01] and Bobenko and Springborn [BS07] establishes that such triangulations can always be found via a simple edge flip algorithm (Section 4.2). Intrinsic Delaunay triangulations in turn lead to a canonical *Laplacian* for polyhedral surfaces [BS07]—the Laplacian plays a fundamental role throughout geometry and physics, and in particular in geometry processing algorithms based on partial differential equations (PDEs). The key benefit of the intrinsic Laplace operator is that it depends only on the shape of a polyhedral surface, rather than the quality of the input triangulation. Gu et al. [Gu+10] shows that in fact the discrete Laplace operator is itself sufficient to describe the geometry of the polyhedron. Glickenstein [Gli05] and others generalize the Delaunay property, edge flips, and several other concepts to a richer class of Euclidean triangulations which includes *weighted*, *Thurston* [Thu79], and *duality* triangulations, showing that many of the same properties still hold. These triangulations have numerous applications in geometry processing [Goe+14].

The machinery of intrinsic triangulations also has roots in conformal geometry processing [Luo04; KSS06]. In particular, a recent *discrete uniformization theorem* [Gu+18b; Gu+18a; Spr19] guarantees that any triangle mesh (no matter how poorly triangulated) admits a high-quality conformal parameterization useful for computer graphics and geometry processing [SSP08b; GSC21b], but only if one is allowed to change the intrinsic triangulation. Here, even though the initial and final meshes are embeddable in $\mathbb{R}^n$, intermediate triangulations may not be—providing an excellent example of how working in a "relaxed" space provides fundamentally new opportunities for mesh processing. These algorithms also build on an unexpected connection between Euclidean polyhedra and *ideal hyperbolic polyhedra* [BPS15], which is also explored in recent work by Gillespie, Springborn, and Crane [GSC21b].

Here, we always take the perspective of a fixed, discrete cone metric which is to be triangulated. The perspective of triangulating a smooth manifold embedded in $\mathbb{R}^n$ via sampling has also been considered [BG10; BDG13]; under sufficient sampling assumptions a Delaunay complex can always be constructed, even in high dimension. One can also consider a more general set of triangulations on a surface which have geodesic (locally-shortest) edges, but relax the requirement that the interior of each triangle be Euclidean (contain no vertices). Such triangulations arise as the dual of geodesic Voronoi diagrams [Ye+19], and similar techniques have been used for self-parameterization of meshes [Lee+98; Liu+20; Liu+21], though the properties of such triangulations have yet to be deeply studied.

Despite the great potential of intrinsic triangulations, there are several reasons that these methods have not yet seen broader adoption in practice. One is simply the scarcity of material aimed at a computational audience—which this thesis aims to address. Even in mathematics it took a very long time (until the 19th century) for the intrinsic perspective to be developed and accepted, yet the shift to intrinsic formulation was a major step forward for the field. We are likewise optimistic that the intrinsic perspective has the potential for major impact on geometry processing and scientific computing.

Another major deficit is that, until quite recently [Fis+07; SSC19a; GSC21a], there has been very little work on practical data structures for intrinsic triangulations. Basic representations developed in mathematics do not support many of the operations needed for digital geometry processing. For instance, when studying problems in, say, geometric topology, one is often happy to consider the *coarsest* triangulation of a space—and may have no need to insert vertices or split edges. In Chapter 3 we discuss several different data structures for intrinsic triangulations that support a more complete set of operations, and examine their trade offs for practical computation.

Finally, given that general-purpose intrinsic data structures are quite new, it is not surprising that we are just starting to see algorithms that fully take advantage of the intrinsic perspective. Chapter 4 explores how several classic algorithms for Delaunay triangulation can be generalized to the intrinsic setting, which in turn improve accuracy and robustness for a variety of PDE-based geometry processing tasks. Chapter 5 explores an especially interesting example, where retriangulation *unrelated* to the Delaunay property is used to find geodesic curves and curve networks on surfaces. Other algorithms consider the intrinsic perspective (*e.g.*, [Goe+14; Liu+21]), but do not yet take advantage of the full space of triangulations accessible via intrinsic edge flips and other intrinsic operations. Beyond this recent work, a great deal remains to be done—Section 7.1 explores a variety of open questions and directions for future work.

Chapter 2

Intrinsic Triangulations

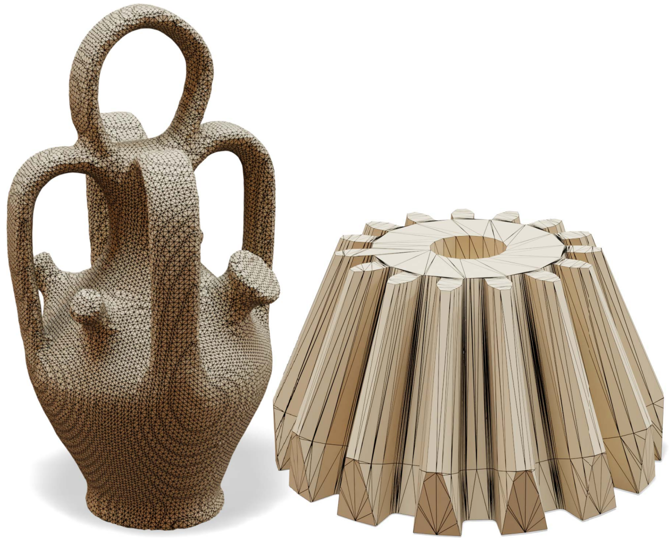


Figure 2.1: An intrinsic triangulation exactly preserves the input geometry, while changing the mesh connectivity. Hence, if the input mesh gives an exact description of the geometry (as with the CAD model at right), it will not be corrupted; if the input exhibits noise or approximation error (as with the marching cubes approximation at left), these errors will get neither better nor worse.

This thesis expands 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. 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 Figure 2.1).

The most important idea is that we will often 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.

The framework of intrinsic triangulations makes two important assumptions. First, we imagine that the input geometry is an *exact* 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 7.1. 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 Delaunay Laplacian (Section 4.1), 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 2.3.1 and 2.3.3 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].)

2.1 Connectivity

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.

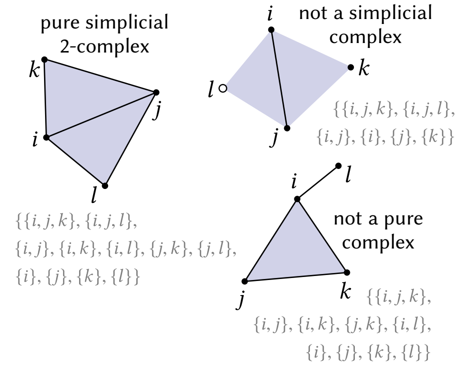
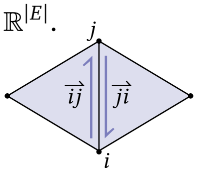

pure simplicial 2-complex
k
j
i
l
$\{\{i,j,k\},\{i,j,l\},$
$\{i,j\},\{i,k\},\{i,l\},\{j,k\},\{j,l\},$
$\{i\},\{j\},\{k\},\{l\}\}$
i
not a simplicial complex
l
k
j
$\{\{i,j,k\},\{i,j,l\},$
$\{i,j\},\{i\},\{j\},\{k\}\}$
i
l
not a pure complex
j
k
$\{\{i,j,k\},$
$\{i,j\},\{i,k\},\{j,k\},\{i,l\},$
$\{i\},\{j\},\{k\},\{l\}\}$

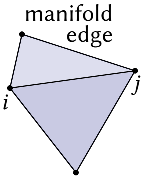
Figure 2.2: In a simplicial complex, an edge must have two distinct endpoints, and a triangle must have three distinct vertices.

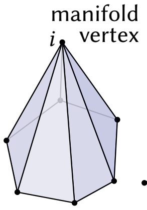
Notation. We will refer to the vertices, edges, and faces of any topological triangulation T as V, E, and F, *resp.*, so that $T = (V, E, F)$. We use $\partial E \subset E$ to denote the set of boundary edges, *i.e.*, edges contained in exactly one triangle, and $\partial V \subset V$ to denote boundary 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}. 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 typically express a quantity as a map from mesh elements to some set of values—for instance, edge lengths can be viewed as a map $\ell : E \to \mathbb{R}_{>0}$ assigning a positive number to each edge, and triangle normals can be viewed as a map $N : F \to \mathbb{R}^3$ assigning three coordinates to each face. If helpful, one can also think of this data as being encoded by column vectors, *e.g.*, $\ell \in \mathbb{R}^{|E|}$.

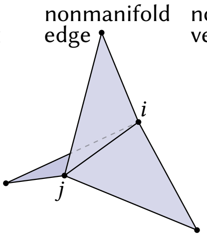
We will also consider the set of *halfedges* H_{ext}, directed edges associated with each edge: for an edge *ij* the are two associated halfedges, one pointing from $i \to j$ and one from $j \to i$.

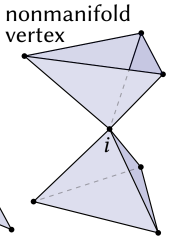


We will often (but not always) assume that T is *manifold* and *orientable*, as defined below. These assumptions simplify data structures and algorithms, and are often sufficient for working with real data—especially since one can sometimes build a "bridge" between nonmanifold meshes and algorithms that operate only on manifold data (see Section 6.1).









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.

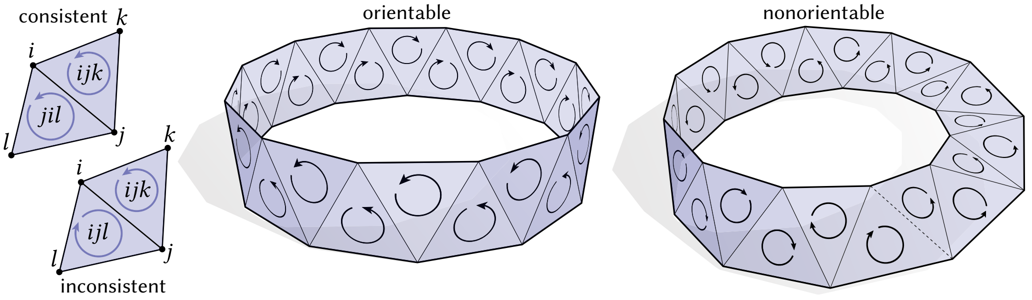


Figure 2.3: A pair of triangles is consistently oriented if they *disagree* on the orientation of the shared edge *(left)*. A triangulation is orientable if all triangles can be assigned consistent orientations *(center)* and nonorientable otherwise *(right)*.

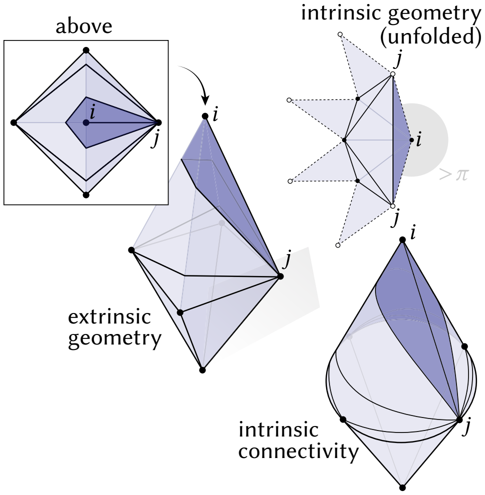


Figure 2.4: Intrinsic triangles can "wrap around" extrinsic polyhedra, allowing them to have unusual connectivity. Here, for instance, the dark blue triangle connects once to vertex i and twice to vertex j—effectively gluing two of its sides to each other along edge ij.

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.*, $\overrightarrow{kji}$). 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.

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 2.5).

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 $i_0j_0k_0, \ldots i_{|F|}j_{|F|}k_{|F|}$.. 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 2.2, where we describe data structures for Δ-complexes. Every simplicial complex is also a Δ-complex. As in the simplicial case we will consider only *pure*, 2-dimensional Δ-complexes, *i.e.*, every vertex and edge is contained in some triangle.

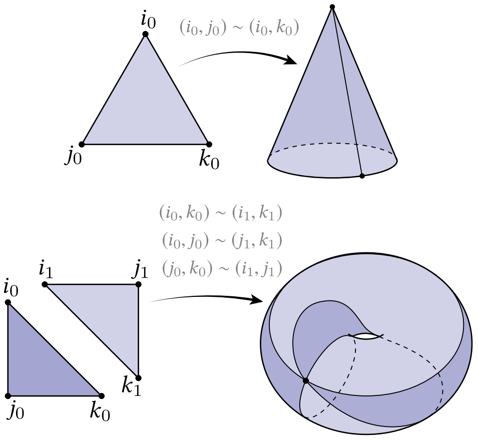
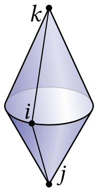

i_0
$(i_0, j_0) \sim (i_0, k_0)$
j_0
k_0
$(i_0, k_0) \sim (i_1, k_1)$
$(i_0, j_0) \sim (j_1, k_1)$
$(j_0, k_0) \sim (i_1, j_1)$
i_1
j_1
i_0
k_1
j_0
k_0

Figure 2.5: In a Δ-complex, the vertices of an edge or triangle need not be distinct. 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).

Notation becomes more challenging for a Δ-complex, since edges and triangles are no longer uniquely determined by their vertices (Figure 2.5). For instance, we may have a *self edge* where the same vertex is found at both endpoints. One possibility might be 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 stick with the convention that a k-dimensional mesh element is specified by $k + 1$ 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 indices need not refer to 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... "*, 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.

Defining the *degree* of a vertex in a Δ-complex also requires some care. In a simplicial complex the degree of a vertex i is simply the number of edges incident on i, but in a Δ complex the same edge may be incident on a vertex more than once. We therefore define the degree $\deg(i)$ as the number of incident edges counted with multiplicity, *i.e.*, +2 for a self-edge from i back to i, and +1 for any other edge ij with $j \neq i$. For instance, in the inset figure vertex i has degree four, even though it is contained in only three distinct edges; vertices j and k both have degree one.



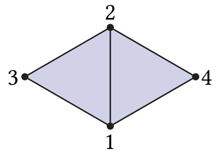
In general one must take care when translating results from the simplicial case to the setting of Δ-complexes: this more general setting often demands new nontrivial proofs even for seemingly intuitive properties (see for instance Bobenko and Springborn [BS07] and Sharp and Crane [SC20b]).

---

[1] pronounced "Delta complex"

2.2 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

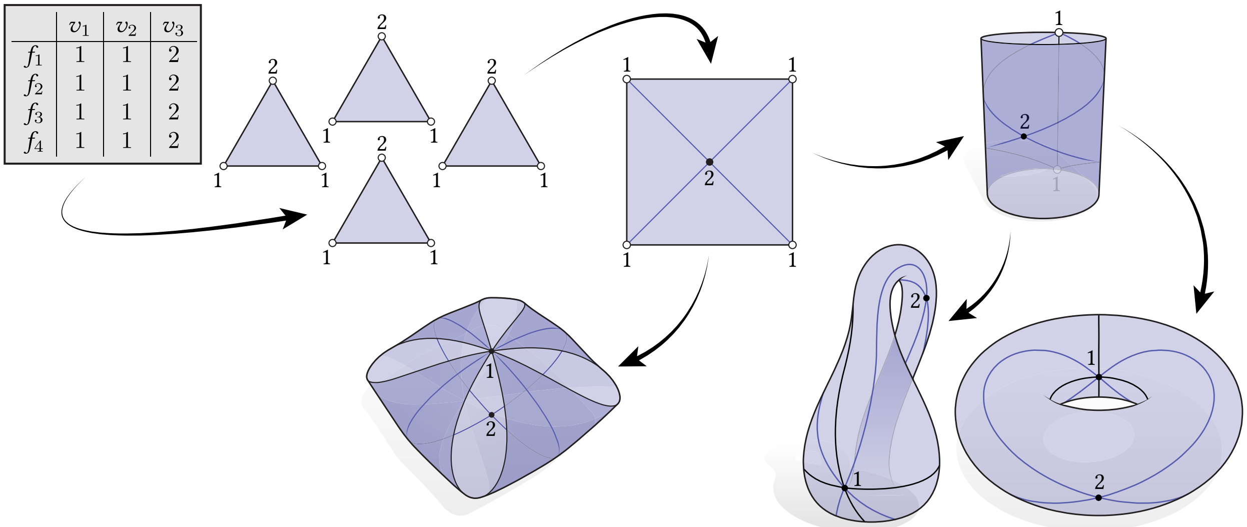


	v_1	v_2	v_3
f_1	1	2	3
f_2	2	1	4

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. 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] which all support traversals and modifications in constant time; see Botsch et al. [Bot+10] for an introduction. 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.



	v_1	v_2	v_3
f_1	1	1	2
f_2	1	1	2
f_3	1	1	2
f_4	1	1	2

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.

2.3 Geometry

A cell complex (or the data structure that encodes it) captures only the mesh connectivity. To describe the geometry, we will assign additional information to mesh elements—such as lengths, areas, or positions in space. Clearly separating a mesh into connectivity and geometry is essential to developing the intrinsic point of view. A tempting alternative perspective is to imagine that, from the beginning, the mesh is just a subset of $\mathbb{R}^n$, obtained by taking a union of triangles (or other elements). However, this viewpoint leads to confusion and inflexibility. For instance, one common conundrum is how to think about self-intersections—for instance, if two triangles intersect, is the mesh still "manifold?" By separating connectivity and geometry, manifoldness becomes a clear-cut condition on the mesh connectivity alone (given in Chapter 2); self-intersections are merely an artifact of the way this connectivity gets mapped into space [2]. Hence, one retains a valuable simplifying assumption even in the presence of intersections (contrast with volumetric approaches, where self-intersections are often a major nuisance!). More importantly, separating out geometry and connectivity is what enables us to consider the much larger space of intrinsic triangulations.

In general, *extrinsic* geometric quantities 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. Likewise, the unit normal of a triangle must be described with respect to some global coordinate system (extrinsic), whereas the area of a triangle can be described without reference to any coordinates (intrinsic). In many cases, it will be the interplay between intrinsic and extrinsic geometry that is most interesting—hence a mesh will store both kinds of data, on two different triangulations.

Historically, the extrinsic perspective is much older than the intrinsic one. For instance, at the

---

[2] In the language of differential geometry: a surface with self-intersections is not an *embedding*, though it may still be an *immersion*.

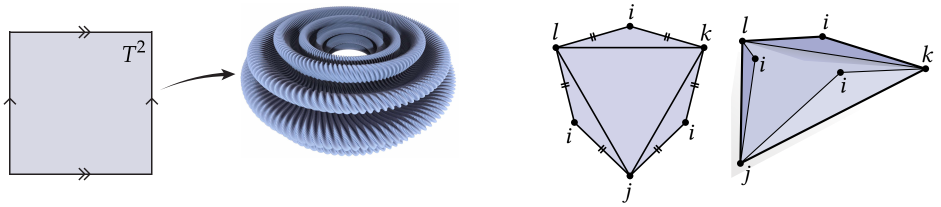


Figure 2.7: Shapes that are difficult or impossible to embed in $\mathbb{R}^n$ are often much easier to describe intrinsically. For instance, the *flat torus (left)* can be thought of intrinsically as a square with opposite edges identified—whereas finding a flat embedding into $\mathbb{R}^3$ is a major challenge (image from [Bor+13]). Likewise intrinsic triangulations that describe a perfectly reasonable metric space, such as the tetrahedron depicted in the right, may be impossible to embed into $\mathbb{R}^3$ while preserving lengths.

beginning of the 19th century a smooth surface was most typically described via a parameterization $f : \mathbb{R}^2 \to \mathbb{R}^n$, assigning explicit Cartesian coordinates to each point. The intrinsic perspective later developed by Gauss, Riemann, and others, provided a more flexible way of describing surfaces. For instance, a *flat torus* is a surface that has the connectivity of a donut, but (unlike a donut) is not curved at any point. This space is very easy to think about intrinsically: just imagine a square where walking off one side "wraps around" to the other side (Figure 2.7, *left*). But it is *incredibly* hard to find a global embedding of the flat torus into $\mathbb{R}^3$ that preserves its flat geometry: such an embedding was not constructed until the 21st century—and would be very awkward to work with directly (Figure 2.7).

In this section we carefully review both extrinsic and intrinsic encodings of mesh geometry, which will play a key role in the development of data structures for intrinsic triangulations.

2.3.1 Extrinsic Geometry

A common way to describe the extrinsic geometry of a triangle mesh is by assigning coordinates to the vertices, which are then interpolated over the rest of the mesh. In particular, suppose we have vertex coordinates $\mathbf{f} : V \to \mathbb{R}^3$. The geometry associated with an edge $ij \in E$ is then given by a straight line segment connecting the coordinates at vertices i and j. We can express this segment as a linear combination

$$\mathbf{f}_{ij} := \left\{ t_i \mathbf{f}_i + t_j \mathbf{f}_j \in \mathbb{R}^3 \,\middle|\, t_i + t_j = 1,\ t_i, t_j \geq 0 \right\}$$

Similarly, the geometry associated with a triangle is then given by

$$\mathbf{f}_{ijk} := \left\{ t_i \mathbf{f}_i + t_j \mathbf{f}_j + t_k \mathbf{f}_j \in \mathbb{R}^3 \,\middle|\, t_i + t_j + t_k = 1,\ t_i, t_j, t_k \geq 0 \right\}$$

2.3.2 Barycentric Coordinates

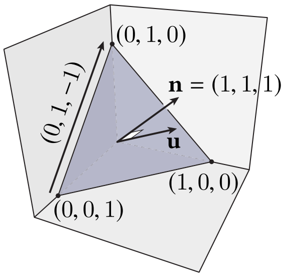
For any point $\mathbf{p} = \sum_i t_i \mathbf{f}_i$ in an edge or triangle, the values t are called the *barycentric coordinates*[3]. Barycentric coordinates are useful because they enable one to express points of a triangle without reference to how it is embedded in space—for instance, in barycentric coordinates the corners of a triangle are always expressed as $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$, and center of mass is always given by $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. For this reason, barycentric coordinates will be essential for linking together the intrinsic and extrinsic geometry of our mesh.

---

[3] Note that since barycentric coordinates always sum to one, in the case of edges we will sometimes consider use just a single barycentric coordinate s; the other coordinate is then $1 - s$.

A helpful mindset when reasoning about barycentric coordinates is to think of them not as weights, but as actually defining a reference triangle (or edge) which gets mapped to the triangle in the mesh. In particular, the set of all barycentric coordinates

$$\sigma := \left\{(t_i, t_j, t_k) \in \mathbb{R}^3 | t_i + t_j + t_k = 1,\ t_i, t_j, t_k \geq 0\right\}$$



defines a copy of the *standard triangle* σ, which is an equilateral triangle sitting in the positive octant of $\mathbb{R}^3$. One can then imagine that there is a disjoint copy σ_{ijk} of the standard simplex σ for each triangle $ijk \in F$. Since neighboring triangles share vertex coordinates, edges of these disjoint copies get mapped to the same segments in space, effectively gluing them together.

Barycentric coordinates can also be used to describe vectors tangent to a triangle. Consider in particular any two points $\mathbf{p}, \mathbf{q}$ of the same triangle, given in barycentric coordinates (s_1, s_2, s_3) and (t_1, t_2, t_3), *resp.* Since the barycentric coordinates of each point sum to one, the barycentric coordinates of their difference sums to zero:

$$\sum_{i=1}^{3}(\mathbf{p}-\mathbf{q})_i = \sum_{i=1}^{3}(s_i - t_i) = \sum_{i=1}^{3} s_i - \sum_{i=1}^{3} t_i = 1 - 1 = 0.$$

Another way to see that the components of a vector must sum to zero is to consider the normal vector $\mathbf{n} = (1, 1, 1)$ of the standard simplex σ. Any vector $u \in \mathbb{R}^3$ tangent to this simplex must be orthogonal to this normal, *i.e.*,

$$0 = \langle \mathbf{n}, \mathbf{u} \rangle = \sum_{i=1}^{3} u_i.$$

For example, in barycentric coordinates the three edge vectors can be expressed as $(-1, 1, 0)$, $(0, -1, 1)$, and $(1, 0, -1)$. Notice that the components of these vectors do not depend on the length of the edges.

2.3.3 Intrinsic Geometry

Perhaps the most fundamental difference between an intrinsic and extrinsic triangulation is that the geometry of an intrinsic triangulation is encoded by edge lengths $\ell_{ij} \in \mathbb{R}$ rather than vertex positions $\mathbf{f}_i \in \mathbb{R}^3$. Since edge lengths are invariant to extrinsic motions such as rotations or isometric bending of the surface, they cannot be used to recover extrinsic quantities such as normals, dihedral angles, or discrete mean curvature. They can however be used to extract important intrinsic information, such as areas, interior angles of triangles, and discrete Gaussian curvature.

The initial values for these lengths will most often be obtained by measuring the distance between vertex positions, *i.e.*, by letting $\ell_{ij} := |\mathbf{f}_i - \mathbf{f}_j|$. However, subsequent operations on the triangulation (such as intrinsic edge flips—see Section 2.3.4) will then modify the initial lengths in ways that may not correspond to any operation on the extrinsic mesh.

More explicitly, the geometry of an intrinsic triangulation is given by an assignment $\ell : E \to \mathbb{R}_{>0}$ of positive lengths to edges[4]. These lengths must satisfy three *triangle inequalities* in each face $ijk \in F$, namely

$$\begin{array}{rcl} \ell_{ij} + \ell_{jk} & \geq & \ell_{ki}, \\ \ell_{jk} + \ell_{ki} & \geq & \ell_{ij}, \\ \ell_{ki} + \ell_{ij} & \geq & \ell_{jk}. \end{array}$$

---

[4] In some cases it is fine to consider zero-length edges, though for simplicity we will generally assume that all lengths are positive.

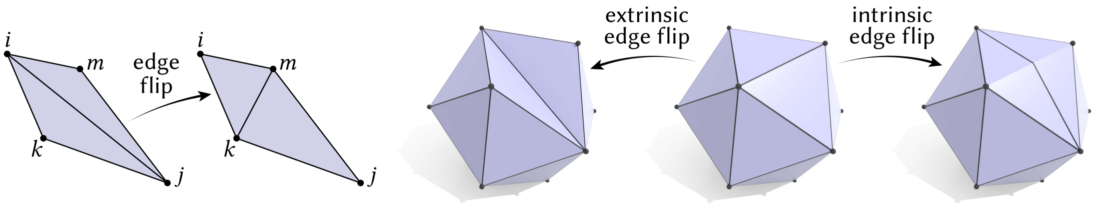


Figure 2.8: *Left:* an edge flip modifies the connectivity of a triangle mesh, replacing triangles ijk, jim with triangles kmi, mkj. *Right:* whereas an ordinary extrinsic edge flip changes the mesh geometry, an intrinsic flip leaves the original geometry untouched.

Any triple of lengths satisfying these inequalities determines a triangle in the plane—hence, one can derive other geometric quantities (areas, angles, *etc.*) from just the three edge lengths (see Section 2.3.6 for further discussion).

2.3.4 Edge Flips

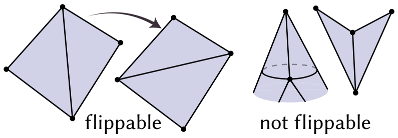
A basic operation in mesh processing is an *edge flip*, which provides the starting point for many of the algorithms we will study in Sections 4 and 5. This operation is apparently so natural that it has been (re-)invented many times, and given many different names: an *exchange* [Law72], *(edge) flip* [BJW90, Figure 1], *diagonal switch* [Lei99, Figure 3.5], *diagonal flip* [Eis85], *diagonal transformation* [NN+93], *Whitehead move* [Riv94b, Figure 2], *stellar exchange* [Pac90, Definition 2], *elementary move* [Mos88, Figure 20], *2-2 Pachner move* [Pac90], *2-2 bistellar flip* [Gli05, Section 3.3], or just *2-2 flip* [Mor96].

