Evolving Intrinsic Triangulations

Mark Gillespie

CMU-CS-24-116

April 2024

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Keenan Crane, Chair
James McCann
Ioannis Gkioulekas
Boris Springborn (TU Berlin)

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

This research was sponsored by National Science Foundation awards IIS1943123, IIS2212290, and CCF1717320, by The David and Lucile Packard Foundation award 201868047, and by Deutsche Forschungsgemeinschaft TRR 109. The material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program. 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.

Last updated: April 22, 2024

Abstract

This thesis presents algorithms and data structures for performing robust computation on surfaces that evolve over time. Throughout scientific and geometric computing, surfaces are often modeled as *triangle meshes*. However, finding high-quality meshes remains a challenge because meshes play two distinct and often-conflicting roles: defining both the surface geometry and a space of functions on that surface.

One solution to this dilemma, which has proven quite powerful in recent years, is the use of *intrinsic triangulations* to decouple these two concerns. The key idea is that given a triangle mesh representing an input surface, one can find many alternative triangulations which encode the exact same intrinsic geometry but offer alternative function spaces to work in. This technique makes it easy to find high-quality intrinsic triangle meshes, sidestepping the tradeoffs of classical mesh construction. However, the fact that intrinsic triangulations exactly preserve the input geometry—one of the central benefits of the technique—also makes it challenging to apply to surfaces whose geometry changes over time.

In this thesis we relax the assumption of exact geometry preservation, allowing the intrinsic perspective to be applied to time-evolving surfaces. We take as examples the problems of mesh simplification and surface parameterization. In the case of mesh simplification, we provide a general-purpose data structure for intrinsic triangulations which share only the topological class of the input surface, but may feature different geometry. In the case of surface parameterization, we build more efficient data structures and algorithms for the special case where the geometry changes *conformally*, using a connection between discrete conformal maps and hyperbolic geometry. In both cases, we find that the intrinsic perspective leads to simple algorithms which are still robust and efficient on a variety of examples.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> *"You see," Mrs. Whatsit said, "if a very small insect were to move from the section of skirt in Mrs. Who's right hand to that in her left, it would be quite a long walk for him if he had to walk straight across."*
>
> Madeleine L'Engle, *A Wrinkle in Time* [1962]

The distinction between intrinsic and extrinsic properties has played a central role throughout the history of differential geometry, dating back to pioneering work by Gauß [1825] and Riemann [1854] in the early 19th century. Intrinsic geometry describes the properties of a surface which depend on local measurements *along* the surface like lengths or angles, independent of the surface's embedding in space. A common metaphor—referenced above—is that intrinsic geometry takes the perspective of an ant walking along the surface. By contrast, extrinsic geometry encompasses the properties which do depend on the embedding of a surface. Importantly, one can consider surfaces which have only intrinsic geometry, without any embedding into ambient space. This intrinsic perspective is famously used in general relativity, where one considers the curved spacetime of our universe without requiring any larger space for the universe to "curve into".

More recently, the intrinsic perspective has become a useful tool in geometry processing, where it allows one to work with triangle meshes which are not embedded into $\mathbb{R}^3$—and even meshes which *cannot* be embedded into $\mathbb{R}^3$. This opens up a larger space of meshes to work in, providing meshes of much higher quality than is possible extrinsically, while still supporting a wide variety of geometry processing tasks. However, existing techniques apply only as a precomputation for static objects: they find alternative representations of fixed objects, but these representations immediately become invalid if the object deforms.

This thesis begins with a discussion of our *integer coordinates* data structure for efficiently encoding static intrinsic triangulations (Chapter 3), before moving on to explore two settings where intrinsic triangulations sit atop changing geometries:

1. Maintaining an intrinsic triangulation while coarsening a surface to perform intrinsic simplification (Chapter 5).

2. Using an intrinsic triangulation while parameterizing a surface via discrete conformal maps (Chapter 4).

Thesis Content

The work described in this thesis was published in several articles, from which much of the text and figures have been drawn. In particular, see:

Mark Gillespie, Nicholas Sharp, and Keenan Crane (2021a). "Integer Coordinates for Intrinsic Geometry Processing". *ACM Transactions on Graphics* 40.6, pp. 1–13. DOI: 10.1145/3478513.3480522.
(code: https://github.com/MarkGillespie/intrinsic-triangulations-demo)

Nicholas Sharp, Mark Gillespie, and Keenan Crane (2021). "Geometry Processing with Intrinsic Triangulations". *ACM SIGGRAPH 2021 Courses*. DOI: 10.1145/3450508.3464592.
(code: https://github.com/nmwsharp/intrinsic-triangulations-tutorial)

Mark Gillespie, Boris Springborn, and Keenan Crane (2021b). "Discrete Conformal Equivalence of Polyhedral Surfaces". *ACM Transactions on Graphics* 40.4, pp. 1–20. DOI: 10.1145/3592401.
(code: https://github.com/MarkGillespie/CEPS)

Hsueh-Ti Derek Liu, Mark Gillespie, Benjamin Chislett, Nicholas Sharp, Alec Jacobson, and Keenan Crane (2023). "Surface Simplification using Intrinsic Error Metrics". *ACM Transactions on Graphics* 42.4, pp. 1–17. DOI: 10.1145/3592403.
(code: https://github.com/HTDerekLiu/intrinsic-simplification/)

Chapter 2

Background & Related Work

> *The start of intrinsic geometry was made by Gauss' paper "Disquisitiones generales circa superficies curvas," which appeared in 1827. Since that time, intrinsic geometry has advanced so far that, at present, all of its major issues can be considered solved, at least those that deal with the geometry of small pieces of regular surfaces … Meanwhile, irregular surfaces merit no less consideration, as they often occur in real life and can be made from, say, a sheet of paper. For example, any polyhedron or cone, or the surface of a lens with sharp edges are not regular. It is no wonder then that there is a need to study irregular surfaces, too.*
>
> A. D. Alexandrov, *Intrinsic Geometry of Convex Surfaces* [1948]

Perhaps the first major result concerning the intrinsic geometry of *polyhedra*—rather than smooth surfaces—was Alexandrov's uniqueness theorem for embeddings of convex polyhedra in the 1940s [Alexandrov 1942]. A decade and a half later Regge [1961] took inspiration from Alexandrov's work and used intrinsic polyhedra in his study of numerical general relativity. It is fitting that general relativity, which motivated much of the development of smooth differential geometry, was also a key impetus behind the devlopment of discrete differential geometry.

From there, the study of polyhedral intrinsic geometry branched in several direction: Troyanov [1986] developed the smooth theory of polyhedra in his study of smooth conformal maps between polyhedral surfaces, while Rivin [1994a] defined intrinsic Delaunay triangulations of polyhedral surfaces and introduced the deep connections between Euclidean and ideal hyperbolic polyhedra. He also introduced the flip algorithm for computing intrinsic Delaunay triangulations. Several years later Indermitte et al. [2001] fixed a flaw in Rivin's proof of correctness of the flip algorithm, although they themselves left open the possibility of a topological obstruction which was only ruled out by Bobenko & Springborn [2007]. Glickenstein [2005, 2023] generalized the intrinsic Delaunay triangulation and edge flip algorithm, introducing increasingly broad classes of triangulations such as weighted, Thurston, and duality triangulations, sharing many of the important properties of intrinsic Delaunay triangulations. Bobenko & Izmestiev [2008] used weighted Delaunay triangulations to provide a constructive proof of Alexandrov's theorem on isometric embeddings of convex polytopes.

In parallel, intrinsic geometry has developed through the study of discrete conformal maps, starting with the work of Roček & Williams [1984] on discrete conformal field theories. Discrete

conformal maps were rediscovered in mathematics by Luo [2004] in his work on combinatorial Yamabe flow, starting a line of work which culiminated in the discrete uniformization theorem for polyhedral surfaces [Gu et al. 2018b; a; Springborn 2019].

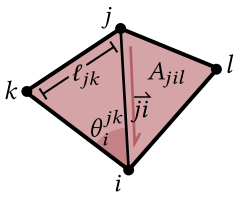
The computer graphics community was introduced to intrinsic geometry by the work of Kharevych et al. [2006] and Springborn et al. [2008] on discrete conformal parameterization, and the work of Fisher et al. [2007] exploring the benefits of intrinsic Delaunay triangulations. de Goes et al. [2014] went on to explore applications of weighted triangulations in geometry processing, including architectural design and mesh generation. More recently, Sharp et al. [2019] proposed a lightweight data structure for computing with intrinsic triangulations, alongside a suite of novel intrinsic retriangulation algorithms beyond the intrinsic Delaunay flips introduced by Rivin in the '90s. Gillespie et al. [2021a] introduced a more robust data structure for encoding intrinsic triangulations. Since then, intrinsic triangulation have proved useful in a variety of contexts, from spectral geometry processing [Fumero et al. 2020] to mesh deformation [Finnendahl et al. 2023]. A more exhaustive survey of the literature on intrinsic triangulations and their applications was provided by Sharp et al. [2021].

Notably, all past work has considered intrinsic triangulations which are *isometric* to the reference input surface—*i.e.* intrinsic triangulations which preserve the input geometry exactly. In this thesis, we introduce data strucures an algorithms for manipulating intrinsic triangulations of surfaces whose geometry changes over time. These data structures necessarily accomodate intrinsic triangulations whose geometry differs in various ways from the input geometry.

In the rest of this chapter, we review our notation and conventions (Section 2.1), before providing an introduction to the concepts from smooth (Section 2.2) and discrete (Section 2.3) differential geometry used in this thesis.

2.1 Notation & Conventions

Throughout, we consider a manifold triangle mesh M with vertex set V, edge set E and face set F. We denote vertices by indices $i \in \mathsf{V}$ and edges and faces by tuples $ij \in \mathsf{E}$, $ijk \in \mathsf{F}$ respectively. We also denote the oriented *halfedge* pointing from vertex i to vertex j by $\overrightarrow{ij} \in \mathsf{H}$, and the corner of face ijk at vertex i as ${}^{jk}_{i} \in \mathsf{C}$.



Quantities and functions. The value of a function $u : V \to \mathbb{R}$ at vertex i is written as u_i; similarly, values on edges are denoted u_{ij} and values on faces are denoted u_{ijk}. A value at the corner of face ijk incident on vertex i is denoted u_i^{jk}. For instance, the position of a vertex may be denoted p_i, the length of an edge ℓ_{ij}, the area of a face A_{ijk}, or the angle of a corner θ_i^{jk}. It will sometimes be helpful to think of a function $u : \mathsf{V} \to \mathbb{R}$ defined on vertices as a $|\mathsf{V}|$-dimensional vector—in this case, we will also write $u \in \mathbb{R}^{\mathsf{V}}$. In general for any set S, we use $\mathbb{R}^S$ to denote the space of functions $S \to \mathbb{R}$ (so, *e.g.*, we write a function defined on edges as $\ell \in \mathbb{R}^{\mathsf{E}}$ or a function defined on faces as $A \in \mathbb{R}^{\mathsf{F}}$).

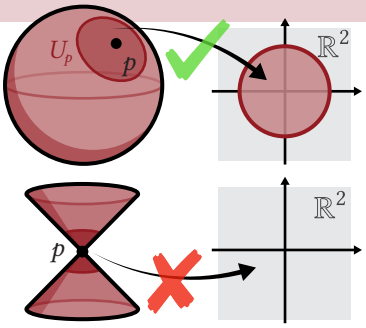
Δ-complexes. We allow meshes to feature multiple edges between the same pair of vertices. Formally, our meshes are general Δ-complexes (see Section 2.3.1 for more discussion of the

details). Consequently, our edge notation ij may not specify a unique edge from the mesh—it is merely used to indicate a particular edge that happense to go from vertex i to vertex j (where i may equal j). Similar caveats apply to the notation for faces.

Defining the *degree* of a vertex in a Δ-complex also requires some care, as the same edge may be incident on a vertex more than once. We 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.

Time-evolving meshes. When our mesh changes over times, we denote the original mesh by $M = (V, E, F)$ and the modified mesh by $\widetilde{M} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$. Quantities on the modified mesh are indicated in the same way, so *e.g.* the edge lengths on the modified mesh are denoted by $\tilde{\ell} : \widetilde{E} \to \mathbb{R}_{\geq 0}$.

2.2 Manifolds

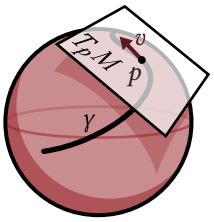


Informally speaking, a manifold is a space which "looks like $\mathbb{R}^n$ everywhere". More formally, a manifold M is a topological space where every point $p \in M$ is contained in some open set U_p which is in continuous bijection with an open set in $\mathbb{R}^n$. So a sphere is a manifold, since every point on the sphere is contained in a disk which can be mapped continuously to the unit disk in the plane. By contrast, a double-knapped cone is not a manifold, because no neighborhood of the tip can be mapped to the plane by a continuous bijection. Each mapping to the plane is called a *chart*, as it describes the "terrain" around a point p, and the collection of all of these charts is referred to as an *atlas*.

So far, our notion of a manifold is purely topological—since our charts are continuous maps, we have defined manifolds as spaces which look like $\mathbb{R}^n$ everywhere *as topological spaces*. To move from the world of topology to geometry, we have to equip our manifolds with additional structure. The two most important examples in this thesis will be *smooth structure* (Section 2.2.1)—allowing us to talk about differentiability of functions in addition to continuity—and *Riemannian structure* (Section 2.2.2)—allowing us to measure lengths and angles along the surface.

2.2.1 Smooth Structure

A smooth structure on a manifold M determines which functions $f : M \to \mathbb{R}$ are smooth (*i.e.* infinitely differentiable). Traditionally, this is expressed using a special atlas of charts which is declared to be smooth: then a function f is smooth on M if its expression in each chart is a smooth function on $\mathbb{R}^n$. As long as the charts satisfy some simple compatibility conditions on their overlaps, then this definition is independent of which particular charts we select from the atlas to check f's smoothness [Lee 2012, Chapter 1].



Tangent Spaces A smooth structure on M allows one to define *tangent spaces* associated to points on M. Conceptually, the tangent space T_pM represents the plane tangent to M at point p, giving a linear approximation of the domain around this point. Somewhat counterintuitively, these tangent spaces can be defined purely intrinsically, without any embedding to provide the position of M in $\mathbb{R}^n$. The key idea is that tangent vectors can be thought of as derivatives of curves lying on M. Since the smooth structure allows us to talk about derivatives of curves, this is enough to define tangent vectors, and hence tangent spaces. The *tangent bundle* TM is the collection of all tangent spaces:

$$TM := \bigsqcup_{p \in M} T_pM. \tag{2.1}$$

Any smooth mapping $f : M \to N$ between manifolds M and N has a linear approximation $df : TM \to TN$ which sends tangent vectors on M to tangent vectors on N. This map is often called the differential, or push-forward, since it pushes vectors from M to N.

It is important to note that these tangent spaces are defined only as abstract vector spaces, without any canonical choice of basis or inner product. So we can do arithmetic with tangent vectors—we can add them together or scale them up and down—but we cannot yet measure the lengths of vectors or angles between them. In the next section we will equip our surfaces with a Riemannian metric which will provide us with an inner product on our tangent spaces.

Similarly, since tangent vectors based at different points on the surface live in different tangent spaces, we cannot compare vectors which live at different points—*e.g.* we cannot ask whether two vectors point in the "same" direction, since they are elements of different vector spaces. Later on, we will see how a Riemannian metric also allows us to relate vectors in different tangent spaces through parallel transport.

Uniqueness of Smooth Structure Topological manifolds of dimension $n \leq 3$ have a unique smooth structure (up to diffeomorphism). Interestingly, the standard proof begins by showing that these manifolds can be triangulated piecewise-linearly [Moise 1952], and then proceeding to show that such triangulations can be smoothed to obtain a smooth structure [Hirsch & Mazur 1974]. Triangulations, which we use to encode polyhedral surfaces in Section 2.3.1 are also an essential tool in the continuous setting. On the other hand, higher-dimensional manifolds may have different smooth structures; for instance, Milnor [1956] famously constructed smooth structures on the 7-sphere inequivalent to the standard one.

2.2.2 Riemannian Structure

A Riemannian structure on M allows us to start doing *geometry*, measuring lengths and angles and so forth for curves running along M. Formally, this structure is usually encoded via a Riemannian metric g, which provides an inner product on each tangent space of M. That is to say, at each point p we have a symmetric, bilinear, positive-definite form $g_p : T_pM \times T_pM \to \mathbb{R}$. To emphasize that g_p is an inner product on T_pM, we will sometimes write $g_p(X, Y)$ as $\langle X, Y \rangle_g$ for vectors $X, Y \in T_pM$, and similarly, we will write $g_p(X, X)$ as $\|X\|_p^2$ for vectors $X \in T_pM$.

Isometries An isometry is a smooth mapping $f : M \to N$ between manifolds M and N which preserves the metric. So, for instance, if γ is a curve on M, then $f \circ \gamma$ is a curve of the same length on N. Similarly, if two curves γ_1 and γ_2 meet at an angle θ on M, then the curves $f \circ \gamma_1$ and $f \circ \gamma_1$ must meet at the same angle θ on N.

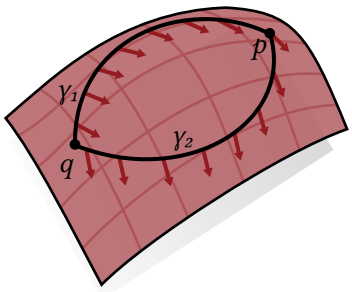
Geodesics To measure the length of a curve $\gamma : [0, T] \to M$, we add up the size of its velocity at all times from 0 to T. More formally, the length of γ can be expressed as the integral

$$L(\gamma) := \int_0^T \|\dot{\gamma}(t)\|_g \, dt, \tag{2.2}$$

where $\dot{\gamma} : [0, T] \to TM$ is the derivative of γ. Now that we can measure the lengths of curves along M, we can also define distances between points in M: the distance between points x and y on M is simply the length of the shortest path from x to y. If M is equal to $\mathbb{R}^n$ with the standard metric, then the shortest line between two points will be a straight line connecting them. On general Riemannian manifolds, then, we thing of shortest paths as generalizations of straight lines.

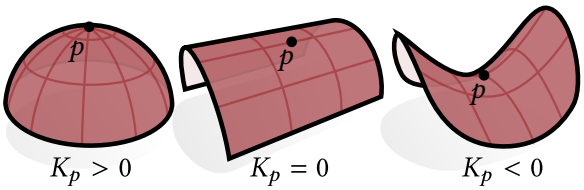
A *geodesic* is a (unit-speed) curve γ which is a locally shortest path, in the sense that for sufficiently close times s and t, γ follows the shortest path between $\gamma(s)$ and $\gamma(t)$. However, γ itself may not be the shortest path between its endpoints. A classic example of a geodesic which is not a shortest path is a curve wrapping around the equator of the sphere. This is locally shortest, since over any short time interval it follows a shortest path, but of course walking all of the way around the sphere is longer than not moving at all.

Parallel Transport If M is merely a smooth surface with no metric, then vectors in different tangent spaces live in totally separate worlds. It does not even make sense to ask questions like "do vectors $v \in T_pM$ and $w \in T_qM$ point in the same direction?" or "which vector $w \in T_qM$ is most similar to $v \in T_pM$?" But if M has a Riemannian metric, we can start to relate the different tangent spaces together. In particular, parallel transport allows us to pick some curve γ starting at a point p and ending at q, and allows us to take a vector $v \in T_pM$ and "transport" v along the curve γ to obtain a resulting vector in T_qM. Indeed, we can transport our starting vector to any point along the curve γ, yielding a collection of vectors which are as "parallel" as possible, according to our metric. However, the result of parallel transport depends in an essential way on the curve γ—as depicted in the inset, parallel transporting v along different paths γ_1 and γ_2 will generally result in different vectors in T_qM.



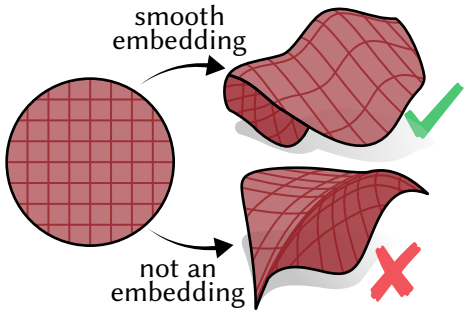
If γ is a geodesic, then parallel transport is quite simple. Starting from a vector $v \in T_pM$, we use the metric to measure its length $\|v\|_g$, as well as the angle $\theta \in [0, 2\pi)$ between v and the derivative vector γ' at p. The result of transporting v along γ to another point $q \in M$ is the vector $w \in T_qM$ with the same magnitude which makes the same angle with γ' at q. Parallel transport along general curves, or on manifolds of higher dimension, is more complicated—we will not use such general parallel transport in this thesis, but the curious reader can find an introduction in do Carmo [1992, Chapter 2].

Curvature At any point p of a Riemannian manifold M, the *Gaussian curvature* K_p measures how "non-flat" M is around point p. Intuitively, points where K_p is positive are "round" like the sphere, points where K_p is negative are "saddle-shaped", and points where K_p is exactly equal to zero are "intrinsically-flat"—they can be unfolded and laid out in the plane without stretching out the surface. Even though these descriptions sound extrinsic, as they refer to the shape of the surface in space, one can compute the Gaussian curvature purely intrinsically, using only the Riemannian metric on M.



2.2.3 Embeddings

So far, all of our treatment of manifold has been *intrinsic*—we have discussed measurements of lengths, angles, and curvatures, independent of any ambient space. If we want work extrinsically—to think about quantities like positions in space, or surface normals—we can equip our surface with an *embedding* $f : M \to \mathbb{R}^3$ defining where in space each point of our surface is situated. An embedding must satisfy two conditions to define meaningful geometry on our surface: (i) it must be smooth and bijective[1] and (ii) its derivative must have full rank everywhere, in define valid normal vectors along the surface. The inset depicts two mappings from the unit disk into $\mathbb{R}^3$. The top map is a smooth embedding, while the bottom is not, as there are no well-defined normal vectors for points on the "crease" lying along the center of the surface.



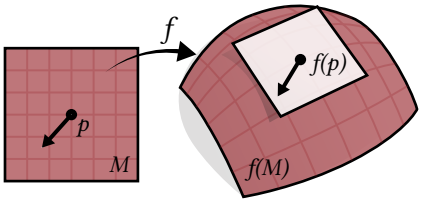


Figure 2.1: An embedding of a surface into $\mathbb{R}^3$ allows us to realize tangent vectors as vectors in $\mathbb{R}^3$ pointing tangent to the surface.

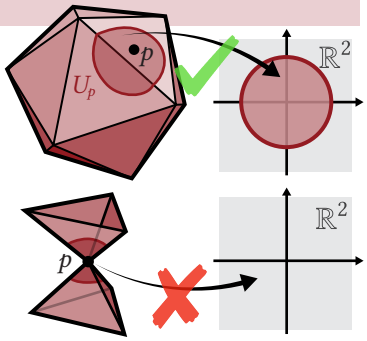
An embedding of a smooth surface M into $\mathbb{R}^3$ also gives us a notion of geometry on M. For instance, we can measure the length of any curve on M by viewing it as a space curve in $\mathbb{R}^3$ and measuring its length there. Formally, an embedding $f : M \to \mathbb{R}^3$ allows us to view any tangent vector $v \in T_pM$ as a vector pointing away from the position $f(p) \in \mathbb{R}^3$. So we can define a Riemannian metric on M by saying that the inner product of two tangent vectors is given by mapping them into $\mathbb{R}^3$ and taking the dot product of the resulting vectors in $\mathbb{R}^3$. The mapping of tangent vectors into $\mathbb{R}^3$ is given precisely by the differential df, so we can write the induced Riemannian metric on M as $g(X, Y) = \langle df(X), df(Y) \rangle_{\mathbb{R}^3}$.

Isometric Embeddings If M already has a Riemannian metric g, then we often want to find isometric embeddings $f : M \to \mathbb{R}^3$ where the inner product defined by the embedding coincides with the existing metric g. Not every surface has a smooth isometric embedding into $\mathbb{R}^3$—for

[1] technically, one must also require the inverse be smooth. Unlike the case of, say, linear maps (where any invertible linear map has a linear inverse), the inverse of a smooth bijective function is not necessarily smooth

instance, the *hyperbolic plane* has no such embedding (Appendix A). However, it is always possible to construct an isometric embedding of any orientable surface into $\mathbb{R}^n$ for a sufficiently large dimension n [Nash 1956]. Additionally, if we relax our smoothness assumption, we can find C^1 isometric embeddings of surfaces into $\mathbb{R}^3$ [Kuiper 1955; Nash 1954].

2.3 Polyhedral Surfaces



Informally speaking, a polyhedral surface is a collection of triangles which have been glued together to form a manifold. Just as in the smooth setting, a triangulated sphere is a manifold, whereas a double-knapped pyramid is not. And like in the smooth setting, we can consider different structures on a polyhedral surface: we can consider a triangulation, which yields topological information about the surface (Section 2.3.1), and we can consider a metric (Section 2.3.2), which yields geometric information about the surface.

2.3.1 Triangulations

In this thesis, we will represent the connectivity of a polyhedral surfaces using a triangulation. More precisely, we use a Δ-complex, which consists of a collection of disjoint triangles along with a prescribed gluing which attaches their vertices and edges together. Explicitly, this amounts to a collection of triangles $i_0 j_0 k_0, \ldots i_{|F|} j_{|F|} k_{|F|}$, alongside a list of vertex gluings $a \sim b$ and a list of edge gluings $(a, b) \sim (c, d)$, where a, b, c, d are vertices from the disjoint triangles. Figure 2.2 shows some examples; a more formal definition is provided by Hatcher [2002, Section 2.1]. Finally, throughout this thesis we consider only pure 2-complexes, *i.e.*, we require that every vertex and edge is contained in some triangle (and triangles are the cells of greatest dimension).

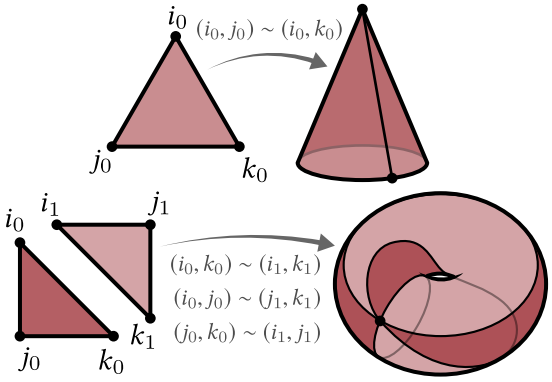


Figure 2.2: In a Δ-complex, the vertices of an edge or triangle may not to be distinct. One can build a cone by gluing together two edges of the same triangle (top), or a torus out of two triangles and just a single vertex (bottom).

Existence Radó [1925] showed that every surface can be triangulated, which is to say that every topological 2-manifold is homeomorphic to some Δ-complex [Moise 2013, Chapter 8]. Perhaps the more surprising fact is that this theorem is not true in higher dimensions: Kirby & Siebenmann [1969] showed that topological manifolds of dimension ≥ 6 are triangulable if and only if a certain cohomology class $\kappa(M) \in H^4(M; \mathbb{Z}/2\mathbb{Z})$ vanishes. On the other hand, Whitehead [1940] showed that every smooth (or even just C^1) manifold can be triangulated—meaning that the non-triangulable manifolds must be quite pathological. In any case, this thesis only considers surfaces, where working with triangulations is much more straightforward.

Equivalence Two triangulations of a manifold are said to be combinatorially equivalent if they have a *common subdivision*, *i.e.* if there is a finer triangulation which can express all faces of both triangulations as unions of its finer faces. In 1908, Steinitz [1908] and Tietze [1908] conjectured that any two triangulations of a topological manifold are combinatorially equivalent, which came to be known as the *Hauptvermutung* (main conjecture) of geometric topology. This conjecture holds true in dimensions two [Radó 1925] and three [Moise 1952], but again fails in higher dimensions [Kirby & Siebenmann 1969].

Although any two triangulations of a given surface are combinatorially equivalent, the choice of triangulation can have dramatic impacts in practice.

Data Structures Perhaps the most common mesh data structure is the *vertex-face adjacency list*, which simply stores the three three vertices associated with each triangle. This representation is simple and easy to use, as it requires only an $|F| \times 3$ matrix. However, a vertex-face adjacency list alone does not provide enough information to encode a general Δ-complex: it provides a vertex gluing map, but leaves the edge gluing map implicit. When working with extrinsic triangle meshes in $\mathbb{R}^3$ this information is sufficient to recover the triangulation, but when working intrinsically one must store more information.

Fortunately, many of the other standard mesh data structures can be used out of the box to represent general Δ-complexes. For instance one can use winged-edge or halfedge structures [Baumgart 1975; Weiler 1985; Kettner 1999]; Botsch et al. [2010] provide an accessible introduction to these data structures. Alternatively, Sharp & Crane [2020a] observe that one can simply augment the vertex-face adjacency list with an additional array storing the edge gluing map to fully encode a general Δ-complex.

Tangent Spaces Away from vertices, tangent vectors on a manifold triangulation are straightforward to reason about, especially in our setting of interest where the triangles are given flat Euclidean metrics (Section 2.3.2). But even at vertices, there are still well-defined tangent spaces. After all, a manifold 2-dimensional Δ-complex is in particular a topological surface, which has a unique smooth structure. In the next section, we will use a metric on the triangulation to construct a convenient parameterization for these tangent spaces.

2.3.2 Polyhedral Geometry

A polyhedral cone metric on a surface M with vertex set V is a smooth Riemannian metric on the punctured surface $M \setminus \mathsf{V}$ which is intrinsically flat everywhere. Such a metric can be encoded by fixing a triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ of M (with the same vertex set V) and picking set of positive edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$ satisfying the triangle inequality $\ell_{ij} + \ell_{jk} > \ell_{ki}$ at each triangle corner; conversely, any such set of lengths determines a valid intrinsic metric. Under this metric, each triangle is isometric to a standard Euclidean triangle with the presecribed edge lengths. One typically obtains initial edge lengths $\ell_{ij} = \|p_i - p_j\|$ from input vertex positions $p : V \to \mathbb{R}^3$, but in principle this could be any abstract metric (e.g., coming from a *cone flattening* [Bobenko & Springborn 2004]). As usual, this metric allows us to measure lengths and angles along the surface. For instance, triangle corner angles $\theta_i^{jk} \in (0, \pi)$ can be determined from the edge

lengths via the law of cosines. Sharp et al. [2021, Appendix A] provide a detailed explanation of how to compute many geometric quantities of interest from edge lengths.

Curvature Although a polyhedral cone metric is flat almost everywhere, it still has a meaningful notion of curvature: each interior vertex i has an associated discrete Gaussian curvature

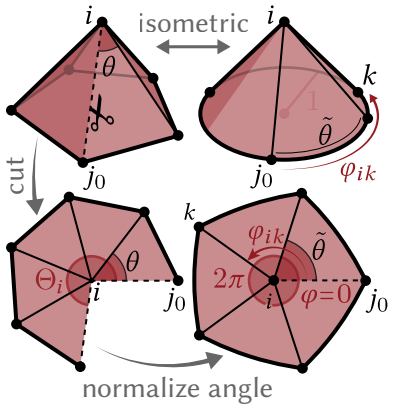
$$\Omega_i := 2\pi - \sum_{ijk \ni i} \theta_i^{jk}. \tag{2.3}$$

This angle defect measures the deviation of vertex i from being flat, and can be interpreted as the integral of Gaussian curvature over a small surface patch containing vertex i. Similarly, each boundary vertex has an associate discrete geodesic curvature

$$\kappa_i := \pi - \sum_{ijk \ni i} \theta_i^{jk}, \tag{2.4}$$

measuring the deviation of the boundary from a straight line around i.

Vertex Tangent Spaces Even though the smooth polyhedral metric is not technically defined at a vertex i, it still provides us with a convenient parameterization of the tangent space T_iM. Any sufficiently small neighborhood of vertex i is isometric a cone of total angle Θ_i. Following Knöppel et al. [2013, Section 6], we express the direction of any tangent vector $v \in T_iM$ as a normalized angle $\varphi := 2\pi\theta/\Theta \in [0, 2\pi)$, where θ is the angle of v relative to an arbitrary but fixed reference edge ij_0, and Θ is the total angle sum at vertex i. The vector itself is then encoded as a complex number $re^{\mathbf{i}\varphi} \in \mathbb{C}$, where $\mathbf{i}$ is the imaginary unit and r is the vector's magnitude. Note that although we have used the metric to define a particular coordinate system on T_iM, the tangent space itself is well-defined independent of our choice of metric.



Discrete Parallel Transport The tangent spaces at adjacent vertices i and j are *a priori* just a pair of abstract vector spaces which are entirely unrelated to each other. However, once we have equipped our surface with a polyhedral metric we can use parallel transport to map vectors between the two tangent spaces. Concretely, we let the angular coordinate $\varphi_{ij} \in [0, 2\pi)$ encode the outgoing direction of an oriented edge ij from vertex i; we use $e_{ij} \in T_iM$ to denote the vector with direction φ_{ij} and magnitude ℓ_{ij}. The corresponding direction at vertex j is given by $\varphi_{ji} + \pi$. Hence, we can parallel transport vectors from T_iM to T_jM following edge ij by applying a rotation $R_{ij} := e^{\mathbf{i}((\varphi_{ji}+\pi)-\varphi_{ij})}$ (encoded as a unit complex number). See Sharp et al. [2019, §3.3 & §5.2] for further discussion.

Barycentric Coordinates Barycentric coordinates provide a convenient coordinate system for performing calculations on points within a triangle, or defining functions on a triangle. Given a

triangle ijk with vertex positions $p_i, p_j, p_k \in \mathbb{R}^2$, the three barycentric coordinates $b_i, b_j, b_k \in \mathbb{R}$ describe the point $x = b_i p_i + b_j p_j + b_k p_k$. This point lies within triangle ijk precisely when the b_i describe a convex combination, *i.e.* when:

1. $b_i \geq 0$ for all i, and
2. $b_i + b_j + b_k = 1$.

The barycentric coordinates for a point $x = \sum_i b_i p_i$ are invariant under linear maps. That is to say, if we have some linear function $f : \mathbb{R}^2 \to \mathbb{R}^2$, then the barycentric coordinates for x in terms of p_i, p_j and p_k are precisely the same as the barycentric coordinates for $f(x)$ in terms of $f(p_i), f(p_j)$ and $f(p_k)$. This equivalence follows directly from the linearity of f, just using the fact that $f\left(\sum_i b_i p_i\right) = \sum_i b_i f(p_i)$, but has the important consequence that we can use barycentric coordinates to refer to points in a triangle *independent of its vertex positions*. For instance, the coordinates $(1, 0, 0)$ always refer to vertex i, and the coordinates $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ always refer to the center of mass, no matter where the vertices of the triangle are located. Consequently, we will often use barycentric coordinates to refer to points on an intrinsic triangulation, even when the triangles have no canonically-defined vertex positions.

Homogeneous barycentric coordinates. When working with conformal maps, it will often be convenient to use barycentric coordinates which are not normalized to have unit sum. Given three vertex positions $p_i, p_j, p_k \in \mathbb{R}^2$ we say that a triple of coordinates $b_i, b_j, b_k \in \mathbb{R}$ (which does not necessarily sum to 1) corresponds to the point

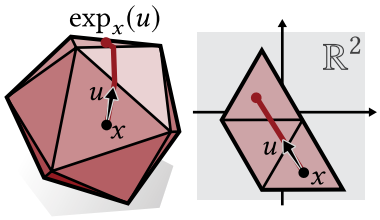
$$x = \frac{1}{b_i + b_j + b_k}\left(b_i p_i + b_j p_j + b_k p_k\right). \tag{2.5}$$

These non-normalized barycentric coordinates can be viewed as homogeneous coordinates on $\mathbb{R}^2$, so we will also refer to them as homogeneous barycentric coordinates.

Barycentric coordinates on edges. In addition to using barycentric coordinates on triangles, we will often use barycentric coordinates to refer to points lying along edges of our mesh. The coordinates (b_i, b_j) correspond to the point $x = b_i p_i + b_j p_j$ along edge ij. If we also require that the coordinates sum to 1, then one of these values is redundant. We use the convention that the scalar barycentric coordinate $t_{ij} \in \mathbb{R}$ corresponds to the point $x = (1 - t_{ij})p_i + t_{ij} p_j$, so that $t = 0$ corresponds to the starting vertex i and $t = 1$ corresponds to the ending vertex j.

Of course, if we wish to use homogeneous barycentric coordinates along edge ij, then there is no redundancy and we need to store the values of both b_i and b_j explicitly.

Exponential and Logarithmic Map The exponential map $\exp_x(u)$ of a tangent vector u at point x computes the point p reached by starting at point x and walking straight (*i.e.*, along a geodesic) with initial direction u for a distance $\|u\|$ (inset, *left*). Concretely, this can be evaluated by laying out the relevant sequence of triangles in the plane and drawing a straight line (inset, *right*). Note that for any oriented edge ij we have $\exp_i(e_{ij}) = j$. Conversely, the logarithmic



map $\log_x(p)$ of a given point $p \in M$ taken at point x gives the smallest tangent vector u at x such that $\exp_x(u) = p$. Hence, for any point p and veretx i, we have that $\exp_x(\log_x(p)) = p$. However, it is not necessarily the case that for any tangent vector v we have $\log_x(\exp_x(v)) = v$, since there may be a shorter path leading to the same destination. In particular, $\log_i(j)$ may not always yield the edge vector e_{ij}.

2.3.3 Retriangulation

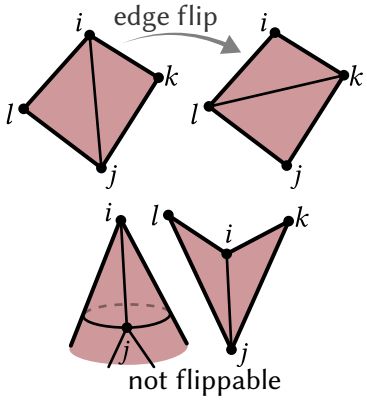
We can represent the intrinsic geometry of a polyhedral surface by recording the edge lengths for *any* triangulation of its vertices. However, not all triangulations serve equally well when we start trying to do computations on the surface. For instance, when considering scalar functions defined on the surface, we often work with piecewise-linear functions which are linear on each triangle. So even if two triangulations may encode the exact same geometry, they might still provide us with different function spaces—and as we will see in Section 3.6, some function spaces work much better than others. The most basic way of improving the quality of a triangulation is to perform intrinsic edge flips, which modify the triangulation while preserving the vertex set.

Intrinsic Edge Flip Consider an edge ij contained in triangles ijk, jil. An edge flip replaces ij with the opposite diagonal kl. An edge ij is *flippable* if and only if

(i) $\deg i, \deg j \geq 2$ and

(ii) triangles ijk, jil form a convex quadrilateral,

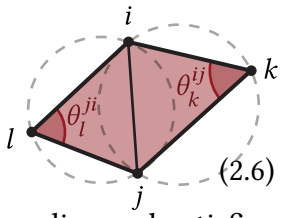
i.e., if both $\theta_i^{jk} + \theta_i^{lj}$ and $\theta_j^{ki} + \theta_j^{il}$ are less than π [Sharp & Crane 2020c, §3.1.3]. Note that these conditions are considerably easier to check than in the extrinsic case [Liu et al. 2020, Appendix C].



One can write down explicit formulas to compute the new length ℓ_{lk}, as well as the angular coordinates $\varphi_{\vec{lk}}$ and $\varphi_{\vec{kl}}$ if desired (see *e.g.* Section 3.4.2 and Appendix A of Sharp et al. [2021]).

Intrinsic Delaunay Triangulations A planar triangulation is Delaunay if there are no vertices inside any triangle circumcircle. Equivalently, we can ask that every interior edge ij satisfy the *local Delaunay condition*

$$\theta_k^{ij} + \theta_l^{ji} \leq \pi. \tag{2.6}$$



Note that if both triangles are inscribed in a common circle, then either diagonal satisfies Equation (2.6). This characterization generalizes to Euclidean polyhedra, since the edge lengths ℓ are sufficient to determine the angles θ. Such *intrinsic Delaunay triangulations* extend many useful properties of 2D Delaunay triangulations to surface meshes—[Sharp et al. 2021, §4.1.1] gives a detailed list. Intrinsic delaunay triangulations can be found via a simple greedy algorithm: flip any non-Delaunay edge (*i.e.* any edge violating Equation (2.6)) until none remain [Bobenko & Springborn 2007]. This algorithm terminates after finitely many flips [Indermitte et al. 2001; Bobenko & Springborn 2007], and in practice takes about $|\mathsf{E}|$ flips on real-world meshes [Sharp et al. 2019, Figure 10].

2.3.4 Polyhedral Embeddings

We can describe the *extrinsic geometry* of a polyhedral surface M by picking a triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ and equipping it with a set of vertex positions $p : \mathsf{V} \to \mathbb{R}^3$. As in the smooth setting, this embedding into $\mathbb{R}^3$ immediately defines the intrinsic geometry of our polyhedron as well—we can easily read off the edge lengths $\ell_{ij} := \|p_i - p_j\|_{\mathbb{R}^3}$. However, the reverse direction is far harder.

Alexandrov [1942] showed that any *convex* intrinsic polyhedron M (*i.e.* any polyhedral cone metric whose angle defects are all nonnegative) has a unique isometric embedding into $\mathbb{R}^3$ as a convex polyhedron. Bobenko & Izmestiev [2008] provide a constructive proof by leveraging a surprising connection to *weighted Delaunay triangulations*. One challenging aspect of the problem is the choice of triangulation—although the theorem guarantees that *some* triangulation of our polyhedron M can be embedded into $\mathbb{R}^3$, not every triangulation can be isometrically embedded. Consequently, in order to construct an isometric embedding one must also identify which triangulation of M may be embedded. However, if the polyhedral surface is convex, then one can always find an embeddable triangulation simply by performing edge flips—adding additional vertices is never necessary.

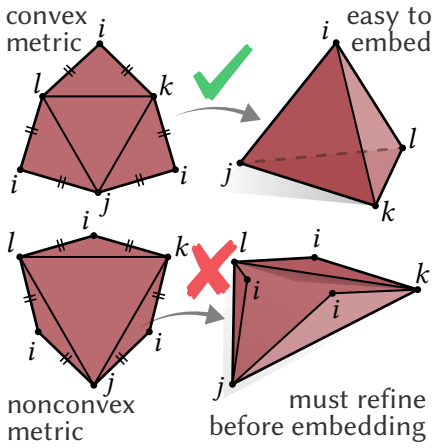


Figure 2.3: *Top*: Any *convex* polyhedral surface can be isometrically embedded into $\mathbb{R}^3$, possibly after performing intrinsic edge flips. *Bottom*: A nonconvex polyhedron (with negative angle defect at i) cannot be isometrically embedded without refining the triangulation.

In the general, nonconvex case isometric embeddings are still guaranteed to exist, but our triangulation may need to be refined. Burago & Zalgaller [1960, 1995] showed that after subdividing the triangulation finitely many times, one can always construct a piecewise-linear isometric embedding of any polyhedral surface M into $\mathbb{R}^3$. However, their construction produces highly corrugated surfaces which can feature wrinkles of arbitrarily high frequency—constructing smoother isometric embeddings of triangle meshes is still an active area of research (see, *e.g.*, [Chern et al. 2018] or [**Sassen:2024:RS**]).

Chapter 3

Integer Coordinates for Intrinsic Triangulations

God made the integers, all else is the work of man.
Leopold Kronecker [1886]

Before diving in to time-evolving intrinsic triangulations, we start with a simpler problem: how should you encode an ordinary intrinsic triangulation? If you only care about the intrinsic geometry itself, the answer is easy: the geometry is determined by the edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$. But often, we care not only about the geometry of the intrinsic triangulation, but also about its relationship to some input mesh. For instance, we often start with an extrinsic triangle mesh and compute an alternative triangulation of the same surface by performing intrinsic edge flips. In this case we might want to know how to map points between the two triangulations: given some point p lying in face ijk of the input triangulation, where does p lie on the intrinsic triangulation? Or we might want to map functions between the triangulations: given a piecewise-linear function f computed on the intrinsic triangulation, how can we plot this function over the input triangles? These questions concern not just the intrinsic triangulation, but its *correspondence* with the input mesh. In this chapter we describe a data structure for encoding the correspondence between two triangulations. While we specialize to the traditional case of two triangulations sharing the same geometry, ideas developed here will play important roles in our treatment of time-evolving geometries in the following chapters.

Problem Statement

Explicitly, suppose we have two manifold triangulations $\mathsf{T}_1 = (\mathsf{V}_1, \mathsf{E}_1, \mathsf{F}_1)$ and $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$, each equipped with a set of edge lengths $\ell_i : \mathsf{E}_i \to \mathbb{R}_{>0}$ defining an intrinsic metric on each surface. We additionally assume that the two surfaces are *isometric*—*i.e.* that there is a bijective mapping between the two surfaces which preserves distances. Our goal in this chapter is to develop a data structure which can efficiently encode this mapping, and can be efficiently updated as the triangulations are modified (*e.g.* by flipping edges or inserting vertices).