To flip an edge $ij \in E$ we replace the two triangles ijk, jim containing ij with two triangles kmi, mkj (Figure 2.8, *left*). This description tells us how to update the connectivity, but how should we define the new geometry? Traditionally, one just adopts the ordinary extrinsic geometry *à la* Section 2.3.1: the vertex positions $\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_k, \mathbf{f}_l \in \mathbb{R}^n$ are now linearly interpolated over the new triangles kmi, mkj, rather than the original triangles ijk, jim. However, unless the two triangles share a common plane, an extrinsic edge flip will change the geometry of the surface, as seen in Figure 2.8, *center*. Hence, if we try to use an edge flip to, say, improve element shape, we do so at the cost of a lower-quality approximation of the original geometry. This situation highlights a fundamental tension between element quality and approximation error that pervades ordinary extrinsic mesh processing.

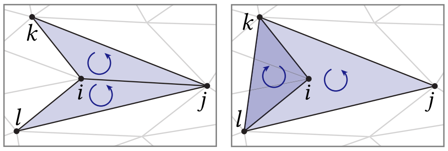
An alternative is to perform an *intrinsic edge flip* (Figure 2.8, *right*). Intuitively, rather than connect vertices k and m along a straight line segment through Euclidean space, we connect them by a straight path *along* the domain (formally, a *geodesic*—see Section 2.4.1). To do so, we update the edge lengths $\ell : E \to \mathbb{R}_{>0}$ that describe the intrinsic geometry of our surface. In particular, we can imagine that we lay out the two original triangles ijk, jim in the plane; the distance between $\mathbf{f}_k$ and $\mathbf{f}_m$ in this layout defines the new edge length ℓ_{km}. In fact, one can compute ℓ_{km} directly from the known edge lengths according to a simple formula—see Appendix A for further discussion.

Intrinsic Edge Flips Preserve Geometry. The key benefit of intrinsic edge flips is that they exactly preserve the metric of the surface described by the triangulation. Hence, quantities like surface area, the shortest path between any pair of points, and the angle sum around each vertex is exactly preserved. One may therefore freely flip edges toward a more desirable triangulation, without "damaging" the geometric approximation of the underlying surface. In turn, one side-steps some of the fundamental trade offs encountered in extrinsic mesh processing, opening the door to new retriangulation strategies—as discussed in Chapter 4.

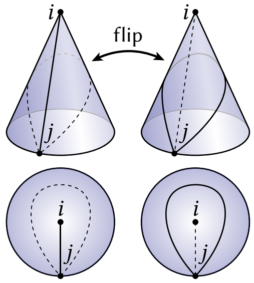
Flippable Edges. Importantly, not all edges can be flipped. Assuming the triangulation is treated as a general Δ-complex, an edge ij can be flipped if and only if it satisfies two conditions:



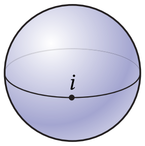
1. triangles ijk and jil form a convex quad, and
2. the endpoints i and j both have degree at least two.



If the first condition does not hold, then flipping the edge essentially causes the mesh to fold over itself. See for instance the inset figure, where flipping the edge ij yields a new triangle klj covering the convex hull of the original figure, and a smaller triangle lki sitting on top of it. In the original configuration, both triangles have the same orientation relative to the background domain, but in the new configuration the orientation of lki is reversed. Moreover, since the triangles overlap, the total area of the original mesh is not preserved, nor is the total angle around vertices i, j, and k (for instance, if we assume the original triangulation is planar then these vertices effectively acquire curvature—see Equation 2.3).



If the second condition does not hold, then flipping the edge would lead to a degree-zero vertex, which cannot be represented by a triangulation (even a Δ-complex). Consider for instance the inset figure, which is drawn both from the side and from above. On the left we have a cone obtained by gluing a triangle to itself along one edge; edge ij hence has a degree-1 endpoint at vertex i. Attempting to flip this edge causes problems for both connectivity and geometry. For one thing, in the flipped figure vertex i now has degree zero, and the initial triangle has been transformed into two faces: a one-sided *monogon* (inside) and a two-sided *digon* (outside). Geometrically, we now have a cone point on the interior of the monogon, meaning that this face is no longer intrinsically flat. On the whole, it becomes impossible to describe this space as a collection of flat Euclidean triangles encoded by edge lengths.



Note that one might also worry about flipping an edge with degree-two endpoints, 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 cannot be triangles. For instance, the inset figure shows a topological complex with a degree-2 self edge, but both faces in this complex are monogons.

Flip Graph. The *flip graph* is a graph whose vertices correspond to intrinsic triangulations, and two triangulations are connected if they are related by a single edge flip. Analyzing the correctness and complexity of many edge flip-based algorithms amounts to proving various properties of the flip graph.

A basic property is *connectivity* of the flip graph, *i.e.*, given two triangulations T_1, T_2 of the same vertex set, can T_1 always be transformed into T_2 by a sequence of edge flips? The answer depends on what kind of flips one considers, *e.g.*, whether one considers only the combinatorics of the edge graph, or also views the flips as transforming the geometry of some domain. [Wag36] showed that the flip graph is connected in the combinatorial case; [Law72] shows that it is connected for point sets in the Euclidean plane; and Mosher [Mos88, Connectivity Theorem for Elementary Moves] gives an algorithm for flipping between *hyperbolic* triangulations[5] using the normal coordinates described in Section 3.5.1.

---

[5] Mosher's algorithm actually works on combinatorial triangulations as well, as all valid combinatorial flips are also possible in hyperbolic triangulations. This is one way in which Euclidean triangulations are actually *more* complicated than ideal hyperbolic triangulations—the conditions on a valid flip are more restrictive.

The flip graph of Euclidean intrinsic triangulations is connected as a consequence of the Delaunay triangulation (Section 4.1)—all triangulations can be flipped to a Delaunay configuration [BS07], and these configurations are necessarily connected by flips between polygons inscribed in circles.

2.3.5 Cone Metric

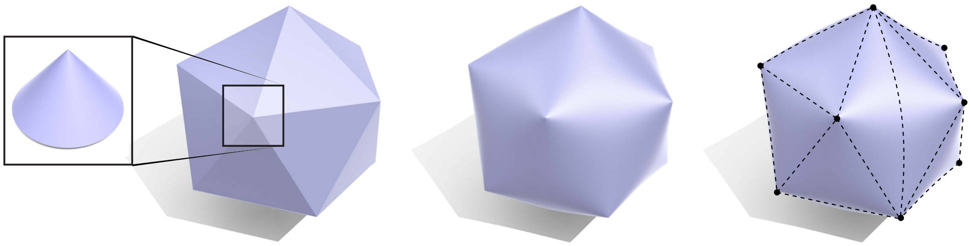


Figure 2.9: *Left:* intrinsically, a small neighborhood around any vertex of a polyhedral surface is indistinguishable from a round circular cone. For instance, a vertex made from triangular pieces of paper can be easily smoothed out into a circular cone without stretching or ripping. *Center:* a good mental "cartoon" of an intrinsic triangulation is hence a surface where the edges are completely smooth, and only the vertices are visible. *Right:* from this perspective, no one triangulation is special—there are many intrinsic triangulations that describe the exact same geometry.

Since neighboring triangles in an intrinsic triangulation have equal edge lengths, they can be "glued together" to define the global geometry of the surface[6]. Unlike the extrinsic case, however, these triangles do not sit in $\mathbb{R}^3$, but rather just define an abstract metric where one knows how to walk from one triangle into the next. There are two very important observations to make about this metric:

1. The geometry in a small neighborhood of any vertex i is indistinguishable from a smooth circular cone. Consider, for instance, physically gluing together triangular pieces of paper around a common point—gluing the last edge to the first forms a paper cone, and since the paper is flexible, this cone can be smoothed out into a circular shape (without distorting lengths along the surface).
2. The edges are "invisible" from an intrinsic point of view: as one walks from one triangle to the next across an edge, there is no geometric quantity that can be measured to determine the moment when the edge is crossed. One way to make this observation is to consider an intrinsic edge flip (as described in Section 2.3.4). Whether we lay out the original triangles ijk, jim or the new triangles kmi, mkj in the plane, they both describe an identical quadrilateral piece of the surface $ikjm$.

For this reason, the space described by the complex T and the edge lengths ℓ is called a *Euclidean cone metric*: it looks like a cone in the vicinity of each point, and otherwise just looks like the flat Euclidean plane everywhere else (Figure 2.9). The edges initially used to define a cone metric are then completely superficial: there are many other triangulations that could be used to describe the exact same geometry. In other words, the initial triangulation is merely "scaffolding" used to get our hands on a definition of the shape. But we can freely change this triangulation without changing the geometry of the underlying surface—just as retriangulating (say) a rectangular region of the plane does not change its shape.

---

[6] A bit more formally, the Riemannian metric of the surface is defined by taking the quotient of the disjoint union of the individual triangles, where two points are equivalent if they share barycentric coordinates along a common edge of the complex K.

2.3.6 Length Based Formulas

In an ordinary extrinsic mesh, local geometric quantities such as areas, lengths, angles, *etc.*, are expressed in terms of the vertex coordinates $\mathbf{f}_i$. For instance, the area A_{ijk} of a triangle $ijk \in F$ can be expressed by taking half the magnitude of the cross product of two of the edge vectors:

$$A_{ijk} = \tfrac{1}{2}|(\mathbf{f}_i - \mathbf{f}_j) \times (\mathbf{f}_i - \mathbf{f}_k)|.$$

Since these expressions use extrinsic operations on coordinates, we must seek other ways to express geometric quantities in the intrinsic setting.

The three edge lengths of a triangle are sufficient to determine any intrinsic property of a triangle—in fact, a good way to understand which quantities are intrinsic (versus extrinsic) is to ask whether they can be obtained from edge lengths alone. For instance, the normal vector of a triangle cannot be determined from lengths, because there are motions of a triangle (such as rotations) that change the normal but do not change the lengths. Likewise, the dihedral angle at an edge must be an extrinsic property, since one can bend two triangles along a shared edge without changing any of the edge lengths. On the other hand, the area A_{ijk} of any triangle $ijk \in F$ is invariant to length-preserving motions, and can be deduced via *Heron's formula*

$$A_{ijk} = \sqrt{s(s-\ell_{ij})(s-\ell_{jk})(s-\ell_{ki})}, \tag{2.1}$$

where here $s = (\ell_{ij} + \ell_{jk} + \ell_{ki})/2$ is the *semi-perimeter*. The interior angle θ_i^{jk} at corner i of triangle ijk is likewise intrinsic, and can be obtained via the *law of cosines*:

$$\ell_{ij}^2 - \ell_{jk}^2 + \ell_{ki}^2 = 2\ell_{ik}\ell_{ki}\cos\theta_i^{jk}. \tag{2.2}$$

Hence, the interior angle itself can be expressed as

$$\theta_i^{jk} = \arccos\left(\frac{\ell_{ij}^2 - \ell_{jk}^2 + \ell_{ki}^2}{2\ell_{ik}\ell_{ki}}\right),$$

though quite often there are ways to avoid evaluating the arc cosine directly. Appendix A gives a more extensive list of such formulas.

Using these local, per-element formulas one can deduce more global quantities by taking sums over mesh elements. For instance, the total surface area is just $\sum_{ijk\in F} A_{ijk}$. Another very important example of an intrinsic quantity is the *vertex angle deficit*

$$\Omega_i := 2\pi - \sum_{ijk} \theta_i^{jk}, \tag{2.3}$$

which measures how much the sum of angles around a vertex deviates from the sum of 2π that one would find in the plane. This quantity provides a notion of Gaussian curvature for polyhedral surfaces—since the angles θ_i^{jk} can be expressed in terms of edge lengths, this notion of curvature can, remarkably enough, be measured without embedding the surface in space[7].

---

[7] This fact is captured in the smooth setting by Gauss' *theorema egregium* or "remarkable theorem."

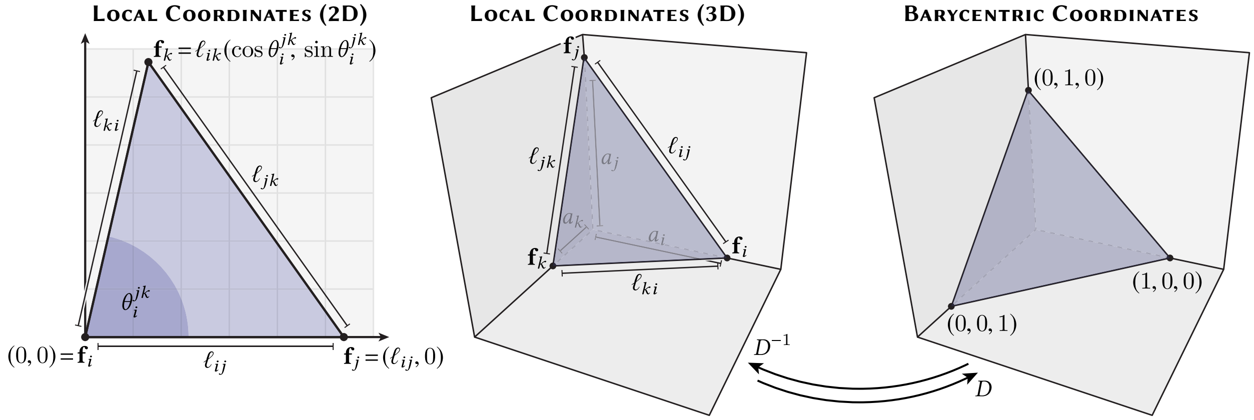

Local Coordinates (2D)
$\mathbf{f}_k = \ell_{ik}(\cos\theta_i^{jk}, \sin\theta_i^{jk})$
ℓ_{ki}
ℓ_{jk}
θ_i^{jk}
$(0,0) = \mathbf{f}_i$
ℓ_{ij}
$\mathbf{f}_j = (\ell_{ij}, 0)$
Local Coordinates (3D)
$\mathbf{f}_j$
ℓ_{jk}
a_j
ℓ_{ij}
a_k
a_i
$\mathbf{f}_k$
$\mathbf{f}_i$
ℓ_{ki}
Barycentric Coordinates
$(0,1,0)$
$(1,0,0)$
$(0,0,1)$
D^{-1}
D

Figure 2.10: Given the three edge lengths of a triangle $\ell_{ij}, \ell_{jk}, \ell_{ki}$, one can always find a corresponding set of vertex positions $\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_k$ in either 2D or 3D. If the vertices of the 3D embedding are placed along the principal axes, then they are related to the barycentric coordinates by a simple diagonal scaling D.

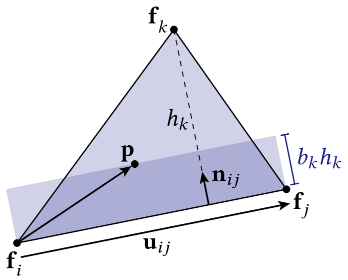
2.3.7 Local Coordinates

2D Local Coordinates. In cases where an intrinsic quantity has no obvious expression in terms of edge lengths, one can still construct local vertex coordinates for the triangle in $\mathbb{R}^n$ and take measurements of this triangle via ordinary expressions from vector calculus. As depicted in Figure 2.10, *left*, we can construct a triangle with edge lengths $\ell_{ij}, \ell_{jk}, \ell_{ki} \in \mathbb{R}_{>0}$ via three points in the plane:

$$\begin{aligned} \mathbf{f}_i &:= (0,0), \\ \mathbf{f}_j &:= (\ell_{ij}, 0), \\ \mathbf{f}_k &:= \ell_{ki}(\cos\theta_i^{jk}, \sin\theta_i^{jk}), \end{aligned}$$

where θ_i^{jk} is obtained from the law of cosines (Equation 2.2). In other words, we put the first vertex at the origin, and the second at a distance ℓ_{ij} along the x-axis. The third vertex is likewise placed a distance ℓ_{ki} from the origin, making the appropriate angle θ_i^{jk} with the first edge.

From 2D to Barycentric Coordinates. Often we will need to express a point $\mathbf{p} \in \mathbb{R}^2$ in a triangle in barycentric coordinates. There are several ways to express these coordinates.


$\mathbf{f}_k$
h_k
$\mathbf{p}$
$b_k h_k$
$\mathbf{n}_{ij}$
$\mathbf{f}_j$
$\mathbf{u}_{ij}$
$\mathbf{f}_i$

One way is to measure the distance from each edge, normalized by the corresponding triangle height. More explicitly, if $\mathbf{u}_{ij} := \mathbf{f}_j - \mathbf{f}_i$ is the vector along edge ij in 2D local coordinates, and $\mathcal{J}$ denotes a 90-degree rotation in the counter-clockwise direction[8], then we can write the inward unit normal along ij as

$$\mathbf{n}_{ij} = \mathcal{J}\frac{\mathbf{u}_{ij}}{|\mathbf{u}_{ij}|} = \mathcal{J}\mathbf{u}_{ij}/\ell_{ij}.$$

The kth barycentric coordinate of $\mathbf{p}$ is then given by

$$b_k = \frac{1}{h_k}\langle \mathbf{n}_{ij}, \mathbf{p} - \mathbf{f}_i \rangle,$$

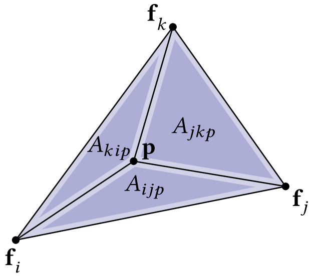
---

[8] A simple way to express this operation is to just exchange the two components and then negate the first one: $\mathcal{J}(x,y) = (-y,x)$.

where h_k is the height of vertex k above edge ij, *i.e.*, the barycentric coordinate is equal to the fraction of the height covered by the point $\mathbf{p}$ (yielding a value between zero and one). We can simplify this expression by noting that $A_{ijk} = \frac{1}{2}\ell_{ij}h_k$, *i.e.*, the area of a triangle is one-half its base length times its height (with respect to any edge). Hence,

$$b_k = \frac{1}{\ell_{ij}h_k}\langle \mathcal{J}\mathbf{u}_{ij}, \mathbf{p}-\mathbf{f}_i\rangle = \frac{1}{2A_{ijk}}\langle \mathcal{J}\mathbf{u}_{ij}, \mathbf{p}-\mathbf{f}_i\rangle = \frac{1}{2A_{ijk}}\langle \mathcal{J}(\mathbf{f}_j-\mathbf{f}_i), \mathbf{p}-\mathbf{f}_i\rangle,$$

which involves only the point $\mathbf{p}$, the vertex positions $\mathbf{f}$, and the triangle area A_{ijk}. Note that these expressions are linear in $\mathbf{p}$, and yield barycentric coordinates $(1,0,0), (0,1,0), (0,0,1)$ when $\mathbf{p}$ is placed at $\mathbf{f}_i, \mathbf{f}_j, \mathbf{f}_k$, respectively, *i.e.*, they do indeed give the barycentric coordinates as defined in Section 2.3.2.



Taking this calculation further reveals that the barycentric coordinates can be expressed as ratios of triangle areas. Consider that $\langle \mathcal{J}\mathbf{u}, \mathbf{v}\rangle = \mathbf{u} \times \mathbf{v}$, where here $\mathbf{u} \times \mathbf{v} := \mathbf{u}_1\mathbf{v}_2 - \mathbf{u}_2\mathbf{v}_1$ gives the nonzero component of the cross product of two vectors in the plane, interpreted as vectors in 3D. Also recall that that the area of a triangle is equal to half the cross product of two of its edge vectors. Hence,

$$b_k = \frac{1}{2A_{ijk}}(\mathbf{f}_j - \mathbf{f}_i) \times (\mathbf{p} - \mathbf{f}_i) = \frac{A_{ijp}}{A_{ijk}},$$

where A_{ijp} is the area of the triangle made by $\mathbf{f}_i$, $\mathbf{f}_j$ and the point $\mathbf{p}$.

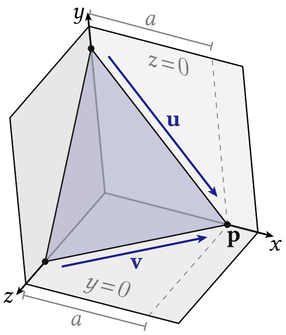
For near-degenerate triangles (*e.g.*, small/large angles, or lengths near zero) it can be more accurate to directly solve a linear system for the barycentric coordinates. Consider in particular the matrix[9]

$$A := \begin{bmatrix} \mathbf{f}_i^x & \mathbf{f}_j^x & \mathbf{f}_k^x \\ \mathbf{f}_i^y & \mathbf{f}_j^y & \mathbf{f}_k^y \\ 1 & 1 & 1 \end{bmatrix}.$$

If we let $\overline{\mathbf{p}} := (\mathbf{p}^x, \mathbf{p}^y, 1)$, then solving the matrix equation

$$A\mathbf{b} = \overline{\mathbf{p}}$$

yields barycentric coordinates $\mathbf{b} = (b_i, b_j, b_k)$ that describe the point $\mathbf{p}$, and such that $b_i + b_j + b_k = 1$. Likewise, given a vector $\overline{\mathbf{v}} := (\mathbf{v}^x, \mathbf{v}^y, 0)$, the corresponding vector in barycentric coordinates is found by solving $A\mathbf{w} = \overline{\mathbf{v}}$. We find that solving this system with a (dense) QR solver with Householder pivoting works quite well in practice.



3D Local Coordinates. Alternatively, we can find 3D vertex coordinates along the three axes that realize the given lengths (Figure 2.10, *center*)—which will also provide a straightforward way to convert points and vectors on our triangle to barycentric coordinates. In particular, suppose that $\mathbf{u} = (a, b, 0)$ and $\mathbf{v} = (a, 0, c)$ are two edge vectors incident on a common vertex $\mathbf{p} = (a, 0, 0)$ a distance a along the x-axis (see inset). Then we have

$$\langle \mathbf{u}, \mathbf{v}\rangle = a \cdot a + b \cdot 0 + 0 \cdot c = a^2.$$

Hence, the distance of $\mathbf{p}$ along the x-axis is just $a = \sqrt{\langle \mathbf{u}, \mathbf{v}\rangle}$. If we are given just three edge lengths then we do not yet know the edge vectors of the triangle in $\mathbb{R}^3$—but can nonetheless compute their inner products, via the law of cosines. Recall that if two vectors $\mathbf{u}$ and $\mathbf{v}$ make an angle θ, then $\langle \mathbf{u}, \mathbf{v} \rangle = |\mathbf{u}||\mathbf{v}| \cos\theta$. Hence, applying the law of cosines (Equation 2.2) yields

---

[9] Geometrically this matrix represents a linear map from barycentric coordinates to 2D homogeneous coordinates.

$$a^2 = \langle \mathbf{u}, \mathbf{v} \rangle = \frac{|u|^2 + |v|^2 - |w|^2}{2},$$

where $w := u - v$ is the third edge of the triangle.

We can hence draw a triangle with edge lengths $\ell_{ij}, \ell_{jk}, \ell_{ki}$ as three points

$$\begin{array}{rcl} \mathbf{f}_i & := & (a_i, 0, 0), \\ \mathbf{f}_j & := & (0, a_j, 0), \\ \mathbf{f}_k & := & (0, 0, a_k), \end{array}$$

where

$$a_i := \sqrt{\frac{\ell_{ij}^2 - \ell_{jk}^2 + \ell_{ki}^2}{2}},$$

and a_j, a_k are obtained by permuting indices[10].

From 3D to Barycentric Coordinates. The 3D local coordinates have a close relationship to barycentric coordinates: by just sliding each point $\mathbf{f}_i$ along its respective axis to a distance "1" from the origin, we transform this triangle into the standard triangle σ. More explicitly, consider the diagonal matrix

$$D := \begin{bmatrix} a_i & 0 & 0 \\ 0 & a_j & 0 \\ 0 & 0 & a_k \end{bmatrix}.$$

Given a point $\mathbf{b} = (b_i, b_j, b_k)$ in barycentric coordinates, $\mathbf{p} = D\mathbf{b}$ gives the corresponding point on the triangle, and likewise, $\mathbf{b} = D^{-1}\mathbf{p}$ (where D is easily inverted by just taking the reciprocal of the diagonal elements). Likewise, if $\mathbf{w} = (w_i, w_j, w_k)$ is a tangent vector in barycentric coordinates, then $\mathbf{u} = D\mathbf{w}$ is the corresponding vector in local 3D coordinates and vice-versa.

Local Coordinates and Length-Based Formulas. The local embeddings provide a unified way to derive expressions that depend only on edge lengths (like those given at the beginning of this section). Suppose, for instance, we have a vector $\mathbf{u}$ expressed in barycentric coordinates (u_i, u_j, u_k) and want to measure its length. Simply taking the Euclidean norm $\sqrt{u_i^2 + u_j^2 + u_k^2}$ will not give the correct length, since the standard triangle σ_{ijk} gets stretched out when we map it to the true geometric triangle $\mathbf{f}_{ijk}$. Instead, we can account for this stretching by measuring the length of the vector $\mathbf{v} = D\mathbf{u}$. In particular,

$$|\mathbf{v}|^2 = x_i^2 u_i^2 + x_j^2 u_j^2 + x_k^2 u_k^2 = \tfrac{1}{2}\left((\ell_{ij}^2 - \ell_{jk}^2 + \ell_{ki}^2)u_i + (\ell_{jk}^2 - \ell_{ki}^2 + \ell_{ij}^2)u_j + (\ell_{ki}^2 - \ell_{ij}^2 + \ell_{jk}^2)u_k\right).$$

Noting that $u_k = -u_i - u_j$ and rearranging terms then yields an expression for the norm involving only the three edge lengths[11]:

$$|\mathbf{v}|^2 = -\left(\ell_{ij}^2 u_i u_j + \ell_{jk}^2 u_j u_k + \ell_{ki}^2 u_k u_i\right).$$

---

[10] Since these values will be different for each corner of each triangle, it may be better to write them as a_i^{jk}, a_j^{ki} and a_k^{ij}; we stick to single indices here for brevity.

[11] In spite of the negative sign, this expression really is positive for all vectors with coordinates satisfying $u_i + u_j + u_k = 0$.

Other quantities can often be derived in a similar way, avoiding the need to construct an explicit local embedding. Avoiding the local embedding saves some small amount of computation, but more importantly it can help to avoid numerical operations that may be inaccurate in extreme situations, *e.g.*, when edge lengths are close to zero, or interior angles are close to zero or π.

2.4 Tangent Vectors

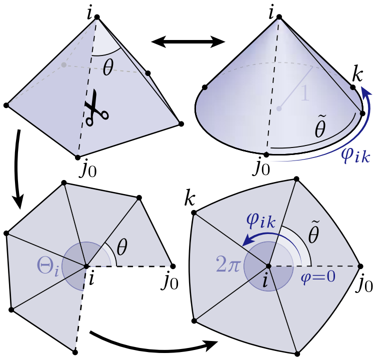


Figure 2.11: Local coordinate system for tangent vectors at vertices.

Vectors tangent to the surface play an important role in geometry processing. In our correspondence data structures, they help to describe the direction of intrinsic edges over the extrinsic mesh and vice versa. At points interior to a face or edge, where the surface is intrinsically flat, tangent vectors have a straightforward encoding, *e.g.*, using Cartesian or barycentric coordinates as discussed in Section 2.3.2. At vertices, however, where the surface locally looks like a cone, we need a different description.