We begin by considering the special case where both triangulations share the same vertex set. This case already allows us to compute intrinsic Delaunay triangulations (Section 2.3.3), which have a wide variety of applications and play a key role in Chapter 4. In Section 3.5.1 we describe how the data structure can be adapted to support the insertion of new vertices into T_2, opening the doors to more advanced retriangulations schemes like intrinsic Delaunay refinement (Section 3.6). The ideas discussed in this chapter were published in Gillespie et al. [2021b] and Gillespie et al. [2021a]—note that the two works used slightly different conventions[1]; this chapter follows the latter, which supports a wider variety of mesh operations.

3.1 Correspondence Data

Our data structure consists of two pieces types data. We maintain

(1) *normal coordinates* $n : \mathsf{E}_2 \to \mathbb{Z}$, which count how many times T_1 crosses each edge of T_2 (Section 3.1.1),

(2) *roundabouts* $r : \mathsf{H}_2 \to \mathbb{Z}$, which give the circular ordering of halfedges from both T_1 and T_2 around each vertex (Section 3.1.2),

which amounts to $3|\mathsf{E}_2|$ integer values, a small fraction of the cost already required to store triangulations T_1 and T_2. The normal coordinates can be used to trace geodesic curves from each vertex i to all vertices j adjacent to i in T_1, yielding a collection of curves lying along T_2. However, they provide only the unordered set of curves, but do not identify which curve corresponds to which logical edge of T_1. Determining which curve corresponds to which edge can be surprisingly difficult since our meshes may have multiple edges between the same pair of vertices—we use the roundabouts to identify traced curves along T_2 with logical edges of T_1.

3.1.1 Normal Coordinates

Normal coordinates represent a curve sitting atop a triangulated surface by recording the number of times that the curve crosses each edge[2] (Figure 3.1). They were originally developed one dimension higher, to study surfaces embedded in 3-manifolds [Kneser 1929; Haken 1961; Hass & Trnkova 2020], but have spread throughout topology—*e.g.*, providing a key tool for studying the "mapping class group" of a surface [Farb & Margalit 2011; Bell 2015; 2013; Schaefer et al. 2008]. Normal coordinates have also found use as an efficient curve encoding for computational geometry [Erickson & Nayyeri 2013], since storing the number of intersections requires exponentially less space than recording each intersection individually.

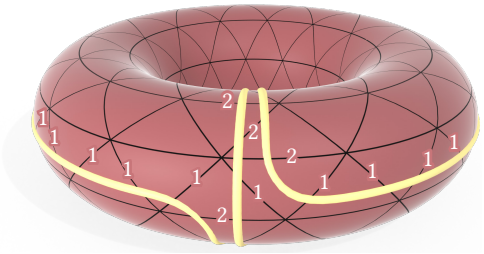


Figure 3.1: Traditionally, normal coordinates encode a curve on a triangulated surface by counting how many times the curve crosses each edge.

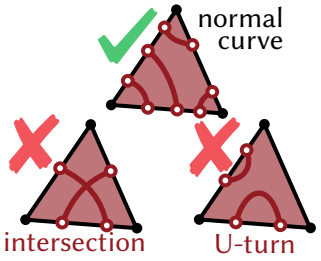
[1] In particular, Gillespie et al. [2021b] set $n_{ij} = 0$ for edges ij shared by both triangulations, whereas Gillespie et al. [2021a] set $n_{ij} = -1$ for edges shared by both triangulations and reserve $n_{ij} = 0$ to mark edges of E_2 which are not present in T_1 but also intersect no edges from T_1. This situation is impossible if T_1 and T_2 share the same vertex set, but is important to handle correctly if one wants to insert new vertices into T_2.

[2] Not to be confused with *geodesic normal coordinates* from Riemannian geometry, which are entirely unrelated.

There are just two *normality conditions* that the curve must satisfy in each face in order to be encoded via normal coordinates:

(1) it cannot intersect itself, and

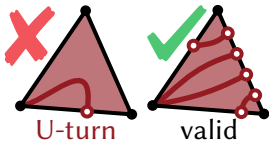
(2) it cannot make a "U-turn"—*i.e.* it cannot enter a face and then exit through the same edge.



If these two conditions are satisfied then the normal coordinates are sufficient to determine the curve, modulo homotopies which do not pass through vertices. Equivalently, normal coordinates determine the sequence of triangles that the curve passes through. And in addition to representing a single curve, normal coordinates can simultaneously encode several curves sitting on the same surface. No matter how many curves are present, one still stores a single integer per edge of the triangulation counting how many times *any* curve crosses that edge.

Our use of normal coordinates deviates from the standard treatment in several. First, rather than working in the topological setting of curves on smooth surfaces, we consider geodesic curves on Riemannian manifolds with Euclidean metrics. Using geodesics allows us to recover the exact path of the curve along the surface without having to work modulo homotopy.

Second, rather than working with closed curves, we assume that our normal coordinates encode the edges of a second triangulation of the surface. In particular, each curve is topologically equivalent to a line segment connecting two vertices. Such curves must obey the two ordinary normality conditions, and we additionally require that curves which enter a triangle through a vertex must exit via the opposite edge (see inset). These assumptions allow us to introduce a new edge flip formula, given in Section 3.1.1.



Finally, we need to decide which triangulation to store normal coordinates on. In the setting of intrinsic triangulations, where we think of T_2 as sitting atop T_1, it may feel more natural to store normal coordinates on the edges of T_1 to encode the paths taken by the edges of T_2. However, the situation is symmetric and we can also store normal coordinates on T_2 to encode the paths taken by the edges of T_1. And indeed, this second option allows us to implement intrinsic edge flips more efficiently: if we wish to flip an edge ij of T_2, but store normal coordinates on edges of T_1, then we must update the normal coordinates on every edge of T_1 which ij crosses (Figure 3.2, *left*). Since ij could cross arbitrarily many edges of T_1, this update could be quite

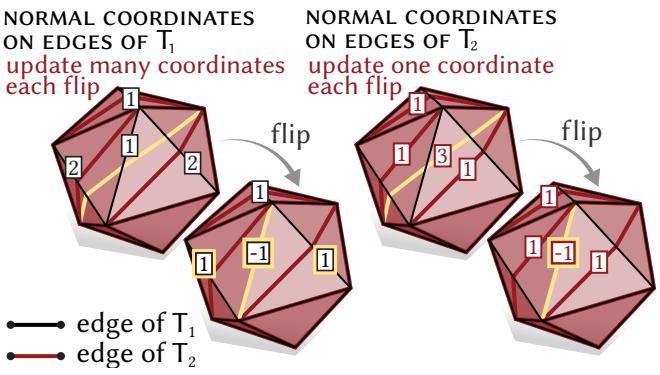


Figure 3.2: We could represent the correspondence between triangulations T_1 and T_2 using normal coordinates on the edges of T_1 *(left)*, or on edges of T_2 *(right)*. The latter is easier to update following an edge flip: if we flip the highlighted edge, we need to update the three highlighted normal coordinates in the former case, but we only need to update the normal coordinate on the flipped edge in the latter case.

expensive. However, if we store normal coordinates on the edges of T_2, then when flipping ij we only have to update the normal coordinate on ij itself, so our update will always run in constant time (Figure 3.2, *right*). In addition, storing normal coordinates on edges of T_2 is essential when the adding new vertices into triangulation T_2 (Section 3.5.1).

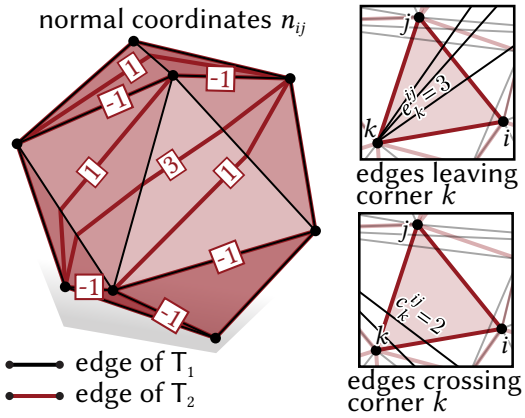


Figure 3.3: The normal coordinates n_{ij} count the number of times each edge $ij \in \mathsf{E}_2$ crosses any edge of the other triangulation T_1. These counts can be used to determine other quantities, *e.g.* how many edges of T_1 cross or leave a corner of a triangle from T_2.

Formal Definition We store normal coordinates on the edges of T_2 which count intersections with edges of T_1. In particular, for each edge ij of T_2, we store the number of times $n_{ij} \in \mathbb{Z}$ that any edge of T_1 crosses ij transversally (Figure 3.3, *left*); if ij is shared between T_2 and T_1, we assign it a special value of $n_{ij} = -1$. The value $n_{ij}^+ := \max(n_{ij}, 0)$ hence gives the number of transversal crossings; the value $n_{ij}^- := -\min(n_{ij}, 0)$ is 1 on shared edges and 0 otherwise[3].

From these numbers we can determine how many edges in T_1 emanate from corner $\overset{ij}{k}$ in T_2 (excluding those along edges of T_2) :

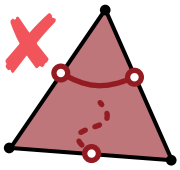
$$e_k^{ij} := \max(0, n_{ij}^+ - n_{jk}^+ - n_{ki}^+). \tag{3.1}$$

Likewise, the number of edges crossing corner $\overset{ij}{k}$ is

$$c_k^{ij} := \tfrac{1}{2}\left(\max\left(0, n_{jk}^+ + n_{ki}^+ - n_{ij}^+\right) - e_i^{jk} - e_j^{ki}\right). \tag{3.2}$$

See Figure 3.3, *right* for examples. For curves that do not touch vertices, the corner coordinates c are essentially dual to the normal coordinates n—see Erickson & Nayyeri [2013, Section 2.3].

Validity Not every assignment of integers to edges of T_2 form a valid set of normal coordinates: for instance, it is impossible for a closed curve to intersect each edge in a given face exactly once. The normal coordinates used in our data structure are always valid by construction, so an explicit treatment of the validity conditions for normal coordinates is not necessary to use them to encode intrinsic triangulations, but it may nonetheless be illuminating to consider which assignments of integers to edges serve as valid normal coordinates.

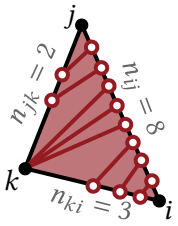


In the traditional setting, when the normal coordinates encode a collection of closed curves, there are two validity conditions on the normal coordinates on the edges of for each triangle:

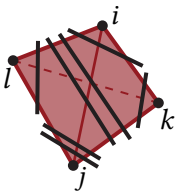
1. their sum must be even, and
2. they must obey the triangle inequality.

(see *e.g.* Thurston & Yuan [2012, Lecture 2]). The first condition must hold, since every curve which enters the triangle via one crossing must then exit through another, leaving an even number of crossings on the triangle's boundary. The second condition must hold since a normal curve which enters the triangle through one edge must exit through a different edge.

[3]Note that we follow the convention of Gillespie et al. [2021a], rather than Gillespie et al. [2021b] who set $n_{ij} = 0$ for parallel edges.



In our setting, where we use normal coordinates to encode edges of a geodesic triangulation, the validity conditions become slightly more complicated. Of course, if the values n_{ij}, n_{jk}, n_{ki} satisfy the two original conditions then they are valid. But we now have other valid normal coordinates as well. For instance, the inset depicts a triangle where the normal coordinates have odd sum, and fail to obey the triangle inequality, but still correspond to a valid set of curves passing through the triangle. The validity conditions for our normal coordinates may be stated as follows: *if* the values $n^+_{ij}, n^+_{jk}, n^+_{ki}$ obey the triangle inequality, then they must have even sum. If they violate the triangle inequality, then any values are valid.



Normal Coordinate Edge Flip Consider two triangles ijk, jil from T_2. In the simple case where no edge from T_1 terminates in a corner of either triangle (see inset), there is an edge flip update that resembles the Ptolemy relation for inscribed quadrilaterals [Mosher 1988; Thurston & Yuan 2012, Equation 1]:

$$n_{kl} = \max(n_{ki} + n_{lj}, n_{jk} + n_{li}) - n_{ij}. \tag{3.3}$$

In the general case, we must derive a more complicated formula:

$$n_{kl} = c_l^{jk} + c_k^{ij} + \tfrac{1}{2}\left|c_j^{il} - c_j^{ki}\right| + \tfrac{1}{2}\left|c_i^{lj} - c_i^{jk}\right| - \tfrac{1}{2}e_l^{ji} - \tfrac{1}{2}e_k^{ij} + e_i^{lj} + e_i^{jk} + e_j^{il} + e_j^{ki} + n_{ij}^{-}. \tag{3.4}$$

To understand Equation (3.4) of Section 3.1.1, consider first the case that lk is not an edge of T_1. Then the edges of T_1 intersect the interior of the quadrilateral in segments of the following types (Figure 3.4):

1. crossing corner l of jil or crossing corner k of ijk
2. crossing corners i of jil and j of ijk, or crossing corners j of jil and i of ijk
3. emanating in corner i or j of jil or ijk
4. the edge ij
5. crossing both corners i of ijk and jil or both corners j of ijk and jil
6. emanating in corner l of jil or emanating in corner k of ijk

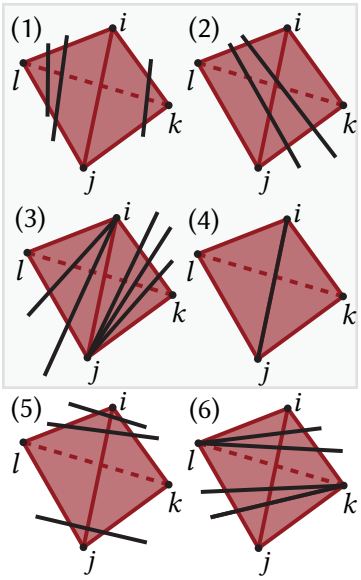


Figure 3.4: Edges of quadrilateral $ikjl$ come in 6 types. The first four each intersect edge lk, contributing to Equation (3.4), while the last two do not.

Segments of types 1–4 are counted by

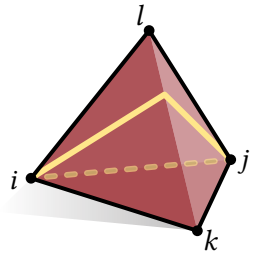
1. $c_l^{ij} + c_k^{ij}$
2. $\tfrac{1}{2}\left|c_j^{il} - c_j^{ki}\right| + \tfrac{1}{2}\left|c_i^{lj} - c_i^{jk}\right| - \tfrac{1}{2}e_l^{ji} - \tfrac{1}{2}e_k^{ij}$
3. $e_i^{lj} + e_i^{jk} + e_j^{il} + e_j^{ki}$
4. n_{ij}^{-}

and each contributes 1 to n_{jk}, while segments of types 5–6 contribute 0. To see the counting formulas for cases 2 and 4, note that $\frac{1}{2}\left|c_j^{il} - c_j^{ki}\right| + \frac{1}{2}\left|c_i^{lj} - c_i^{jk}\right|$ counts #{type 2} + $\frac{1}{2}$ #{type 6}, and that $n_{ij}^- = 1$ if and only if ij is also an edge of T_1.

Finally, if lk is an edge of T_1, then term 2 above is -1 and terms 1, 3, and 4 are zero (since lk cannot intersect other edges of T_1). So Equation (3.4) is satisfied with both sides equal to -1.

3.1.2 Roundabouts

Although normal coordinates completely describe a triangulation sitting on top of T_2, they do not tell us how the edges of this triangulation correspond to the edges of T_1: as noted in Section 2.1, two endpoints may not uniquely identify an edge. For instance, the two highlighted edges on the tetrahedron drawn in the inset both connect vertex i to vertex j, so if we obtain a path from i to j from our normal coordinates, it is hard to tell *a priori* which edge the path describes.



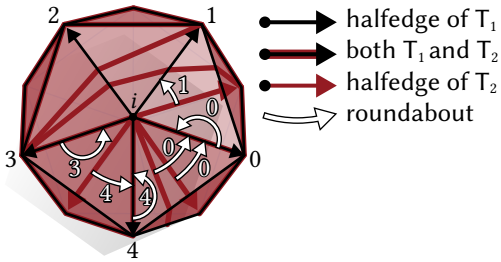


Figure 3.5: For each halfedge of T_2, the roundabout gives the next halfedge of T_1.

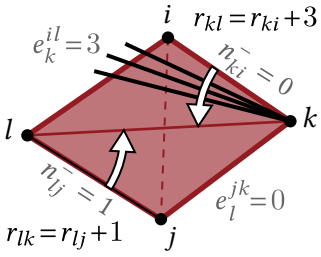
We therefore augment our normal coordinates with what we call *roundabouts*, in analogy with roundabouts or traffic circles found on roadways. At each vertex $i \in \mathsf{V}$, these roundabouts describe how the outgoing halfedges of the two triangulations are interleaved.

Conceptually, for each halfedge $\overrightarrow{ij} \in \mathsf{H}_2$, the roundabout tells us the first halfedge from T_1 counterclockwise from $\overrightarrow{ij}$ around vertex i. In practice, we encode the halfedge from T_1 as in index $r_{\overrightarrow{ij}} \in \mathbb{Z}_{\geq 0}$ local to vertex i (Figure 3.5), which allows us to update the roundabout values in constant time when flipping an edge of T_2 (Section 3.1.2). These indices start at zero, and enumerate the halfedges of T_1 which emanate from i in counter-clockwise order, starting at some arbitrary but fixed halfedge. Note that if a halfedge from T_2 coincides with a halfedge from T_1, the roundabout points to this halfedge, as indicated by self-arrows.

Roundabout Edge Flip To update our roundabouts after flipping an edge ij with opposite vertices k, l, we first update the normal coordinates as described in Section 3.1.1. We then set

$$\begin{aligned} r_{\overrightarrow{kl}} &= \operatorname{mod}(r_{\overrightarrow{ki}} + e_k^{il} + n_{ki}^-, \deg_1(k)), \\ r_{\overrightarrow{lk}} &= \operatorname{mod}(r_{\overrightarrow{lj}} + e_l^{jk} + n_{lj}^-, \deg_1(l)), \end{aligned} \tag{3.5}$$

where $\deg_1(i)$ is the degree of vertex i in the triangulation T_1. In other words, to find the first outgoing halfedge of T_1 following $\overrightarrow{kl} \in \mathsf{H}_2$, we start at $\overrightarrow{ki}$ and add the number of edges e_k^{il} of T_1 that emanate from corner k of triangle kil. Also, if $\overrightarrow{ki}$ is coincident with a halfedge from T_1, we add 1 to advance past this halfedge. The mod operation accounts for wraparound. See inset for an example. This update resembles a combinatorial version of the signpost update from Sharp et al. [2019, p. 3.2.1]: integer indices $r_{\overrightarrow{ij}}$ play the role of real-valued directions; the integer counts e_i^{jk} play the role of real-valued angles.



3.2 Tracing Edges

Using normal coordinates and roundabouts, we can pick any edge in T_1 and trace out the sequence of triangles in T_2 that it passes through (Section 3.2.1). To get the curve geometry, we lay out this triangle strip in the plane and draw a straight line between endpoints (Section 3.2.2). The final curve is encoded by 1D barycentric coordinates $s, t \in [0, 1]$ on each intersected edge.

3.2.1 Topological Tracing

To identify the sequence of edges in T_2 crossed by some edge in T_1, we start at one crossing and repeatedly identify the next edge crossed until the edge of T_1 terminates at a vertex. We can determine the next edge crossed purely from the stored normal coordinates, by considering the three cases illustrated in Figure 3.6.

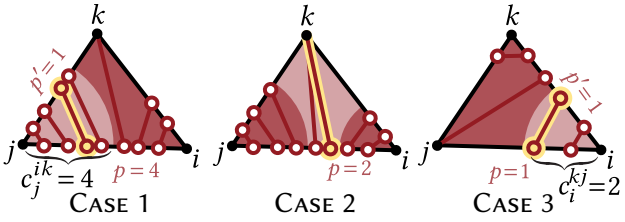


Figure 3.6: A curve entering triangle jil along edge ij can proceed in 3 ways: the left-most c_j^{ik} crossings go left (*left*), the rightmost c_i^{kj} crossings go right (*right*), and the rest terminate at vertex k (*center*).

Now we just need to find the first crossing, which we do using the roundabouts. Suppose we want to trace edge il of T_1, starting from vertex i. Since T_1 and T_2 share the same vertex set, we know that the curve also starts at vertex i in T_2. And for every corner $\overset{jk}{i}$ of T_2 incident on i, we can use the roundabouts r_{ij} and r_{ik} to determine which edges of T_1 start in $\overset{jk}{i}$. Once we find the corner containing il, we use the normal coordinates and roundabouts to work out the index of that first crossing along jk, from which we can trace out the rest of the curve.

Algorithm 1 GetFirstCrossing(T_2, n, r, il)

Input: A triangulation $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$, equipped with normal coordinates $n : \mathsf{E}_2 \to \mathbb{Z}$ and roundabouts $r : \mathsf{H}_2 \to \mathbb{Z}$ encoding a second triangulation T_1 of the same vertex set, along with an edge $il \in \mathsf{E}_1$ from the second triangulation.

Output: The first intersection between il and an edge of T_2, encoded as a pair $(\overrightarrow{kl}, p)$ where $\overrightarrow{kl} \in \mathsf{H}_2$ and p is the index of the crossing along halfedge $\overrightarrow{kl}$. If il coincides with an edge of T_2, returns $(il, -1)$ instead.

for $ijk \in \mathsf{F}_2$ incident on i **do**
  $iH \leftarrow$ LocalIndex$(\overrightarrow{il})$
  $e_i^{jk} \leftarrow \max(0, n^+_{jk} - n^+_{ki} - n^+_{ij})$ ▷*Equation (3.1)*
  ▷ *#curves emanating from $\overset{jk}{i}$, including along ij or ki*
  width $\leftarrow e_i^{jk} + n^-_{ij} + n^-_{ki}$
  if $r_{ij} \le iH < r_{ij} +$ width **then**
    ▷ *If iH lies in this range, it emanates from $\overset{jk}{i}$.*
    if $iH = r_{ij} +$ width $- n^-_{ki}$ **then** ▷*runs along ik*
      return $(\overrightarrow{ik}, -1)$
    else if $iH < r_{ij} + n^-_{ij}$ **then** ▷*runs along ik*
      return $(\overrightarrow{ij}, -1)$
    else ▷*crosses jk*
      return $(\overrightarrow{jk}, iH - r_{ij} - n^-_{ij})$

Algorithm 2 TraceEdge(T_2, n, r, lm)

Input: A triangulation $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$, equipped with normal coordinates $n : \mathsf{E}_2 \to \mathbb{Z}$ and roundabouts $r : \mathsf{H}_2 \to \mathbb{Z}$ encoding a second triangulation T_1 of the same vertex set, along with an edge $lm \in \mathsf{E}_1$ from the second triangulation.

Output: The path of lm as a sequence of crossings $(l, \zeta_1, \ldots, \zeta_n, m)$, where $\zeta_i = (\overrightarrow{ij}, p)$ is a crossings encoded as a halfedge $\overrightarrow{ij} \in \mathsf{H}_2$ and an index p for the crossing along the halfedge.

(currHalfedge, p) $\leftarrow$ GetFirstCrossing(T_2, n, r, lm)
if $p < 0$ **then** ▷*Shared halfedge, exit early*
  return $[l, m]$
$\gamma \leftarrow [l]$ ▷*Start path at vertex l*
while True **do** ▷*Walk until γ terminates at a vertex*
  $\overrightarrow{ij} \leftarrow$ currHalfedge
  $k \leftarrow$ OppVertex(Twin(currHalfedge))
  if $p < c_i^{kj}$ **then** ▷*turn right (Figure 3.6, right)*
    currHalfedge $\leftarrow \overrightarrow{ik}$ ▷*Move to $\overrightarrow{ik}$*
    $p \leftarrow p$
    Append(γ, (currHalfedge, p))
  else if $p \ge n_{ij} - c_j^{ik}$ **then** ▷*turn left (Figure 3.6, left)*
    currHalfedge $\leftarrow \overrightarrow{kj}$ ▷*Move to $\overrightarrow{kj}$*
    $p \leftarrow n_{kj} + p - n_{ij}$
    Append(γ, (currHalfedge, p))
  else ▷*terminate at k (Figure 3.6, center)*
    return (γ, k)

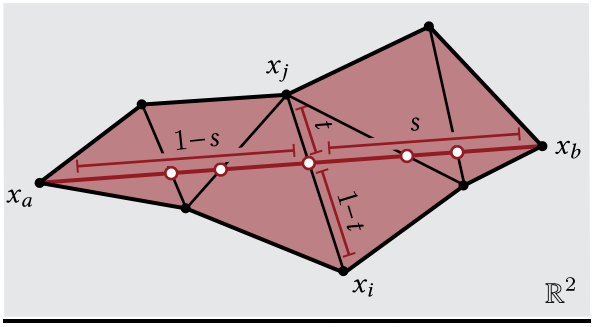
Note that the tracing procedure gives us each edge from T_1 as a sequence of edge crossings on T_2. To express the edges from T_2 as sequences of T_1 edge crossings, we allocate an array of size n_{ij} for each edge $ij \in \mathsf{E}_2$. Each time a traced edge $ab \in \mathsf{T}_1$ crosses ij, we store a reference to ab in entry p of the array (using roundabouts to get the edge index).

3.2.2 Recovering Geodesics

To get the geometry of each traced edge $ab \in \mathsf{E}_2$, we use the crossing sequences computed in Section 3.2 and the edge lengths ℓ to incrementally lay out a triangle strip in the plane. We then intersect each interior edge ij of this strip with the line from a to b—by construction, this line will be contained entirely inside the strip. In particular, if $x_i \in \mathbb{R}^2$ are the vertices of a Euclidean triangle strip, we can solve the equation

$$(1-s)x_a + sx_b = (1-t)x_i + tx_j \tag{3.6}$$

for the barycentric coordinates $s, t \in [0,1]$ of the intersection point. Algorithm 3 provides pseudocode for a basic version of this procedure, which recovers the path taken by a geodesic γ lying on T_2, represented as a sequence of points on T_2 encoded in barycentric coordinates[4]. In Section 4.2.1, we detail an analogous procedure for recovering geodesic paths in the hyperbolic plane which is used to compute discrete conformal parameterizations.



Algorithm 3 RecoverGeodesic$(\mathsf{T}_2, \ell, \gamma)$

Input: A triangulation $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$ with edge lengths $\ell : \mathsf{E}_2 \to \mathbb{R}_{>0}$, along with a path $\gamma = (l, \zeta_1, \ldots, \zeta_n, m)$ from vertex $l \in \mathsf{V}_2$ to $m \in \mathsf{V}_2$. Each $\zeta = (\overrightarrow{ij}, p)$ is a crossing encoded as a halfedge $ij \in \mathsf{H}_2$ and the index p of the crossing along the halfedge.

Output: The trajectory of γ as a sequence of points $(a, z_1, \ldots, z_n, m)$ along T_2. Intermediate crossings are encoded as pairs $z = (\overrightarrow{ij}, t)$ where $ij \in \mathsf{H}_2$ and t is the barycentric coordinate of point z along halfedge ij.

▷ *Compute positions in $\mathbb{R}^2$ for triangle strip containing γ*
$\mu \leftarrow$ LayOutTriangleStrip(γ)
trajectory $\leftarrow [l]$
for $\zeta = (\overrightarrow{ij}, p) \in \gamma$ **do**
  ▷ *Intersection of ab and ij in the plane (see inset above)*
  $s, t \leftarrow$ IntersectionBarycentric$(\mu_a, \mu_b, \mu_i, \mu_j)$
  Append(trajectory, $z = (\overrightarrow{ij}, t)$)
Append(trajectory, m)
return trajectory

3.3 Common Subdivision

The *common subdivision*[5] S of T_1 and T_2 is the polygon mesh obtained by "slicing up" the underlying surface along the edges of both T_1 and T_2. The vertices of S are hence a superset of V_1 and V_2, and every edge or face of T_1 and T_2 can be expressed as union of edges or faces of S respectively. Moreover, when T_1 is an extrinsic triangulation, the faces of S are always planar and convex. They are planar since they are subsets of the faces of T_1, which are planar. And they are convex because each face is obtained by cutting a triangle along a collection of straight lines, which always yields convex polygons (Figure 3.7). And most importantly, any piecewise-linear function on T_1 or T_2 can be represented exactly as a piecewise-linear function on S.

[4] If more detailed correspondence information is required, one can use the algorithm described in Gillespie et al. [2021a, Section 3.1], which computes the barycentric coordinates of each crossing point on T_1 as well as on T_2.

[5] Also known as the *common subdivision*, or even the *supermesh* (in FEM literature, *e.g.* Farrell et al. [2009, Section 2]).

The common subdivision serves as an essential "bridge" between an intrinsic triangulation and the original extrinsic domain: it provides the minimal piecewise linear basis on which both intrinsic data at vertices and extrinsic vertex positions can simultaneously be interpolated. S can then be used to pull back functions from the abstract intrinsic setting to an ordinary mesh sitting in space.

Note however that even if T_1 and T_2 have nice elements, S is not in general a high-quality mesh, and may not itself be suitable for, *e.g.*, solving PDEs. Rather, it plays a complementary role in the geometry processing pipeline, enabling (for instance) transfer of data between triangulations [Gillespie et al. 2021a, Section 4.3], or visualization of data downstream via standard rendering tools.

Tracing out the edges allows us to construct the common subdivision S of T_1 and T_2. To determine the connectivity of S we slice up each triangle $ijk \in \mathsf{F}^B$ independently. First we extract the connectivity of S, using only the normal coordinates n_{ij}. Then we recover the intersection geometry, allowing us to interpolate data stored at the vertices of T_1 or T_2 to S—most commonly, vertex positions on T_1 along with any solution data on T_2. Note that one can construct pathological cases in which there are quadratically many intersections between T_1 and T_2 (or worse—consider a triangulation with many *Dehn twists*, as pictured in [Sharp et al. 2019, Figure 4]). However, we do not observe such extreme behavior in practice.

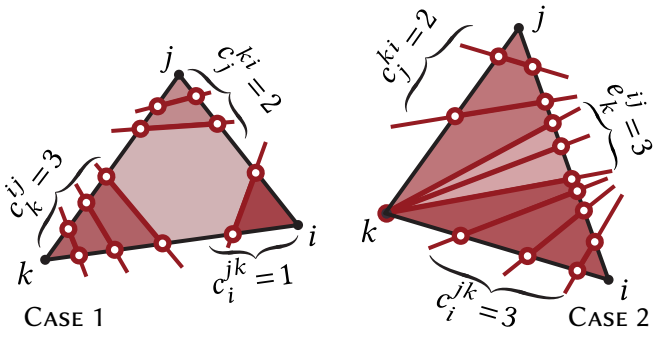


Figure 3.7: We find the connectivity of common subdivision within each triangle using its normal coordinates.

Connectivity We subdivide T_2 independently in each face ijk (Figure 3.7). There two cases to consider. In case 1, when no curves emanate from any corner, we simply connect the first c_i^{jk} crossings along edge ij to the first c_i^{jk} crossings along ik (in order), and likewise for corners j and k. In case 2 curves emanate from some corner; without loss of generality, let this corner be k so that the number of emanating curves is $e_k^{ij} > 0$. We walk from i to j, connecting the first c_i^{jk} crossings to those along ik, the next e_i^{jk} crossings to vertex k, and the remaining c_j^{ki} crossings to those along edge kj. Note that curves running along edges ($n_{ij} < 0$) require no special treatment.

Intersection Geometry Next, we associate each vertex i of the common subdivision with a point in T_1 and a point in T_2, encoded in barycentric coordinates. Using these values, one can linearly interpolate data from T_1 or T_2 to the vertices of S. Again, there are just two cases: each vertex i in S is either a vertex of T_2 or the intersection of an edge of T_1 with an edge of T_2. In the first case, the position on both triangulations is known. In the second case, we can compute the desired barycentric coordinates using the tracing procedures described in Section 3.2.

3.4 Flipping to a Given Triangulation

We primarily focus on modifying our triangulations by flipping edges. At first, one might worry that this local operation could be too restrictive—if you start with one triangulation of a surface, are there other triangulations (with the same vertex set) that you cannot reach merely by flipping edges? Fortunately, this concern turns out to be false: any pair of triangulations are connected by a finite sequence of edge flips. There are many proofs in the literature, but the most relevant to us is a constructive proof by Mosher [1988], who presents a simple algorithm for finding the sequence of edge flips to move from one triangulation to another. Unfortunately, Mosher's algorithm is embedded in a complicated analysis of its correctness and is moreover framed the language of hyperbolic geometry:

> *Our proof has the advantage that it implicitly gives an algorithm for constructing a sequence of elementary moves [*i.e. *edge flips] connecting [triangulations] δ and δ', when δ' is given in terms of certain intersection numbers with the arcs of δ. We shall not explicitly describe this algorithm; we leave that to the interested reader.*
>
> Mosher [1988, pages 37–38]

In this section, we give a more detailed description of the algorithm for flipping from one triangulation to another. In practice we generally modify triangulations by flipping to Delaunay (Section 2.3.3) or applying adaptive retriangulation schemes (Section 3.6), rather than trying to flip to a particular set of normal coordinates, but Mosher's algorithm is nonetheless important from a theoretical perspective and provides an interesting application of normal coordinates.

Suppose we have a triangulation T_2 equipped with normal coordinates $n : \mathsf{E}_2 \to \mathbb{Z}$ encoding another triangulation T_1 sitting atop it. Our goal is to flip a sequence of edges on T_2 to transform it into T_1. The idea underlying the algorithm is quite simple: we identify any edge ij of T_1 (the target triangulation) which is not already contained in T_2, and we flip the first edge of T_2 intersecting this edge. Then we check if any more edges of T_2 intersect ij. If they do, we continue flipping the first edge of T_2 intersecting ij until none remain. At this point, edge ij is shared by both triangulations. We repeat for some other edge of T_1 until all edges are shared, and both triangulations are the same.

Algorithm 4 FlipTo(T_2, n)

Input: A triangulation $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$, equipped with normal coordinates $n : \mathsf{E}_2 \to \mathbb{Z}$ encoding a second triangulation T_1 of the same vertex set.
Output: Perform edge flips on T_2 to transform it into T_1, *i.e.* all normal coordinates equal -1.

for $m \in \mathsf{V}_2$ **do** ▷ *Iterate over edges of T_1 incident on m*
  for crossing $(\overrightarrow{ij}, p)$ emanating from m **do**
    curveEnded $\leftarrow$ False
    ▷ *Flip each edge of T_2 which this edge crosses*
    while not end **do**
      $k \leftarrow$ OppVertex(Twin($\overrightarrow{ij}$))
      ▷ *Before getting next crossing, record which edge we mean to flip*
      edgeToFlip $\leftarrow ij$
      ▷ *Get next crossing on curve (*à la *Figure 3.6)*
      if $p < c_i^{kj}$ **then**
        $(\overrightarrow{ij}, p) \leftarrow (\overrightarrow{ik}, p)$
      else if $p \geq n_{ij} - c_j^{ik}$ **then**
        $(\overrightarrow{ij}, p) \leftarrow (\overrightarrow{kj}, n_{kj} + p - n_{ij})$
      else
        curveEnded $\leftarrow$ True ▷ *curve ends at k*
      $\mathsf{T}_2, n \leftarrow$ Flip(T_2, n, edgeToFlip)
return T_2

The one difficulty lies in identifying which edges of T_2 are crossed by some edge of T_1—but this is essentially the "tracing" problem discussed in Section 3.2.1. Starting at any vertex of T_2, we can use our normal coordinates to identify how many edges of T_1 emanate from this vertex, and to locate their first crossings with other edges *à la* Equation (3.1). And once we have the first crossing along an edge, we can find all subsequent crossings *à la* Algorithm 2. We just have to flip each edge of T_2 as we cross it. Algorithm 4 provides pseudocode for this procedure.

An important caveat is that Mosher's algorithm is formulated in the combinatorial setting[6], where one can flip any edge as long as it is not incident on a degree-1 vertex. In particular, the edge flips performed in Algorithm 4 are often impossible if T_2 is a Euclidean polyhedron, since its edges also have to satisfy a convexity condition in order to be flipped. Analyzing the behavior of this algorithm in the Euclidean setting, and determining whether a version can be used to flip between any two Euclidean triangulations would make for interesting future work.

Termination. Analyzing Algorithm 4 is tricky. In order to prove that it terminates, Mosher defines a particular quantity $\iota(\mathsf{E}_2, ij)$ as the total number of intersections between some edge $ij \in \mathsf{E}_1$ and all edges of E_2 *except* for the first edge intersected by ij, and shows that $\iota(\mathsf{E}_2, ij)$ decreases with each flip made as we try to incorporate ij into triangulation T_2. Note that $\iota(\mathsf{E}_2, ij)$ is at most the total number if intersections between ij and E_2, so we can bound the number of flips required by this total number of intersections. However, each flip used to incorporate ij into E_2 could create many more intersections between E_2 and other edges of T_1, so this analysis does not directly yield a useful quantitative bound on the number of flips required. In the end, much like the case of flipping to Delaunay, the flip algorithm is known to terminates in a finite amount of time, but not even guaranteed to have polynomial complexity (although runtimes are quite reasonable in practice—see Figure 3.8).

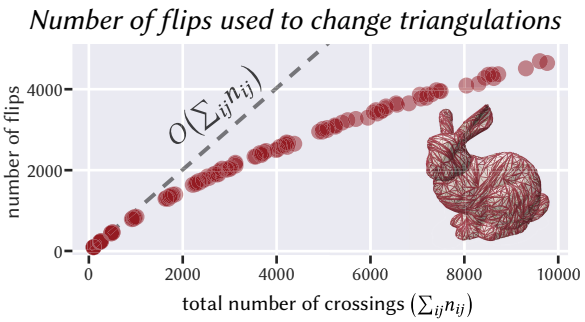


Figure 3.8: Theoretically, we only know that Mosher's algorithm terminates in finite time. But in practice, its cost seems to scale sublinearly with the total number of intersection between the two triangulations, *i.e.* with $\sum_{ij \in \mathsf{E}_2} n_{ij}$. Here we plot the number of flips used by Algorithm 4 to flip from a mesh of the bunny to a randomly-chosen triangulation of the same surface.

3.5 Modifying the Vertex Set

We now extend the integer coordinates data structure to allow the addition of new vertices into T_2. The main idea remains the same—we still store normal coordinates and roundabouts on the edges of T_2—but we also explicitly track where the new vertices lie on T_1. Letting $\mathsf{V}^\star := \mathsf{V}_2 \setminus \mathsf{V}_1$ denote the set of inserted vertices, our complete correspondence data structure consists of

(1) *normal coordinates* $n : \mathsf{E}_2 \to \mathbb{Z}$, counting how many times T_1 crosses each edge of T_2.

(2) *roundabouts* $r : \mathsf{H}_2 \to \mathbb{Z}$, giving the ordering of halfedges of T_1 and T_2 about each vertex[7].

(3) *locations* $q : \mathsf{V}^\star \to \mathsf{T}_1$, recording where on T_1 each new vertex of T_2 is located.

Each location q_m is encoded by storing the face $ijk \in \mathsf{F}_1$ in which vertex $m \in \mathsf{V}^\star$ lies, along with the barycentric coordinates (b_i, b_j, b_k) describing the exact location of m in face ijk. And of course, as we modify T_2, we also update the edge lengths used to encode its intrinsic metric.

[6] or equivalently, in the setting of hyperbolic geometry

[7] for halfedges ij emanating from inserted vertices $i \in \mathsf{V}^\star$, we set $r_{ij} = 0$.

3.5.1 Vertex Insertion

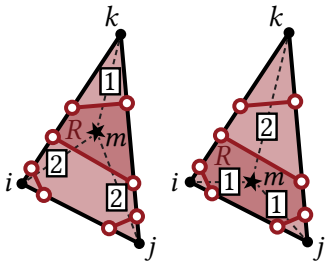
Suppose we want to insert a new vertex at a point x on T_2 expressed as barycentric coordinates v relative to a face $ijk \in \mathsf{F}_2$. To do so, we insert a new vertex m into face ijk, and must compute edge lengths, normal coordinates, and roundabouts for the new edges mi, mj, mk, as well as the positions q_m of m on triangulation T_1. Note that similar operations have been described in the topological setting (*e.g.* by Schaefer et al. [2002, Section 5.4]), but they do not provide the ability to insert a point at a particular geometric location, which is essential in many applications (*e.g.* Section 3.6).

Mesh Update We first insert m into T_2, splitting ijk into three new triangles. The new edge lengths can be computed directly from the barycentric coordinates—in particular, any tangent vector w expressed in barycentric coordinates has length

$$\|w\|^2 = -\ell_{ij}^2 w_i w_j - \ell_{jk}^2 w_j w_k - \ell_{ki}^2 w_k w_i \tag{3.7}$$

(see [Schindler & Chen 2012, Section 3.2] or [Sharp et al. 2021, Sections 2.3.2 and 2.3.7]). We can hence compute $\ell_{im} = \|(b_i, b_j, b_k) - (1, 0, 0)\|$, and similarly for ℓ_{jm}, ℓ_{km}.

Normal Coordinates & Roundabouts Unlike edge flips, where the new normal coordinates depend solely on the old ones, normal coordinates resulting from a vertex insertion depend on the particular geometric region R containing the inserted point m (see inset). We hence compute geometric crossings for all curves passing through face ijk, then determine the region R via line-side tests (implemented via a simple cross product). If m is extremely close to a region boundary we may pick the wrong region (due to floating-point error), but will still produce valid connectivity for a nearly identical vertex location. Moreover, barycentric coordinates v arising from, say, Delaunay refinement (Section 3.6) will not be exact anyway. New roundabouts emanating from vertices in $\{i, j, k\} \cap \mathsf{V}_1$ are set via Equation (3.5).



Position on T_1 The geometric crossings at R's corners provide barycentric coordinates u and v relative to T_1 and T_2 *resp.* Hence, to get q_m we simply express x as a linear combination of the corners' v–coordinates (*i.e.* their barycentric coordinates in face ijk), then take the same linear combination of the corners' u-coordinates.

Explicitly, each corner of the region R is either a vertex i of T_2 (in which case we know its location q_i on T_1), or it is an intersection between an edge of T_1 and an edge of T_2, in which case we can work out its position on T_1 during the geodesic tracing routine. And by definition, R is the intersection of ijk with some face $abc \in \mathsf{F}_1$, so we can write all of these positions in barycentric coordinates on abc.

Then we recover barycentric coordinates u for x within face abc by solving a small linear system. In principle one could use any 3 corners of R to determine the desired barycentric coordinates, but we make use of all corners of R for numerical stability. To be precise, let $3 \leq \rho \leq 6$ denote the number of corners of R. Let the m^{th} corner of R have barycentric

coordinates $u_a^{(m)}, u_b^{(m)}, u_c^{(m)}$ on $abc \in \mathsf{F}_1$ and barycentric coordinates $v_i^{(m)}, v_j^{(m)}, v_k^{(m)}$ on $ijk \in \mathsf{F}_2$, all of which are know. We also know the barycentric coordinates v_i for x in ijk. We then want to solve for the corresponding u_a on abc. We proceed in two steps: first, we express v as a linear combination ξ of the $v^{(m)}$. Then, we apply this same linear combination to the $u^{(m)}$ to obtain u. Concretely, we first solve for the minimum-norm solution of the underdetermined system

$$\begin{pmatrix} v_i^{(0)} & v_i^{(1)} & \cdots & v_i^{(\rho)} \\ v_j^{(0)} & v_j^{(1)} & \cdots & v_j^{(\rho)} \\ v_k^{(0)} & v_k^{(1)} & \cdots & v_k^{(\rho)} \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \vdots \\ \xi_\rho \end{pmatrix} = \begin{pmatrix} v_i \\ v_j \\ v_k \end{pmatrix}, \tag{3.8}$$

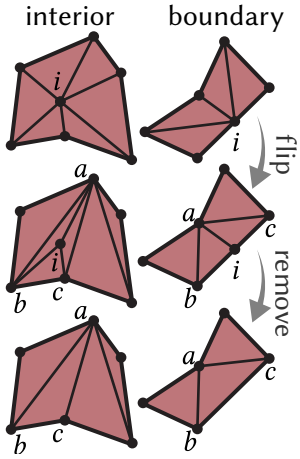
and then set

$$u_a := \sum_m u_a^{(m)} \xi_m, \quad u_b := \sum_m u_b^{(m)} \xi_m, \quad u_c := \sum_m u_c^{(m)} \xi_m. \tag{3.9}$$

Note that while one often seeks a nonnegative ξ, any solution will suffice here: we only use ξ to interpolate in Equation (3.9).

3.5.2 Flat Vertex Removal

Generally, a vertex of the original triangulation cannot be removed without distorting the intrinsic metric: any curvature at that vertex would be lost. However, inserted vertices are intrinsically flat (*i.e.* have no Gaussian curvature), and can hence be removed safely. In fact this operation is necessary for Delaunay refinement of domains with boundary (Section 3.6).