For an extrinsic mesh, one idea is to simply pick some tangent plane at each vertex—however, the best choice of tangent plane is not always clear (much like picking a vertex normal), and this approach will not work in the intrinsic setting. A more canonical approach is to encode tangent vectors at a vertex i in local polar coordinates $(r, \varphi) \in \mathbb{R}_{\geq 0} \times [0, 2\pi)$, *i.e.*, as a radius r and angle φ relative to some reference direction $\varphi = 0$. For instance, suppose we pick some arbitrary (but fixed) edge ij_0 to serve as the reference direction, and let

$$\tilde{\theta}_i^{jk} := \frac{2\pi}{\Theta_i} \theta_i^{jk},$$

where $\Theta_i := \sum_{ijk} \theta_i^{jk}$. Then the directions of all the outgoing edges with respect to this local coordinate system are given by

$$\varphi_{ij_a} = \frac{2\pi}{\Theta_i} \sum_{n=0}^{a-1} \tilde{\theta}_i^{j_n, j_{n+1}},$$

In other words, we take the cumulative angle sum up to edge ij_a, normalized by the total angle around the vertex. Geometrically, these are the angles we would get by isometrically smoothing a polyhedral vertex neighborhood out into a smooth cone (Figure 2.11, top). Alternatively, we can imagine that we cut the vertex open, flatten it into the plane, and then stretch it[12] into a closed figure (Figure 2.11, bottom), effectively normalizing all vectors to the range $[0, 2\pi)$.

2.4.1 Geodesics

A *geodesic* is the generalization of the notion of a "straight line" to a curved surface. On a smooth surface, a geodesic is a curve γ that is both straightest and locally shortest. *Straightest* essentially means that the curve experiences no tangential acceleration—extrinsically, the direction of the curve changes purely in order to remain on the surface. *Locally shortest* means that the restriction of γ to a sufficiently small neighborhood around each point $\mathbf{p} \in \gamma$ is the shortest possible path between

---

[12] In fact, stretching this figure out via the complex map $z \mapsto z^{2\pi/\Theta_i}$ effectively defines a Riemannian atlas that gives the polyhedral surface not only a smooth structure, but also a conformal structure, as studied by Troyanov [Tro91].

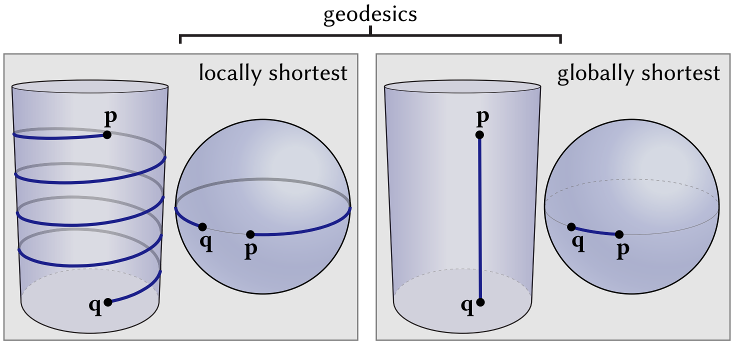


Figure 2.12: A *geodesic* is often confused with a “shortest path” between points **p** and **q**, but in general a geodesic can be any path that locally minimizes length, or equivalently, that exhibits no tangential acceleration.

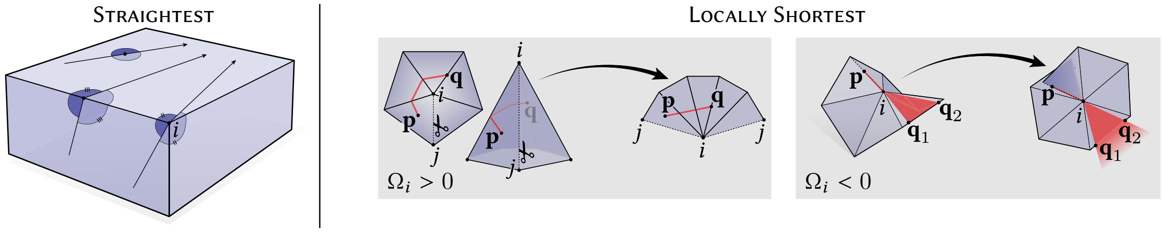


Figure 2.13: There are two distinct ways to define geodesics on polyhedra: as “straightest” curves, with equal angle on both sides *(left)*, or as “locally shortest” curves that cannot be pulled tighter *(right)*. A curve through a positively-curved vertex ($\Omega_i > 0$) can always be made shorter, whereas there are many ways to continue a path through a negatively-curved vertex ($\Omega_i < 0$) while remaining locally shortest—for example, a path from **p** through i and then to either $\mathbf{q}_1$ or $\mathbf{q}_2$ is locally shortest. For intrinsic triangulations, we need only consider locally shortest geodesics. *(Leftmost figure adapted from [PS06])*

the endpoints of this smaller curve. Importantly, a geodesic need not be globally shortest curve: for instance, a straightest curve that winds many times around a cylinder is still a geodesic. The globally shortest geodesic between two points is called a *minimal geodesic*. Figure 2.12 shows some examples.

On a polyhedral surfaces, straightest paths are not always locally shortest (Figure 2.13). In particular, a *straightest* geodesic through a vertex can be defined as a curve with equal angle on both sides [PS06]. However, a path through a positively-curved vertex ($\Omega_i > 0$) can always be made shorter by going around the vertex. Likewise, for any segment entering a negatively-curved vertex ($\Omega_i < 0$) there are *many* outgoing segments that make a locally shortest path. For intrinsic triangulations, we need only consider locally shortest paths. In particular, edges of an intrinsic triangulation will never pass through a cone point; geodesics computed by the algorithm in Chapter 5 are locally shortest.

Algorithmically, there are several distinct types of problems where one might need to compute a geodesic. One is an *initial value problem* where one must trace out a geodesic starting at a given point **p** in a given direction **u**—this problem is easily solved by evaluating the *discrete exponential map*, as discussed in Section 2.4.2, and will be essential for tracing out edges of one triangulation over another. One might also seek to compute a minimal geodesic between two distinct points **p**, **q** (see [Cra+20] for a detailed survey), or to shorten an existing curve until it becomes a locally shortest geodesic—Chapter 5 describes an algorithm for this final task that takes advantage of intrinsic triangulations.

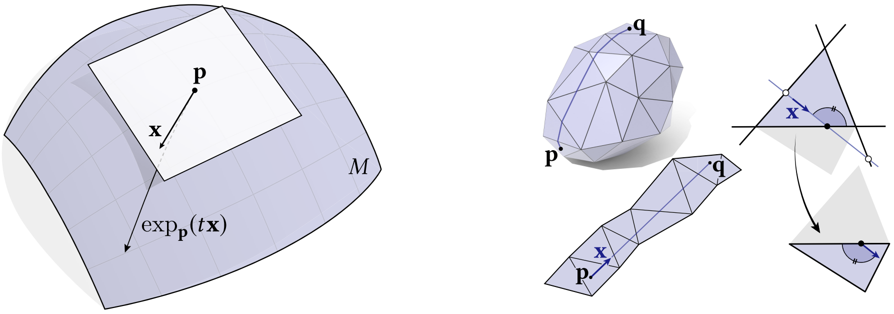
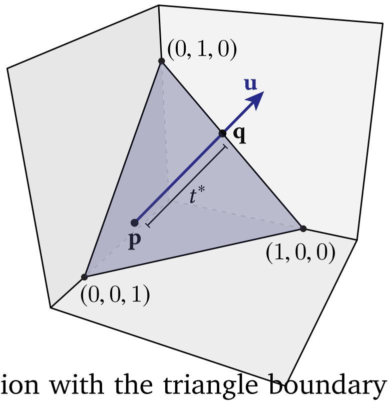


Figure 2.14: *Left:* the exponential map simply walks along the surface in a given direction $\mathbf{x}$, starting at a given point $\mathbf{p}$. *Right:* on a triangle mesh, the exponential map can be computed by tracing out a straight line through a strip of triangles. Each step amounts to performing ray-edge intersections, then transporting the ray direction vector to the next triangle.

2.4.2 Exponential Map

Intuitively, the *exponential map* simply captures the idea of "walking straight along the surface" (Figure 2.14, *left*). More precisely, for any surface M, $\exp_{\mathbf{p}}(\mathbf{x})$ gives the point $\mathbf{q} \in M$ reached by starting at a point $\mathbf{p} \in M$ and traveling in a straight or *geodesic* path for a distance $L := |\mathbf{x}|$, starting in the direction $\mathbf{u} := \mathbf{x}/|\mathbf{x}|$ (see inset). On a polyhedral surface the exponential map traces out straight line segments in a sequence of triangles—if these triangles are laid out in the plane, it is simply a straight line. If the path reaches a vertex, it continues in the direction that maintains equal angles on both sides [PS06]—though as noted before when working with intrinsic triangulations all the paths we consider will terminate at vertices (or at interior points).

To evaluate the map on a polyhedral surface (Figure 2.14, *right*), one can start in the triangle ijk containing the point p and find the intersection of the ray $\mathbf{r}(t) = \mathbf{p} + t\mathbf{u}$ with the three edges, *i.e.*, the smallest positive value of t for which $r(t)$ is on the boundary of ijk. The vector $\mathbf{u}$ is then transported to the neighboring triangle, and the process is repeated until the total distance traveled is equal to L. Below we describe this procedure in greater detail, letting $\mathbf{f}_i^{jk}, \mathbf{f}_j^{ki}, \mathbf{f}_k^{ij} \in \mathbb{R}^n$ be 2D or 3D local coordinates for any triangle ijk.

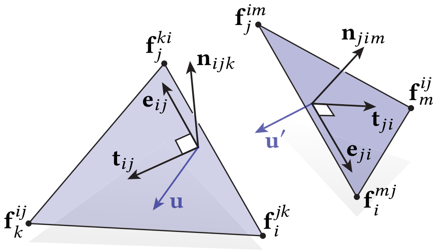


Ray-Edge Intersections. To compute the ray-edge intersections, assume that the point $\mathbf{p}$ and the unit vector $\mathbf{u}$ are given in local coordinates, and let $\mathbf{b}, \mathbf{w}$ be the corresponding barycentric coordinates (as described in Section 2.3.7). Finding an intersection with the three edges then amounts to finding the smallest positive t values at which each of the three barycentric coordinates vanishes—viewed from the perspective of the standard triangle, we are just looking for the points where ray hits the planes $x = 0$, $y = 0$, and $z = 0$. Solving the ray equation $\mathbf{r}(t) = 0$ for these three planes then amounts to just evaluating

$$t_n := -\mathbf{b}_n/\mathbf{w}_n, \quad n \in \{i, j, k\}.$$

Letting t^* be the smallest positive value from this set, the intersection with the triangle boundary is then $\mathbf{a} := \mathbf{b} + t^*\mathbf{w}$. Suppose, without loss of generality, that the ray intersects edge ij. Then the intersection will have barycentric coordinates $(b_i, b_j, 0)$, and the origin of the ray in the next triangle (in local coordinates) is given by

$$\mathbf{p}' = b_i \mathbf{f}_i^{mj} + b_j \mathbf{f}_j^{im}.$$



Parallel Transport. To transport the ray direction vector from one triangle to the next, consider an orthonormal basis for ijk aligned with edge ij:

$$\mathbf{e}_{ij} := \frac{\mathbf{f}_j^{ik} - \mathbf{f}_i^{jk}}{|\mathbf{f}_j^{ik} - \mathbf{f}_i^{jk}|} \quad \text{and} \quad \mathbf{t}_{ij} := \widehat{\mathbf{n}}_{ijk} \times \mathbf{e}_{ij}.$$

Here $\widehat{\mathbf{n}}_{ijk}$ is the unit vector in the normal direction

$$\mathbf{n}_{ijk} := (\mathbf{f}_j^{ki} - \mathbf{f}_i^{jk}) \times (\mathbf{f}_k^{ij} - \mathbf{f}_i^{jk});$$

a cross product with this vector just represents a 90-degree rotation in the plane of the triangle[13]. Let $\mathbf{e}_{ji}, \mathbf{t}_{ji}$ be the corresponding basis for jim, aligned with edge ji. We can then transport the vector $\mathbf{u} \in \mathbb{R}^n$ tangent to ijk to the corresponding vector $\mathbf{u}' \in \mathbb{R}^n$ tangent to jim via

$$\mathbf{u}' = -\big(\langle \mathbf{u}, \mathbf{e}_{ij} \rangle \mathbf{e}_{ji} + \langle \mathbf{u}, \mathbf{t}_{ij} \rangle \mathbf{t}_{ji}\big),$$

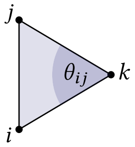
i.e., by measuring its components in the basis for ijk, and re-expressing it in the basis for jim. The negative sign accounts for the fact that the orientation of the shared edge is reversed.

This process is then repeated in the next triangle jim for the ray $r'(t) := \mathbf{p}' + t\mathbf{u}'$, stopping when the sum of all t values equals L.

2.5 The Laplace Matrix

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

$$w_{ij} = \sum_{ijk} \frac{1}{2} \cot \theta_{ij} \qquad L_{ij} = -w_{ij} \qquad L_{ii} = \sum_{j \in \mathcal{N}_i} w_{ij} \tag{2.4}$$



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 any 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 (Equation A.3), like the many other intrinsic properties in this section.

---

[13] In 2D one can also just apply the transformation $(x, y) \mapsto (-y, x)$

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 will have extrema only on the boundary. Edges with positive cotangent weights are *Delaunay* edges, corresponding to the widely-studied Delaunay property is from planar geometry. A key benefit of intrinsic triangulations will be the ability to construct a special high-quality Laplace matrix with all nonnegative cotangent weights called the *intrinsic Delaunay Laplacian*, which not only guarantees the maximum principle but also generally improves accuracy across applications (Section 4.10).

Perfect Laplacians. Wardetzky et al. [War+07] identify several properties that one would commonly desire in a discrete Laplace matrix for a triangle mesh, including linear precision, a maximum principle, and locality, and prove that there can be no Laplacian which simultaneously satisfies all of these properties. The intrinsic Delaunay Laplacian [BS07], which we discuss at length beginning in Section 4.1 technically fails the locality property as defined by Wardetzky et al. [War+07]. We note, however, that the connectivity for the intrinsic Delaunay Laplacian is actually geometrically *more* local than that of the original triangulation, in the sense that each vertex is necessarily connected to its nearest neighbor (Section 4.1.1), and the matrix still has the same sparsity ratio as the cotangent Laplacian of the original mesh. In some contexts, the geometric perspective may be a more meaningful notion of locality, and in this sense the intrinsic Delaunay Laplacian is indeed a "perfect Laplacian", reflecting its overwhelming effectiveness in practice, as explored in Section 4.10. (See also Sharp and Crane [SC20a, Section 3.3] for further discussion.)

2.5.1 The Mass Matrix

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. There are several common choices. The *vertex lumped mass matrix* is a diagonal $|V| \times |V|$ matrix whose i^{th} entry is one third of the areas of the triangles incident on vertex i:

$$(\mathsf{M}_{\text{vertex lumped}})_{ii} = \tfrac{1}{3}\sum_{ijk}\mathcal{A}_{ijk}. \tag{2.5}$$

This is easy to construct, and to invert, but is not always very accurate. For higher accuracy, one can instead use the *Galerkin mass matrix*:

$$(\mathsf{M}_{\text{Galerkin}})_{ii} = \tfrac{1}{6}\sum_{ijk}\mathcal{A}_{ijk}, \quad (\mathsf{M}_{\text{Galerkin}})_{ij} = \tfrac{1}{12}\sum_{ijk}\mathcal{A}_{ijk}. \tag{2.6}$$

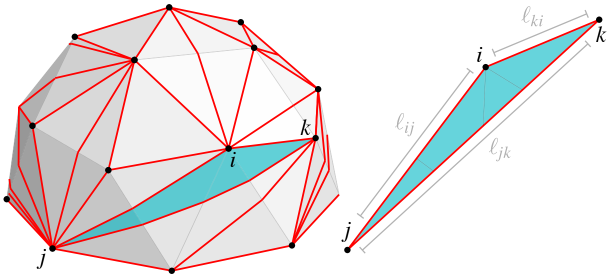
If we think of u and v as piecewise-linear functions on our domain, then $u^T \mathsf{M}_{\text{Galerkin}} v$ is precisely equal to the L^2 inner product $\int_M u(x)v(x)\,dx$ [SF08, Chapter 10, (32)].

Chapter 3

Representing Correspondence

To represent an abstract intrinsic triangulation, one needs only a mesh data structure that can encode a Δ-complex (Section 2.1), and a list of edge lengths (Section 2.3.3). This representation completely encodes the intrinsic geometry of the surface. However, in the common case where the intrinsic triangulation is defined on top of an extrinsic input mesh, it may not be sufficient: it says nothing about how the intrinsic triangulation is situated relative to the extrinsic mesh. The abstract representation alone does not specify the *correspondence* between the two meshes. For example, one cannot use it to translate points between the extrinsic and intrinsic triangulations, or even to transfer tangent data at shared vertices. In this section, we discuss concepts that arise from an intrinsic triangulation sitting on top of an embedded mesh, and introduce several data structures encode the relationship between the two triangulations while supporting essential queries and operations.

3.1 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.*, see Chapter 4). Because these operations preserve the geometry of 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[1].

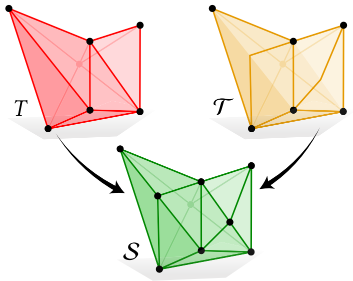
In some contexts, such as constructing operators like the intrinsic Delaunay Laplacian (Section 4.1), the abstract intrinsic representation is entirely sufficient. However in other contexts, it will be important to directly represent and evaluate this relationship between an intrinsic triangulation and the underlying extrinsic mesh. A basic example is visualization, where we seek to draw the intrinsic triangulation sitting along the embedded surface in space. More fundamentally, evaluating this relationship is necessary to translate data between the intrinsic triangulation and extrinsic mesh: common operations will include interpolating data defined at the vertices the extrinsic mesh to the vertices of a finer intrinsic triangulation, or mapping a point defined in barycentric coordinates along the intrinsic triangulation to the corresponding point on the embedded mesh, or vice versa.

---

[1] Note that in principle, there is no reason that the underlying carrier mesh needs to be extrinsic—we could just as easily define one intrinsic triangulation on top of an initial intrinsic triangulation. All the machinery described here still applies, although we cannot construct and embedding for visualization. Nevertheless, we will refer to the underlying mesh as the "extrinsic mesh" because that is the overwhelmingly common case in practice.

Whenever we are dealing with an extrinsic and intrinsic triangulation of the same surface, we will denote the extrinsic triangulation by $T_{\text{ext}} = (V_{\text{ext}}, E_{\text{ext}}, F_{\text{ext}})$ and the intrinsic triangulation by $T_{\text{int}} = (V_{\text{int}}, E_{\text{int}}, F_{\text{int}})$.

3.1.1 Common Subdivision



An intrinsic triangulation T_{int} and the corresponding extrinsic mesh T_{ext} are related via their *common subdivision* $\mathcal{S}$. Intuitively, the common subdivision is the intersection of T_{int} and T_{ext}, an extrinsic polygon mesh obtained by "cutting up" T_{ext} along the edges of T_{int}. More formally, given two triangulations of a surface T_1, T_2, the common subdivision $\mathcal{S}(T_1, T_2)$ is the coarsest polygonal decomposition[2] such that every edge of T_1 and T_2 can be expressed as a union of edges in $\mathcal{S}(T_1, T_2)$. A consequence of this construction is that every vertex in T_1 and T_2 appears in $\mathcal{S}(T_1, T_2)$, and every face of T_1 and T_2 can be expressed as as union of faces in $\mathcal{S}(T_1, T_2)$. In this text we will refer to $\mathcal{S}(T_{\text{ext}}, T_{\text{int}})$ as simply *the* common subdivision $\mathcal{S}$. Any piecewise-linear function on T_{ext} or T_{int} can be represented exactly as a piecewise-linear function on $\mathcal{S}$. We can then interpolate the vertex positions f_i from the extrinsic triangulation to obtain vertex positions on $\mathcal{S}$, which make each face planar and convex. The faces of $\mathcal{S}$ can then be triangulated arbitrarily if desired.

These properties make the common subdivision well-suited for interpolation and visualization of quantities defined on T_{ext} and T_{int}. For example, in figures throughout this text, we visualize an intrinsic triangulation T_{int} of an embedded surface T_{ext} by extracting the common subdivision and rendering it as an ordinary mesh in space, with some appropriate shading policy (*e.g.* Figure 1.3, Figure 4.1). The common subdivision can also be used to rigorously transfer data between the bases of T_{ext} and T_{int}, as described in Section 4.11. The rich data structures described below will support explicitly constructing common subdivision from a representation of an intrinsic triangulation.

We emphasize that even if T_{ext} and T_{int} 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, the common subdivision serves a complementary role, *e.g.* for visualization and data transfer.

3.2 Correspondence Data Structures

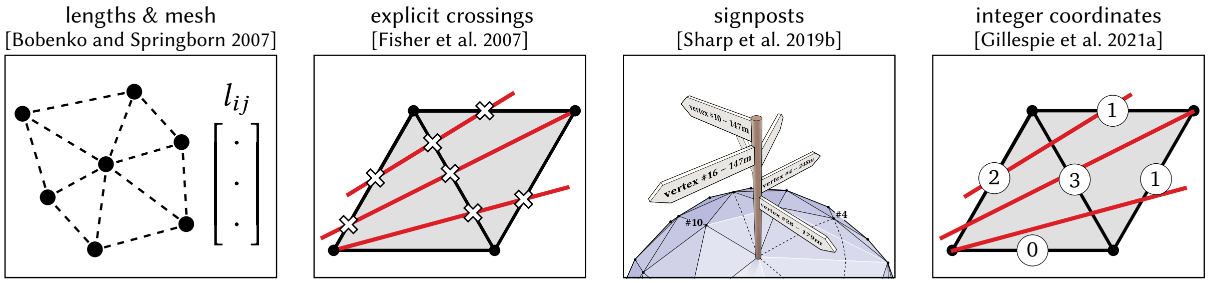
To manipulate intrinsic triangulations in geometry processing, we will often require richer data structures to represent the correspondence of an intrinsic triangulation with an extrinsic mesh, encoding not just intrinsic edge lengths, but also the paths of these geodesic edge along the surface. These data structures support operations like constructing the common subdivision, extracting the trajectory of geodesic intrinsic edges, or querying the bijection between a point on the intrinsic triangulation and the equivalent point on the extrinsic mesh. The remainder of this section we will introduce three data structures capable of representing the correspondence (Figure 3.1). Table 3.1 outlines the capabilities of the different approaches.

---

[2] Formally, a 2-manifold CW complex, as defined in Hatcher [Hat02]

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 3.1: 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.


lengths & mesh [Bobenko and Springborn 2007]

explicit crossings [Fisher et al. 2007]

signposts [Sharp et al. 2019b]

integer coordinates [Gillespie et al. 2021a]

Figure 3.1: The representations for intrinsic triangulations considered in this thesis.

3.3 Explicit Crossings

The most direct approach to encoding correspondence is to maintain an explicit representation of the common subdivision (Section 3.1.1), which is immediately updated as the intrinsic triangulation is modified—this strategy was first explored by Fisher et al. [Fis+07]. To encode the common subdivision $\mathcal{S}$, this representation explicitly stores all *crossings*, locations where an intrinsic edge crosses an edge of the underlying mesh, which become additional vertices of $\mathcal{S}$. Accordingly we will refer to this data structure as *explicit crossings*, or simply the *explicit* approach. An annotated mesh data structure stores the connectivity of $\mathcal{S}$; this mesh data structure must support general polygonal faces, as well as fast insertion and removal operations to facilitate edge flips—Fisher et al. [Fis+07] suggest a halfedge mesh. The vertices of the common subdivision are labelled as actual triangulation vertices or as crossings; each edge is likewise labelled as coming from T_{ext} (an "o" edge), T_{int} (a "c" edge), or both (an "oc" edge). In this context we refer to the edges of $\mathcal{S}$ *segments* and reserve the word "edge" to mean the sequence of segments that make up an edge of T_{ext} or T_{int}.

An advantage of the explicit crossing representation is that the common subdivision is always available, and can be queried at any time without additional cost. Furthermore, the connectivity of the common subdivision is always correct by construction, unlike *e.g.* the signpost representation in Section 3.4. The price for these properties is that flipping an edge in the explicit crossing representation is a nontrivial operation which must traverse and update the common subdivision, in contrast to the formulaic constant-time updates offered by other representations. Additionally, explicit crossings as described by Fisher et al. [Fis+07] support only edge flip operations, with a restriction that the vertex sets of T_{ext} and T_{int} are identical. Other operations such as vertex insertions have not yet been defined, though in principle the crossing data structure could be generalized to support such additional operations, at the expense of additional complexity.

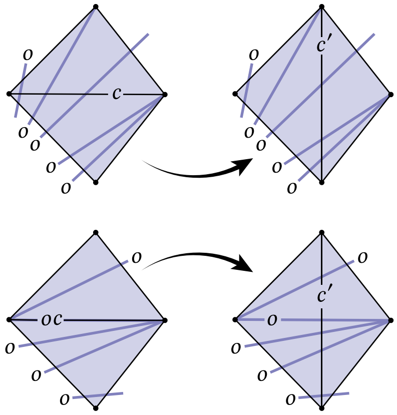


Figure 3.2: Flipping an edge in the explicit crossing representation [Fis+07].

3.3.1 Edge Flips

To perform an edge flip in the explicit representation, we begin by removing the c edge, and merging the pairs of o segments that it split (Figure 3.2, *top*). Then we insert the opposite diagonal c', splitting any o segments that it crosses. Note that the necessary connectivity changes to not depend at all on the mesh geometry: all information about which segments to merge is present in the original common subdivision mesh, and we can tell if c' crosses an edge by checking if it touches both the left and right boundary of the diamond. For this reason, the explicit representation always encodes the correct connectivity of the common subdivision. Once the connectivity is determined, we can measure the location of crossings by laying the diamond out in the plane.

Some care is required when flipping a shared edge (Figure 3.2, *bottom*). In this case, the edge is not removed, but simply relabeled from oc to o. Otherwise, the procedure proceeds in the same way as before. Similar treatment is required when flipping a c edge to lie along an o edge.

3.4 Signposts

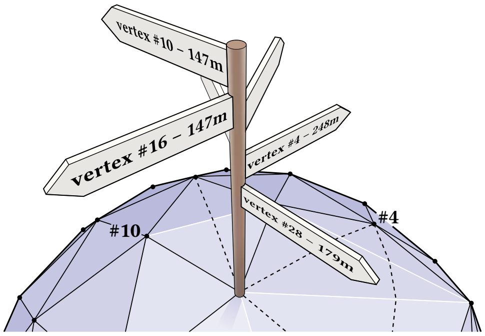


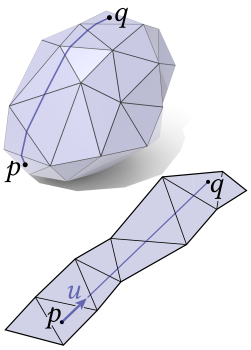
Figure 3.3: Signposts encode edges of an intrinsic by storing the length and direction of each edge in the tangent space each vertex.

Rather than explicitly encoding the 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 straightforward to generalize to the case of inserting additional vertices, and facilitate the manipulation of vector-valued quantities on a surface. A disadvantage of this approach is that correspondence is stored only inexactly in floating point coordinates, and thus accuracy may degrade for numerically degenerate inputs.

More precisely, in addition to the connectivity and edge lengths always used to represent an intrinsic triangulation, the signpost data structure stores a set of angles $\varphi : H_{\text{int}} \to [0, 2\pi)$. Each angle φ_{ij} stores the direction of the halfedge from vertex $i \in V_{\text{int}}$ to vertex $j \in V_{\text{int}}$, in the local polar coordinate system at vertex i. These polar coordinate systems are chosen to coincide with tangent coordinates frames on T_{ext}, constructed as described in Section 2.4. Because the tangent coordinate systems are in correspondence between T_{ext} and T_{int}, tangent vectors on one triangulation can be directly interpreted as tangent vectors on the other surface—this is they key property which makes the signpost representation work, and which we must carefully maintain as we modify the intrinsic triangulation. 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.

3.4.1 Tracing Through Triangulations

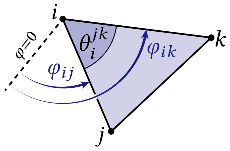
The path of an intrinsic edge along the extrinsic mesh is recovered by "tracing" the edge along the surface—in the signpost representation this amounts to evaluating the exponential map (Section 2.4.2), where the direction and distance of each edge provide the initial conditions. This is an easy and efficient local geometric operation, which essentially amounts to evaluating many ray-line intersections while traversing the extrinsic mesh, to compute where the 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 3.6).

3.4.2 Local Mesh Operations

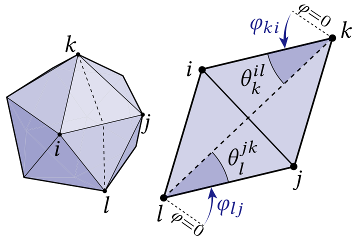
In Sharp, Soliman, and Crane [SSC19a], we define a wide variety of mesh-processing algorithms on the signpost representation of intrinsic triangulations. For the most basic edge flip, signposts can be updated via a simple local formula, just as edge lengths are updated [SSC19a, Section 3.3.1]. However, the key advance 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, all while maintaining an encoding of correspondence for queries like extracting the common subdivision or evaluating the bijection. These operations in turn enable higher-level retriangulation algorithms, like intrinsic Delaunay *refinement* and optimal Delaunay relaxation, discussed at length in Chapter 4.



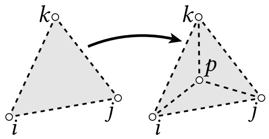
Signpost Update. While working with the signpost data structure, one often needs to update the direction of a halfedge from known direction of adjacent halfedges. In particular, suppose we know the direction φ_{ij} and the three edge lengths $\ell_{ij}, \ell_{jk}, \ell_{ki}$. Then we can evaluate the angle θ_k^{jk} as in Equation A.2, and update the direction of halfedge $\overrightarrow{ik}$ as

$$\varphi_{ik} = \varphi_{ij} + \tfrac{2\pi}{\Theta_i}\theta_i^{jk}, \tag{3.1}$$

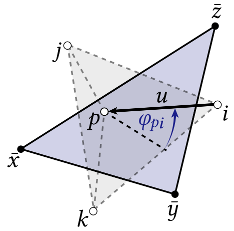
where Θ_i is the angle sum at vertex i.



Edge Flip. An edge flip replaces an edge ij with its opposite diagonal kl. As always, we must first compute the length of the flipped edge (Equation A.4). In addition, the signpost data structure requires the angles of the two new halfedges $\overrightarrow{kl}$ and $\overrightarrow{lk}$, computed via the signpost update above.



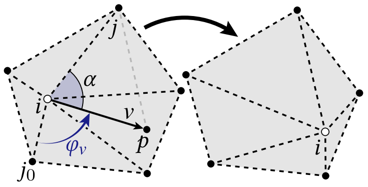
Face Split. Given barycentric coordinates for a point p inside triangle $ijk \in F_{\text{int}}$, a face split replaces ijk with three new triangles: ijp, jkp, kip. The lengths of the new edges can be computed via Equation A.5, and the angles at incoming halfedges φ_{ip} can be computed via a signpost update.



The angles at outgoing halfedges are more complicated, because we must be sure to construct a tangent basis for p which is aligned with basis for the face or edge of T_{ext} on which the new vertex sits. To do so, we will trace one of the incoming halfedges $\overrightarrow{ip}$ along T_{ext} to p. Not only does this operation give the location of p on T_{ext}, but we can also use the final incoming direction of this vector to construct an aligned tangent space. More precisely, first we trace along any halfedge $\overrightarrow{ip}$ from i to p—recall that tracing just means means evaluating the discrete exponential map (Section 2.4.2). This yields barycentric coordinates for p within some extrinsic triangle $\overline{xyz} \in F_{\text{ext}}$. Furthermore, the direction vector at the conclusion of tracing gives us the vector u pointing from i to p in the tangent basis of $\overline{xyz}$. We then set φ_{pi} to be the angle between $-u$ and the reference direction of triangle $\overline{xyz}$. This ensures that our signposts at p are expressed in the coordinate system of its parent extrinsic triangle, which allows us to perform tracing queries from p in future. Now that we have determined φ_{pi}, we can compute φ_{pj} and φ_{pk} via signpost updates.

Edge Split. Edge splits proceed in essentially the same manner as face splits, by first updating connectivity, lengths, and angles for the incoming halfedges, then tracing out one of the incoming edges to the new vertex to determine its location and construct angles for outgoing halfedges in an aligned tangent coordinate system. Splitting a boundary edge of E_{int} is again essentially the same, but we create only three incoming edges to the new vertex, and ensure that the new vertex location is along the corresponding boundary edge of E_{ext}.

Vertex Removal. We can remove any inserted vertex simply by flipping any incident edges until it has degree three and then removing the vertex and its 3 remaining edges, leaving behind a triangular face. In [GSC21a] we argue for the correctness of this flipping strategy. We emphasize that vertices of the extrinsic mesh cannot be removed in this manner, only vertices which were previously inserted. This is not merely a limitation of the signpost data structure, but a more fundamental limitation of our formulation for Euclidean intrinsic triangulations of a piecewise-flat domain—geometry can no longer be exactly preserved if extrinsic vertices were allowed to be removed. A more general formulation will be needed to support removal of arbitrary vertices from the intrinsic triangulation.



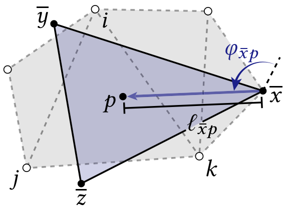
Vertex Repositioning. Inserted vertices $i \in V_{\text{int}}$ can also be moved to another location p on the surface. One can always move vertices by first[3] inserting p at the target position and then removing i. For smaller motions within a local vertex neighborhood, we can instead perform an explicit update. Let v be a vector pointing from i to p. The new edge lengths are given by

$$\ell_{jp} = \sqrt{|v|^2 + \ell_{ij}^2 - 2|v|\ell_{ij}\cos\alpha}, \tag{3.2}$$

where $\alpha := \varphi_{ij} - \varphi_v$ is the angle between v and edge ij. Angles on halfedges incident on p can be computed just as in a face split.

3.4.3 Queries

The signpost representation encodes the correspondence between an intrinsic triangulation and an underlying surface, so it can be queried for data like edge trajectories, or translating a location from one triangulation to the other. Because this correspondence is stored implicitly via edge vectors, correspondence queries typically involve evaluating the exponential map.



Point Query. One basic operation is a *point query*, evaluating the bijection between T_{ext} and T_{int}. Given a point p described by barycentric coordinates on some element of T_{ext}, we can find the corresponding point on T_{int} by constructing a vector pointing to p from any adjacent vertex, then evaluating the exponential map to trace out that vector on T_{int}. The same procedure can likewise be applied to map a location from T_{int} to T_{ext}.

---

[3] It is important to insert p first, as removing i changes the triangulation, and could invalidate the barycentric coordinates which specify p.

Construct Edge Trajectory. Signposts can also be used to efficiently reconstruct the path of a single edge $\overrightarrow{ij}$ along the surface. This simply requires tracing the edge by evaluating the exponential map from i in the direction and distance of $\overrightarrow{ij}$, resulting in a path represented as a polyline of barycentric points along the other triangulation. Again, this procedure can be applied identically for an edge from T_{ext} along T_{int}, or an edge from T_{int} along T_{ext}. This operation is as efficient as possible—the time complexity of the operation is equal to the number of edge crossings in the generated trajectory. Because these trajectories are encoded only inexactly in floating point coordinates, the traced edge will generally not terminate exactly at the target j, but rather at some very nearby location; Sharp, Soliman, and Crane [SSC19a, Appendix A] describe some simple policies for projecting to the expected termination point.

Construct the Common Subdivision. If the trajectories of all edges are extracted, they can be assembled to form the common subdivision of T_{ext} and T_{int} (Section 3.1.1). This assembly is described in detail in Section 3.6.

Transfer Tangent Data. Signposts define a very particular choice of tangent coordinates at vertices of T_{int} which is in correspondence with the tangent coordinates on T_{ext}. When these tangent coordinates are used for computation on T_{int} (*e.g.*, generating a tangent vector field at vertices), the resulting values can be directly transferred between T_{int} and T_{ext}. In particular, any vertex which appears in both T_{int} and T_{ext} has matching tangent coordinates on both triangulations, so tangent-valued quantities can be trivially copied at vertices. Newly inserted vertices on T_{int} have tangent spaces in correspondence with the canonical tangent space on the face or edge of T_{ext} along which they sit.

3.4.4 Robustness

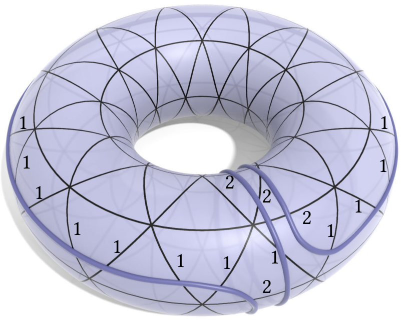
The implicit encoding of intrinsic edge trajectories in a signpost triangulation is discretely exact: in proper real arithmetic, it would always perfectly reconstruct the paths and the corresponding crossing connectivity. Recall also that in practice, it is not necessary that tracing the edge ij "exactly" hit the vertex j, merely arriving in the correct vertex neighborhood is sufficient. However, even with this observation, recovering the trajectory of an edge in inexact floating-point arithmetic may fail for nearly-degenerate inputs. 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 (Section 3.5) 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.

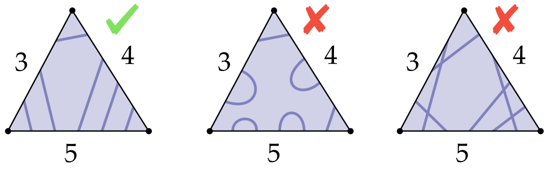
3.5 Integer Coordinates

Rather than encoding the correspondence with floating-point signposts, we 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 $n : E_{\text{int}} \to \mathbb{Z}$ and roundabouts $r : H_{\text{int}} \to \mathbb{Z}_{\geq 0}$ on the intrinsic mesh. As with signposts, when new vertices are inserted into the intrinsic mesh one must also store the corresponding location on the extrinsic mesh as barycentric coordinates in a face or edge.

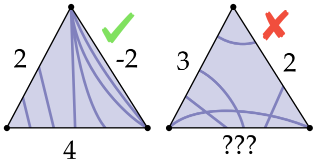
3.5.1 Normal Coordinates

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 that one can represent a curve on a triangulated surface simply by counting how many times it crosses each edge (inset). These crossing counts uniquely determine the curve so long as it *normal*, *i.e.* it never intersects itself and always enters and exits triangles on different sides[4]. To see why, consider a single triangle. Given valid crossing counts on its edges, there is always a unique way of connecting them up to form segments which do not cross and do not loop back on themselves: you simply pair up endpoints near corners. Connecting up the crossings like this in each triangle determines the topology of the curve along the surface.





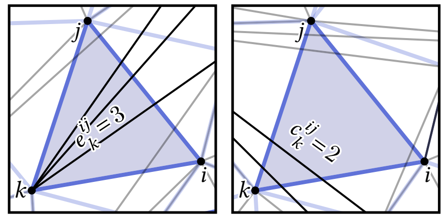
In order to encode intrinsic triangulations, we will need to allow curves to begin and end at vertices, and to run *parallel* to mesh edges between vertices. This only requires a small change: we set the normal coordinate n_{ij} to k for edges with k crossings, and set it to $-k$ for edges that have k parallel curves. Note that since our curves are not allowed to cross each other, we cannot have any curves running parallel to an edge which also intersects curves transversally, so these cases are mutually exclusive. It is often convenient to define $n_{ij}^+ := \max(n_{ij}, 0)$ and $n_{ij}^- := -\min(n_{ij}, 0)$ which count crossings and parallel curves respectively.



In this setting, there are two useful quantities that we can define at each corner: the number of edges leaving the corner, and the number of edges crossing the corner. These are simple functions of the triangle's normal coordinates [GSC21a]:

$$e_k^{ij} = \max(0, n_{ij}^+ - n_{jk}^+ - n_{ki}^+), \tag{3.3}$$

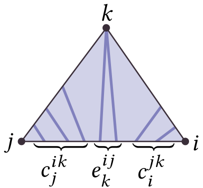
$$c_k^{ij} = \tfrac{1}{2}\left(\max\left(0, n_{jk}^+ + n_{ki}^+ - n_{ij}^+\right) - e_i^{jk} - e_j^{ki}\right). \tag{3.4}$$



emanating from corner k crossing corner k

We can also reconstruct the normal coordinates from these quantities, since each curve crossing edge ij must have either crossed an adjacent corner or emanated from the opposite vertex. Explicitly, the normal coordinate n_{ij} is given by:

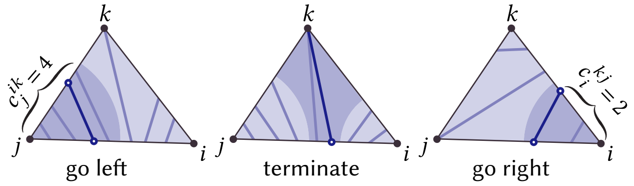
$$n_{ij} = c_j^{ik} + e_k^{ij} + c_i^{jk}. \tag{3.5}$$



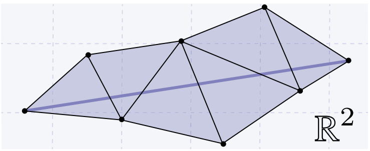
---

[4] This condition that the curves never cross back on themselves is a combinatorial version of the *geodesic* property: normal curves are locally-shortest in the sense that they cross as few edges as possible [TY12].

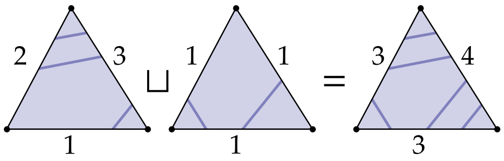
In the classical setting of closed curves, these “corner coordinates” are themselves often used to represent curves, *e.g.* [EN13]. This should also be possible in the setting of intrinsic triangulations, but has not yet appeared in the literature.



Using the corner coordinates, we can easily trace out the curve. A curve entering a triangle can either go left, stop at the opposite vertex, or go right. The number of curves which go left is precisely c_j^{ik}, and the number of curves which go right is c_i^{kj}, so we can determine which way the curve goes from its index along edge ij.

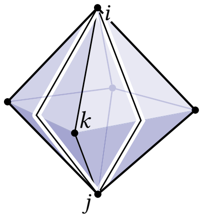


This tracing procedure only determines the triangle strip that the curve passes through. But when the curves are geodesic—as in the case of intrinsic triangulations—this actually determine the geometry of the curve. Once we know the triangle strip, we can simply lay the triangles out in the plane and connect the endpoints with a straight line to obtain the exact geometry of the curve. Although this final layout step may suffer from floating point error on near-degenerate meshes, everything up until this point has depended only on integer data. In particular, the computed curve is guaranteed to pass through the correct triangle strip.



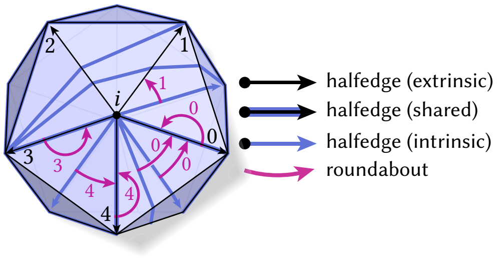
And finally, rather than just using normal coordinates to represent a single curve, we can use them to represent a whole collection of curves. The key idea is that if you have two disjoint curves, their union is represented by the sum of their normal coordinates. Hence, we can represent an entire intrinsic triangulation simply by counting how many times it intersects each extrinsic edge.

3.5.2 Roundabouts

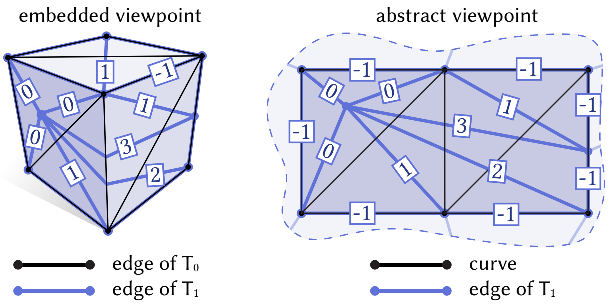


While normal coordinates determine the geometry of curves along a triangulation, they fail to fully encode the full correspondence between two triangulations, as they do not specify which logical edge each curve corresponds to. Since the triangulations are Δ-complexes, the endpoints of an edge may not 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 the extrinsic triangulation T_{ext}.

Precisely, for each intrinsic halfedge $\overrightarrow{aj}$ starting at a shared vertex $a \in V_{\text{ext}} \cap V_{\text{int}}$, the roundabout stores the first extrinsic halfedge $ab \in H_{\text{ext}}$ following $\overrightarrow{aj}$. This is encoded as an index $r_{\overrightarrow{aj}} \in \mathbb{Z}_{\geq 0}$, where we enumerate the halfedges of T_{ext} about vertex a in counterclockwise order (inset). Note that we only need roundabouts at halfedges pointing away from shared vertices $a \in V_{\text{ext}} \cap V_{\text{int}}$ since all edges of the extrinsic triangulation must start and end at such vertices. Even if such an edge has been split, the corresponding *sequence* of curves starts and ends at such vertices.



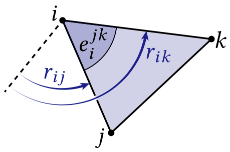
3.5.3 The Abstract Viewpoint



In principle, normal coordinates could either be defined as a value per edge of T_{ext}, counting the number of crossings from T_{int}, or as a value per edge of T_{int}. The latter approach (insert, *left*) works better for representing intrinsic triangulations, as it remains fully-informative even when we insert new vertices in to T_{int}. It is then natural to think of T_{int} as the primary triangulation, with T_{ext} as a collection of geodesic curves sitting along it (inset, *right*).

3.5.4 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.



Roundabout Update. Just as we can update signposts based on some known edge lengths and angles, we can update roundabouts based on known roundabouts and normal coordinates

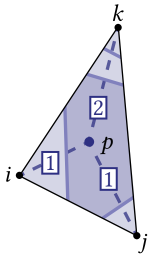
$$r_{ik} = r_{ij} + e_i^{jk} + n_{ij}^-. \tag{3.6}$$

The quantity e_k^{jk} counts how many extrinsic edges emanate strictly between $\vec{ij}$ and $\vec{ik}$, while n_{ij}^- detects whether an extrinsic edge lies exactly along ij. One can optionally reduce r_{ik} modulo the degree of vertex i in the extrinsic triangulation, like taking an angle module 2π.

Edge Flip. Again, we consider replacing edge ij with its opposite diagonal kl. In addition to computing the length of ℓ_{kl}, we compute new normal coordinates as

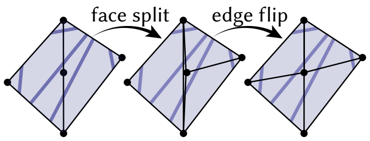
$$n_{kl} = c_l^{jk} + c_k^{ij} + \tfrac{1}{2}\left|c_j^{il} - c_j^{ki}\right| + \tfrac{1}{2}\left|c_i^{lj} - c_i^{jk}\right| - \tfrac{1}{2}e_l^{ji} - \tfrac{1}{2}e_k^{ij} + e_i^{lj} + e_i^{jk} + e_j^{il} + e_j^{ki} + n_{ij}^-, \tag{3.7}$$

and compute new roundabouts via a roundabout update.

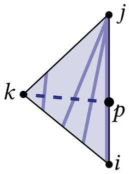


Face Split. We take as input a point p within intrinsic triangle ijk, represented by barycentric coordinates. We update the mesh connectivity and compute new edge lengths just as with signposts. Updating the normal coordinates is more involved since unlike edge flips, where the new normal coordinates depend only on the old ones, face splits are a fundamentally geometric operation. The result depends essentially on the barycentric coordinates of p.

We begin by tracing curves over the extrinsic mesh to compute the barycentric coordinates of each crossing along the edges of ijk. Once we have located the crossings, we use line side tests to determine which region p lies in. This determines the new normal coordinates, as well as the location of p along the extrinsic mesh [GSC21a, Appendix B]. After the normal coordinates have been computed, the new roundabouts can be obtained via roundabout updates. Note that we only have to compute roundabouts for halfedges pointing at p; halfedges emanating from p start at an inserted vertex and thus are not assigned roundabouts.



Edge Split. We take as input a point p along intrinsic edge ij, represented by barycentric coordinates. If ij does not lie along an extrinsic edge (*i.e.* $n_{ij} \geq 0$), then we simply perform a face split in one of the adjacent faces and then flip edge ij. However, if ij lies along an extrinsic edge (*i.e.* $n_{ij} < 0$), then we perform a different edge split routine, which ensures that the new vertex is inserted along the extrinsic edge.



In this case, the new normal coordinate is simply $n_{kp} = \max(n_{ki}, n_{kj}, 0)$, since edge pk intersects every curve in the face. We can compute the length of edge pk using the same barycentric coordinate formula that we use to perform face splits, and we can perform roundabout updates on any necessary halfedges. All that remains is computing the barycentric coordinates of p on T_{ext}, which we do by linearly interpolating the barycentric coordinates at intrinsic vertices i and j.

Vertex Removal. As in the signpost representation, inserted vertices can be removed by flipping to degree-three, and then deleting the vertex and its three adjacent edges. Again, only previously-inserted vertices can be removed in this manner; a more generally representation would be needed to allow the removal of extrinsic vertices.

Vertex Repositioning. Vertices can be moved by inserting a new vertex and then removing the old one. In principle, a local update procedure similar to the signpost one should be possible, but no such procedure exists at the moment.

Point Query. We can convert points from the intrinsic mesh to the extrinsic mesh as described in the face split operation. However, converting points from the extrinsic mesh to the intrinsic mesh is more complicated. One possibility is to use the common subdivison: given a point p on the extrinsic mesh one can identify the face of the common subdivison containing p. This common subdivison face is itself contained within some face of the intrinsic mesh, and by converting between barycentric coordinates within the different faces, we can locate p on the intrinsic mesh.

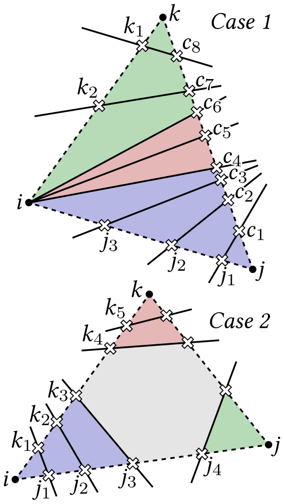
3.5.5 Robustness

The key property of the integer coordinate representation is that topologically-correct connectivity of edge trajectories and the common subdivision is always maintained through any sequence of operations. As always, geometric accuracy may still degrade in floating point, but a guarantee of correct connectivity is an important building block for robust systems. This property is shared with the explicit crossing representation, but is in contrast to the signpost representation, where connectivity must be recovered from implicit floating point data. However, unlike explicit crossings, integer coordinates offer a formulaic constant-time edge flip operation, as well as operations beyond edge flips such as insertions.

3.6 Extracting the Common Subdivision

Both the signpost and integer coordinate representations do not explicitly store the common subdivision $\mathcal{S}$, but instead store other implicit data from which edge crossings can be recovered. We can then construct a mesh encoding of the common subdivision from these edge crossings using a procedure which applies identically to either representation.

This procedure operates independently in each face of the intrinsic triangulation $ijk \in F_{\text{int}}$, cutting it along the edges of the extrinsic mesh T_{ext}. We first determine where each edge of ijk is crossed by edges of T_{ext} by tracing edges, an operation supported by our rich representations. These crossings are then gathered and ordered along each edge, and will become vertices of the common subdivision. The choice to define this procedure per-face of F_{int} rather than F_{ext} is intentional—because our intrinsic triangulations support inserting additional vertices, faces of F_{ext} may contain arbitrary configurations of vertices from V_{int} on their interior, which would greatly complicate extraction of the common subdivision if we attempted to define it per-face of F_{ext}. In contrast, the faces of F_{int} always have empty interiors, and within each the connectivity of the common subdivision falls in to just two cases (see inset). The first case is when there is some corner of the face from which edges of E_{ext} emanate, labelled here as the corner at vertex i without loss of generality; this can be detected when the number of crossings along jk is greater than the sum of the crossings along ij and ki. Otherwise, in the second case there are no edges of E_{ext} emanating from any corner.



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.

Repeating this procedure for each face of F_{int} yields the entire common subdivision. To assemble polygons, we then emit faces corresponding to consecutive segments crossing the intrinsic triangle, as well as the final central polygon in Case 2. Assigning unique indices to each crossing ensures that the output is a standard vertex-face adjacency list representation of the common subdivision. If desired, the emitted vertices and faces of can be labelled according the element of T_{ext} and T_{int} from which they originated, and geometry can be associated with the mesh as barycentric coordinates on T_{ext}, T_{int}, or as an interpolated extrinsic embedding from T_{ext}. Note also that the generated faces are planar polygons of degree ranging from $3-6$, which can be optionally triangulated if desired.

Chapter 4

Retriangulation

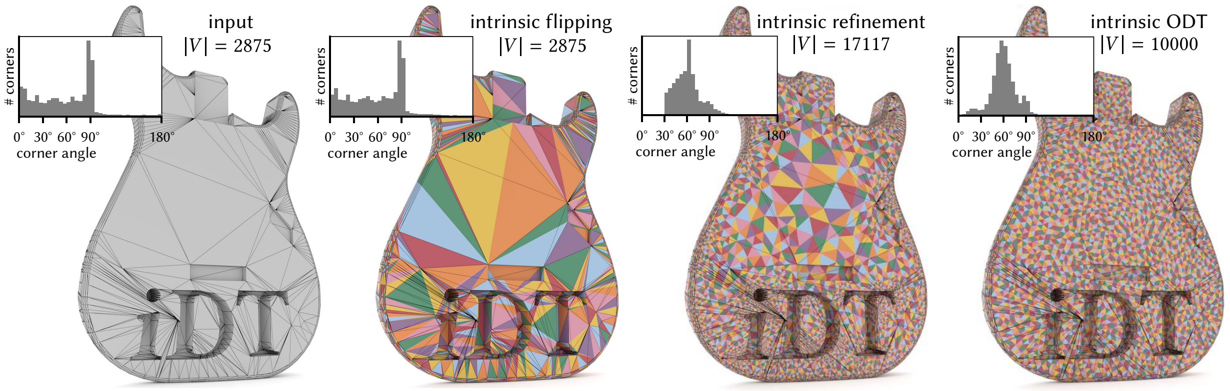

input
|V| = 2875
# corners
0° 30° 60° 90° 180°
corner angle
intrinsic flipping
|V| = 2875
# corners
0° 30° 60° 90° 180°
corner angle
intrinsic refinement
|V| = 17117
# corners
0° 30° 60° 90° 180°
corner angle
intrinsic ODT
|V| = 10000
# corners
0° 30° 60° 90° 180°
corner angle
iDT

Figure 4.1: Intrinsic retriangulation schemes applied to a computed-aided design model with poor triangle quality. The black wireframe denotes the extrinsic 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.