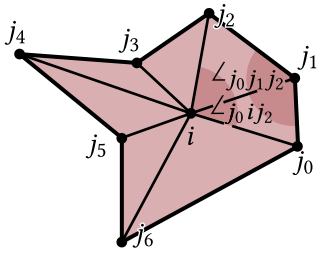


To remove an interior vertex i with zero Gaussian curvature, we perform edge flips until i has degree 3 and then replace the three triangles iab, ibc, ica incident on i with the single triangle abc (inset, *left*). Since the vertex is intrinsically flat, this change preserves the surface geometry. And we do not have to update any edge lengths, normal coordinates, or roundabouts beyond the updates required to perform the edge flips. We can remove a boundary vertex i with zero geodesic curvature by an analogous procedure: we perform edge flips until $\deg i = 3$ and replace the two resulting triangles iab, ica with the single triangle abc (inset, *right*). Note that when i is an ear vertex (*i.e.* a degree-2 boundary vertex), its degree can easily be increased to 3 by flipping the opposite edge. Again, the surface geometry remains unchanged, since i has no geodesic curvature. Theorem 1 and Theorem 2 prove the correctness of this procedure for removing interior and boundary vertices respectively, under the assumption that the neighborhood of i remains a simplicial complex throughout. We hence find that a useful heuristic is to first flip any self-edges ($i = j$); if there are none, we flip the edge ij with largest angle sum $\theta_k^{ij} + \theta_l^{ji}$ (which are, in some sense, the "most convex"). Schaefer et al. [2002, Section 5.4] also suggest a similar flipping procedure for removing interior vertices, but do so in the topological setting where the necessary edge flips are always valid—they do not consider the convexity condition (Section 2.3.3).

Theorem 1. *If an intrinsically-flat vertex i in the interior of a simplicial complex has degree $d > 3$, then some incident edge can be flipped to decrease the degree of i.*

Proof. Recall that an edge can be flipped if both endpoints will have degree at least 1 after the flip, and the edge is contained in a convex quadrilateral (Section 2.3.3). As always, the convex quadrilateral is defined in the sense of the intrinsic geometry determined by edge lengths. The endpoint degree constraint is automatically satisfied on a simplicial complex, so we only need to show that the geometric convexity constraint is satisfied, which is equivalent to showing that all angles of the edge's quadrilateral are at most π.

We denote the neighboring vertices of i by j_k (with j_{k+1} *etc.* implicitly indexed modulo the vertex degree d), as depicted in the inset. The outer angles $\angle ij_{k-1}j_k$ and $\angle ij_{k+1}k$ are corners of Euclidean triangles, and thus are necessarily at most π, so we need to find an edge ij_k for which the angles $\angle j_{k-1}ij_{k+1}$ and $\angle j_{k+1}j_kj_{k-1}$ are also at most π. First we consider the inner corners $\angle j_{k-1}ij_{k+1}$.

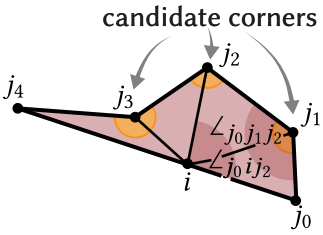


At most two of these angles can be greater than π. To see why, suppose there were three $\angle j_{k-1}ij_{k+1} > \pi$. Since the degree of i is $d > 3$, then some pair of those three large angles would correspond to disjoint angular sectors around the vertex, and summing their angles yields a value greater than 2π, which is impossible because the angle sum of i is 2π. Thus all but at most two of the edges incident on i have inner corners with angle at most π. Likewise, at least three of the outer corners $\angle j_{k+1}j_kj_{k-1}$ are at most π. This is because the sum of all d outer corners must be $(d-2)\pi$. Since they are nonnegative, at most $d-3$ of them can be strictly greater than π, implying that at least 3 will be less than or equal to π.

Thus at least three outer corners are at most π, and at most two of the inner corners are *not* at most π, so there must be at least one edge for which both the inner and outer corners are at most π. This edge can then be flipped, reducing the vertex degree. □

Theorem 2. *If a vertex i in the boundary of a simplicial complex has cone angle π and degree $d > 3$, then some edge ij incident on i can be flipped to decrease the degree of i.*

Proof. Again, our goal is to find a neighboring quadrilateral whose angles are all at most π. Since none of the inner angles $\angle j_{k-1}ij_{k+1}$ can exceed π (after all, the sum of *all* angles incident on i is only π), the only difficulty is showing that one of the $d-2$ "candidate corners" around the outside of the neighborhood must be at most π. Since i has angle sum π, we can view this neighborhood as a d-sided polygon—as noted above, such a polygon must have at least 3 corners with angle less than or equal to π. Since there are $d-2$ candidate corners, we conclude that at least one candidate corner must have angle at most π, and hence we can find a flippable edge incident on i. as desired. □



Importantly, these proofs *do not* handle the full general case of a Δ-complex, where there may exist self-edges which cause flips to not make progress. However, we note that Sharp & Crane [2020b, Appendix A] proves that a similar flip-removal strategy works in the case of a Δ-complex, and we conjecture that an analogous technique could be applied to generalize the

Figure 3.9: Intrinsic Delaunay refinement inserts new vertices intrinsically into a mesh to improve the triangle quality. We show that (under a few assumptions), intrinsic Delaunay refinement is guaranteed to produce a mesh whose triangles all have corner angles of at least 30°.

above theorems. Also, note that the "equality" case of Theorem 1 is a possibility, such as a degree four cross configuration where all angles $= \pi/2$. Fortunately the resulting skinny triangle after the edge is a non-issue, because the center vertex is about to be removed.

3.6 Intrinsic Delaunay Refinement

Delaunay refinement inserts vertices in order to produce a Delaunay mesh whose triangles all satisfy a minimum angle bound (Figure 3.9). Here we modify *Chew's second algorithm* to perform intrinsic Delaunay refinement [Chew 1993; Shewchuk 1997]. This problem has been extensively studied in the plane, but an intrinsic (*i.e.* geodesic) scheme was only recently proposed by Sharp et al. [2019, Section 4.2]. However, they did not handle meshes with boundary—here we resolve the essential difficulties of the boundary case, and show how refinement can be implemented using our integer-based data structure.

In the plane, the basic algorithm is to greedily pick any triangle which violates the minimum angle bound, insert a vertex at its circumcenter, then flip to Delaunay. This process continues until all triangles satisfy the angle bound. If a triangle's circumcenter is outside the domain, then the boundary edge ij separating the triangle from its circumcenter is split at its midpoint; subsequently, all interior vertices within at least a distance of $\ell_{ij}/2$ are removed—though removing additional interior vertices causes no issues (Section 3.6). One can prove that this process succeeds for minimum angle bounds up to 25.65 degrees on planar

Algorithm 5 DelaunayRefinement$(\mathsf{T}_2, \ell, \theta_{\min})$

Input: An intrinsic triangulation $\mathsf{T}_2 = (\mathsf{V}_2, \mathsf{E}_2, \mathsf{F}_2)$, equipped with edge lengths $\ell : \mathsf{E}_2 \to \mathbb{R}_{>0}$, along with a minimum allowed angle $\theta_{\min}$.
Output: An intrinsic triangulation T_2 whose corner angles are all at least θ_{min}

$\mathsf{T}_2, \ell \leftarrow$ FlipToDelaunay(T_2, ℓ)
while T_2 has triangles with angles less than $\theta_{\min}$ **do**
  $ijk \leftarrow$ any triangle with an angle less than $\theta_{\min}$
  ▷ *Find the circumcenter of ijk using Equation (3.10)*
  ▷ *and Equation (3.11)*
  $\hat{v}_i \leftarrow \ell_{jk}^2(\ell_{ij}^2 + \ell_{ki}^2 - \ell_{jk}^2)$
  $\hat{v}_j \leftarrow \ell_{ki}^2(\ell_{ij}^2 + \ell_{jk}^2 - \ell_{ki}^2)$
  $\hat{v}_k \leftarrow \ell_{ij}^2(\ell_{jk}^2 + \ell_{ki}^2 - \ell_{ij}^2)$
  $v \leftarrow \frac{1}{\hat{v}_i + \hat{v}_j + \hat{v}_k}(\hat{v}_i, \hat{v}_j, \hat{v}_k)$
  ▷ *Barycentric offset from barycenter to circumcenter*
  $\delta v \leftarrow v - (1/3, 1/3, 1/3)$
  ▷ *Evaluate exponential map from face barycenter*
  $c \leftarrow$ Exp(Barycenter(ijk), δv)
  if c lies inside the mesh **then**
    $\mathsf{T}_2, \ell \leftarrow$ InsertVertex(T_2, ℓ, c)
  else
    $lm \leftarrow$ boundary edge separating c from ijk
    $m \leftarrow$ SplitEdge$(lm, 0.5)$
    ▷ *Flip to Delaunay before getting Dijkstra ball*
    $\mathsf{T}_2, \ell \leftarrow$ FlipToDelaunay(T_2, ℓ)
    ▷ *Remove inserted vertices in lm's diametral ball*
    $\mathcal{B} = \{i \in \mathsf{V}_2 : \text{DijkstraDist}(\mathsf{E}_2, i, m) < \ell_{lm}\}$
    for $i \in \mathcal{B}$ **do**
      $\mathsf{T}_2, \ell \leftarrow$ RemoveVertex(T_2, ℓ, i)
  $\mathsf{T}_2, \ell \leftarrow$ FlipToDelaunay(T_2, ℓ)

domains with boundary angles at least 60° [Shewchuk 1997, Section 3.4.2]. More advanced versions of this procedure can achieve better angle bounds, *e.g.* [Rand 2011], but here we restrict our attention to the basic algorithm for simplicity.

There are two difficulties in adapting this algorithm to the intrinsic setting: locating circumcenters and computing (geodesic) distances. As mentioned earlier, intrinsic Delaunay triangulations obey the empty circumcircle property; hence each triangle has an intrinsically-flat circumdisk with a well-defined center (Figure 3.10, *left*). So long as this center corresponds to a point on the surface, it can be found by walking from the triangle's barycenter (Figure 3.10, *right*). In practice, we compute triangle ijk's circumcenter in homogeneous (*i.e.*, unnormalized) barycentric coordinates $\hat{v}_i$ via the following formula [Schindler & Chen 2012, Section 2.3]:

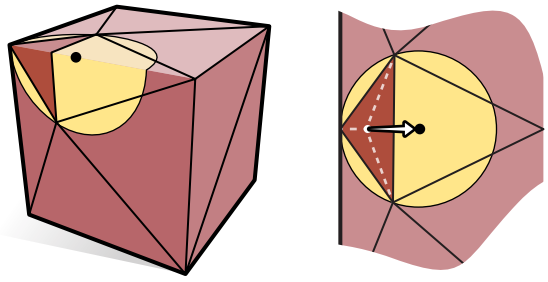


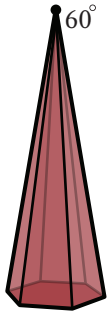
Figure 3.10: Triangles in Delaunay meshes have empty circumdisks, and thus well-defined circumcenters (*left*). When necessary, we locate a triangle's circumcenter by walking outwards from its barycenter (*right*).

$$\hat{v}_i := \ell_{jk}^2(\ell_{ij}^2 + \ell_{ki}^2 - \ell_{jk}^2), \tag{3.10}$$

and then normalize to obtain barycentric coordinates

$$v_i := \frac{\hat{v}_i}{\hat{v}_i+\hat{v}_j+\hat{v}_k}. \tag{3.11}$$

To locate the circumcenter on the surface, we then evaluate the exponential map (Section 2.3.2) starting at the barycenter $w_i = w_j = w_k = 1/3$, along the vector $v - w$. If we hit a boundary edge ij while tracing out this path, then the circumcenter is not contained in the surface, so we split ij at its midpoint and flip to Delaunay. We must then remove all inserted interior vertices within a geodesic ball of radius $\ell_{ij}/2$ centered at the inserted point. Computing geodesic distance on a surface mesh is nontrivial, but Xia [2013, Corollary 1] shows that on a Delaunay triangulation any vertex inside a geodesic ball of radius r will also be inside the Dijkstra ball of radius $2r$ (*i.e.* points whose distance along the edge graph are at most $2r$). We hence remove all interior inserted vertices within a Dijkstra distance of ℓ_{ij}. While Xia considers only the planar setting, their proof (which is based on triangle strips) applies without modification to intrinsic Delaunay triangulations of surfaces. Observe also that, as in the planar case, Delaunay refinement only ever removes *previously-inserted* vertices. Hence, as assumed in Section 3.5.2, the original extrinsic vertex set V_1 is still preserved.



On meshes with narrow cone vertices or boundary angles, it may be impossible to find any triangulation satisfying a given angle bound. In such cases, we do not insert circumcenters of intrinsic triangles incident on exactly one narrow vertex, or are entirely contained in a triangle of T_1 incident on a narrow vertex, and ignore such triangles when computing the minimum corner angle of the output mesh. While the final output may violate the angle bound, violations occur only near narrow vertices. In analogy with the planar case, we set 60° as the minimum allowed angle sum (see inset); in practice the vast majority of meshes obey this constraint at all vertices (97.2% of Thingi10k), and we can obtain high-quality triangulations even on those which do not.

Removing Extra Vertices When Chew's second algorithm splits an edge, it removes all inserted circumcenters within a geodesic ball centered at the edge's midpoint. These vertices must be removed, but it is okay to removes additional interior inserted vertices. Shewchuk [1997, Section 3.4.2] observes that the algorithm can only perform finitely many edge splits. As long as one removes all interior inserted vertices within the geodesic ball—and never removes vertices along the boundary—the algorithm will still perform only finitely many edge splits. Hence, it must terminate as usual following the final edge split, even if one removes extra circumcenters during edge splits.

3.6.1 Refinement Results

As a stress test, we successfully compute an intrinsic Delaunay refinement and associated subdivision for all manifold meshes in the Thingi10k dataset of Zhou & Jacobson [2016]; in turn, these high-quality intrinsic triangulations allow users compute reliable and highly accurate solutions to partial differential equations even on extremely low-quality meshes. In particular, we used *MeshLab* to convert each mesh to the PLY file format [Cignoni et al. 2008], resulting in 7696 valid manifold meshes. We begin by mollifying each mesh to a tolerance of 10^{-5} (Section 3.7). For each model we compute the intrinsic Delaunay triangulation (Section 2.3.3), as well as an intrinsic Delaunay refinement (Section 3.6) with a 25° angle bound. We verify that the algorithms terminate with the expected conditions. Additionally, we successfully extract an explicit mesh of the common subdivision in both cases, except for 1 model in the case of refinement whose common subdivision contains around 30 million vertices (Figure 3.12, *left*).

We compare against the explicit overlay representation of Fisher et al. [2006] and the signpost representation of Sharp et al. [2019] (Table 3.1). The overlay representation similarly offers a guarantee of valid connectivity, but does not provide a constant-time edge flip operation (like normal coordinates do). More importantly it does not support operations beyond edge flips and thus cannot perform Delaunay refinement. Signposts support a wide range of operations, but may not successfully recover the common subdivision on degenerate inputs (Figure 3.11). The statistic reported here differs from the result in Sharp et al. [2019], because no preprocessing of meshes is per-

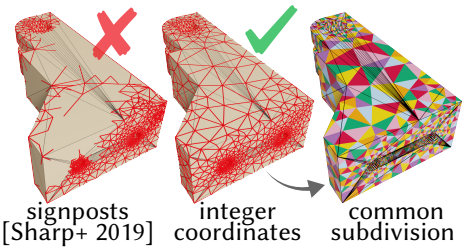


Figure 3.11: Signposts may fail to recover the common subdivision on near-degenerate inputs. By contrast, integer coordinates always yield a valid common subdivision.

Method	Intrinsic Delaunay Triangulation	Intrinsic Delaunay Refinement
Explicit Overlay	100 %	not supported
Signpost Tracing	96.0 %	69.1 %
Integer Coordinates	100 %	100 %

Table 3.1: Success rate of integer coordinates compared to other approaches on the Thingi10k dataset. We construct a Delaunay triangulation and Delaunay refinement on each model, and attempt to recover the connectivity of the common subdivision.

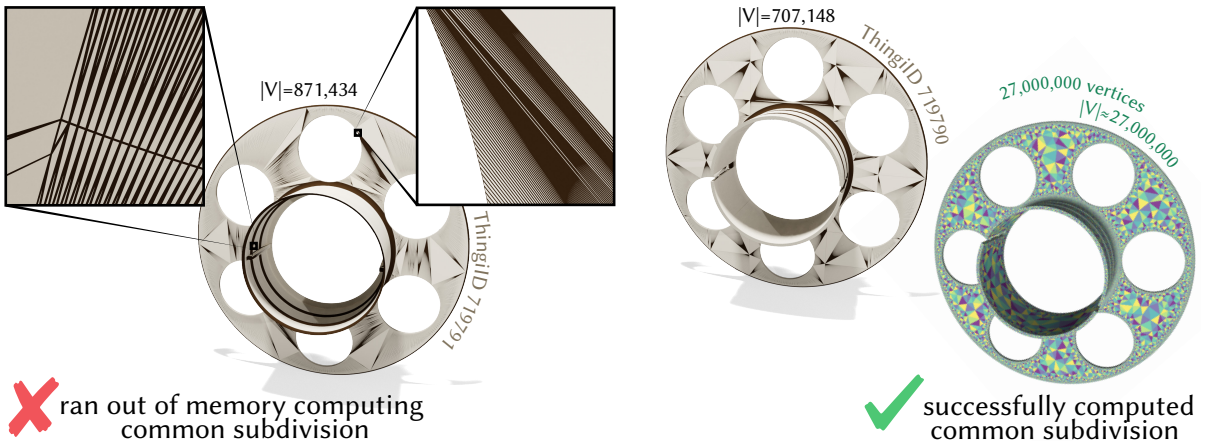


Figure 3.12: We fail to compute an explicit mesh of the common subdivision following Delaunay refinement on one Thingi10k model (*left*). Its common subdivision would contain 34 million vertices and our program runs out of memory. We succeed on a nearly identical model (*right*), whose common subdivision contains merely 27 million vertices.

formed. For refinement Sharp et al. [2019] do not treat the boundary case, so we compare only on models without boundary.

Performance and Complexity Our data structure is able to compute Delaunay refinements for complex meshes in seconds[8]. For example, computing the Delaunay refinement in Figure 15 of Gillespie et al. [2021a] took 0.2s on a mesh with 3000 vertices, and the Delaunay refinement in Figure 16 of the same paper took 0.6s on a mesh with 10,000 vertices. Because we lazily recover intersection geometry from our integer coordinates when inserting vertices, routines such as Delaunay refinement which perform many insertions may become moderately expensive on large near-degenerate inputs. For instance we take 4 minutes to perform Delaunay refinement on 719791 (Figure 3.12, *left*) whereas signposts take only 1.5 minutes—but on such meshes signposts generally fail to compute a valid common subdivision.

Our data structure is carefully constructed to ensure that edge flips can be performed in constant time; each edge flip simply amounts to a few arithmetic operations. Removing a vertex of degree d takes time $O(d)$, since one must perform $O(d)$ edge flips before removing the resulting degree-3 vertex. On the other hand, inserting a new vertex into face ijk of T_2 requires tracing geodesics for each edge of T_1 which intersects face ijk, and takes time proportional to the number of intersections between those geodesics and the edges of T_2.

Similarly, the time and memory cost of computing the common subdivision scales linearly with the number of intersections between edges of T_1 and edges of T_2—or equivalently, with the size of the common subdivision. For most meshes, this does not pose an issue, but on one model (depicted in Figure 3.12, *left*) we ran out of memory while trying to extract the common subdivision. In such cases, it may be helpful to simplify the input mesh before running intrinsic Delaunay refinement (Chapter 5).

[8]Timings are measured on a single core of an Intel i9-9980XE with 32 GB of RAM.

3.6.2 Proof of Correctness on Manifold Meshes without Boundary

Here we seek to prove that DelaunayRefinement (Algorithm 5) succeeds, in the basic case of a closed surface with bounded cone angles. We will not prove the more general boundary case here, but experimentally we observe success on a large dataset (Section 3.6.1).

Theorem 3. *On meshes without boundary, with vertex angle sums at least* $60°$*, Algorithm 5 produces a Delaunay mesh with triangle corner angles at least* $30°$*.*

Proof. By definition, DelaunayRefinement only terminates when the triangulation is a Delaunay triangulation which satisfies the angle bound, so we just need to prove that termination occurs after a finite number of iterations. We will show this by establishing that DelaunayRefinement maintains a minimum spacing between all vertices in the mesh, so the number of insertions is bounded by surface area. Our argument will generally follow the planar proof of Shewchuk [1997, Section 3.2.1], though extra care is needed in the intrinsic case, where self edges may connect a vertex to itself.

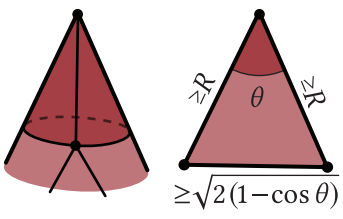
In particular, we consider the length of the shortest edge in the initial mesh's intrinsic Delaunay triangulation, $\delta := \min_{ij} \ell_{ij}$. We will show that the minimum edge length in each subsequent Delaunay triangulations is at least δ. Then all vertices must be separated by a distance at least δ, since Lemma 2, each vertex is connected to its geodesic nearest neighbor. Hence, each vertex is contained in an open disk of radius $\frac{1}{2}\delta$ which is disjoint from all other disks. As the input mesh has finite surface area, we conclude that Algorithm 5 can only insert finitely many vertices, and thus must terminate.

It remains to show that DelaunayRefinement never creates an edge of length less than δ. First, we convert angle bound α to a *circumradius-to-shortest-edge ratio* bound $B = \frac{1}{2\sin\alpha}$ [Shewchuk 1997, Section 3.1]: a triangle has corner angles at least $\alpha = 30°$ whenever $B \leq 1$. Note that we insert circumcenters for triangles with $B > 1$.

We proceed by induction. First, all initial edges have length at least δ by definition. Now consider inserting vertex i at the circumcenter of triangle jkl with circumradius R. Since we only split triangles with $B > 1$, and jkl's edges have length at least δ, we must have $R > \delta$. All of the new edges that we create must be incident on i (Lemma 3), and since jkl had an empty circumcircle, there can be no other vertices within distance $R > \delta$. So any new edges to other vertices have length at least δ. But we must also consider self edges connecting i to itself.

Gluing together the two ends of a self edge yields a loop; we will split into cases based on the homotopy class of this loop on the punctured surface (before the insertion of i). First, note that the loop cannot be contractible to a point, since the original edge is geodesic. We split into two cases: either the loop contracts around a single vertex, or it does not.

If the loop contracts around a single vertex, then the self edge encloses a degree-1 vertex. The degree-1 vertex must have distance at least R to the inserted vertex, and has angle sum at least $60°$. Thus, by the law of cosines, the length of the self edge is at least



$$\sqrt{R^2 + R^2 - 2R^2\cos\theta} = R\sqrt{2(1-\cos\theta)}.$$

Since $\cos 60° = \frac{1}{2}$, and $1 - \cos\theta$ is increasing with θ, this shows that the self edge has length at least R whenever θ is at least $60°$.

If the loop is not in a homotopy class contractible about a single vertex, then the shortest loop $\gamma_{\min}$ in the homotopy class has nonzero length. By Lemma 4, we can take $\gamma_{\min}$ to touch some vertex a, and note that since $\gamma_{\min}$ is the *shortest* loop our original self edge must be at least as long as $\gamma_{\min}$. Then by Lemma 2 a has an edge at least a long as $\gamma_{\min}$, and thus the self edge has length $\geq |\gamma_{\min}| \geq \delta$.

Thus, we conclude that Algorithm 5 never introduces an edge of length less than δ, which means that it must terminate after inserting finitely many vertices. □

Lemma 1. *For any pair of vertices $i, j \in V$, let Γ_{ij} be the set of non-constant geodesics connecting i to j. Then*

$$d_{ij} := \inf_{\gamma \in \Gamma_{ij}} length(\gamma) > 0.$$

Proof. This follows directly from [Indermitte et al. 2001, Proposition 1], which states that for any $L > 0$, the number of geodesic arcs from i to j of length at most L is finite. Since any geodesic of length 0 is constant, and thus not in Γ_{ij}, this implies that $d_{ij} > 0$. □

Lemma 2. *For any vertex $i \in V$, the intrinsic Delaunay triangulation contains an edge to i's nearest neighbor.*

Proof. This is a standard result, which we include for completeness. Let j be i's nearest neighbor, *i.e.* $j := \text{argmin}_j d_{ij}$. Note that j may equal i, and $d_{ij} > 0$ by Lemma 1. Consider the disk D of radius d_{ij} centered at i. Since j is i's nearest neighbor, D contains no vertex other than i. Thus, the circle which goes through i and j and is tangent to D at j has empty interior, and its boundary contains no vertices other than i and j. We conclude that ij is in the Delaunay triangulation [Bobenko & Springborn 2007, Definition 3]. □

Lemma 3. *All edges created in DelaunayRefinement following the insertion of a vertex i and flipping to Delaunay are incident on i.*

Proof. Again, we follow the planar proof of Shewchuk [1997, Lemma 12]. We wish to prove that all Delaunay edges which are not incident on i were Delaunay before inserting i. This follows from the fact that edges of a Delaunay triangulation satisfy an empty circumcircle condition [Bobenko & Springborn 2007, Definition 3]. If an edge's circumcircle is empty after inserting vertex i, it must have been empty before too, so the edge was already Delaunay. □

Lemma 4. *Any geodesic loop γ is isotopic to a geodesic loop γ' of the same length which touches a vertex.*

Proof. γ can "slide" until it touches a vertex without changing its length. Precisely, consider a unit-speed motion of γ within the surface along its outward normal direction. During the motion, $\frac{d}{dt}|\gamma| = \int_\gamma \kappa(s)\, ds$, where κ is the geodesic curvature of γ. Since γ is a geodesic, $\kappa = 0$: its length does not change. Thus we can construct γ' by sliding γ along the surface until it touches a vertex. □

As an aside, we note that geodesic loops which do not touch a vertex only occur in non-generic configurations.

3.7 Robust Implementation

Our integer coordinates are guaranteed to encode a triangulation sitting atop T_2. The geometric accuracy of this triangulation, of course, depends on floating point arithmetic, which can become inaccurate in near-degenerate configurations. *Exact predicates* have been applied with great success to similar problems [Devillers & Pion 2003]. Unfortunately they do not directly apply to intrinsic triangulations, as the predicates that we evaluate are not fixed functions of the input data; an intrinsic edge length can depend upon arbitrarily many input edge lengths. Hence, we focus on fast and robust implementations using ordinary floating point arithmetic.

One essential tool for manipulating intrinsic triangulations on near-degenerate input meshes is *intrinsic mollification*, introduced by Sharp & Crane [2020a]. Mollification provably ameliorates near-degenerate meshes by adding a small value δ to every edge length, ensuring that every triangle satisfies the triangle inequality with slack at least ϵ. This operation only changes the geometry if some triangle is within ϵ of being degenerate, and even then changes the geometry by a negligible amount. Intrinsic mollification works particularly well with our data structure compared to past approaches: the signpost data structure of Sharp et al. [2019] relies on tracing edges along the surface, which become *less* accurate when mollification is applied. Integer coordinates have no such problem. In our experiments we mollify with $\epsilon = 10^{-5}h$, where h is the mean edge length, and find that it resolves almost all numerical difficulties.

Even after mollification, it is still beneficial to use care when working with floating point. For example, there are well-conditioned triangles on which the Delaunay condition (Equation (2.6)) is difficult to evaluate; in practice, we only enforce Equation (2.6) up to some ϵ tolerance. As a further example, when computing new normal coordinates in SplitFace, one could lay out the face in the plane, and independently count intersections along the new edges. However, this can produce invalid normal coordinates in floating point. We apply a more nuanced policy, described in the paper, which always yields valid normal coordinates.

Chapter 4

Surface Parameterization

The famous uniformization theorem of Koebe is perhaps the single most important theorem in the whole theory of analytic functions of one variable. … As soon as the uniformization theorem is proved, it is not necessary to consider Riemann surfaces more general than the disk, the plane, and the sphere. It must be admitted, of course, that the reduction to these cases does not always simplify matters.

Lars Ahlfors [1973]

The first instance of dynamic surfaces that we will consider arises from surface parameterization. We describe a numerical method which computes maps from that are locally injective and discretely conformal in an exact sense. Unlike previous methods for discrete conformal parameterization, the method is guaranteed to work for any manifold triangle mesh, with no restrictions on triangulation quality or cone singularities. In particular we consider maps from surfaces to the plane, and globally bijective maps from genus zero surfaces to the sphere. Recent theoretical developments have shown that each task can be formulated as a convex problem where the triangulation is allowed to change—we complete the picture by introducing the machinery needed to actually construct a discrete conformal map. A key challenge lies in tracking the correspondence between our evolving intrinsic triangulation and the original mesh.

Problem Statement

We start with a manifold triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ equipped with edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$. The triangulation T may have any genus, and may have a nonempty boundary. Our goal in this chapter is to construct a discrete conformal parameterization, *i.e.* a map $\varphi : \mathsf{T} \to \mathbb{R}^2$ which is discretely conformal in the exact sense defined in Section 4.1. The parameterization φ is locally-injective, meaning that it does not flip any triangles of T. Unlike many common parameterizations, φ is not linear on the faces of T, rather it is defined to be projective on the faces of a refined triangulation $\tilde{\mathsf{T}}$.

When T has genus zero, we also describe an algorithm for computing globally bijective maps $\varphi : \mathsf{T} \to S^2$ to the sphere, which are also discretely conformal in an exact sense.

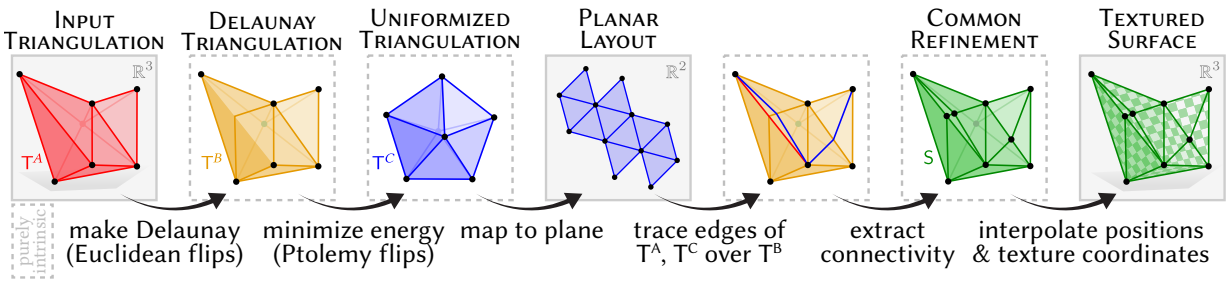


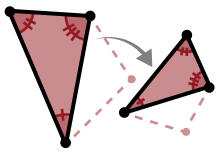
Figure 4.1: Steps of our algorithm. Throughout we color the input mesh T^A red, its intrinsic Delaunay triangulation T^B yellow, the uniformized triangulation T^C blue, and the common subdivision S of all three green. (Note: triangulations in dashed boxes are purely intrinsic and never actually embedded in $\mathbb{R}^n$.)

In the smooth setting, existence of conformal maps is guaranteed by the *uniformization theorem* [Abikoff 1981]. Very recently, Gu et al. [2018b,a] and Springborn [2019] established an analogous *discrete uniformization theorem* for triangle meshes. However, these theoretical results fall short of providing practical algorithms, since they do not describe how to construct the mapping between the input and target domain. We present the first end-to-end algorithm for computing and evaluating this map—in particular, we provide:

- a combinatorial data structure for correspondences between triangulations (Section 4.2.1),
- a scheme for evaluating discrete conformal maps based on the *light cone* (Section 4.2.3), and
- critical details needed to implement discrete uniformization including a careful treatment of numerics and boundary conditions (Section 4.3), and subtleties of the spherical case (Section 4.4).

Our optimization procedure is a simple modification of the *CETM algorithm* (from Springborn et al. [2008], *Conformal Equivalence of Triangle Meshes*): we minimize the same energy, but evaluate it on a triangulation that changes according to the current scale factors. However, since the triangulation may now change, this procedure does not yield an explicit parameterization of the input. To improve the quality of the map, we also need to flip the input to an *intrinsic Delaunay triangulation* before starting the optimization. The main difficulty in developing a practical algorithm is therefore tracking and evaluating the correspondence between these three triangulations—Figure 4.1 gives an overview of the whole process. Importantly, even though the Euclidean geometry of our polyhedron changes during optimization, it can always be seen as different reflections of the same underlying hyperbolic polyhedron, which greatly simplifies the problem of correspondence. This perspective lets we can adapt the integer coordinates data structure from Chapter 3 to store the correspondence between two hyperbolic polyhedra, which are guaranteed to be isometric, even if the Euclidean polyhedra which they represent are not.

4.1 Discrete Conformal Equivalence

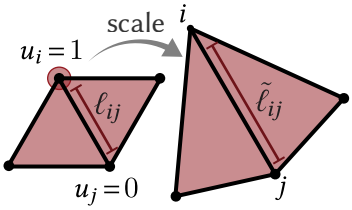


In the smooth setting, conformal maps preserve angles—naïvely, one might therefore require that for triangle meshes, discrete conformal maps preserve the angles at all corners. However, this condition is far too rigid: since each triangle can only scale and rotate, its neighbors—and in turn, the entire surface—may only scale by a constant amount. As a result, many other notions of discrete conformal maps have been explored; Crane [2020] gives a detailed account.

A particularly successful approach is the notion of *discrete conformal equivalence*. In the smooth setting, two Riemannian metrics $g, \tilde{g}$ are conformally equivalent if they are related by a positive scaling $\tilde{g} = e^{2u} g$ for some real-valued function u. On a triangle mesh two discrete metrics (*i.e.* sets of edge lengths) $\ell, \tilde{\ell}$ are called discretely conformally equivalent if

$$\tilde{\ell}_{ij} = e^{(u_i + u_j)/2} \ell_{ij} \tag{4.1}$$

for some assignment of *scale factors* $u_i \in \mathbb{R}$ to vertices i [Roček & Williams 1984; Luo 2004]. This innocent-looking definition leads to a rich discrete theory which is just as flexible as the smooth one [Bobenko et al. 2015]. Bücking [2016, 2018] and Gu et al. [2019] consider convergence under refinement.



Two sets of edge lengths are discretely conformally equivalent if and only if they induce the same *length cross ratios* [Springborn et al. 2008, Section 2]

$$\mathfrak{c}_{ij} = \frac{\ell_{il} \ell_{jk}}{\ell_{lj} \ell_{ki}}. \tag{4.2}$$

We will give a definition of conformal equivalence for Euclidean polyhedra with *different* connectivity while discussing the variable triangulation setting below.

4.1.1 Discrete Uniformization

Conformal equivalence offers an appealing strategy for parameterization: rather than solve directly for a map to the plane, first find scale factors that describe a discretely conformally equivalent flat surface—perhaps with target angle defects Ω_i^* prescribed at just a few isolated *cone points* (Figure 4.2, *center*). This new surface is then cut open and unfolded into the plane (Figure 4.2, *right*). In the smooth setting, existence of such scale factors is guaranteed by the *uniformization theorem* [Abikoff 1981] and its generalization to cone metrics [Troyanov 1991]. In the discrete setting, however, there is a critical problem: for a fixed triangulation, there may be no scale factors that achieve the target angle defects. One must therefore adopt an expanded notion of discrete conformal equivalence that allows the triangulation to change (Section 4.1.1).

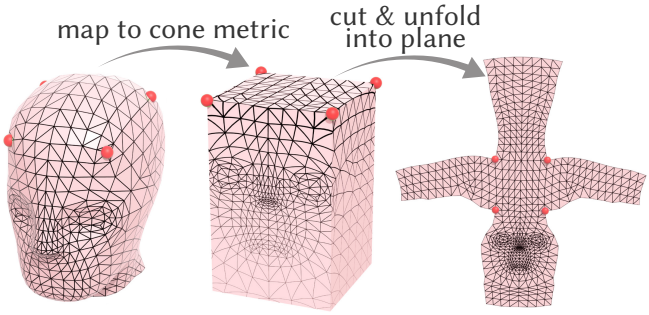


Figure 4.2: Conformal parameterization with cones.

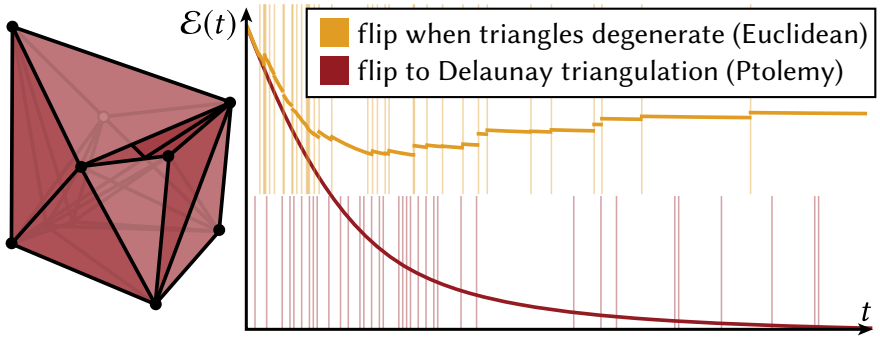


Figure 4.3: Flipping edges when triangles degenerate causes the energy $\mathcal{E}$ to jump discontinuously—voiding any guarantee of convergence *(top)*. In contrast, flipping to Delaunay via Ptolemy flips before evaluating the energy ensures that we always reach the correct solution *(bottom)*. Here we consider a coarse double torus with target angle defects $+3\pi/4$ at all but one vertex, which has large negative curvature. We take small steps to clearly plot the energy; vertical lines indicate flip times.

To actually compute the scale factors, Luo [2004] proposed the *discrete Yamabe flow*

$$\frac{d}{dt}u_i(t) = \Omega_i^* - \widetilde{\Omega}_i(t). \tag{4.3}$$

Here $\widetilde{\Omega}_i(t)$ are the angle defects induced by the scale factors $u(t)$. However, since there may be no scale factors that achieve the target angle defects, this flow can fail to reach a critical point $\frac{d}{dt}u_i = 0$, where $\widetilde{\Omega}_i = \Omega_i^*$. In this case, the scaled edge lengths $\tilde{\ell}$ will eventually violate the triangle inequality—at which point the flow becomes ill-defined and cannot continue. Springborn et al. [2008] and Bobenko et al. [2015] describe this flow as gradient descent on an explicit convex energy $\mathcal{E}$, leading to the more efficient, 2nd-order *CETM algorithm*. CETM extends $\mathcal{E}$ to be well-defined even for invalid edge lengths—but if the minimizer is found in this extended region, it fails to describe a valid parameterization (Figure 4.18).

Flipping Edges. Luo [2004] conjectured that degenerate triangles might be avoided by applying Euclidean edge flips at the exact moment when triangles degenerate, as implemented by Campen & Zorin [2017b, Section 7.3.1], but this idea has two fatal flaws. First, mixing flips with vertex scaling can yield lengths that are not conformally equivalent to the original ones (Figure 4.4). Second, it can cause discontinuities in the value of $\mathcal{E}$, voiding any guarantee that the flow will converge (Figure 4.3). This lack of guarantees is a problem even for methods that care only about injectivity, and not conformal maps [Chien et al. 2016; Campen & Zorin 2017b; a; Campen et al. 2019]. Likewise, the generalized method of Chen et al. [2016, Algorithm

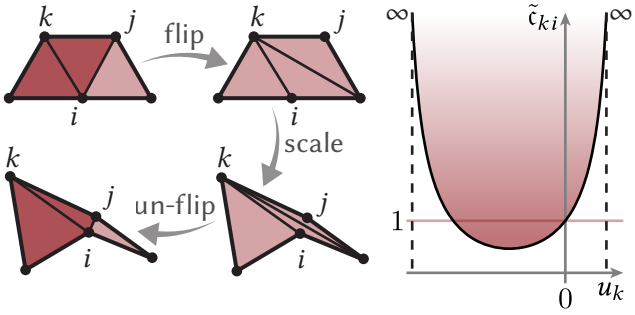


Figure 4.4: Performing Euclidean edge flips at arbitrary moments in the flow can badly distort the conformal structure. Here, we flip edge ij, scale edges incident on k by a factor $e^{u_k/2}$, and undo the flip. The cross ratio $\tilde{c}_{ki}$ of edge ki (Equation (4.2)) is not preserved, and in fact can take almost any value.

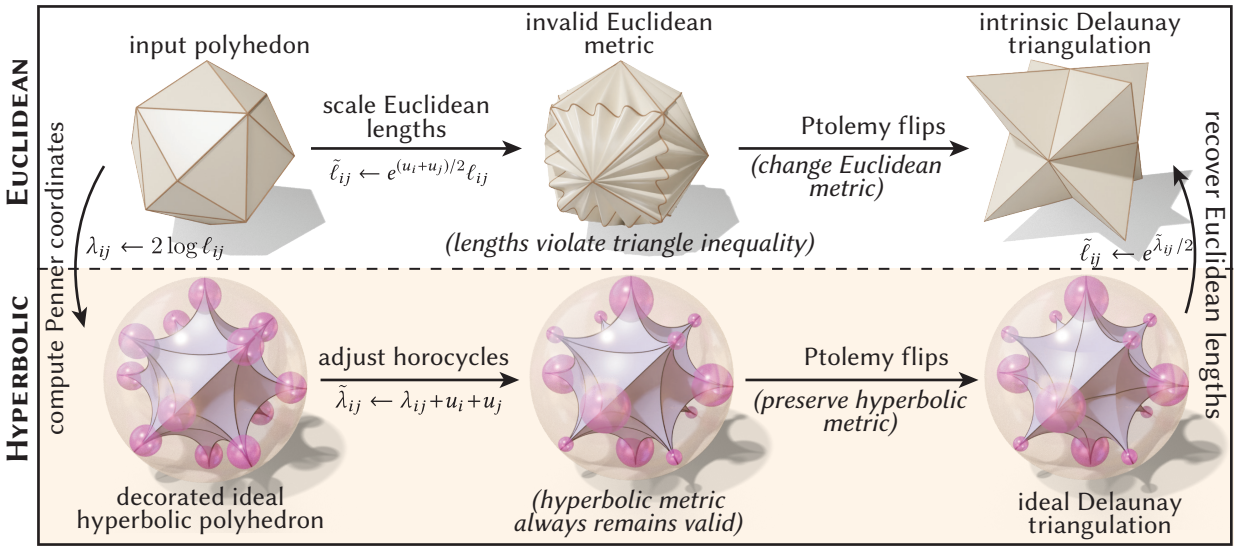


Figure 4.6: *Top:* Triangle meshes with different connectivity (but the same vertices) are considered discretely conformally equivalent if they are the same up to a conformal rescaling of edge lengths, followed by Ptolemy edge flips to a Delaunay triangulation. *Bottom:* This definition, and the use of Ptolemy (rather than Euclidean) edge flips, arises from a hyperbolic perspective, where we simply retriangulate a hyperbolic polyhedron without changing its geometry.

1] takes a step of arbitrary size before performing power Delaunay flips, and Yu et al. [2017, Algorithm 1] take an arbitrary step before performing Euclidean flips. Both algorithms can hence distort conformal structure, or even violate the triangle inequality—at which point the flow is undefined and cannot continue. Our use of *Ptolemy flips* ensures the flow is always well-defined and exactly preserves the conformal structure (see Appendix A.2.2).

Variable Triangulations A recent theoretical breakthrough is a notion of discrete conformal equivalence that does not depend on how a polyhedral surface is triangulated (Figure 4.5), along with associated *discrete uniformization theorems* for the Euclidean [Gu et al. 2018b], hyperbolic [Gu et al. 2018a], and spherical [Springborn 2019] cases. This work is intimately linked to realization results for ideal hyperbolic polyhedra [Rivin 1994b; Fillastre 2008; Prosanov 2020]. The theorems guarantee one can *always* find a conformally equivalent triangulation with prescribed angle defects Ω^*, so long as they satisfy Gauss-Bonnet. This solution is unique up to scale (Euclidean case) or Möbius transformations (spherical case).

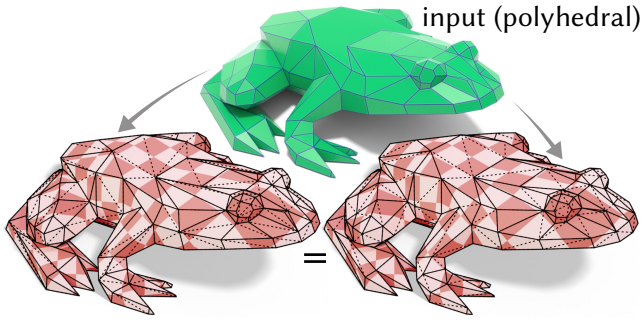


Figure 4.5: We adopt a notion of conformal equivalence that yields the same discrete conformal map, no matter how the input polyhedral surface is triangulated. Here a mesh with planar faces is triangulated two different ways, yielding identical results.

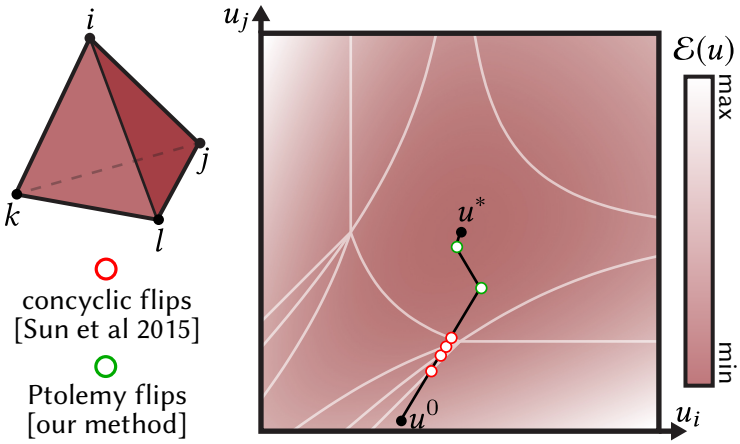


Figure 4.8: A slice of the energy landscape for a tetrahedron. Each conformal scaling u induces a Delaunay triangulation—white curves delineate regions with a common triangulation. Previous algorithms must stop and flip at each region boundary (where triangles become concyclic), whereas we can flip at any moment—since Ptolemy flips commute with scaling.