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, performing edge flips, vertex insertions, and other operations to construct triangulations with improved numerical conditioning and other desirable properties.

4.1 Intrinsic Delaunay Triangulations

The *Delaunay triangulation* of a point set in the plane is a fundamental concept in computational geometry which has a beautiful analogue on intrinsic triangulations. 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]. In the plane, there are many equivalent characterizations of the Delaunay triangulation of a point set (Figure 4.2), including:

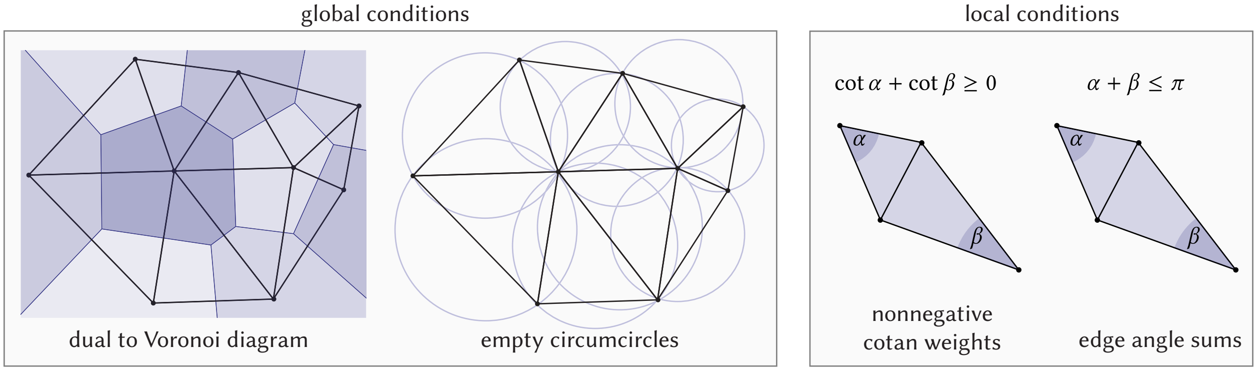
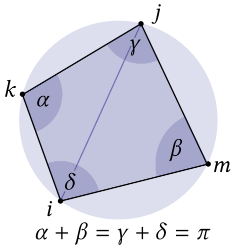


Figure 4.2: There are many equivalent characterizations of the Delaunay triangulation. Here we illustrate a few examples.

1. The Delaunay triangulation is the dual of the *Voronoi diagram*.
2. The Delaunay triangulation is the set of all triangles with empty circumcircles.
3. The Delaunay triangulation is the triangulation where all cotangent weights are nonnegative.
4. The Delaunay triangulation is the triangulation where for each edge ij, the sum of the angles opposite ij in the adjacent triangles is at most π.
5. 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 $z := x^2 + y^2$.

All but the last characterization can be easily generalized to intrinsic triangulations of curved surfaces. Amazingly they still coincide, defining the *intrinsic Delaunay triangulation*. Formally we take (4) as our definition, since it is concrete and was the definition that Rivin [Riv94a] originally proposed, but we could take any one since they are all equivalent.

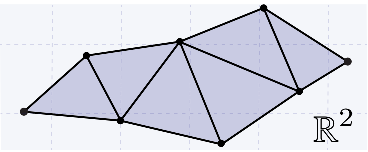


Concyclic Quads. For a generic polyhedron, these equivalent conditions uniquely determine the intrinsic Delaunay triangulation; any given polyhedron admits exactly one intrinsic triangulation of its vertices which satisfies them. However, this may not be the case in non-generic configurations; in particular, if the neighboring triangles around edge ij form an intrinsic concylic quadrilateral, then both ij and the opposite diagonal km will have angle sum π (and equivalently have cotan weight 0, *etc.*) Thus, there may be several triangulations which satisfy all of the conditions, so in general one should speak of *an* intrinsic Delaunay triangulation, rather than *the* intrinsic Delaunay triangulation[1]. 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].

[1] Some authors instead choose to define the *Delaunay tessellation* in such scenarios, which is still uniquely-defined, but takes the form of a polygon mesh rather than a triangle mesh. However, in our applications we will always work with triangle meshes, and generally will not be concerned with the existence of multiple Delaunay triangulations.

4.1.1 Properties of Intrinsic Delaunay Triangulations

Here, we gather a collection of useful properties of intrinsic Delaunay triangulations, along with their proof sketches. Properties of planar Delaunay triangulations usually extend to the intrinsic surface case, when defined in an appropriate manner. In particular, we carefully clarify these properties in non-generic configurations, where there may be multiple Delaunay triangulations.



Generally, two main proof techniques are used to generalize results from the planar setting. The first is to consider the change in the some quantity when a non-Delaunay edge is flipped; if the quantity can be shown to decrease, then it must be minimized on a Delaunay triangulation, though careful treatment of the 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 we can apply properties from planar computational geometry to all possible unfoldings to generalize them to the intrinsic case.

Lastly, we note that when the notion of a circumcircle or disk arises in the intrinsic setting, it can be formalized as an isometric immersion of a Euclidean disk into the surface. Such an immersion is only well-defined if the immersed disk does not strictly contain any cone points; we will only need to consider such empty disks for our arguments. A disk immersed in this manner may overlap itself along the surface.

4.1.1.1 Empty Triangle Circumcircles & Edge Disks

If a triangle ijk appears in a Delaunay triangulation, then it has a geodesic circumcircle with empty interior [BS07]. Conversely, 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 every Delaunay triangulation.

Similarly, each edge ij in a Delaunay triangulation has a geodesic disk with i,j on its boundary and an empty interior [BS07]. Conversely, if i and j are the only vertices on the boundary of the disk, then the edge ij necessarily appears in every 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 every Delaunay triangulation.

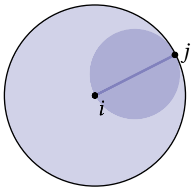
Bobenko and Springborn [BS07] actually define the Delaunay triangulation[2] as a cell complex whose edges are precisely the geodesic segments between vertices which have empty circumcircles, whose faces are disk-inscribed geodesic polygons with empty circumcircles. In Proposition 10, they prove that this construction is equivalent to the opposite angle sum characterization of the Delaunay property, which proves that intrinsic Delaunay triangulations obey the empty circumcircle property for faces and edges.

4.1.1.2 Contains Nearest Neighbors

For vertex i, every Delaunay triangulation has an edge connecting to its nearest geodesic neighbor j. In a non-generic configuration, if there are multiple other vertices at the same minimal geodesic distance from i, then all of these edges must appear in every Delaunay triangulation. Another subtlety in the intrinsic setting is that vertex i may be its own nearest neighbor: if there is a non-constant geodesic from i to itself which is shorter than all paths to other vertices, we say that i is its own nearest neighbor, and it is this edge which must appear in the Delaunay triangulation.

---

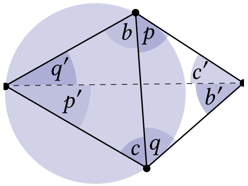
[2] Technically the Delaunay *tessellation*: the polygon mesh of edges which strictly satisfy the Delaunay property. They then obtain Delaunay triangulations by arbitrarily triangulating any non-triangular face immediately preceding their Definition 8.



This follows from the empty edge disk property in a straightforward way. Suppose that i's nearest neighbor is j, and they are separated by a length of ℓ_{ij}. Now consider the geodesic disk of radius ℓ_{ij} about i. Since j is the nearest neighbor, this disk must be empty. This empty disk contains a smaller disk which goes through i and is tangent to the larger disk at j. Hence, the curve connecting i to j has an empty disk about it, and must therefore be a Delaunay edge.

4.1.1.3 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 maximized [Sib78].



The proof of this property proceeds exactly like the planar case. In order to prove that the planar Delaunay triangulation maximizes angles, Sibson observed that flipping a non-Delaunay edge increases the lexicographic rank of the set of angles. In particular, Sibson notes that in the inset diagram, each primed angle is strictly smaller than the corresponding unprimed angle. This implies that only a Delaunay triangulation can maximize the lexicographic ordering—for any non-Delaunay triangulation, we can flip some non-Delaunay edge to increase its lexicographic rank.

Some difficulty arises in the case of non-unique Delaunay triangulations: not all Delaunay triangulations have the same set of angles. However, they each have the same *minimum* angle[3], so it is still true that all Delaunay triangulations maximize the minimum angle.

4.1.1.4 Smoothest Piecewise-Linear Interpolation (Rippa's Theorem)

For any function defined at the vertices, a Delaunay triangulation yields the smoothest piecewise-linear interpolation over the domain, in the sense of Dirichlet energy [Rip90; BS07; Che+10].

Bobenko and Springborn [BS07, Rippa's Theorem 1] observe that Rippa's proof holds without change in the intrinsic setting. Much like the angle maximization property, the basic idea is to show that flipping a non-Delaunay edge always makes any piecewise-linear interpolant smoother. This then implies that only a Delaunay triangulation can minimize Dirichlet energy—for any non-Delaunay triangulation, flipping some non-Delaunay would decrease the Dirichlet energy. Unlike the angle maximization case, however, there is no difficulty for surfaces with multiple Delaunay triangulations: all Delaunay triangulations have equally-smooth piecewise-linear functions[4].

[3] Here the Delaunay tessellation perspective is quite useful. Recall that the Delaunay tessellation is a polygon mesh whose faces are all concyclic polygons. All Delaunay triangulations arise by triangulating these polygonal faces in different ways. So to prove that all Delaunay triangulations have the same minimum angle, it suffices to prove that all triangulations of a concyclic polygon have the same minimum angle. This follows from elementary geometry, using the fact that the angles of an inscribed triangle are proportional to the length of the opposite arc.

[4] One way of seeing this is that the smoothness of a piecewise-linear function is measured by the cotan Laplacian. Because all Delaunay triangulations induce the same cotan Laplacian, they all produce equally-smooth interpolations. .

4.1.1.5 Minimal Spectrum

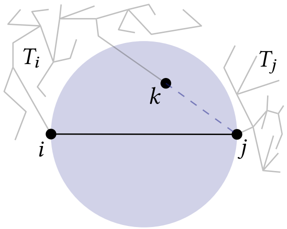
The eigenvalues of the Laplace matrix are minimized on a Delaunay triangulation, that is for the i'th eigenvalue λ_i, all other triangulations have $\lambda_i' \geq \lambda_i$ [Che+10].

Chen *et al.* prove this property by again observing that flipping a non-Delaunay edge can never increase the eigenvalues of the Laplacian. Note that no special treatment of non-generic surfaces is needed as all Delaunay triangulations have the same Laplace matrix.

4.1.1.6 Minimal Minimum Spanning Tree

A Delaunay triangulation has the shortest possible minimum spanning tree among all triangulations of the vertex set [Tou80].

Toussaint originally proved this theorem in the planar case in relation to *relative neighborhood graphs*. We instead follow the planar proof found in [ORo98] which generalizes immediately to the intrinsic setting. We consider spanning trees of the vertices of an intrinsic triangulation whose edges are geodesic paths. Let MST be the shortest such spanning tree. We will show that the edges of MST must appear the Delaunay. Assume for contradiction that an edge $ij \in$ MST does not appear in the Delaunay triangulation. Thus, the disk which has ij as a diameter[5] is not empty. There must be some third vertex k in this disk (or on its boundary).



Now, we consider removing ij from MST splits the tree into two parts, which we will call T_i and T_j. Without loss of generality, let $k \in T_i$. Now, we will show that there is a geodesic connecting j to k with length strictly less than ℓ_{ij}. Replacing edge ij with edge jk will thus yield a spanning tree of strictly less weight, contradicting our assumption that MST was minimal.

Suppose k is in the interior of the disk. Then the distance from k the midpoint of edge ij is strictly less than $\ell_{ij}/2$. Since the distance from this midpoint to j is $\ell_{ij}/2$, we conclude that the distance from j to k is strictly less than ℓ_{ij} as desired. On the other hand, suppose that there are no other vertices in the interior of the disk and that k lies on the boundary. Then the disk is intrinsically flat. In a Euclidean disk, the distance from j to any point on the boundary other than i is strictly less than the diameter ℓ_{ij}.

Hence, we can remove edge ij from the spanning tree and add a shorter edge jk instead. This contradiction implies that every edge of the MST must lie in the Delaunay triangulation, as desired.

4.1.1.7 Geometric Spanner

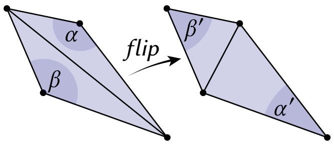
A geometric spanner is a graph with the property that distance along edges between any two vertices is at most some constant factor more than the actual geometric distance between those vertices—here, the geodesic distance along the surface. Any intrinsic Delaunay triangulation is a 2-spanner: between any two vertices the graph distance along edges is at most twice the geodesic distance [Xia13].

Xia considers only the case of planar Delaunay triangulations, but the analogous result for surfaces follows almost immediately. Consider vertices i, j on an intrinsic triangulation, with shortest path γ between them. γ is contained in some triangle strip in the Delaunay triangulation, or possibly a sequence of strips (due to saddle vertices). We can lay each triangle strip out in the plane to obtain a planar Delaunay triangulation, preserving the distance between the beginning and end of the strip. From the planar result, each strip is a geometric spanner, where the graph distance is at most twice the straight-line distance. Applying this argument to each triangle strip composing γ implies that there must be a path along the edges of the intrinsic Delaunay triangulation within the desired bound.

---

[5] Formally, this disk can be defined as the geodesic ball of radius $\ell_{ij}/2$ centered at the midpoint of edge ij

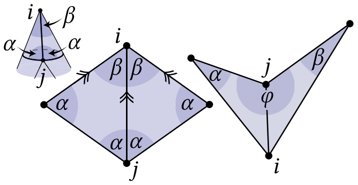
4.2 Delaunay Flipping



A key result for intrinsic geometry processing is that any triangulation can be transformed to be an intrinsic Delaunay triangulation by repeatedly flipping any edge with $\alpha + \beta > \pi$ (inset) [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.

Theorem 1 (Indermitte et al. [Ind+01] and Bobenko and Springborn [BS07])**.** *Repeatedly flipping non-Delaunay edges yields an intrinsic Delaunay triangulation after a finite number of flips.*

Proof. See Bobenko and Springborn [BS07, Proposition 12]. Essentially, Delaunay edge flips always decrease Musin's harmonic index [Mus97], and there are a finite number of intrinsic triangulations of a given domain with bounded harmonic index. ☐



An important fact is that every non-Delaunay edge is indeed flippable. Recall that an edge is flippable if and only if its endpoints have degree greater than 1, and its neighboring faces form a convex quadrilateral (Section 2.3.4). For an edge *ij* incident on a degree-1 vertex (inset, *left*), both neighboring faces must actually be the same isosceles triangle. Since $2\alpha < 2\alpha + 2\beta$, the sum of opposite angles is less than the sum of the angles at *ij*'s endpoints, and is hence less than π. So *ij* must be Delaunay. Similarly, if *ij*'s neighbors form a nonconvex quadrilateral (inset, *right*), then the angle at one endpoint must be strictly greater than π. Thus, the sum of opposite angles must be strictly less than π, so *ij* must be Delaunay.

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 use the resulting mesh for subsequent computation. 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 4.3) showed linear scaling on a challenging 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 $y = x^2$).

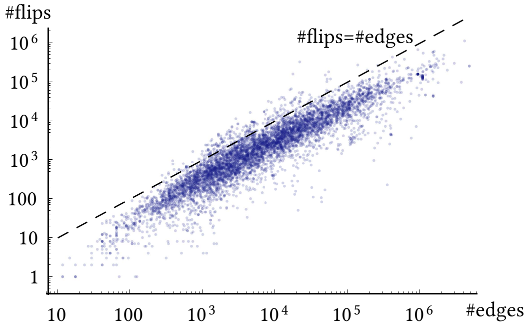
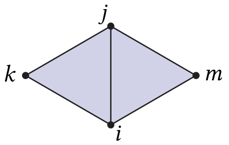


Figure 4.3: An empirical study of the number of edge flips to produce an intrinsic Delaunay triangulation [SSC19a]. Each point is a 3D model from the Thingi10k dataset [ZJ16].



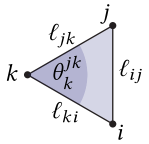
To implement the flip algorithm, we maintain a queue of possibly non-Delaunay edges. For each edge in the queue, we check the Delaunay condition, which is implemented by checking the sign of the edge's cotan weight. In practice, we allow some epsilon tolerance on this check due to floating point errors—a tolerance of 10^{-5} is generally sufficient. If the cotan weight is negative, then the edge is not Delaunay and we flip it.

After flipping, a non-Delaunay edge necessarily becomes Delaunay. However, any of the 4 neighboring edges on the boundary of the diamond may be become non-Delaunay due to the flip—these edges must then be enqueued to be checked themselves. There is no reason to keep multiple copies of an edge in the queue to be checked, so as an efficiency optimization we can optionally maintain an auxiliary boolean array indicating which edges are currently in the queue, updating it as necessary when edges are pushed or popped. The process is summarized in Algorithm 1, with the inset notation.

```
Algorithm 1 FLIPTODELAUNAY($T_{int}$)
Input: An intrinsic triangulation $T_{int}$. This may be represented via any of the data structures from Chapter 3.
Output: An updated intrinsic Delaunay triangulation $T_{int}$.
toCheck ← $\mathcal{E}$                                   ▷Enqueue all edges
while toCheck is not empty do
    $ij$ ← POPFRONT(toCheck)
    ▷Check if $ij$ violates the Delaunay condition
    if COTANWEIGHT($T_{int}$, $ij$) $< -\epsilon$ then
        $T_{int}$ ← FLIPEDGE($T_{int}$, $ij$)
        ▷Now push neighboring edges onto the queue
        ▷(the flip may have made them non Delaunay)
        neighbors ← $\{im, mj, jk, ki\}$
        for edge $e \in$ neighbors do
            if $e \notin$ toCheck then
                toCheck ← PUSHBACK(toCheck, $e$)
end while
return $T_{int}$
```

```
Algorithm 2 EDGECOTANWEIGHT($T_{int}$, $ij$)
Input: An edge $ij$ of an intrinsic triangulation $T_{int}$.
Output: The cotan weight of edge $ij$.
totalWeight ← 0
for face $ijk$ neighboring edge $ij$ do
    ▷Compute face area with Heron's formula
    $s \leftarrow (\ell_{ij} + \ell_{jk} + \ell_{ki})/2$
    area ← $\sqrt{s(s-\ell_{ij})(s-\ell_{jk})(s-\ell_{ki})}$
    angleCotan ← $(\ell_{jk}^2 + \ell_{ki}^2 - \ell_{ij}^2)/(4\,\text{area})$
    totalWeight += angleCotan/2
return totalWeight
```

There are many ways to compute edge cotan weights. After some trigonometry, we can compute edge cotan weights directly from edge lengths without using any inverse trigonometric functions. Consider a single triangle:



The area of the triangle is $\frac{1}{2}\ell_{ki}\ell_{jk}\sin\theta_k^{ij}$, and we can compute $2\ell_{ki}\ell_{jk}\cos\theta_k^{ij}$ via the law of cosines. By dividing the two expressions, we obtain $\cot\theta_k^{jk}$ (Equation A.3, Algorithm 2).

Alternative Strategies. Many other 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. In modern planar geometry, the edge-flipping approach is often avoided in favor of spatial partitioning schemes which are asymptotically more efficient [GS85; Dwy87]. However unlike flipping, spatial decompositions do not generalize immediately to the setting

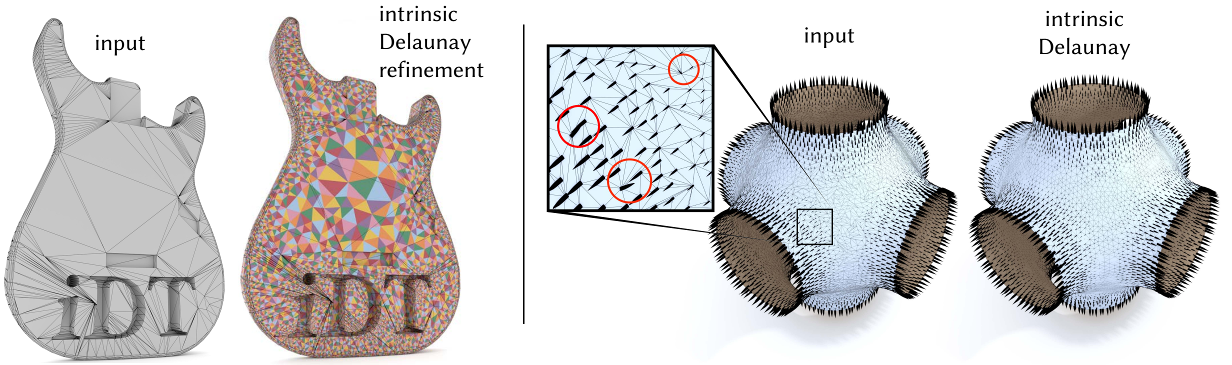


Figure 4.4: *Left*: Rich data structures enable intrinsic Delaunay refinement, generating triangulations with good angle bounds. The black wireframe denotes the extrinsic mesh, while colored triangles give the intrinsic triangulation. *Right*: Signposts further enable vector field processing; the Laplacian of the intrinsic Delaunay triangulation offers a maximum principle for tangent vector fields, which here avoids unexpected flipped vectors when generating a smooth field.

of intrinsic triangulations. Furthermore, we observe empirically that flipping exhibits essentially linear scaling on real data (Figure 4.3). On surfaces, an alternate approach is to construct the geodesic Voronoi diagram and take its dual [Liu+17b]. This algorithm provably terminates in $O(n^2 \log n)$ time, though it requires the construction of shortest paths along the mesh as subroutine (*e.g.* via the MMP algorithm [MMP87]), which can be complex and difficult to implement robustly in practice.

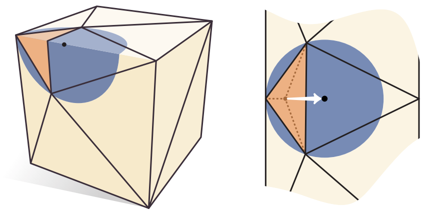
4.3 Delaunay Refinement

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, making use of the insertion operations offered by the signpost data structure. The same procedure can be applied on any intrinsic triangulation which supports operations like insertion; Gillespie, Sharp, and Crane [GSC21a] further the develop the routine with a proof and treatment of surfaces with boundary. 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 intrinsic triangle ijk that violates the angle bound
- Insert the circumcenter p of ijk

The only remaining difficulty with implementing this algorithm intrinsically is locating the circumcenter of ijk. However, this is also fairly straightforward because our triangulation necessarily satisfies 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].



Robust Triangulation. 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 aligns with the treatment of the planar case [Che89], though formally extending the analysis to the intrinsic setting on general meshes with boundary is an area of ongoing work. In Gillespie, Sharp, and Crane [GSC21a] we prove that the algorithm succeeds on meshes without boundaries or needle vertices, though a full proof of the most 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 4.9).

Now we can elaborate on the implementation of intrinsic Delaunay refinement. Similar to the Delaunay flipping algorithm, we maintain a queue of possibly-invalid faces to check. We initialize the queue with all faces of the mesh. Then, for each face in the queue, we check whether it satisfies the minimum angle bound. If it does not, we insert its circumcenter into the mesh, and push all of the faces incident on the new vertex onto the queue of faces to check. Finally, we flip to Delaunay. Each time we flip an edge, the two new faces are pushed onto the queue if they violate the minimum angle bound.

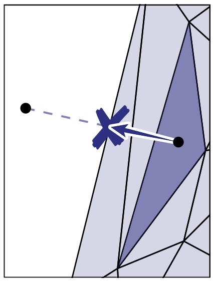
Algorithm 3 FLIPQUEUETODELAUNAY(T_{int}, edgesToCheck, θ_{min})

Input: A triangulation T_{int}, a queue edgesToCheck of possibly non-Delaunay edges, and a minimum angle bound θ_{min}.

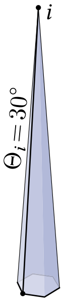
Output: An updated Delaunay triangulation T_{int}, and a list facesToCheck of all newly-created faces which violate the minimum angle bound

```
facesToCheck ← $\emptyset$
while edgesToCheck is not empty do
    $ij$ ← PopFront(edgesToCheck)
    if $ij$ is not Delaunay then
        $T_{\text{int}}$ ← FlipEdge($T_{\text{int}}, ij$)
        for neighboring edge $\widetilde{ij}$ which is not in edgesToCheck do
            edgesToCheck ← PushBack(edgesToCheck, $\widetilde{ij}$)
        for neighboring face $ijk$ which is not in facesToCheck do
            if ShouldRefine($T_{\text{int}}, ijk, \theta_{\text{min}}$) then
                facesToCheck ← PushBack(facesToCheck, $ijk$)
end while
return $T_{\text{int}}$, facesToCheck
```

We can accelerate this algorithm by integrating the Delaunay flip algorithm. Using Algorithm 1 each time would require checking the Delaunay condition at every edge of the mesh after inserting each vertex, which quickly becomes expensive. Fortunately, the mesh was Delaunay before inserting the vertex, so the only edges which could fail to satisfy the Delaunay condition are those neighboring to the newly-inserted vertex. Accordingly, we define an alternative flipping procedure (Algorithm 3) which takes in the list of all possibly non-Delaunay edges, and then runs the ordinary flipping algorithm. This flipping procedure also returns the set of all faces which were created by flips and violate the minimum angle bound—we process these faces later in the outer loop of the algorithm.



In the presence of boundary, triangle circumcircles may not be contained in the mesh. In the planar case, Chew's 2nd algorithm splits boundary segments, then removes previously inserted vertices within the diametral ball as a technical requirement to ensure the algorithm converges. We employ an analogous geodesic procedure for the intrinsic case. To find vertices within a ball around the edge without a complicated geodesic distance query, we instead remove vertices within twice the graph distance; by the geometric spanner property of Delaunay triangulations this includes all vertices within the desired geodesic ball (Section 4.1.1). Precisely, whenever a circumcenter tracing query hits a boundary edge, we instead split that edge at its midpoint. Then, we flip the triangulation to Delaunay, and remove all inserted vertices within a Dijkstra ball [6] of radius ℓ_{ij} centered at the new vertex. We assume that the RemoveVertex routine returns the list of edges that were modified due to the removal; these edges and their adjacent faces must be checked for the Delaunay property and refinement criteria respectively.



Finally, it can be impossible to achieve a given minimum angle bound near extremely skinny needle-like vertices. At best a single triangle wraps around the vertex, making it a degree-1 vertex. For this reason, we only refine triangles which have a corner angle less that θ_{min} that are incident on a vertex with degree greater than one (Algorithm 4). One can also incorporate other conditions, *e.g.* refining triangles whose circumradii exceed a given bound to impose triangle area constraints, but for simplicity we consider only angle bounds here.

Algorithm 4 ShouldRefine($T_{\text{int}}, ijk, \theta_{\text{min}}$)

```
Input: A face $ijk$ in triangulation $T_{\text{int}}$, and the angle bound $\theta_{\text{min}}$.
Output: Whether or not the face should get refined
for vertex $m$ in $ijk$ do
    if CornerAngle($ijk, m$) $< \theta_{\text{min}}$ then
        if Degree($m$) $> 1$ then
            return true
return false
```

Algorithm 5 DelaunayRefine($T_{\text{int}}, \theta_{\text{min}}$)

```
Input: A triangulation $T_{\text{int}} = (V_{\text{int}}, E_{\text{int}}, F_{\text{int}})$ and a desired minimum corner angle $\theta_{\text{min}} \leq 30°$.
Output: An updated triangulation $T_{\text{int}}$ whose faces all have corner angles at least $\theta_{\text{min}}$.
$T_{\text{int}}, \_ \leftarrow$ FlipQueueToDelauany($T_{\text{int}}, E_{\text{int}}$)    ▷Initially flip to Delaunay, checking all edges
facesToCheck $\leftarrow F_{\text{int}}$    ▷Maintain an explicit queue of possibly-invalid faces to check
while facesToCheck is not empty do
    edgesToCheck $\leftarrow \emptyset$    ▷List of possibly non-Delaunay edges to check at the end of this iteration
    $ijk \leftarrow$ PopFront(facesToCheck)    ▷Get next face. Skip faces which no longer exist.
    if ShouldRefine($T_{\text{int}}, ijk, \theta_{\text{min}}$) then    ▷Check face for refinement
        ▷Compute the barycentric coordinates of the circumcenter
        $\hat{v}_i = \ell_{jk}^2(\ell_{ij}^2 + \ell_{ki}^2 - \ell_{jk}^2)$    ▷And similarly for $\hat{v}_j, \hat{v}_k$
        $(v_i, v_j, v_k) = (\hat{v}_i, \hat{v}_j, \hat{v}_k)/(\hat{v}_i + \hat{v}_j + \hat{v}_k)$    ▷Normalize to obtain true barycentric coordinates
        $p \leftarrow$ Exp(Barycenter($ijk$), $v - (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$)    ▷Find circumcenter by tracing from barycenter
```

---

[6] *i.e.* the set of vertices within a certain distance along the edge graph.

```
        if $p$ lies in the mesh interior then
            $T_{\text{int}}, q \leftarrow$ InsertVertex($T_{\text{int}}, p$)    ▷Insert circumcenter. The new vertex is named q
        else
            ▷If the circumcenter lies outside the mesh, then we split the boundary edge separating ijk
            ▷from its circumcenter, and remove all vertices within a Dijkstra ball of the new vertex
            $\widetilde{ij} \leftarrow$ the edge that $p$ lies on
            $T_{\text{int}}, q \leftarrow$ SplitEdgeAtMidpoint($T_{\text{int}}, \widetilde{ij}$)
            for face $f$ adjacent to inserted point $q$ do
                for edge $e$ of face $f$ which is not in edgesToCheck do
                    edgesToCheck $\leftarrow$ PushBack(edgesToCheck, $e$)
            ▷We need to flip to Delaunay before computing the Dijkstra ball centered at q, since
            ▷in a Delaunay triangulation the Dijkstra ball approximates a geodesic ball
            $T_{\text{int}}$, newFacesToCheck $\leftarrow$ FlipQueueToDelaunay($T_{\text{int}}$, edgesToCheck)
            facesToCheck $\leftarrow$ AppendBack(facesToCheck, newFacesToCheck)
            edgesToCheck $\leftarrow \emptyset$
            $\mathcal{D} \leftarrow$ DijkstraBall($T_{\text{int}}, q, \ell_{\tilde{ij}}$)
            for inserted vertex $m \in \mathcal{D}$ which is not on the boundary do
                $T_{\text{int}}$, modifiedEdges $\leftarrow$ RemoveVertex($m$)
                ▷RemoveVertex modifies neighbors; enqueue them to check the Delaunay condition.
                ▷We also put their neighboring faces into the face queue as necessary.
                for edge $e \in$ modifiedEdges do
                    edgesToCheck $\leftarrow$ PushBack(edgesToCheck, $e$)
                    for face $f$ adjacent to $e$ which is not in facesToCheck do
                        if ShouldRefine($T_{\text{int}}, f, \theta_{\min}$) then
                            facesToCheck $\leftarrow$ PushBack(facesToCheck, $f$)
        for face $f$ adjacent to $q$ do    ▷Add faces/edges incident on the inserted vertex to queues
            if $f \notin$ facesToCheck then
                facesToCheck $\leftarrow$ PushBack(facesToCheck, $f$)
            for edge $e$ of face $f$ which is not in edgesToCheck do
                edgesToCheck $\leftarrow$ PushBack(edgesToCheck, $e$)
        ▷Flip to Delaunay after vertex insertion. Since the mesh was Delaunay before,
        ▷the only possible non-Delaunay edges are edgesToCheck
        $T_{\text{int}}$, newFacesToCheck $\leftarrow$ FlipQueueToDelaunay($T_{\text{int}}$, edgesToCheck)
        facesToCheck $\leftarrow$ AppendBack(facesToCheck, newFacesToCheck)
end while
```

4.4 Constrained Triangulation

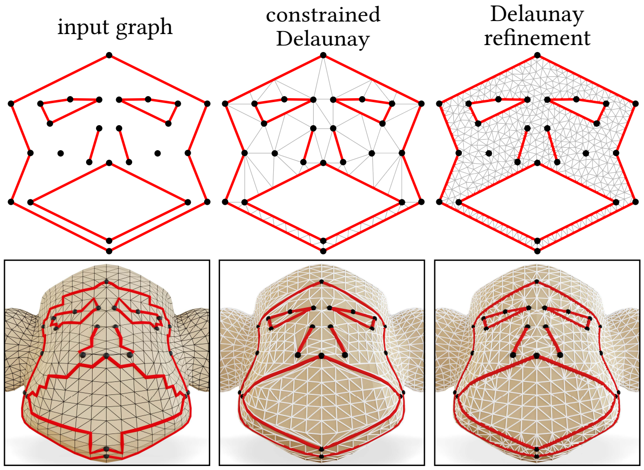


Figure 4.5: *Top.* Planar constrained Delaunay triangulations are used to produce high-quality triangulations which conform to a collection of specified lines. *Bottom.* Constrained intrinsic triangulations play a similar role on surfaces, preserving a set of intrinsic edges in the triangulation.

A *constrained triangulation* is a triangulation required to contain some predefined collection of edges. For traditional mesh generation in $\mathbb{R}^2$ and $\mathbb{R}^3$, constrained Delaunay triangulations are a standard tool, yielding meshes which align to specified boundary geometry [Che89; She02b; CDS12]. With the intrinsic approach, we can can likewise generate *constrained intrinsic triangulations*, which are guaranteed contain specified intrinsic edges of interest. These edges can then be used in applications *e.g.* to impose boundary conditions along predefined regions (Figure 5.8), or to preserve "feature edges" or creases in a mesh[7]. The intrinsic FLIPTODELAUNAY and DELAUNAYREFINE procedures (Algorithm 1 and Algorithm 5) are easily modified to preserve constrained edges, by either declining to flip such edges (allowing them to be non-Delaunay), or by splitting a constrained edge instead of flipping it (which will eventually yield a Delaunay edge).

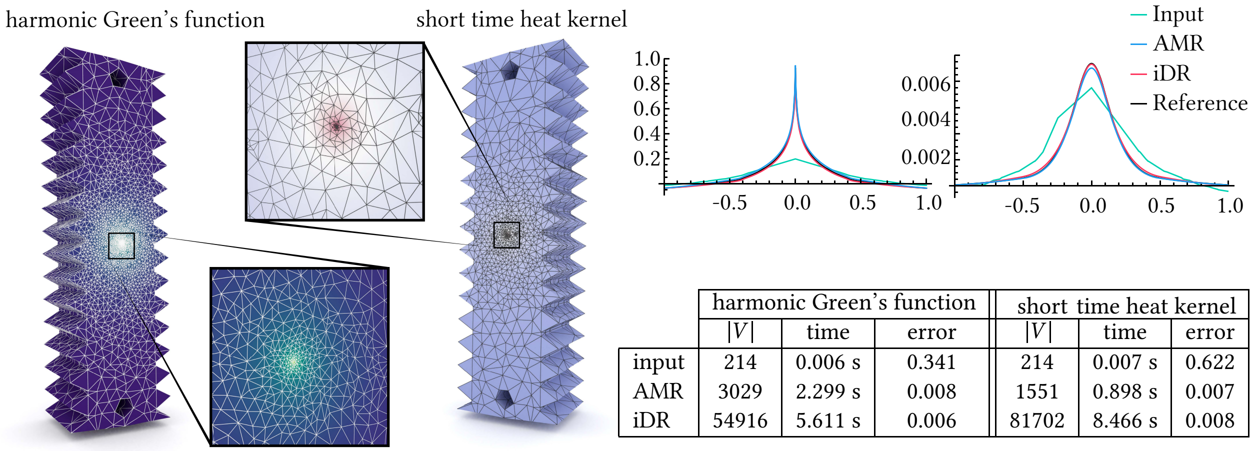
However, if the desired edges are not already present in the initial triangulation, then we face a significant new challenge in the intrinsic setting: unlike in the planar case, we cannot simply draw a straight line between distant vertices to obtain the specified edge. The FlipOut procedure, described in Chapter 5, fills this necessary role, allowing us to introducing long geodesic edges between a specified pair of vertices into the triangulation. The long red geodesic edges used as constraints in Figure 4.5, *bottom* are generated using this procedure.

4.5 Optimal Delaunay Triangulation

An *optimal Delaunay triangulation* improves element 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 original vertices is of course undesirable, since it would change the surface geometry as well as 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 Delaunay refinement; 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 4.1). Crucially, throughout this

---

[7] Crease edges characterized by by sharp dihedral angles are of course indistinguishable in the intrinsic geometry, but may nonetheless be useful to preserve as constrained edges for applications, *e.g.* for directional alignment.



	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 4.6: 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].

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

4.6 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. In [SSC19a, Section 5.4 & Supplemental], we 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 4.6). As another application, we 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 4.10).

4.7 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 setting there is a simple *mollification* procedure which provably resolves degeneracies, while making only a negligible change to the geometry [SC20a].

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

$$\ell_{ij} + \ell_{jk} > \ell_{ki} + \delta, \tag{4.1}$$

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

$$\varepsilon := \max_{ijk} \max(0, \delta - \ell_{ki} - \ell_{ij} + \ell_{jk}) \tag{4.2}$$

is the smallest length we can add to *all* edge lengths to ensure that Equation 4.1 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].

4.8 Metric Scaling

Some problems in geometric computating make use of a customized, possibly-anisotropic metric along the domain, such as a spatially-varying speed function affecting distances along a surface [CHK13], or a preferrential alignment for vector fields [Jia+15]. The most direct approach to compute with such metrics is to modify algorithms to incorporate a metric tensor or scaling factor when deriving relevant operators and expressions. However, the intrinsic perspective offers an appealing alternative strategy: rescale the edge lengths ℓ according to the metric, then simply use any ordinary isotropic algorithm. The intrinsic viewpoint is crucial here, because finding a new extrinsic embedding which respescts a custom metric would be a difficult global problem which may not even admit any solutions, whereas scaling intrinsic edge lengths is generally a simple, local operation.

As one concrete strategy, suppose anistropy is specified by a norm $|v|_g$ in each face ijk which measures the length of a tangent vector v in local coordinates (Section 2.3.7). Scaled edge lengths can then be computed by measuring each edge vector u_{ij} as $\ell'_{ij} = |u_{ij}|_g$, averaged over all faces incident on the edge. Any subsequent computation is performed on the intrinsic triangulation defined by these scaled edge lengths ℓ'. For large scalings, the resulting edge lengths may violate the triangle inequality, invalidating the representation—this can be addressed by constraining the edge lengths, or modifying the mesh. This approach, and related concerns, are discussed at length by Campen, Heistermann, and Kobbelt [CHK13]. It should be noted that applying transformations to edges lengths yields an abstract intrinsic triangulation which does not necessarily have an isometric correspondence with the original underlying surface, and thus the data structures from Chapter 3 cannot be directly applied as described.

4.9 Comparison to Traditional Remeshing

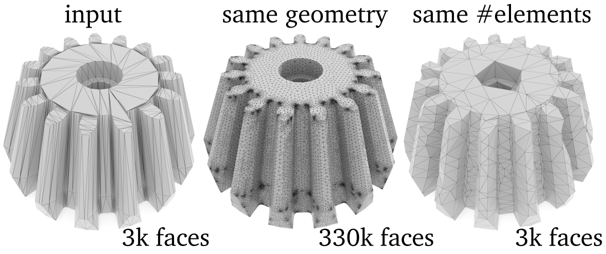


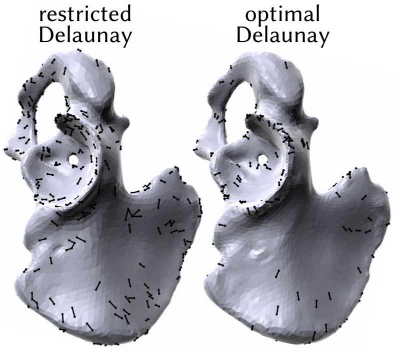
Figure 4.7: Traditional remeshing cannot improve element quality without increasing mesh size or disturbing the geometry; intrinsic triangulations escape this tradeoff.

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.

Another important advantage of the intrinsic approach is efficiency. Whereas surface remeshing often amounts to difficult optimization problems or long-running iterative schemes, intrinsic retriangulation is extremely efficient, more akin to planar triangulation. The procedures described in this section generally have runtimes on the order of milliseconds.

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].

4.9.1 Other Notions of Delaunay



There are many other seemingly-similar notions of Delaunay triangulation which arise in surface mesh generation. One common strategy, referred to as “restricted Delaunay” is to generate a Delaunay tetrahedralization[8] of a point set, and take a subset of the faces of the tetrahedralization as a surface mesh[CDS12, Chapter 13]. Other methods define planar schemes, then project on to surfaces [CH11] In fact, the classic work now known for describing Chew’s 2nd algorithm mainly concerns a notion of circumspheres in 3D space for surface meshes [Che93]. Recent work by Khoury and Shewchuk [KS21] defines a related construction of constrained restricted Delaunay triangulations, though no concrete algorithms are yet known. However, these alternate definitions come without any of the same Delaunay properties, such as positive cotangent weights (inset image, edges with negative cotan weights highlighted in black) or empty geodesic circumballs (Section 4.1.1). The intrinsic Delaunay criterion is the only notion of Delaunay for surface meshes which extends all of these desirable properties of the planar Delaunay triangulations to the surface case.

---

[8] A note on terminology: in some contexts, the term *Delaunay triangulation* may be used for a Delaunay complex of any dimension, such that a “3D Delaunay triangulation” really refers to a set of Delaunay tetrahedra, not triangles.

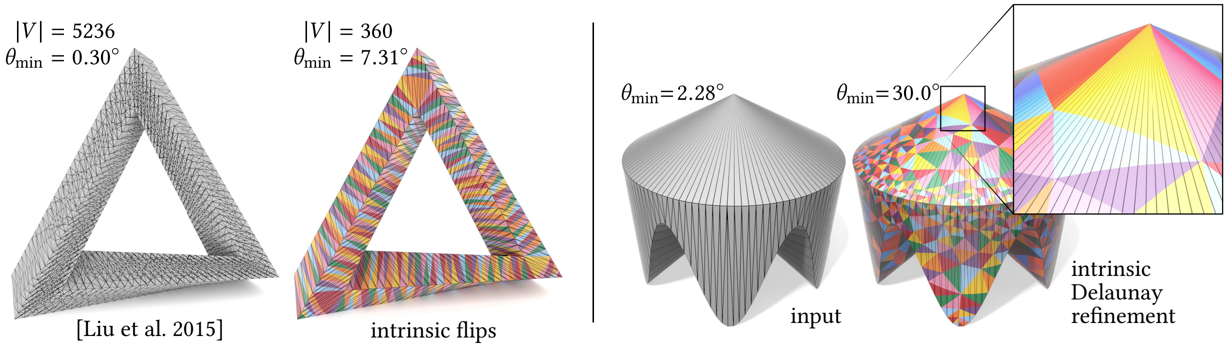


Figure 4.8: *Left*: Extrinsic schemes may need to insert many vertices and create skinny triangles to produce a Delaunay triangulation while preserving the shape, while the intrinsic approach preserves the vertex set and improves triangles quality. *Right*: Small corner angles in an extrinsic triangulation must remain if extrinsic shape is to be preserved, but instead preserving only the intrinsic shape allows these angles to be improved.

4.9.2 Extrinsic Construction

Instead of taking the intrinsic approach, it is also possible to construct traditional, extrinsic triangle meshes which satisfy the intrinsic Delaunay criterion while also preserving the geometry. However, preserving the extrinsic geometry is stricter requirement than preserving only the intrinsic geometry: the resulting meshes may require inserting a large number of elements, whereas the intrinsic approach achieves the Delaunay property without changing the number of elements (Figure 4.8, *left*). In particular, Shewchuk [She02b, Section 7.1] discusses extrinsic Delaunay refinement of surface meshes, which essentially amounts to applying planar Delaunay refinement independently in each triangle. The same can also be achieved using only edge splits and planar flips [DZM07; Liu+15], and in fact Ye et al. [Ye+20] show that interpolating scalar functions to such a mesh yields a Rippa-like smoothness guarantee. However, beyond increasing element counts, a disadvantage of geometry-preserving extrinsic Delaunay remeshing is that such schemes cannot possibly improve small triangle corner angles which appear in an input mesh (Figure 4.8, *right*). In contrast, intrinsic remeshing has significant freedom to improve skinny corner angles while preserving intrinsic geometry.

4.10 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 extrinsic mesh. The basic pipeline is:

1. intrinsically retriangulate the extrinsic mesh,
2. solve the problem on the intrinsic triangulation,
3. transfer the solution back to the extrinsic mesh.

Crucially, we can generally run existing geometric algorithms directly on an intrinsic triangulation: there is no need reinvent or re-derive the algorithms. Intrinsic triangulations still offer a familiar mesh interface, and are equipped with linear basis functions in triangles widely used in geometry processing (higher-order basis can of course likewise be constructed if desired). The most common change is

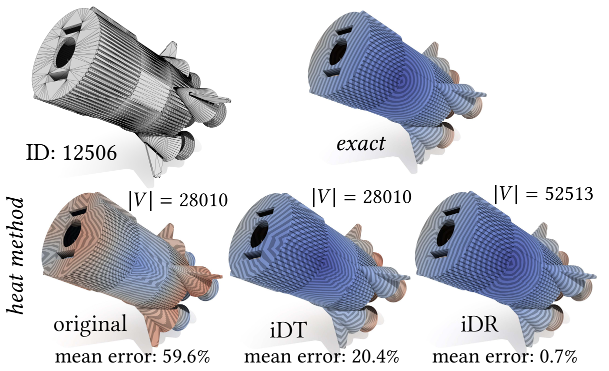


Figure 4.9: Intrinsic triangulations dramatically improve the quality of solutions from PDE-based geometry processing algorithms such as the heat method when run on low-quality geometry. [SSC19a]

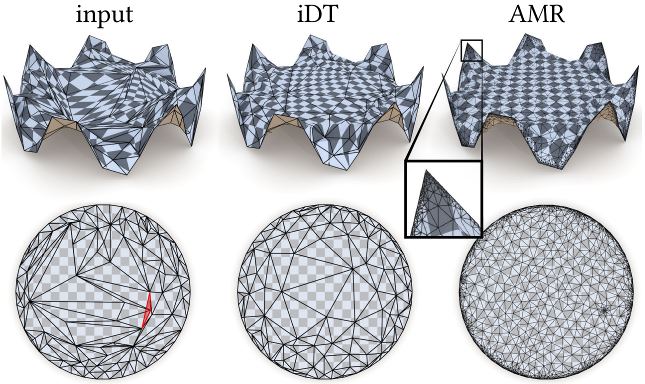


Figure 4.10: Using an intrinsic Delaunay triangulation ensures that a harmonic parameterization is flip-free, while adaptive mesh refinement provides high-resolution in the interesting regions of the mesh. [SSC19a]

simply to evaluate geometric quantities directly from edge lengths rather than vertex positions—see Appendix A for some useful expressions. Many algorithms involve only intrinsic data and operators, but there are some in which extrinsic operations play a crucial role, such as bending energies—it is not yet straightforward to apply intrinsic to these problems. However, a common setting is an intrinsic *operator* (*e.g.* the Laplacian) applied to extrinsic data (*e.g.* vertex positions), such as the surface editing context in Figure 6.1. In this case, intrinsic triangulations can be used to build a high-quality operator, which is then applied to the extrinsic data, improving robustness.

4.10.1 The Intrinsic Delaunay Laplacian

The most widespread usage of intrinsic triangulations is to construct the intrinsic Delaunay triangulation for a low-quality mesh, and then read off the corresponding Laplace matrix, the *intrinsic Delaunay Laplacian* [BS07]. This Laplacian has a variety of desirable properties, including a guarantee of a maximum principle (Section 2.5), and improved element quality (by maximizing minimal corner angles, Section 4.1.1, see [She02a] for further discussion). The most basic usage of the intrinsic Delaunay Laplacian is to simply build a better matrix, and substitute it in place of the ordinary cotan Laplacian. This is already remarkably effective in practice, although further benefits may be had by evaluating an entire algorithm on a high-quality intrinsic triangulation, as opposed to just building the Laplace matrix. The applications we explore will improve accuracy and robustness both through the use the intrinsic Delaunay Laplacian, as well as other techniques such as evaluating other operators beyond the Laplacian, and computing on an intrinsic Delaunay *refinement* with guaranteed element quality.

4.10.2 Examples

We illustrate robustness with intrinsic triangulations through several examples.

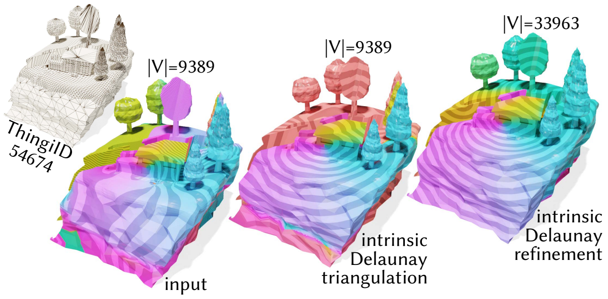

ThingiID 54674
|V|=9389
input
|V|=9389
intrinsic Delaunay triangulation
|V|=33963
intrinsic Delaunay refinement

Figure 4.11: 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]

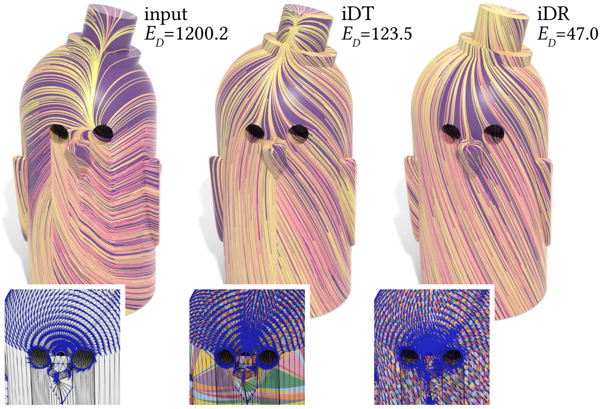


Figure 4.12: The signpost data structure also enables processing tangent data. Here, smoothest vector fields computed on an intrinsic triangulation have much lower Dirichlet energy than one computed on a low-quality mesh. [SSC19a]

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 representative example, computing approximate geodesic distance on a mesh by solving a short-time diffusion equation [CWW13]. Evaluating the algorithm on an intrinsic retriangulation greatly improves the accuracy of the solution to the diffusion equation, which in turn leads to much more accurate distances on low quality meshes. (Figure 4.9)

The Logarithmic Map. The *logarithmic map*, the inverse of the exponential map (Section 2.4.2), is useful as a parameterization of a surface mesh [SGW06] [9]. This parameterization can be computed with the vector heat method [SSC19b], which again amounts to solving a diffusion equation, but this time diffusing tangent vectors rather than just scalar functions. Once again, we apply the method on an intrinsic triangulation without any other modifications, and observe significantly more accurate results. Although this algorithm uses tangent vectors during computation, the end result is simply a pair of functions, and thus the procedure could be evaluated on the simple edge length-only data structure, with no additional correspondence tracking. However, in Figure 4.11 we visualize the intrinsic logarithmic map represented in the bases of the intrinsic triangulation, which requires a more sophisticated data structure to construct the common subdivision.

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 vector field, rather than a scalar function, so even copying values at vertices requires some sort of correpondence data structure. (Figure 4.12)

---

[9] Note that Schmidt *et al.* refer to the logarithmic map as the "exponential map".

4.11 Transferring Solutions Between Triangulations

After computing the solution to a problem on the intrinsic triangulation, one often wants to obtain a solution on the extrinsic mesh. The simplest strategy is to simply copy values at vertices back to the extrinsic mesh. Since intrinsic retriangulation never removes extrinsic vertices, this operation is always well-defined, and can even be performed when using only the abstract edge length representation. This approach is already quite effective in practice, but it is not optimal in any sense, and does not apply to more general tangent vector data, so we will describe some alternative notions of data transfer.

4.11.1 Optimal Attribute Transfer

Rather than simply copying values back at vertices, we can seek the function on the extrinsic mesh which is closest to the intrinsic solution, in the L^2 sense. As described by Gillespie, Sharp, and Crane [GSC21a], this amounts to a sparse linear system constructed over the common subdivision. Other notions of closeness, and other basis functions could easily be treated similarly. Note that this operation is impossible using only intrinsic edge lengths; it requires the common subdivision, as provided by rich mesh data structures (Chapter 3).

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

$$\|f - \hat{f}\|_{L^2}^2 := \int_M \left|f(x) - \hat{f}(x)\right|^2 \, dx. \tag{4.3}$$

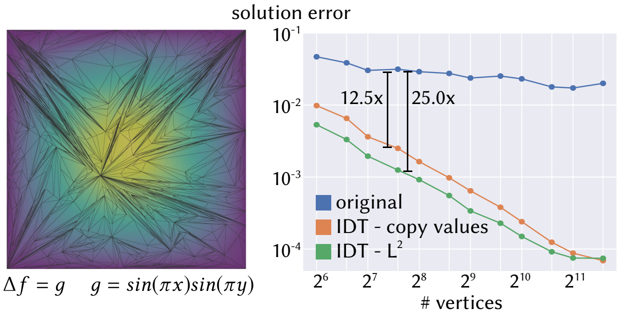
If f and $\hat{f}$ were defined over a *single* mesh, then we could evaluate this distance exactly using the Galerkin mass matrix M (Section 2.5.1):

$$\|f - \hat{f}\|_{L^2}^2 = (f - \hat{f})^T \mathsf{M} (f - \hat{f}), \tag{4.4}$$

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. However, f and $\hat{f}$ are defined on different triangulations: T_{int} and T_{ext}, respectively. Nonetheless, f and $\hat{f}$ are both piecewise-linear over the common subdivision $\mathcal{S}$. Thus, we can evaluate the L^2 distance by interpolating both functions to $\mathcal{S}$ as

$$\|f - \hat{f}\|_{L^2}^2 = (P_{\text{int}} f - P_{\text{ext}} \hat{f})^T \mathsf{M}_{\mathcal{S}} (P_{\text{int}} f - P_{\text{ext}} \hat{f}), \tag{4.5}$$

where $\mathsf{M}_{\mathcal{S}}$ is the Galerkin mass matrix of the common subdivision, and $P_{\text{ext}}, P_{\text{int}}$ are interpolation matrices mapping functions on the extrinsic and intrinsic triangulations to $\mathcal{S}$ *resp.*. In particular, P_{ext} is a $|V_{\text{ext}}| \times |V_{\mathcal{S}}|$ matrix where each row corresponds to a vertex of $\mathcal{S}$ and has that vertex's barycentric coordinates on the extrinsic mesh as entries, and likewise for P_{int}. We can now find the function $\hat{f}$ which is L^2-closest to the intrinsic function f simply by minimizing Equation 4.5, which amounts to solving a linear system.