There are two equivalent definitions of discrete conformal equivalence—a key idea introduced by Gu et al. [2018b] is to consider an *intrinsic Delaunay triangulation* of the input (Section 2.3.3). Note that Delaunay triangulations are used not because they are "nice" in a numerical sense, but because their properties are essential to establishing the discrete uniformization theorem.

One definition is that two Delaunay triangulations are conformally equivalent if they are related by a sequence of vertex scalings (Equation (4.1)) and concyclic Euclidean edge flips (Figure 4.7), which maintain the Delaunay property [Gu et al. 2018b, Definition 1.1]. Algorithms that adopt this definition must stop and flip whenever two triangles become concylic. Wu [2014] shows that only finitely many flips are needed, ensuring that computation terminates. Sun et al. [2015] present an implementation of such a scheme, but do not evaluate the pointwise map between the domain and target (as needed for, *e.g.*, texture mapping or remeshing).

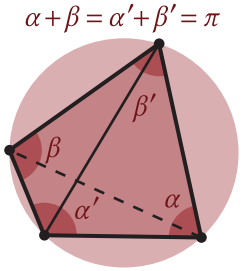


Figure 4.7: Both of the triangulations of a circular quad obey the local Delaunay condition $\alpha + \beta \leq \pi$.

We adopt an alternative definition which is theoretically equivalent—though this is far from obvious: the two triangulations are discretely conformally equivalent if they describe the same *ideal hyperbolic polyhedron* [Bobenko et al. 2015, Definition 5.1.4]. As observed by Springborn [2019], a discretely conformally equivalent triangulation can be obtained by applying a vertex scaling, then flipping to a Delanuay triangulation via *Ptolemy flips*, rather than ordinary Euclidean flips (Figure 4.6, *top*). Ptolemy flips are well-defined even when edge lengths fail to satisfy the triangle inequality, so one need not worry about maintaining a valid Euclidean metric, nor about triangles being concyclic: one can scale to any *invalid* metric, then flip to a valid one, which effectively retriangulates an associated *ideal hyperbolic polyhedron* (Figure 4.6, *bottom*). Campen et al. [2021] also adopt this approach in concurrent work.

By adopting this definition, we cast discrete conformal parameterization as an unconstrained convex optimization problem where the only variables are the scale factors u_i. The optimizer need not worry about edge flips, which can be encapsulated in a routine to evaluate the energy and its derivatives. Moreover, we can use a 2nd-order Newton method to achieve fast convergence, since the energy we minimize is twice continuously differentiable even across different triangulations. Overall this approach is generally faster than stopping to perform flips, and also accommodates the more difficult spherical case, via bounds constraints (Section 4.4).

4.1.2 Working with Hyperbolic Polyhedra

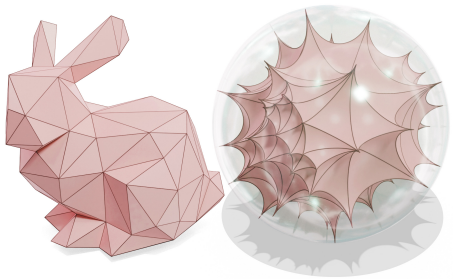


Figure 4.9: An ordinary triangle mesh *(left)* can always be viewed as an *ideal hyperbolic polyhedron (right)*, *i.e.*, surface made from triangles of constant negative curvature and all three vertices lying on the ideal boundary of hyperbolic space.

While the name *ideal hyperbolic polyhedron* may sound intimidating, these objects are closely connected to ordinary triangle meshes, and from a computational perspective they are—if anything—simpler to work with. An introduction to the theory can be found in Appendix A, but in this section we give a brief overview of the details required to implement our parameterization algorithm. Just as a (Euclidean) triangle mesh can be encoded by a triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ along with edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$, an ideal hyperbolic polyhedron can be encoded by a triangulation equipped with *Penner coordinates* $\lambda : \mathsf{E} \to \mathbb{R}$. And unlike the Euclidean case, where the edge lengths must satisfy some validity conditions, these Penner coordinates can be arbitrary. Any assignment of a real number to each edge of the mesh defines a valid ideal polyhedron.

Correspondence with Euclidean Meshes Given a Euclidean triangle mesh with lengths ℓ, we obtain a corresponding ideal polyhedron by taking Penner coordinates $\lambda_{ij} = 2 \log \ell_{ij}$. A standard mesh and its associated ideal polyhedron are illustrated in Figure 4.9.

Ptolemy Flips Penner coordinates are easily updated during edge flips via *Ptolemy's relation* [Penner 2012, Corollary 4.16, p. 40]. Letting $\ell_{ij} = e^{\lambda_{ij}/2}$ for each edge ij, we compute

$$\ell_{kl} = (\ell_{ki}\ell_{lj} + \ell_{jk}\ell_{li})/\ell_{ij}. \tag{4.4}$$

The new Penner coordinate is then $\lambda_{kl} = 2 \log(\ell_{kl})$. Since Equation (4.4) is a rational expression in ℓ, it is often simplest to just store and manipulate the edge lengths ℓ rather than the Penner coordinates λ. See the paper for further discussion of numerics.

Importantly, this so-called *Ptolemy flip* is the same as a Euclidean edge flip whenever the two Euclidean triangles are concyclic. In general, Euclidean flips may distort the discrete conformal structure even though they preserve the Euclidean geometry, while Ptolemy flips always preserve the hyperbolic metric, hence the conformal structure. Moreover, Euclidean flips are well-defined only when the triangle inequalities are satisfied, whereas Ptolemy flips are always well-defined.

Ideal Delaunay Triangulations The hyperbolic analogue of Euclidean Delaunay triangulations are *ideal Delaunay triangulations* (Appendix A.2.3): if $\ell = e^{\lambda/2}$ are edge lengths associated with given Penner coordinates λ, then every edge must satisfy the *local ideal Delaunay condition*

$$\ell_{ij}^2(\ell_{jk}\ell_{ki} + \ell_{il}\ell_{lj}) < (\ell_{il}\ell_{ki} + \ell_{jk}\ell_{lj})(\ell_{il}\ell_{jk} + \ell_{ki}\ell_{lj}). \tag{4.5}$$

We can find such a triangulation by greedily flipping edges, this time using Ptolemy flips. Remarkably, if Equation (4.5) is satisfied globally, then the lengths ℓ always describe a valid *Euclidean* intrinsic Delaunay triangulation [Springborn 2019, p. 4.14]. Yet working in the ideal setting enables us to start with an invalid Euclidean metric and flip to a valid one (Figure 4.6).

4.2 Correspondence

The triangulation produced by uniformization cannot be used in most applications without mapping data back to the input mesh. Two basic strategies have been developed for this purpose. Fisher et al. [2007] and Sun et al. [2015] maintain a mesh of the common subdivision, ensuring correct connectivity at the cost of implementation complexity. Sharp et al. [2019] instead represent correspondence implicitly using signpost vectors at vertices. This floating-point encoding is more efficient, but suffers from errors in extreme situations (Figure 4.10). We opt for integer coordinates (Chapter 3), which provide the best of both worlds: an implicit encoding that can be updated easily, yet guarantees the right connectivity. But in order to do so, we have to solve two problems: first, we need to adapt the data structure to encode hyperbolic polyhedra, in addition to standard Euclidan polyhedra. And second, we need to encode the correspondence between three distinct triangulations, not just two (Figure 4.1). We detail the necessary adjustments in Section 4.2.1 and Section 4.2.2 respectively.

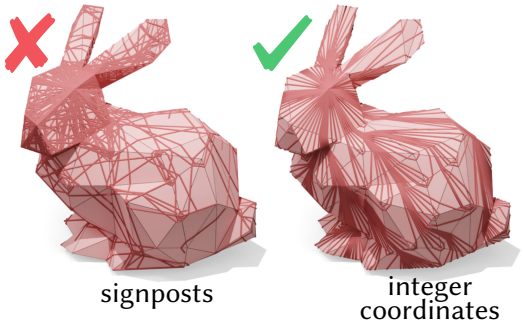


Figure 4.10: *Left:* The signpost data structure suffers from numerical error in extreme situations, like the "peacock triangulation" from Section 4.4.3. *Right:* integer coordinates always provide the correct connectivity.

Another basic question is how to interpolate data across triangulations, such as vertex or texture coordinates. The natural choice for discrete conformal maps is to use piecewise projective interpolation [Bobenko et al. 2015], which can be implemented via standard homogeneous coordinates [Springborn et al. 2008, Section 3.4]. We extend this idea to variable triangulations by laying out triangles in the *light cone* rather than the Euclidean plane (see Section 4.2.1).

Our approach to correspondence depends critically on the hyperbolic picture: our use of integer coordinates depends on hyperbolic straightening, and our interpolation scheme (Figure 4.12) likewise relies crucially on the hyperboloid model of hyperbolic space.

4.2.1 Integer Coordinates for Ideal Hyperbolic Polyhedra

With a small tweak, we can use the integer coordinates data structure described in Chapter 3 to encode the correspondence between two triangulations T_1 and T_2 of the same ideal hyperbolic polyhedron. We still use normal coordinates and roundabouts to encode the sequence of triangles in T_2 that each edge of T_1 passes through. The only difference arises when we compute the exact path taken by an edge through this sequence of triangles (Section 3.2.2). Rather than laying out the sequence of triangles in the Euclidean plane and connecting the endpoints by a straight Euclidean line, we now lay the triangles out in the hyperbolic plane and connect the endpoints by a hyperbolic geodesic. In particular, we found that this calculation was easiest to do in the hyperboloid model of the hyperbolic plane, where vertices live on the so-called *light cone*.

Layout in the Light Cone The hyperboloid model of the hyperbolic plane identifies the hyperbolic plane with the unit hyperboloid $\{(x, y, z) : x^2 + y^2 - z^2 = -1\}$. This hyperboloid

asymptotically approaches the *light cone*[1] $\{(x, y, z) : x^2 + y^2 = z^2\}$, whose rays represent ideal points of the hyperboloic plane. In this model, we can identify ideal hyperbolic triangles with secant triangles whose vertices lie on the light cone (Figure 4.11). So once we have used Algorithm 2 to determine the sequence of triangles in T_2 than some edge ab of T_1 passes through, our next task is to embed those triangles as secant triangles in the light cone.

As derived in Appendix A.3, the first triangle aij has vertices

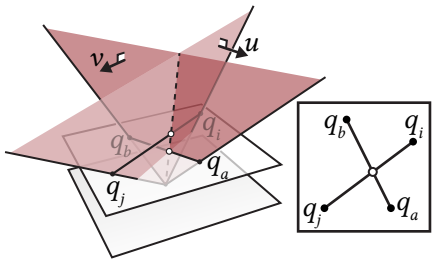
$$\begin{aligned} q_a &= \tfrac{2}{\sqrt{3}} \ell_{ai} \ell_{aj} \ell_{ij}^{-1} (1, 0, 1), \\ q_i &= \tfrac{2}{\sqrt{3}} \ell_{ai} \ell_{ij} \ell_{aj}^{-1} (\cos(2\pi/3), \sin(2\pi/3), 1), \\ q_j &= \tfrac{2}{\sqrt{3}} \ell_{aj} \ell_{ij} \ell_{ai}^{-1} (\cos(4\pi/3), \sin(4\pi/3), 1). \end{aligned} \tag{4.6}$$

For any triangle kjl following a known triangle ijk, we use the Ptolemy relation to get ℓ_{il}, then solve for the unknown position

$$q_l = \frac{\ell_{il}}{\ell_{ik} \ell_{ij}} \left(-\frac{\ell_{jl} \ell_{kl}}{\ell_{il}} q_i + \frac{\ell_{ik} \ell_{kl}}{\ell_{jk}} q_j + \frac{\ell_{jl} \ell_{ij}}{\ell_{jk}} q_k \right). \tag{4.7}$$

This process repeats until we have laid out the whole strip, including the endpoints q_a and q_b. To account for conformal scaling, we also scale just these endpoints by e^{-u_a} and e^{-u_b}, respectively.

We think of the embedded points $q_i \in \mathbb{R}^3$ as homogeneous coordinates representing points in the two-dimensional hyperbolic plane. So, for each edge ij crossed by edge ab, we can compute the intersection of ab with ij by computing the intersection between segment $q_a q_b$ and segment $q_i q_j$ in homogeneous coordinates. Explicitly, we solve for values $s, t, u \in \mathbb{R}$ such that



$$(1 - t) q_a + t q_b = e^u \left((1 - s) q_i + s q_j \right). \tag{4.8}$$

Letting $v := q_a \times q_b$ and $w := q_i \times q_j$, we can write the solution explicitly as

$$t = \frac{\langle w, q_a \rangle}{\langle w, q_a - q_b \rangle}, \quad s = \frac{\langle v, q_i \rangle}{\langle v, q_i - q_j \rangle}, \quad u = \log \left(\frac{\langle v, q_j - q_i \rangle}{\langle w, q_a - q_b \rangle} \right). \tag{4.9}$$

The barycentric coordinates s, t and scale factors u computed at each intersection are used to construct the common subdivision in Section 4.2.2 and for texture interpolation in Section 4.2.3.

[1] so-named because rays of the light cone represent directions that light can travel in spacetime in the study of special relativity.

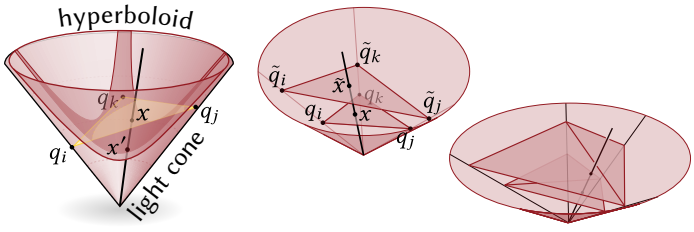


Figure 4.11: By drawing triangles in the light cone *(left)*, the map between surfaces can be found by drawing a straight line through the origin *(center)*, which also works for two different triangulations *(right)*.

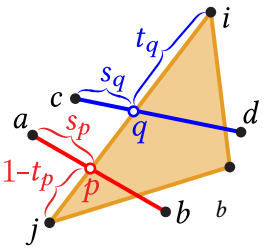
4.2.2 Common Subdivision of Three Triangulations

Following uniformization (Section 4.3), we end up with three triangulations: the input T^A with vertex positions f, its intrinsic Delaunay triangulation T^B, and the parameterized mesh T^C with texture coordinates z (Figure 4.1). We separately use one set of integer coordinates to track the correspondence between T^A and T^B (as Euclidean triangle meshes), and another set of integer coordinates to track the correspondence between T^B and T^C (as ideal hyperbolic polyhedra).

For most tasks, we need an explicit map between T^A and T^C. To construct one, we first use the integer coordinates to identify where the edges of T^A and T^C intersect edges of T^B (Section 3.2); these intersections are used to construct the common subdivision S of all three meshes, similar to Section 3.3. After constructing S, we interpolate f and z across its faces (Section 4.2.3), obtaining an extrinsic polygon mesh with positions f_i at vertices and texture coordinates z_i^{jk} at corners.

4.2.3 Interpolation

The vertex coordinates f_i and texture coordinates z_i^{jk} define piecewise-linear and piecewise-projective functions over the faces of T^A and T^C *resp.*; we now sample these functions onto S. To do so, we also use the scale factors u obtained while tracing hyperbolic geodesics. We process each triangle $ijk \in \mathsf{T}^B$ independently. First, we interpolate data onto each edge ij of the triangle. For each edge point p along an edge $ab \in \mathsf{E}^A$, let s_p, t_p be the barycentric coordinates along ab and ij, *resp.* Then $f_p = (1-s_p)f_a + s_p f_b$. Similarly, for a point q along $cd \in \mathsf{E}^C$ we have homogeneous texture coordinates



$$\hat{z}_q = e^{-u_q}\left((1-s_q)(z_c,1) + s_q(z_d,1)\right) \in \mathbb{R}^3, \tag{4.10}$$

where $(z,1)$ indicates that a 1 has been appended to z. The scale factors e^u arise from projective rather than linear interpolation, as discussed in Section 4.2.1. To get values of f at edge points q, and values of $\hat{z}$ at edge points p, we linearly interpolate between adjacent known values along ij. Finally, to get the values at each face point, we write the endpoints of the two incident fragments in 2D barycentric coordinates relative to ijk, and compute the intersection point in homogeneous coordinates via Equation (4.9). The resulting s, t values are then used to linearly interpolate f and $\hat{z}$ from the segment endpoints. Note that since texture coordinates are discontinuous across cuts, we store $\hat{z}$ at corners rather than vertices. The final surface can be visualized by tessellating polygons into triangles; just as in [Springborn et al. 2008, Section 3.4] we perform a homogeneous divide on texture coordinates $\hat{z}$ at each sample point (*e.g.*, each pixel).

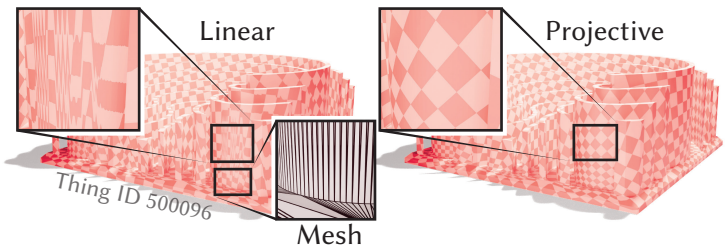


Figure 4.12: For meshes with low-quality triangles, standard linear interpolation yields a poor conformal map *(left)*. We describe how to perform projective interpolation across triangulations, yielding a dramatically smoother map *(right)*.

4.3 Planar Parameterization

Now we describe our procedure for planar parameterization (Figure 4.1)—see Section 4.4 for the spherical case. Given an input mesh T^A, we first flip to an intrinsic Delaunay triangulation T^B, which preserves the Euclidean geometry and defines the discrete conformal structure. We then solve an optimization problem for scale factors u that transform T^B into a triangulation T^C with the prescribed angle defects (Section 4.3.3). After optimization, we lay T^C out in the plane (Section 4.3.5), and pull this layout back to the input mesh as described in Section 4.2.3.

4.3.1 Variational Formulation

The input to our discrete uniformization procedure is the intrinsic Delaunay triangulation T^B, and target angle defects $\Omega^* : \mathsf{V} \to \mathbb{R}$ which must satisfy a discrete Gauss-Bonnet condition:

$$\tfrac{1}{2\pi} \sum_{i \in \mathsf{V}} \Omega_i^* = |\mathsf{V}| - |\mathsf{E}^B| + |\mathsf{F}^B| \tag{4.11}$$

(see Section 4.3.4 for a generalization to surfaces with boundary). Note that target defects Ω_i^* must be smaller than 2π, since the sum of angles around a vertex is always positive. Minimizing a convex energy $\mathcal{E}$ then yields scale factors u relative to T^B. Unlike CETM we flip to Delaunay whenever we need to evaluate the energy or its derivatives (see Section 4.3.2). This process is completely hidden inside a callback routine—from the perspective of the optimizer, one simply has to solve an unconstrained problem that is convex and twice continuously differentiable (C^2).

4.3.2 Energy Evaluation

To evaluate our energy for any given u, we first compute the edge lengths $\tilde{\ell}_{ij} = e^{(u_i+u_j)/2} \ell_{ij}^B$, and flip to the corresponding ideal Delaunay triangulation $\widetilde{T} = (\mathsf{V}, \widetilde{\mathsf{E}}, \widetilde{\mathsf{F}})$ via Ptolemy flips. These flips change the Euclidean geometry but preserve the discrete conformal structure. We use $\tilde{\lambda}$, $\tilde{\theta}$, and $\widetilde{\Omega}$ to denote the corresponding Penner coordinates, interior angles, and angle defects, *resp.*

Energy The discrete conformal energy is then given by

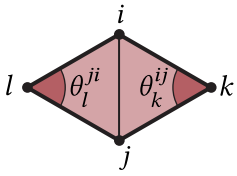
$$\mathcal{E}(u) = \sum_{i \in \mathsf{V}} (2\pi - \Omega_i^*)\, u_i - \sum_{ij \in \widetilde{\mathsf{E}}} \pi \tilde{\lambda}_{ij} + \sum_{ijk \in \widetilde{\mathsf{F}}} 2f(\tilde{\lambda}_{ij}, \tilde{\lambda}_{jk}, \tilde{\lambda}_{ki}), \tag{4.12}$$

where $f(\tilde{\lambda}_{ij}, \tilde{\lambda}_{jk}, \tilde{\lambda}_{ki}) := \frac{1}{2}\left(\tilde{\theta}_i^{jk} \tilde{\lambda}_{jk} + \tilde{\theta}_j^{ki} \tilde{\lambda}_{ki} + \tilde{\theta}_k^{ij} \tilde{\lambda}_{ij}\right) + \text{Л}(\tilde{\theta}_i^{jk}) + \text{Л}(\tilde{\theta}_j^{ki}) + \text{Л}(\tilde{\theta}_k^{ij})$. Here Л denotes *Milnor's Lobachevsky function* $\text{Л}(\theta) := -\int_0^\theta \log|2 \sin u|\, du$, and is related to *Clausen's integral* via $\text{Л}(\theta) = \frac{1}{2}\text{Cl}_2(2\theta)$, which is implemented in standard numerical packages [Galassi et al. 1994].

Gradient At each vertex $i \in \mathsf{V}$, the gradient of the energy is

$$\partial_{u_i} \mathcal{E} = \widetilde{\Omega}_i - \Omega_i^* \tag{4.13}$$

Note, then, that any stationary point $\partial_u \mathcal{E} = 0$ achieves the desired angle defects $\widetilde{\Omega} = \Omega^*$.



Hessian The Hessian is given by the positive-semidefinite *cotan Laplacian* $L \in \mathbb{R}^{\mathsf{V}\times\mathsf{V}}$ [MacNeal 1949, Section 3.2; Crane et al. 2013a, Chapter 6]. Since a Δ complex may contain more than one edge with the same endpoints, the off-diagonal entries L_{ij} and L_{ji} are obtained by summing the values $\frac{1}{2}(\cot\theta_k^{ij} + \cot\theta_l^{ji})$ over all edges $ij \in \widetilde{\mathsf{E}}$ with endpoints i and j, where k, l are the vertices opposite the edge. For each vertex $i \in \mathsf{V}$, we then have a diagonal entry $L_{ii} = -\sum_{ij\in\widetilde{\mathsf{E}}} L_{ij}$, where the sum is taken over all edges incident on i. Note that self-edges (where $i = j$) make no contribution.