$\Delta f = g \quad g = sin(\pi x)sin(\pi y)$

The inset example illustrates the benefits of this technique, improving the accuracy of solutions on near-degenerate triangulations [GSC21a]. We begin by generating a low-quality mesh of the unit square via random edge splits, and solve the Poisson equation $\Delta f = \sin(\pi x)\sin(\pi y)$ on this extrinsic mesh, then solve the same problem on the intrinsic Delaunay triangulation. We transfer the intrinsic solution back to the extrinsic mesh by copying values at vertices, as well as via the L^2 projection described above, yielding two more solutions on the extrinsic mesh. Finally, we compute the error of each solution as represented in the basis of the original extrinsic mesh against an analytic ground truth, and plot the average error after 100 trials on randomly generated meshes of the domain. Copying back intrinsic solution values at vertices provides already improves accuracy by 12.5× compared to solving directly on the extrinsic mesh, and picking the L^2-closest function results in even better solution, decreasing error by a factor of 25× (inset figure).

4.11.2 Transferring Tangent Vectors

Sometimes, the quantity to be transferred is a tangent vector field (Section 2.4) rather than a scalar function. In this case, even copying the solution back at vertices becomes complicated, as tangent vectors on the intrinsic and extrinsic meshes are represented in different coordinate systems. Copying tangent vectors is easiest to do using the signpost data structure, since signposts directly encode the mapping between these coordinate systems, as discussed in Section 3.4.3. In other representations, a change of basis can be computed by comparing the angle between edges in T_{int} and T_{ext} on the common subdivision.

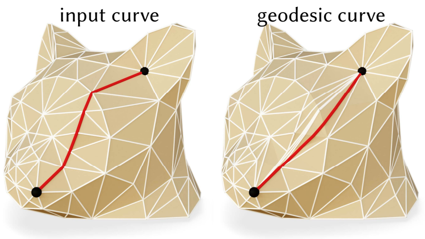
Chapter 5

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 (Section 2.4.1). Fast and accurate computation of geodesics enables a multitude of algorithms throughout science and engineering [Bos+11], and the ability to construct "straight lines" on polyhedral surfaces allows us to run classic algorithms from 2D computational geometry on curved surfaces. Here, we consider *exact, polyhedral* geodesics, which are exactly geodesic along the piecewise-flat geometry of a mesh, 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]: they start at the source, and propagate "windows" of geodesic paths sharing a common history in a traversal similar to Dijkstra's algorithm. 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].

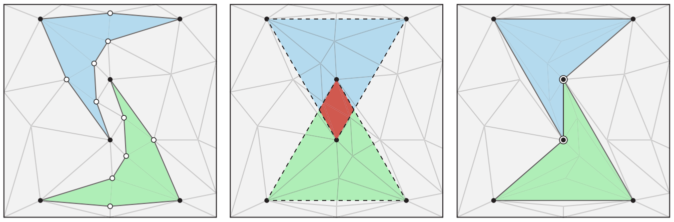
However, in geometry processing we will often also need to consider geodesic paths which are not necessarily globally-shortest geodesics. For instance, every edge of an intrinsic triangulation is a geodesic path, but not necessarily the *shortest* geodesic path between the endpoints. Though less widely-studied 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 as a lists of vertices which move along the surface or in $\mathbb{R}^3$ [HS94; MVC05; XW07; ASS09; XHF11; Han+17; Liu+17a; RŠN19]. On the other hand, *Eulerian* methods encode curves as the level sets of real-valued functions on the surface [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, in Sharp and Crane [SC20b] we show 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 3.5.1). 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.

5.1 Geodesics from Intrinsic Edge Flips

The edges in an intrinsic triangulation are always geodesics along the surface (Section 2.3.4); 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 T_{int} (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 T'_{int}. 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 Chapter 3.


input regions

Lagrangian straightening (regions not well-defined)

flip-based straightening (regions are preserved)

Noncrossing Curves. This algorithm will operate 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 is important in computational fabrication, or satisfy smoothness objectives in optimization. Flip-based geodesics are particularly suited for straighting such curves precisely because they will guarantee to preserve the noncrossing property—otherwise, unintentional crossings would render the curves useless for the corresponding applications (inset). In our algorithms, a *flexible joint* is a local region of the path which can be straightened without introducing a crossing. In the notation of Figure 5.2, the joint is flexible if there are no other path segments in the region swept by the angle labelled α_{abc}. Sharp and Crane [SC20b, Section 3.2] describe a data structure for tracking flexible joints, efficiently handling arbitrary path configurations.

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 2.3.4, 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

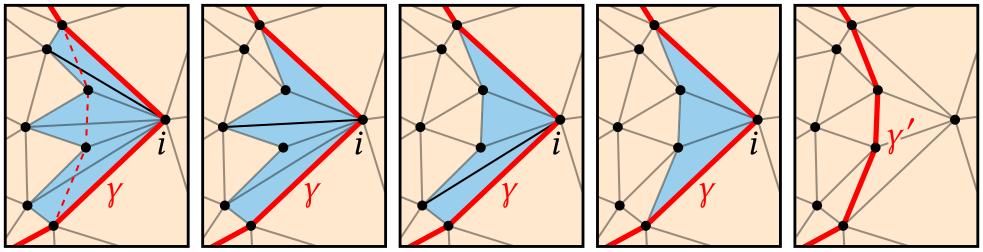


Figure 5.1: FLIPOUT shortens the curve γ by repeatedly flipping edges to introduce a shorter path.

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 6 FLIPOUT($T_{\text{int}}, \gamma_{abc}$)

Input: A triangulation T_{int}, 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 T_{int}.

```
while any $\beta_i < \pi$ do
    $j \leftarrow \min i$ s.t. $\beta_i < \pi$        ▷choose the first edge with $\beta_i < \pi$
    FLIPEDGE($T_{\text{int}}, bn_j$)
end while
$\gamma_{\text{shorter}} \leftarrow (a, n_1, \ldots, n_{k-1}, c)$        ▷path along the outer arc
return $T_{\text{int}}, \gamma_{\text{shorter}}$
```

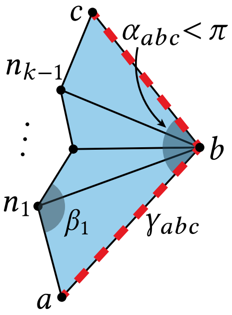


Figure 5.2: FLIPOUT notation.

This procedure is easiest to conceptualize in a planar triangulation (as shown in Figure 5.1). 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 geodesics, along curved surfaces.

Theorem 2 (Sharp and Crane [SC20b])**.** *When* FLIPOUT *terminates,* $|\gamma_{shorter}| < \gamma_{abc}$, i.e. *the new path is shorter than the input path.*

Proof. Upon termination the angles β_i are all greater than or equal to π. Hence, in the planar layout, γ_{shorter} is a convex curve contained in the initial curve γ_{abc}. A corollary of *Crofton's formula* [Cro68] is that for two nested convex curves sharing endpoints, the inner one is shorter. Since the planar layout is isometric, the lengths of curves in this planar diagram exactly match the lengths of edge paths on the surface, and thus $|\gamma_{\text{shorter}}| < |\gamma_{abc}|$. □

For a full proof that FLIPOUT terminates, even in the case where T_{int} may be a general Δ-complex, see Sharp and Crane [SC20b, Appendix A].

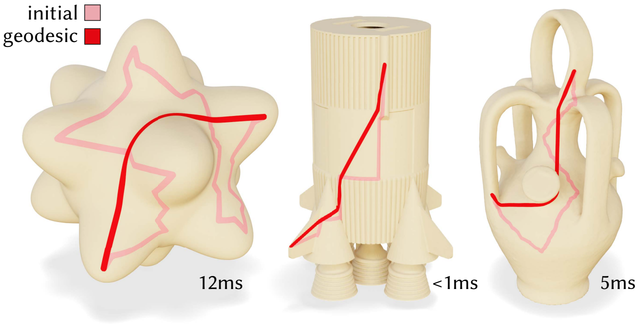


Figure 5.3: Basic results shortening an initial path to a geodesic by flipping edges. Inset values give the runtimes. All of the resulting curves shown are exact polyhedral geodesics.

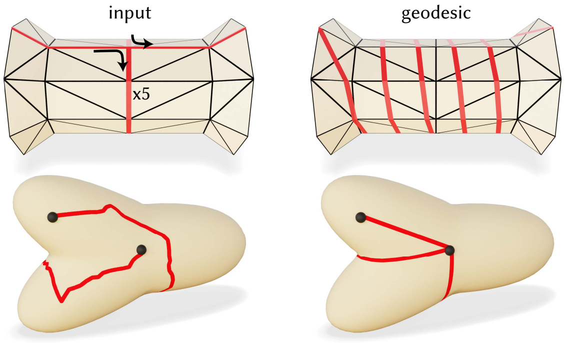


Figure 5.4: Careful treatment of noncrossing curves enables finding geodesics that overlap many times (*top*), or those that get pulled tight around endpoints of the path itself (*bottom*).

Geodesic Paths. Starting from an arbitrary edge path $\gamma_{x\leftrightarrow y}$ between vertices x and y, we can compute an exact polyhedral geodesic in a finite number of steps by simply applying FLIPOUT repeatedly (Figure 5.3). Each iteration shortens the path until we obtain a locally shortest, *i.e.* geodesic, path. Algorithm 7 describes the procedure, while Theorem 3 argues that it must terminate after a finite number of steps. In practice we suggest always processing the flexible joint with smallest curve corner angle, by maintaining a priority queue of flexible joints sorted by corner angle.

Algorithm 7 MAKEPATHGEODESIC($T_{\text{int}}, \gamma_{x\leftrightarrow y}$)

```
Input: A triangulation $T_{\text{int}}$ 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 $T_{\text{int}}$.
while $\gamma_{x\leftrightarrow y}$ is not geodesic do
    $\gamma_{abc} \leftarrow$ flexible joint in $\gamma_{x\leftrightarrow y}$ with smallest angle $\alpha_{abc}$
    $T_{\text{int}}, \gamma_{\text{shorter}} \leftarrow$ FLIPOUT($T_{\text{int}}, \gamma_{abc}$)    ▷locally shorten
    $\gamma_{x\leftrightarrow y} \leftarrow$ UPDATEPATH($\gamma_{x\leftrightarrow y}, \gamma_{abc}, \gamma_{\text{shorter}}$)    ▷Replace old subpath $\gamma_{abc}$ with new subpath $\gamma_{\text{shorter}}$
end while
return $T_{\text{int}}, \gamma_{x\leftrightarrow y}$
```

By definition, if Algorithm 7 terminates then the result is an exact polyhedral geodesic along the surface, so to understand its correctness we need only argue that it must terminate. Note that much like the termination proof for the Delaunay flipping algorithm (Section 4.2), we merely argue that this procedure passes through a collection of states which is finite, albeit exponentially large. Giving any meaningful bound on the time complexity of the procedure is still an open question, though it is very efficient in practice.

Theorem 3 (Sharp and Crane [SC20b]). MAKEPATHGEODESIC *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. □

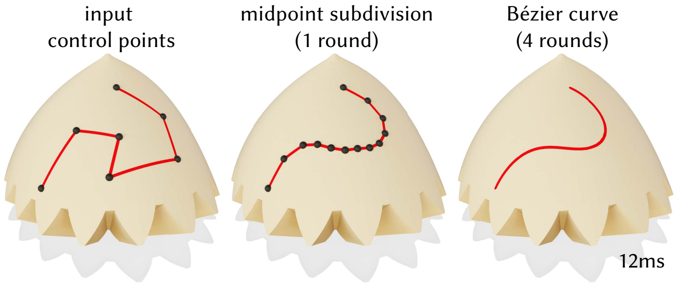
Instead of running MAKEPATHGEODESIC until convergence, one can also stop when the length has decreased by a sufficient amount. This generates curves which are shorter and straighter but not fully geodesic, similar to the intermediate results of a curve-shortening flow (Figure 5.5).

The only other tool 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 mark edges which make up the path, but to support any more complicated configurations which may arise (Figure 5.4), one may employ a data structure which encodes an ordered stack of path segments along each edge of a mesh as in Sharp and Crane [SC20b, Section 3.2]

5.2 Geodesic Loops and Curve Networks

The iterative shortening procedure in Algorithm 7 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 segments ab and bc in Algorithm 6 are in fact the same segment. In this case, one should replace the single segment of the triangle with the two opposite of the containing triangle [1]. Constructing geodesic loops (Figure 5.6) 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 5.7), or as the boundaries of regions on a surface.

5.3 Geodesic Bézier Curves



Bézier curves are an indispensable tool in geometric modeling. Classically defined in the plane, Bézier curves were extended to polygonal surfaces by Morera, Carvalho, and Velho [MCV08] via a geodesic version of de Casteljau's algorithm. The basic idea is that

---

[1] The proof of termination does not handle this case, but termination has always been observed in practice. For details see Sharp and Crane [SC20b, Appendix B].

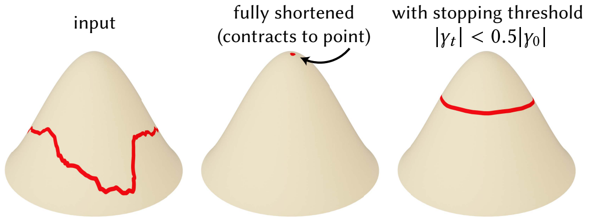


Figure 5.5: Algorithm 7 acts as discrete curve-shortening flow; stopping the procedure early via a length or angle threshold generates straighter curves, without drifting too far from the initialization or contracting to a point.

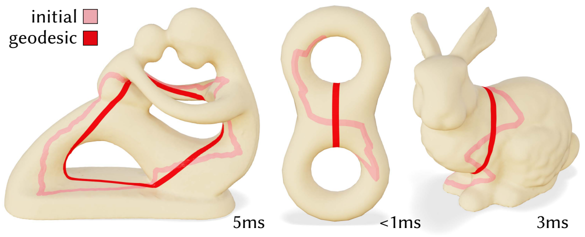


Figure 5.6: Geodesic loops generated by with edge flips in an intrinsic triangulations. Inset values give the runtimes.

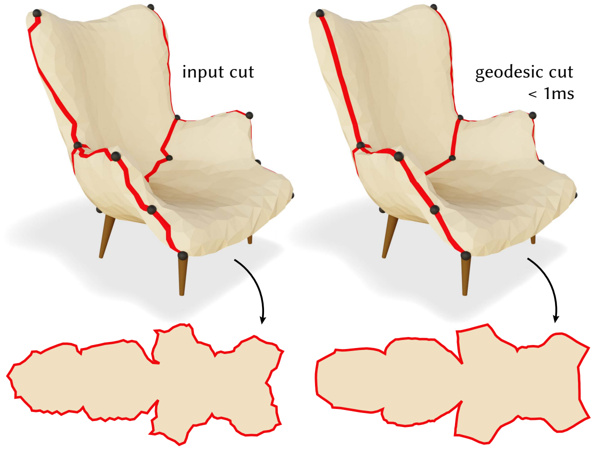


Figure 5.7: Curve networks arise when cutting and flattening a shape for computational fabrication. Our method is perfectly suited to straighten an initial cut network along edges (*left*) to a geodesic network (*right*), yielding a much more natural pattern for fabrication (*bottom*).

given an ordered list of control points (connected by Dijkstra paths), one can transform the control polygon into a smooth Bézier curve by repeatedly applying the following steps:

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 MakePathGeodesic for the first step (inset), we can construct intrinsic triangulations which contain approximate Bézier curves. Incorporating this Bézier curve construction into our retriangulation pipeline has numerous benefits, in particular because we also generate triangulations which conform to these paths.

5.4 Triangulated Geodesic Paths

When constructing geodesics by applying the FlipOut procedure to an intrinsic triangulation, we not only construct the geodesic path itself, but also a triangulation which contains that path among its edges. This triangulation may have many extremely poor quality triangles in it (*e.g.*, with very skinny corner angles), but it can easily be improved as a post-process by applying the retriangulation schemes discussed in Chapter 4 while preserving the path edges. In fact, Section 4.4 discusses how this FlipOut

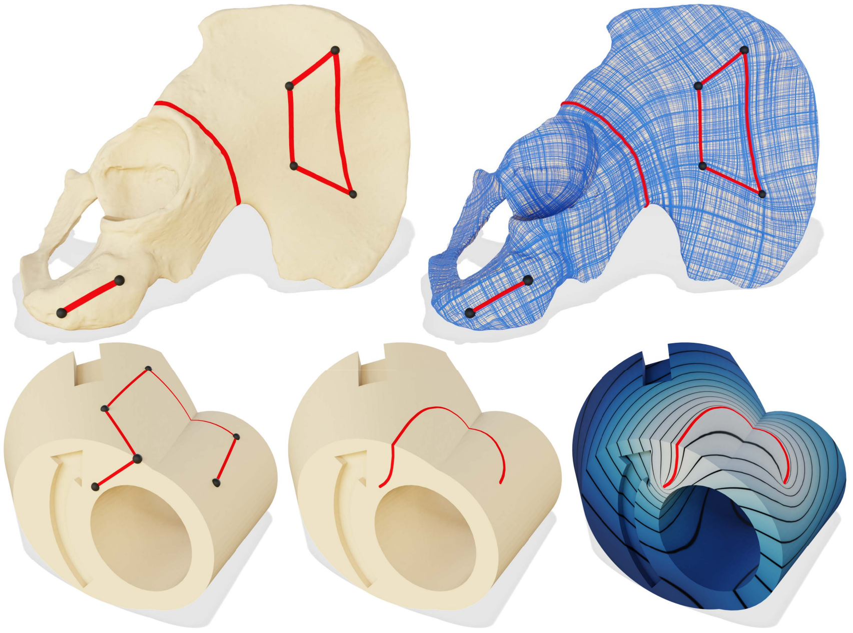


Figure 5.8: PDEs taking boundary conditions from constrained intrinsic triangulations. *Top*, a cross field conforming to curves on a 3D scan of a pelvis [Knö+13], and *bottom*, a Poisson equation with boundary conditions defined along a Bézier curve on a mechanical part.

procedure fills a very important role for generating constrained intrinsic triangulations which align to desired curves. In applications, these constrained edges can then be used as boundary conditions or guiding features for geometry processing algorithms; Figure 5.8 shows two such examples.

5.5 Single-Source Geodesics

The methods described above are a kind of curve-shortening flow: they take some particular curve along the edges of an intrinsic triangulation 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 problem, where one seeks a geodesic path to all other vertices in a mesh.

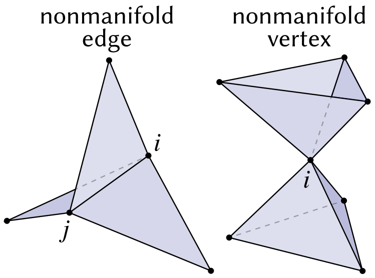
Sharp and Crane [SC20b] also present a simple algorithm for this task based on the FLIPOUT subroutine. The basic idea of which is to run a Dijkstra search outward from the source vertex, while constantly applying the FLIPOUT procedure at the frontier to ensure all accepted paths are geodesic. Of course, 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 6

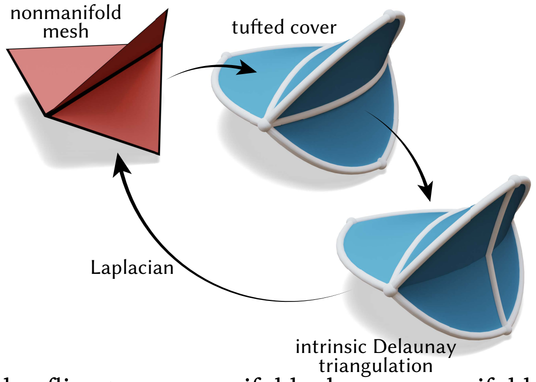
Generalized Domains

The standard definition an intrinsic triangulation relies on strong assumptions about the domain: it must be a manifold, oriented triangle mesh. In this section we relax these assumptions, generalizing intrinsic triangulations to nonmanifold meshes and point clouds, corresponding to the work in [SC20a].

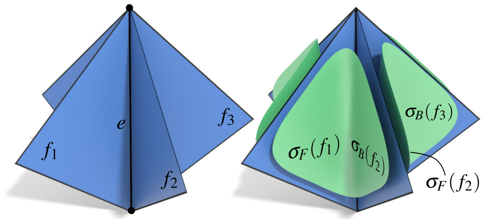
6.1 Nonmanifold Intrinsic Triangulations



Recall that manifold connectivity requires that any neighborhood of the surface looks like the plane (Section 2.1). Manifold connectivity endows a mesh with numerous useful mathematical properties and is essential for many of the constructions in these notes. However, nonmanifold features (see inset) often arise in the geometric data that one encounters in practice, either intentionally or due to noise. Moreover, a lot of data is stored in point clouds which have even less structure: they lack any connectivity information at all. Handling such inputs is crucial to leverage intrinsic triangulations for robust geometry processing. This section introduces a simple technique introduced by Sharp and Crane [SC20a], which enables the construction of an intrinsic Delaunay triangulation all triangle meshes, even those which may be nonmanifold and nonorientable, via a covering space. With this same technique, we can also generate intrinsic Delaunay triangulations of point clouds which have no connectivity information whatsoever (Section 6.2).



The Tufted Cover. To construct intrinsic triangulations of nonmanifold meshes, we use a special covering of the mesh called the *tufted cover*. It has the same vertex set as the nonmanifold mesh but twice as many faces, which we glue together along their edges according to a 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. By construction the tufted cover is edge-manifold, so we can perform edge flips on it, allowing us to *e.g.* obtain an intrinsic Delaunay triangulation.



We build 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 is always an edge manifold, closed, and oriented triangulation. The resulting triangulation has 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. 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 extrinsic mesh $T_{\text{ext}} = (V_{\text{ext}}, E_{\text{ext}}, F_{\text{ext}})$ is a triangle mesh $\tilde{T} = (\tilde{V}, \tilde{E}, \tilde{F})$ with the same vertices ($\tilde{V} = V_{\text{ext}}$), together with a *gluing map* $\tilde{G}$. Recall from Section 2.2 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, encoded as a pair (f, s) of a face f and a side within that face s.

For each face $f \in F_{\text{ext}}$, $\tilde{T}$ 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 $e \in E_{\text{ext}}$ 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 8; here $\textsc{Side}(e, f)$ just gives the side index of e within face f (1, 2, or 3).

Algorithm 8 $\textsc{ConstructTuftedCover}(T_{\text{ext}}, \rho)$

Input: A (possibly nonmanifold) triangle mesh M_{ext} and an ordering ρ of faces around each edge.
Output: The tufted cover mesh $\tilde{T}$ and edge glue map $\tilde{G}$

```
$\tilde{F} \leftarrow \bigcup_{ijk \in F_{\text{ext}}} \{ijk, jik\}$        ▷two copies of each face
$\tilde{G} \leftarrow \{\}$        ▷assemble an edge glue map
for each edge $e \in E_{\text{ext}}$ do
    if $e$ and $\sigma_F(\rho^e_1)$ have the same orientation then
        $f \leftarrow \sigma_F(\rho^e_1)$
    else $f \leftarrow \sigma_B(\rho^e_1)$
    for $i = 1, \ldots, k$ do        ▷letting $k := |\rho_e|$
        $g_1 \leftarrow \sigma_F(\rho^e_{i+1 \bmod k})$
        $g_2 \leftarrow \sigma_B(\rho^e_{i+1 \bmod k})$
        if $f$ and $g_1$ have different orientation along $e$ then
            $\textsc{Swap}(g_1, g_2)$
        $\tilde{G}(f, \textsc{Side}(e, f)) \leftarrow (g_1, \textsc{Side}(e, g_1))$
        $\tilde{G}(g_1, \textsc{Side}(e, g_1)) \leftarrow (f, \textsc{Side}(e, f))$
        $f \leftarrow g_2$
return $\tilde{T}, \tilde{G}$
```

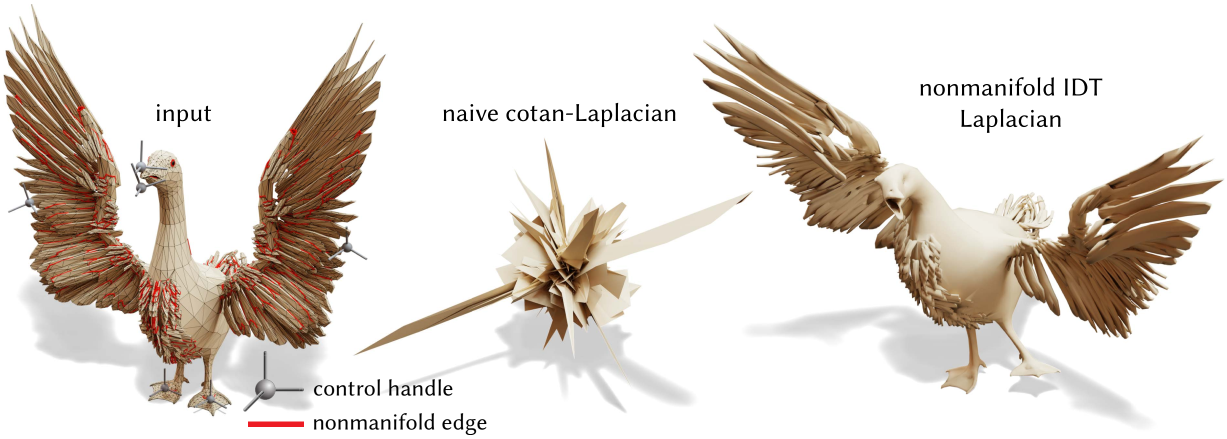


Figure 6.1: Robustness in the context of differential surface editing [Yu+04; Lip+04; Sor+04], where a system of equations involving a Laplacian is solved to deform a 3D model. Applying these techniques naively in the extrinsic mesh, which is nonmanifold and has many low-quality triangles, yields only numerical noise. Substituting the nonmanifold IDT Laplacian constructed on the tufted cover generates the expected smooth deformation.

It should be emphasized that the tufted cover is not globally vertex-manifold; 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 we have not otherwise rectified the nonmanifoldness.

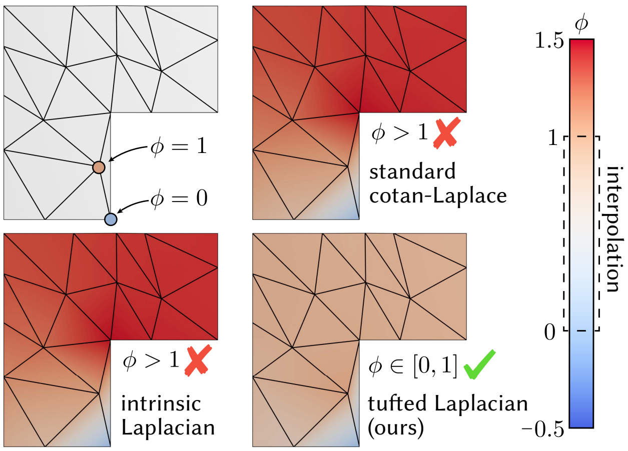
We construct the intrinsic Delaunay Laplacian on the tufted cover exactly as described in Section 4.2, dividing the matrix by a factor of 2, because the tufted cover is a double cover of the original mesh. This construction offers a high-quality cotan-Laplace matrix for nonmanifold meshes for the first 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 6.1).

Beyond Intrinsic Delaunay Laplacians.. In [SC20a] we focus 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 Chapter 4. It is reasonable to wonder whether the other techniques in this text can be likewise generalized to the nonmanifold 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 explicit crossing representation (Section 3.3) applies to the tufted cover without modification[1]. However, signposts (Section 3.4) 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. The integer coordinates representation (Section 3.5) requires similar treatment for its roundabouts. Intrinsic Delaunay refinement (Section 4.3) is well-defined on the tufted cover, with the caveat that new vertices should be inserted as “tufted” vertices, rather than separately on each sheet of the cover. Edge flip geodesics (Chapter 5) can likewise be applied to paths along a sheet of the tufted cover, though the path will stay along the sheet of the cover where it was initially constructed, which may or may not be the expected result.

---

[1] As long as the mesh data structure supports nonmanifold meshes.



Domains with boundary.. The tufted intrinsic Delaunay Laplacian has useful properties not only for nonmanifold meshes, but for any mesh with boundary. Both the extrinsic cotan-Laplacian and the intrinsic Delaunay 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. These 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. However, since the tufted cover is always closed it has no boundary edges, and therefore all weights of its intrinsic Delaunay Laplacian will be nonnegative—even for edges on the boundary of the original nonmanifold mesh. For this reason, harmonically interpolating any set of pinned values will yield a function bounded within the range of these values, whereas for standard intrinsic Delaunay triangulations this property does not hold along the boundary. 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[2]. This property provides additional robustness for algorithms built on top of interpolated weights, Green's functions, *etc.*

6.2 Point Clouds

Point clouds, *i.e.* sets of points $p \in \mathbb{R}^3$, are a another common surface representation in geometry processing; however, the total absence of connectivity information makes many computations more difficult on point clouds than on meshes. 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 take the union of many local triangulations, obtaining connectivity which represents the surface well but creates many nonmanifold edges, repeated triangles, *etc.* We can then build the tufted intrinsic Delaunay triangulation, since the tufted cover frees us of any constraints on the connectivity of our mesh.

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 can then be used to build a Laplacian in a variety of ways. For instance, both Belkin *et al.* [BSW08] and Liu *et al.* [LPG12] use the triangulations to determine the mass matrix M, and determine edge weights via a Gaussian function of the distance in $\mathbb{R}^3$. Such schemes are accurate and have nice theoretical properties, such as pointwise convergence for fairly uniform

---

[2] We note that harmonic interpolation by pinned values at vertices is not a well-posed convergent discretization, but nonetheless it is common and pragmatic.

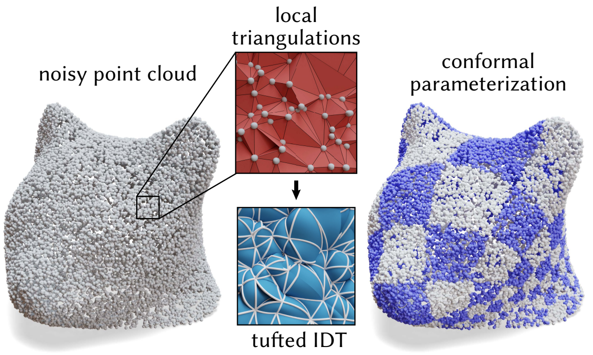


Figure 6.2: The tufted intrinsic Delaunay point cloud Laplacian, demonstrated here for spectral conformal parameterization of a point cloud [Mul+08].

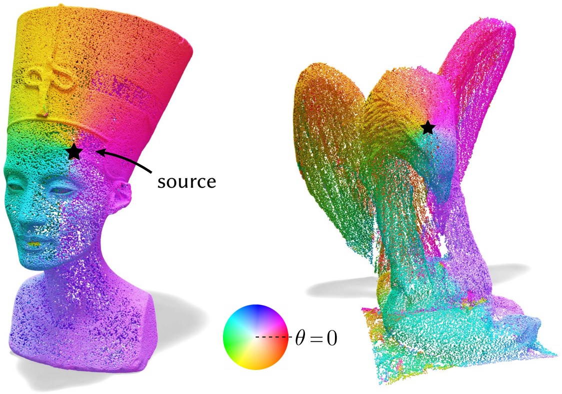
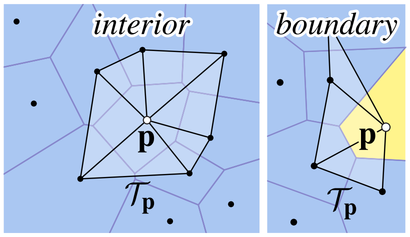


Figure 6.3: Angular parameterizations of point clouds, computed with the tufted intrinsic Delaunay point cloud Laplacian via the vector heat method [SSC19b].

point distributions, but involve numerical parameters which are difficult to estimate and produce matrices that are far more dense than the cotan Laplacian of a mesh. Alternate 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 latter 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 take the union $\mathcal{T} = \bigcup_{p \in P} \mathcal{T}_p$ of 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 the triangles triply-cover the local neighborhoods.) This Laplacian exhibits all the desired properties (symmetry, positive-definiteness, nonnegative edge weights, *etc.*) while remaining very sparse. And 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 6.2 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 6.3 we show 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 7

Conclusion

Geometry processing offers a wealth of algorithmic tools for manipulating 3D surfaces, but these tools are only as good as the representations we apply them to. The intrinsic triangulations explored in this thesis open doors to new and better representations of shapes, offering the freedom to retriangulate a surface while always preserving its geometry. To make use of this flexibility, we must slightly change how we think about triangulations—shifting from an embedding defined by vertex positions to a metric defined only by edge lengths. Although this definition is a slight departure from the norm in geometry processing, it mirrors the turn to intrinsic formulations in differential geometry, and still supports a broad range of common algorithms which require only an intrinsic formulation.

To manipulate intrinsic triangulations in practice, we need data structures which support queries involving the original underlying geometry. The work in this thesis contributes two new data structures, the signpost representation and integer coordinates, which are both efficient and robust (Chapter 3). Furthermore, we concretely state how these representations can be used to support retriangulation operations on intrinsic surfaces (Chapter 4). A benefit of the intrinsic approach is that many planar retriangulation routines, such as Delaunay triangulation and refinement, can be directly generalized to surfaces while retaining the same guarantees. These high-quality intrinsic triangulations are excellent computational domains for geometry processing—they offer guaranteed element quality, exactly preserve intrinsic geometry, and can be constructed extremely efficiently. Many existing routines in geometry processing gain remarkable robustness when applied directly on an intrinsic triangulation. We also show how a carefully-constructed covering space can extend many of the same benefits to more general data, such as nonmanifold meshes and point clouds (Chapter 6). Furthermore, these triangulations hold great promise beyond retriangulation: as a first example, we present a new edge flip-based algorithm for constructing geodesic paths on surfaces (Chapter 5). Together, these contributions develop intrinsic triangulations as a powerful tool in geometry processing, and we are optimistic that there is much more to be built on top of this tool.

Looking forward, an important next phase for intrinsic triangulations is promoting widespread adoption. To again draw a comparison to numerical linear solvers, there was a wide gap between the development of the algorithms and their uptake as a standard automatic tool in numerical packages—this is the gap which we must cross with intrinsic triangulations to realize their full value for the community. Intrinsic triangulations have an important role to play, enabling many powerful algorithms in geometry processing to be applied to low-quality data, but only if we continue to develop them both in theory and in practice. With an investment of energy and creativity, this will be an exciting area of research for many years to come.

7.1 Open Questions

We conclude with open questions in the area of intrinsic triangulations, several of which are the subject of ongoing work.

Asymptotic complexity of Delaunay flipping. In practice, Delaunay flipping seems to run significantly faster than known runtime bounds suggest (Figure 4.3). Are there special classes of triangulations on which the algorithm is guaranteed to be fast, or better general asymptotic bounds on the runtime of the algorithm?

Lifting definitions of Delaunay. Can you generalize the parabolic lifting definition of Delaunay to surfaces (Section 4.1)? There is also a similar lifting definition for the sphere: the connectivity of the Delaunay triangulation of a set of points in $\mathbb{R}^2$ is the convex hull of the stereographic projection of those points onto the sphere. Does this generalize somehow to intrinsic Delaunay triangulations of surfaces?

Hybrid data structures. Can you augment the signpost data structure with normal coordinates to obtain a data structure which provably encodes the correct correspondence while being faster than the full integer coordinate data structure?

Exact predicates. The Delaunay flipping algorithm is only guaranteed to terminate in real arithmetic. In floating point, one often uses various epsilon tolerances to make the algorithm terminate on difficult inputs. Exact predicates have been successfully applied to similar problems [DP03], but have proved difficult to apply in the setting of intrinsic triangulations, as the necessary predicates are not functions of a fixed amount of input data. For example, the length of an intrinsic edge can depend on the lengths of arbitrarily many original edges. Can exact predicates, or similar ideas, help to compute the exact Delaunay triangulation in floating point?

Truly integer-only intrinsic triangulations. The integer coordinate correspondence data structure (Section 3.5.1) does not depend purely on integers—it also stores edge lengths and barycentric coordinates in floating point. However this is, in some sense, merely a performance optimization. In theory, these edge lengths and positions could be computed from the normal coordinates, roundabouts, and the geometry of the input mesh, leading to a true integer-only correspondence data structure. In practice, a naive implementation of this integer-only data structure is prohibitively slow, taking hours to compute Delaunay triangulations on even the simplest of models. But this notion still seems appealing theoretically, and perhaps a more efficient implementation could make it a viable option in practice.

The embedding problem. Suppose you're given only the connectivity of a triangulation and its edge lengths. Can this metric be embedded as a Euclidean polyhedron in $\mathbb{R}^3$, or $\mathbb{R}^n$? If so, which triangulation is needed to construct a piecewise linear embedding? How do you construct this embedding? Can you determine the embeddable triangulation without actually constructing the embedding (and in particular, is the former problem any easier than the latter?). This problem was solved in the convex case by Bobenko and Izmestiev [BI08], who provide an algorithm based on convex optimization and intrinsic edge flips; Kane, Price, and Demaine [KPD09] give a pseudopolynomial time algorithm to actually compute the solution to a given numerical precision. The problem remains open for the general nonconvex case.

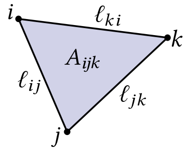
Nonplanar faces. Many polygonal meshes encountered in practice have nonplanar faces, *i.e.*, polygons whose vertices do not all lie in a common plane. A basic question is how to interpret such polygons geometrically—for instance, should one fit a smooth interpolating patch? How do you assign an area to such faces? How do you define (discrete) curvature on such meshes? *Etc.*. Several techniques in geometry processing and architectural geometry seek to *planarize* such meshes before processing [Gly+04; CW07; Liu+06], whereas others define operators that do not require the geometry to be planar [Bun+20]. Planarization is especially tricky, since one must find vertex positions $f : V \to \mathbb{R}^3$ that *simultaneously* make all faces planar—a constraint that is often quite rigid, and can push the vertices far from their original positions [VB15]. The intrinsic point of view offers a potentially interesting alternative: for each face, find the planar polygon with the same side lengths, and whose angles are as close as possible (in some sense) to the original angles. These planar polygons endow the input surface with a new Euclidean cone metric that can be treated in exactly the same way as the intrinsic triangulations discussed so far. Moreover, unlike searching for compatible vertex positions in $\mathbb{R}^3$ (which typically entails global optimization), the geometry of each polygon can be constructed independently, in a straightforward fashion. However, a variety of challenges remain, such as defining and tracking correspondence between the extrinsic and intrinsic polygonal mesh.

Intrinsic tetrahedra. The same intrinsic formulation can be likewise applied to higher dimensional complexes, in particular tetrahedral meshes. In the most basic sense, one can certainly discard the vertex positions of a tetrahedral mesh and retrain only edge lengths, and this representation may already have benefits *e.g.* for anisotropic problems. However, more advanced operations such as producing Delaunay meshes will require deeper thought. There is no direct analogue of the basic Delaunay flipping algorithm (Theorem 1) in higher dimensions, even edge flips must be generalized to more complex *bistellar flips*. One appealing viewpoint is to generalize flipping operations as projections of a higher-order simplex [BEE02]. More general local *cavity operations* may also be a promising alternative [LM14]. Additionally, the notion of curvature on tetrahedral meshes differs from the surface case: input meshes generally have no curvature on their interior, but a general intrinsic metric might induce curvature concentrated along edges.

Appendix A

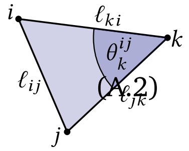
Evaluating Geometric Quantities

Here we express several basic geometric quantities in terms of the intrinsic edge lengths. See also Schindler and Chen [SC12] for useful perspectives on such quantities via barycentric coordinates.



Triangle Area. One can compute the area of triangle ijk via *Heron's formula*:

$$\begin{gathered} A_{ijk} = \sqrt{s(s-\ell_{ij})(s-\ell_{jk})(s-\ell_{ki})}, \\ \text{where } s := \tfrac{\ell_{ij}+\ell_{jk}+\ell_{ki}}{2}. \end{gathered} \tag{A.1}$$

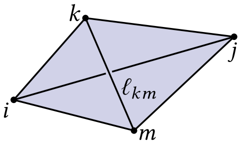


Corner Angles. One can compute the corner angles of triangle ijk via the law of cosines:

$$\theta_k^{ij} = \arccos\left(\frac{\ell_{jk}^2 + \ell_{ki}^2 - \ell_{ij}^2}{2\ell_{jk}\ell_{ki}}\right). \tag{A.2}$$

Cotan Weights. It's possible to compute the cotangent of a corner angle by taking the cotangent of the angle defined above. However, one can avoid inverse trigonometric functions by recalling that the area of a triangle is proportional to the sine of its corner angle. Thus,

$$\cot\theta_k^{ij} := \frac{\cos\theta_k^{ij}}{\sin\theta_k^{ij}} = \frac{\ell_{jk}^2 + \ell_{ki}^2 - \ell_{ij}^2}{4A_{ijk}}. \tag{A.3}$$



Opposite Diagonal. To flip an edge ij contained in a pair of triangles ijk, jim we need to compute the length of the opposite diagonal km. Recall that an edge flip can be performed only if the two triangles form a convex quadrilateral. In this case, the opposite diagonal length is given by the relationship

$$\ell_{km}^2 = \frac{1}{2}\left(\ell_{im}^2 + \ell_{jk}^2 + \ell_{jm}^2 + \ell_{ki}^2 - \ell_{ij}^2 + \frac{(\ell_{im}^2 - \ell_{jm}^2)(\ell_{jk}^2 - \ell_{ki}^2))}{\ell_{ij}^2}\right) + \frac{8A_{ijk}A_{jim}}{\ell_{ij}^2}, \tag{A.4}$$

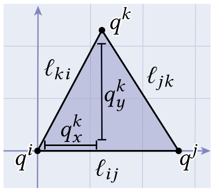
where the triangle areas A_{ijk} and A_{jim} are computed as above. Take care when implementing, that the edge lengths in this diagram are labelled relative to the diamond connectivity *before* the flip is performed. Also note that this expression gives the *square* of the diagonal length—for an edge flip one must then take the square root to recover the new length.

Though this expression is easy to write down and implement, there are many other possible numerical expressions for the same quantity, which may be more accurate or efficient—see especially the discussion in Sharp, Soliman, and Crane [SSC19a, Appendix A].

Distance in Barycentric Coordinates. Given two points in triangle ijk encoded as barycentric coordinates p and q, we can compute the distance between them as a function of the edge lengths, and their barycentric difference $u := p - q$:

$$|u| = \sqrt{-\ell_{ij}^2 u_i u_j - \ell_{jk}^2 u_j u_k - \ell_{ki}^2 u_k u_i}. \tag{A.5}$$

For a derivation, see Section 2.3.6, or Schindler and Chen [SC12, Section 3.2].



Planar Layout. The 2D layout from Section 2.3.7 can be computed as follows. Given a triangle ijk, we can place vertices i and j at positions

$$q^i := (0, 0) \quad q^j := (\ell_{ij}, 0). \tag{A.6}$$

We can compute the height of the triangle (or equivalently the y-coordinate of q^k) as

$$q_y^k = h = \frac{2A_{ijk}}{\ell_{ij}}, \tag{A.7}$$

and we can use the Pythagorean theorem to solve for the x position of q^k:

$$q_x^k = \pm\sqrt{\ell_{ki}^2 - h^2}, \tag{A.8}$$

taking the positive solution if $\theta_i^{jk} < 90°$ and the negative solution otherwise. Note that we can check this condition by checking the sign of $\cos\theta_i^{jk}$. Using the law of cosines, one can show that $\theta_i^{jk} < 90°$ if and only if $\ell_{jk}^2 < \ell_{jk}^2 + \ell_{ki}^2$.

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.

[BDG13] Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. “Constructing intrinsic Delaunay triangulations of submanifolds”. *arXiv preprint arXiv:1303.6493* (2013).

[BEE02] Marshall Bern, David Eppstein, and Jeff Erickson. “Flipping cubical meshes”. *Engineering with Computers* 18.3 (2002), pp. 173–187.

[BG10] Jean-Daniel Boissonnat and Arijit Ghosh. “Triangulating smooth submanifolds with light scaffolding”. *Mathematics in Computer Science* 4.4 (2010), pp. 431–461.

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

[BJW90] Clive F Baillie, Desmond A Johnston, and Roy D Williams. “Crumpling in dynamically triangulated random surfaces with extrinsic curvature”. *Nuclear Physics B* 335.2 (1990), pp. 469–501.

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

[Bor+13] Vincent Borrelli, Saıd Jabrane, Francis Lazarus, and Boris Thibert. “Isometric embeddings of the square flat torus in ambient space”. *Ensaios Matemáticos* 24 (2013), pp. 1–91.

[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.

[BPS15] Alexander I Bobenko, Ulrich Pinkall, and Boris A Springborn. “Discrete conformal maps and ideal hyperbolic polyhedra”. *Geometry & Topology* 19.4 (2015), pp. 2155–2215.

[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.

[Bun+20] Astrid Bunge, Philipp Herholz, Misha Kazhdan, and Mario Botsch. “Polygon laplacian made simple”. *Computer Graphics Forum*. Vol. 39. 2. Wiley Online Library. 2020, pp. 303–313.

[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).

[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.

[CHK13] Marcel Campen, Martin Heistermann, and Leif Kobbelt. “Practical anisotropic geodesy”. *Computer Graphics Forum*. Vol. 32. 5. Wiley Online Library. 2013, pp. 63–71.

[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.

[Cra+20] Keenan Crane, Marco Livesu, Enrico Puppo, and Yipeng Qin. “A Survey of Algorithms for Geodesic Paths and Distances”. *arXiv preprint arXiv:2007.10430* (2020).

[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.

[CW07] Barbara Cutler and Emily Whiting. “Constrained Planar Remeshing for Architecture”. *Graphics Interface*. Montreal, Canada: ACM, 2007, pp. 11–18.

[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).

[DP03] Olivier Devillers and Sylvain Pion. “Efficient Exact Geometric Predicates for Delauny Triangulations.” *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*. 2003.

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

[Dwy87] Rex A Dwyer. “A faster divide-and-conquer algorithm for constructing Delaunay triangulations”. *Algorithmica* 2.1 (1987), pp. 137–151.

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

[DZM07] Ramsay Dyer, Hao Zhang, and Torsten Möller. “Delaunay Mesh Construction”. *Proceedings of the 5th Eurographics Symposium on Geometry Processing*. 2007.

[Eis85] Peter R. Eiseman. “Solution adaptivity using a triangular mesh”. *The Free-Lagrange Method*. Ed. by Martin J. Fritts, W. Patrick Crowley, and Harold Trease. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 205–235. ISBN: 978-3-540-39697-0.

[EN13] Jeff Erickson and Amir Nayyeri. “Tracing compressed curves in triangulated surfaces”. *Discrete & Computational Geometry* 49.4 (2013), pp. 823–863.

[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).

[Gau25] Carl Friedrich Gauss. *General Investigations of Curved Surfaces*. 1825.

[Gli05] David Glickenstein. “Geometric triangulations and discrete Laplacians on manifolds”. *arXiv preprint math/0508188* (2005).

[Gly+04] James Glymph, Dennis Shelden, Cristiano Ceccato, Judith Mussel, and Hans Schober. “A parametric strategy for free-form glass structures using quadrilateral planar facets”. *Automation in construction* 13.2 (2004), pp. 187–202.

[Goe+14] Fernando de Goes, Pooran Memari, Patrick Mullen, and Mathieu Desbrun. “Weighted triangulations for geometry processing”. *ACM Transactions on Graphics (TOG)* 33.3 (2014), pp. 1–13.

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

[GS85] Leonidas Guibas and Jorge Stolfi. “Primitives for the manipulation of general subdivisions and the computation of Voronoi”. *ACM transactions on graphics (TOG)* 4.2 (1985), pp. 74–123.

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

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

[Gu+10] Xianfeng David Gu, Ren Guo, Feng Luo, and Wei Zeng. “Discrete Laplace-Beltrami operator determines discrete Riemannian metric”. *arXiv preprint arXiv:1010.4070* (2010).

[Gu+18a] Xianfeng David Gu, Ren Guo, Feng Luo, Jian Sun, and Tianqi Wu. “A Discrete Uniformization Theorem for Polyhedral Surfaces II”. *Journal of Differential Geometry* 109.3 (2018).

[Gu+18b] Xianfeng David Gu, Feng Luo, Jian Sun, and Tianqi Wu. “A Discrete Uniformization Theorem for Polyhedral Surfaces”. *Journal of Differential Geometry* 109.2 (2018).

[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).

[Jia+15] Tengfei Jiang, Xianzhong Fang, Jin Huang, Hujun Bao, Yiying Tong, and Mathieu Desbrun. “Frame field generation through metric customization”. *ACM Transactions on Graphics (TOG)* 34.4 (2015), pp. 1–11.

[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).

[KPD09] Daniel Kane, Gregory N Price, and Erik D Demaine. “A pseudopolynomial algorithm for Alexandrov’s Theorem”. *Workshop on Algorithms and Data Structures*. Springer. 2009, pp. 435–446.

[KS21] Marc Khoury and Jonathan Richard Shewchuk. “Restricted Constrained Delaunay Triangulations”. *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021.

[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).

[Law72] 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.

[Lee+98] Aaron WF Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. “MAPS: Multiresolution adaptive parameterization of surfaces”. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, pp. 95–104.

[Lei99] Gregory Leibon. *Random Delaunay triangulations, the Thurston-Andreev theorem, and metric uniformization*. University of California, San Diego, 1999.

[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+06] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. "Geometric modeling with conical meshes and developable surfaces". *ACM SIGGRAPH 2006 Papers*. 2006, pp. 681–689.

[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).

[Liu+20] Hsueh-Ti Derek Liu, Vladimir G Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. "Neural subdivision". *ACM Transactions on Graphics (TOG)* 39.4 (2020), pp. 124–1.

[Liu+21] Hsueh-Ti Derek Liu, Jiayi Eris Zhang, Mirela Ben-Chen, and Alec Jacobson. "Surface Multigrid via Intrinsic Prolongation". *arXiv preprint arXiv:2104.13755* (2021).

[LM14] Adrien Loseille and Victorien Menier. "Serial and parallel mesh modification through a unique cavity-based primitive". *Proceedings of the 22nd International Meshing Roundtable*. Springer, 2014, pp. 541–558.

[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).

[Mor96] Patrick Joseph Moran. *Visualization and modeling with shape*. University of Illinois at Urbana-Champaign, 1996.

[Mos88] Lee Mosher. "Tiling the projective foliation space of a punctured surface". *Transactions of the American Mathematical Society* (1988), pp. 1–70.

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

[Mus97] Oleg R Musin. "Properties of the Delaunay triangulation". *Proceedings of the thirteenth annual symposium on Computational geometry*. 1997, pp. 424–426.

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

[NN+93] Seiya Negami, Atsuhiro Nakamoto, et al. "Diagonal transformations of graphs on closed surfaces" (1993).

[ORo98] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[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.

[Pac90] Udo Pachner. "Shellings of simplicial balls and pl manifolds with boundary". *Discrete mathematics* 81.1 (1990), pp. 37–47.

[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).

[PS06] Konrad Polthier and Markus Schmies. "Straightest Geodesics on Polyhedral Surfaces". *ACM SIGGRAPH 2006 Courses*. 2006.

[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).

[Rie54] Bernhard Riemann. *On the Hypotheses which lie at the Bases of Geometry*. 1854.

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

[Riv94a] Igor Rivin. "Euclidean Structures on Simplicial Surfaces and Hyperbolic Volume". *Annals of mathematics* 139.3 (1994).

[Riv94b] Igor Rivin. "Intrinsic geometry of convex ideal polyhedra in hyperbolic 3-space". *Lecture Notes in Pure and Applied Mathematics* (1994), pp. 275–275.

[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.

[SC12] Max Schindler and Evan Chen. "Barycentric Coordinates in Olympiad Geometry". *Olympiad Articles* (2012).

[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).

[SGC21] Nicholas Sharp, Mark Gillespie, and Keenan Crane. "Geometry Processing with Intrinsic Triangulations". *ACM SIGGRAPH 2021 Courses*. 2021.

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

[She02a] Jonathan Shewchuk. “What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint)”. *University of California at Berkeley* 73 (2002), p. 137.

[She02b] Jonathan Richard Shewchuk. “Delaunay refinement algorithms for triangular mesh generation”. *Computational geometry* 22.1-3 (2002), pp. 21–74.

[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).

[SSP08a] Boris Springborn, Peter Schröder, and Ulrich Pinkall. “Conformal Equivalence of Triangle Meshes”. *ACM Transactions on Graphics (TOG)* 27.3 (2008).

[SSP08b] Boris Springborn, Peter Schröder, and Ulrich Pinkall. “Conformal equivalence of triangle meshes”. *ACM SIGGRAPH 2008 papers*. 2008, pp. 1–11.

[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).

[Thu79] William P Thurston. *The geometry and topology of three-manifolds*. Princeton University Princeton, NJ, 1979.

[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).

[Tou80] Godfried T Toussaint. “The relative neighbourhood graph of a finite planar set”. *Pattern recognition* 12.4 (1980), pp. 261–268.

[Tro91] Marc Troyanov. “Prescribing curvature on compact surfaces with conical singularities”. *Transactions of the American Mathematical Society* 324.2 (1991), pp. 793–821.

[TY12] Dylan Thurston and Qiaochu Yuan. “Notes on Curves on Surfaces” (2012).

[VB15] Amir Vaxman and Mirela Ben-Chen. “Dupin Meshing: A Parameterization Approach to Planar Hex-Dominant Meshing”. *Technion IIT CS Technical Report* (2015).

[Wag36] Klaus Wagner. “Bemerkungen zum vierfarbenproblem”. *Jahresbericht der Deutschen Mathematiker-Vereinigung* 46 (1936), pp. 26–32.

[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).

[War+07] Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. "Discrete Laplace operators: no free lunch". *Symposium on Geometry processing*. Aire-la-Ville, Switzerland. 2007, pp. 33–37.

[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).

[Ye+19] Zipeng Ye, Ran Yi, Minjing Yu, Yong-Jin Liu, and Ying He. "Geodesic Centroidal Voronoi Tessellations: Theories, Algorithms and Applications". *arXiv preprint arXiv:1907.00523* (2019).

[Ye+20] Zipeng Ye, Ran Yi, Wenyong Gong, Ying He, and Yong-Jin Liu. "Dirichlet energy of Delaunay meshes and intrinsic Delaunay triangulations". *Computer-Aided Design* 126 (2020), p. 102851.

[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).