4.3.3 Optimization

Since the energy $\mathcal{E}$ is convex and globally C^2, it can be minimized using any standard method for convex optimization. We use Newton's method with backtracking line search, as described in Algorithms 9.5 and 9.2 of Boyd & Vandenberghe [2004], *resp.* In particular, we use the descent direction $v \in \mathbb{R}^{\mathsf{V}}$ obtained by solving the linear system

$$Lv = \partial_u \mathcal{E}, \tag{4.14}$$

where $\partial_u \mathcal{E} \in \mathbb{R}^{\mathsf{V}}$ encodes the gradient defined in Section 4.3.2. Note that the matrix L has a one-dimensional kernel of constant vectors—corresponding to global scaling. We use the solution v with mean zero. Although L is rank deficient, the system is solvable: Gauss-Bonnet ensures that the right-hand side sums to zero. And since the energy is convex, initialization does not affect the final result (apart from a global scale)—we initialize each vertex with $u_i = 0$.

4.3.4 Surfaces with Boundary

For a smooth surface M with boundary ∂M, the space of conformal maps to the plane is parameterized by a real-valued function along the boundary—geometrically, this function can be determined by prescribing either the scale factors u or the curvature density $\kappa\, ds$ along ∂M (see [Sawhney & Crane 2017, Section 4.2] for further discussion). We can specify such conditions by either a scale factor u_i or target exterior angle κ_i^* at each boundary vertex $i \in \partial\mathsf{V}$. To enforce these conditions, we glue together two copies of the input mesh along the boundary (as in Jin et al. [2004]), reducing the problem to the no-boundary case. Unlike CETM, we can hence always find a solution with the prescribed boundary data. Note that this construction extends Springborn [2019], which does not consider surfaces with boundary; Sun et al. [2015] describe a similar scheme in the case of prescribed boundary curvature. Maps to the circular disk are handled in a similar fashion, but using spherical uniformization.

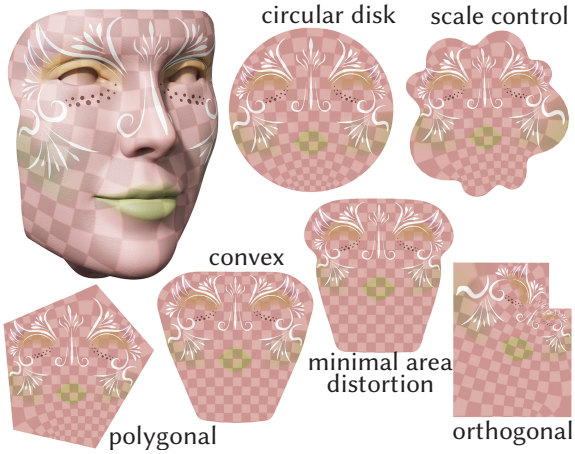
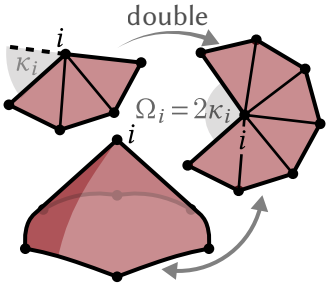


Figure 4.13: Our algorithm guarantees existence of a locally injective discrete conformal map for any prescribed boundary lengths or angles, which can be used to achieve a rich variety of behavior.



Fixed Boundary Curvature Suppose we want our flattened domain to have an exterior angle κ_i^* at a boundary vertex i, corresponding to an angle sum of $\pi - \kappa_i^*$. On the doubled domain, we thus prescribe an angle defect $\Omega_i^* = 2\pi - 2(\pi - \kappa^*) = 2\kappa_i^*$. The intrinsic parameterization which we compute is unique, so in particular it must be symmetric across the two copies of the original mesh. Hence, if we cut the uniformized surface along the original boundary curve, each half will exhibit the desired angles κ^*. The only requirement is that the angle defects and exterior angles satisfy a Gauss-Bonnet condition $\sum_{i \in \mathsf{V}} \Omega_i^* + \sum_{i \in \partial \mathsf{V}} \kappa_i^* = |\mathsf{V}| - |\mathsf{E}| + |\mathsf{F}|$. In Figure 4.13 we assign target angles that yield convex ($\kappa_i^* > 0$), orthogonal ($\kappa_i^* \in \frac{\pi}{2}\mathbb{Z}$), or polygonal boundaries ($\kappa_i^* = 0$ almost everywhere).

Fixed Boundary Scale Factors To prescribe boundary scale factors, we fix the values u_i at vertices i of the doubled domain corresponding to the original boundary. For instance, setting $u_i = 0$ at all boundary vertices yields minimal area distortion [Chebyshev 1899, p. 242] in the sense that it minimizes the variation in scale factors [Springborn et al. 2008, Appendix E]—see Figure 4.13. Fixing these values restricts the convex energy $\mathcal{E}$ to a linear subspace; hence we are still solving a convex problem. To compute the descent direction, we now solve the same system (Equation (4.14)), except that we set zero Dirichlet boundary conditions at the boundary vertices, since we do not want these values to change. The minimizer will exhibit the target angle defects at interior vertices, since the gradient still only vanishes when $\widetilde{\Omega} = \Omega^*$.

4.3.5 Planar Layout

The final scale factors u provide an intrinsic description of the flattened surface, which we can lay out in the plane. We first scale the edge lengths and flip to Delaunay with Ptolemy edge flips to get the final triangulation (T^C, ℓ^C). This triangulation is flat away from cone singularities,so we can simply lay the triangles out in the plane one at a time to get a parameterization with no flipped triangles. (See the paper for numerically robust alternatives.) We store texture coordinates $z_i^{jk} \in \mathbb{R}^2$ at corners to allow for discontinuities across cuts.

4.4 Spherical Parameterization

We now consider conformal maps to the sphere S^2. Given a genus-0 Delaunay triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ with edge lengths $\ell : \mathsf{E}^B \to \mathbb{R}_{>0}$, we give an algorithm to compute vertex positions $f : \mathsf{V} \to S^2 \subset \mathbb{R}^3$ describing a discretely conformally equivalent convex sphere-inscribed polyhedron. The solution is guaranteed to exist, and be unique up to Möbius transformations of the sphere—the same existence and uniqueness exhibited by the smooth uniformization theorem.

The strategy used by CETM is essentially to delete the neighborhood of a special vertex i^*, conformally map this modified surface to a flat disk, and apply stereographic projection to the sphere, where the removed vertex i^* is re-inserted (Figure 4.14, *left*). For a fixed triangulation, there are several problems. First, as discussed previously, a discretely conformally equivalent flat disk with this particular triangulation may not exist. And even if we try to allow the triangulation to vary, it is not immediately clear what to do about boundary edges (which cannot be flipped).

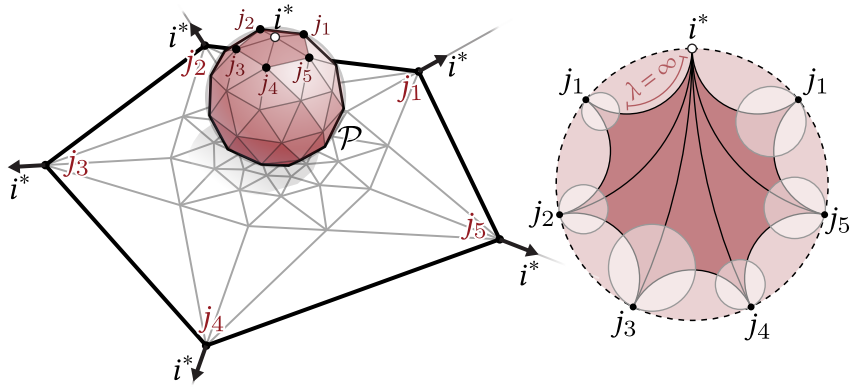


Figure 4.14: *Left:* a convex polyhedron inscribed in the sphere can also be viewed, via stereographic projection, as a planar Delaunay triangulation with all boundary vertices connected to a vertex i^* at infinity. *Right:* in the Poincaré model, the horocycle at i^* shrinks to a point, and the incident Penner coordinates λ_{i^*j} go to infinity.

Second, the final polyhedron may not be convex. In fact, many combinatorial triangulations do not admit *any* convex embedding in the sphere—conformal or otherwise [Rivin 1996]. Third, the map may become non-injective when vertex i^* is re-inserted.

Imagine that we instead start with the object we want: a convex sphere-inscribed polyhedron $\mathcal{P}$ conformally equivalent to the input surface. If we stereographically project this polyhedron to the plane from a vertex i^*, we get a planar disk whose boundary vertices are all connected to the same vertex i^* at infinity (Figure 4.14, *left*). Stereographic projection preserves discrete conformal equivalence, and since the polyhedron is convex, its stereographic projection will be a planar Delaunay triangulation with convex boundary [Brown 1979]. So if we can find such a triangulation, we can obtain the desired spherical conformal map via stereographic projection.

To solve this problem, Springborn [2019] reformulates it in the hyperbolic setting where one can freely flip edges without invalidating the hyperbolic metric. Here, the Penner coordinates $\lambda_{i^*j} = 2 \log \ell_{i^*j}$ incident on the special vertex i^* are now infinite—effectively pushing the horocycle at i^* off to infinity (Figure 4.14, *right*). This decorated polyhedron can be found via the same uniformization procedure as in Section 4.3, but with a few important modifications. For one, it uses a modified Delaunay flipping procedure which accounts for edges of infinite length (Section 4.4.1), and a modified energy which accounts for the boundary vertices j adjacent to i^* (Section 4.4.2). Linear inequality constraints on u ensure that the edges i^*j are convex and have the right cross ratios (Section 4.4.3). Solving a bounds-constrained optimization problem (Section 4.3.3) yields scale factors u that describe the desired planar disk, which we can then stereographically project back onto the sphere.

4.4.1 Modified Delaunay Flips

Since some Penner coordinates are now infinite (Figure 4.14, *right*), we can no longer check the Delaunay condition using Equation (4.5). However, we can still express the ideal Delaunay condition in terms of the horocyclic arc lengths α_i^{jk} (Appendix A.2.3). Initially, all edge lengths ℓ are well-defined and we have

$$\alpha_i^{jk} = \frac{\ell_{jk}}{\ell_{ki}\ell_{ij}}. \tag{4.15}$$

Scaling lengths *à la* Equation (4.1) then gives new arc lengths

$$\tilde{\alpha}_i^{jk} = e^{-u_i} \alpha_i^{jk}.$$

At the special vertex i^*, where $u_{i^*} = \infty$, we hence get $\tilde{\alpha}_i^{jk} = 0$ as expected. An edge ij then satisfies the ideal Delaunay condition if

$$\tilde{\alpha}_k^{ji} + \tilde{\alpha}_l^{ij} < \tilde{\alpha}_i^{jk} + \tilde{\alpha}_i^{lj} + \tilde{\alpha}_j^{ik} + \tilde{\alpha}_j^{li}. \tag{4.16}$$

If this condition is not satisfied, we perform a Ptolemy flip (Equation (4.4)). However, rather than compute $\tilde{\ell}_{kl}$ directly (which may be infinite), we first compute ℓ_{kl} via the Ptolemy relation and then scale to get $\tilde{\ell}_{kl}$. Importantly, if Equation (4.16) is satisfied with equality for an edge kl opposite the special vertex i^*, we also flip kl—since for any sequence of finite horocycles around i^* approaching infinity, this edge that would belong to the ideal Delaunay triangulation.

4.4.2 Spherical Variational Principle

As in the Euclidean case, the energy and its derivatives are always evaluated on the triangulation $\widetilde{T}$ obtained by updating the Penner coordinates of T^B (*à la* Equation (A.3)) and flipping to an ideal Delaunay triangulation (*à la* Section 4.4.1). In particular, let $\mathsf{T}^\circ := (\mathsf{V}^\circ, \mathsf{E}^\circ, \mathsf{F}^\circ)$ be the mesh obtained by removing the special vertex i^* together with its incident edges and faces from $\widetilde{\mathsf{T}}$. The energy for spherical uniformization is then

$$\mathcal{E}_{S^2}(u) = 2\pi \sum_{i \in \mathsf{V}^\circ} u_i - \pi \sum_{ij \in \mathsf{E}^\circ} \tilde{\lambda}_{ij} + \sum_{ijk \in \mathsf{F}^\circ} 2f(\tilde{\lambda}_{ij}, \tilde{\lambda}_{jk}, \tilde{\lambda}_{ki})$$

(see Springborn [2019, Equation 56 and Theorem 7.18], which differs by a constant that does not affect minimizers). For each vertex $i \in \mathsf{V}^\circ$, its gradient is

$$\partial_{u_j} \mathcal{E}_{S^2} = \widetilde{\Omega}_j + \pi(\deg_{\mathsf{F}^\circ}(j) - \deg_{\mathsf{E}^\circ}(j)),$$

where $\deg_{\mathsf{E}^\circ}(j)$ and $\deg_{\mathsf{F}^\circ}(j)$ are the number of edges and faces of T° containing j, *resp.*; this degree difference will be -1 for vertices adjacent to V° (and zero otherwise). Ω^* does not appear because we do not consider cone singularities in the spherical case. The Hessian is again the cotan-Laplacian, where cotangents from any removed face are set to zero.

4.4.3 Constraints

In the fixed triangulation case, Bobenko et al. [2015, Proposition 3.2.1] observe that setting $u_j = -\lambda_{i^*j}$ ensures that the boundary edges i^*j exhibit the right length cross ratio. However, in the variable triangulation case we do not know *a priori* which vertices j will end up adjacent to the removed vertex i^* (since this set may change due to edge flips). Instead, as proposed by Springborn [2019], we impose the inequality constraint $u_j \geq -\lambda_{i^*j}$ for *all* vertices $j \in \mathsf{V}^\circ$, where λ_{i^*j} is the geodesic distance between horocycles in the input triangulation. At a minimizer, these inequalities will be satisfied with equality for vertices j adjacent to i^*.

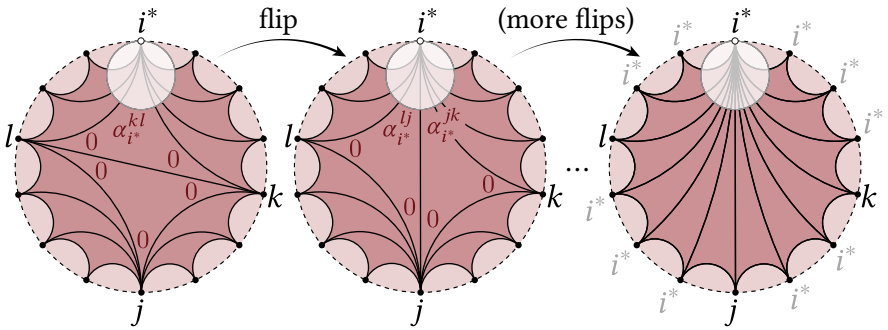


Figure 4.15: To find a triangulation connecting vertex i^* to all other vertices j, we put a finite horocycle at i^* and send all other horocycles to infinity. Modified Delaunay flips then yield the desired triangulation.

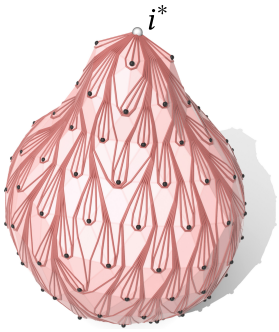


Figure 4.16: Peacock triangulation.

To compute the geodesic distances, we first construct a triangulation that connects i^* to all other vertices $j \in \mathsf{V}^\circ$ by minimal geodesics. To do so, we send all the horocycles *except* the one at i^* to infinity—in the Poincaré model, the representative circles shrink down to points (Figure 4.15, *left*). In general, an edge connecting two vertices $j_1, j_2 \neq i^*$ cannot be Delaunay, since the horocyclic arc length α at both vertices will be zero—hence smaller than the arc length of the complementary vertices (see Figure 4.15 and Equation (A.10)). By flipping to a Delaunay triangulation, we ensure that any edge leaving a vertex $j \neq i^*$ connects only to i^* (Figure 4.15, *right*). Moreover one can show that, due to the global Delaunay property, every such edge is a minimal geodesic [Springborn 2019, Proposition 5.16]. All other edges go from i^* back to i^*, resulting in what we call a *peacock triangulation* (Figure 4.16). To get the values λ_{i^*j}, we then read off the distances between the original horocycles (rather than those that have been scaled down to points).

4.4.4 Optimization

Once we know λ_{i^*j} for each j, we can solve the following convex optimization problem to find our desired scale factors:

$$\begin{aligned} \min_{u:\mathsf{V}^\circ\to\mathbb{R}} \quad & \mathcal{E}_{S^2}(u) \\ \text{s.t.} \quad & u_j \geq -\lambda_{i^*j}, \qquad \forall j \in \mathsf{V}^\circ. \end{aligned}$$

This problem can be solved via bounds-constrained Newton's method with backtracking line search. We use the implementation found in the *PETsc/TAO* library [Balay et al. 2019; 1997; Munson et al. 2014]—specifically, the `TAOBNLS` solver [Benson et al. 2003, Section 4.2.1].

4.4.5 Spherical Layout

After optimization, we have scale factors u at vertices that describe a flat metric on the topological disk T°. We lay this disk out in the plane *à la* Section 4.3.5, then stereographically project to get coordinates z on the unit sphere $S^2 \subset \mathbb{R}^3$ (re-inserting the special vertex i^* at the center of stereographic projection). This final map is unique only up to Möbius transformations of the

sphere; we compute a canonical Möbius transformation via Baden et al. [2018, Algorithm 1], using vertex rather than face areas to express the center of mass.

4.4.6 Spherical Interpolation

Interpolation is done as in Section 4.2.3, but we now interpolate positions on the sphere using homogeneous coordinates $\hat{z} \in \mathbb{R}^4$, and we must account for both uniformization and stereographic projection in our scale factors. If we let $\tilde{\ell}_{ij}$ be the edge lengths of $\mathcal{P}$, and ℓ_{ij} be the lengths from T^B after applying the same sequence of Ptolemy flips used for uniformization, then solving Equation (4.1) within each triangle ijk yields

$$u_i = \log\left(\frac{\tilde{\ell}_{ij}}{\ell_{ij}}\frac{\ell_{jk}}{\tilde{\ell}_{jk}}\frac{\tilde{\ell}_{ki}}{\ell_{ki}}\right)$$

(and similarly for u_j, u_k). Since stereographic projection preserves discrete conformal equivalence, these values agree across triangles. Also, since $\mathcal{P}$ is convex, normalizing interpolated coordinates gives an injective map to the unit sphere (for, *e.g.*, texture mapping).

4.5 Parameterization Results

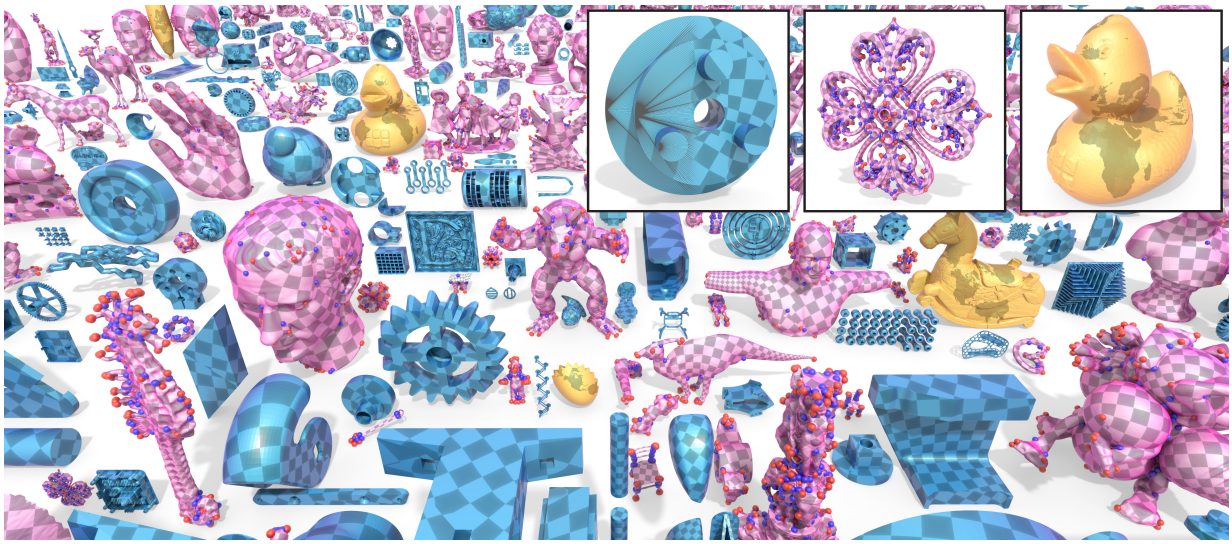


Figure 4.17: Our method computes locally injective, discretely conformal maps even for near-degenerate triangulations *(turquoise meshes)* and extremely difficult configurations of cone singularities *(magenta meshes)*. We also compute globally bijective conformal maps to the sphere *(yellow meshes)*.

This section evaluates the empirical behavior of our method, here referred to as *conformal equivalence of polyhedral surfaces (CEPS)*. Our main points of comparison is the CETM algorithm of Springborn et al. [2008], which does not use flips. All methods use identical code for tracking correspondence, *à la* Section 3.1. The overall observation is that CEPS succeeds on far more models than CETM. Even when CETM does succeed, it may not provide as good of an approximation of a smooth conformal map (Figure 4.18, *top*).

4.5.1 Planar Parameterization Results

Difficult Cone Configurations. We ran our method on the standard MPZ benchmark of Myles et al. [2014], containing challenging cone configurations. CEPS succeeds on all 114 models, including extraction of the common subdivision. Maps were discretely conformal up to floating point error, with an average length cross ratio error of about 10^{-9}, and no worse than about 10^{-4}. In contrast, CETM succeeded on only 73 models (Figure 4.18, *middle*).

Many injective but non-conformal methods do not do as well on this difficult benchmark: as reported by Bright et al. [2017, Section 8.1], their method and the methods of Chien et al. [2016], Aigerman et al. [2014], Levi & Zorin [2014], and Lipman [2012] succeed on 104, 102, 97, 93, and 90 models, *resp.* Many of these methods have running times on the order of minutes or (on the most difficult examples) hours, versus seconds for our method. On the other hand, we must change/refine the triangulation, whereas these methods keep the triangulation fixed. Like CEPS, the combinatorial method of Zhou et al. [2020] succeeds on all MPZ models, but can yield highly distorted maps that are expensive to optimize; cost is again on the order of minutes to hours.

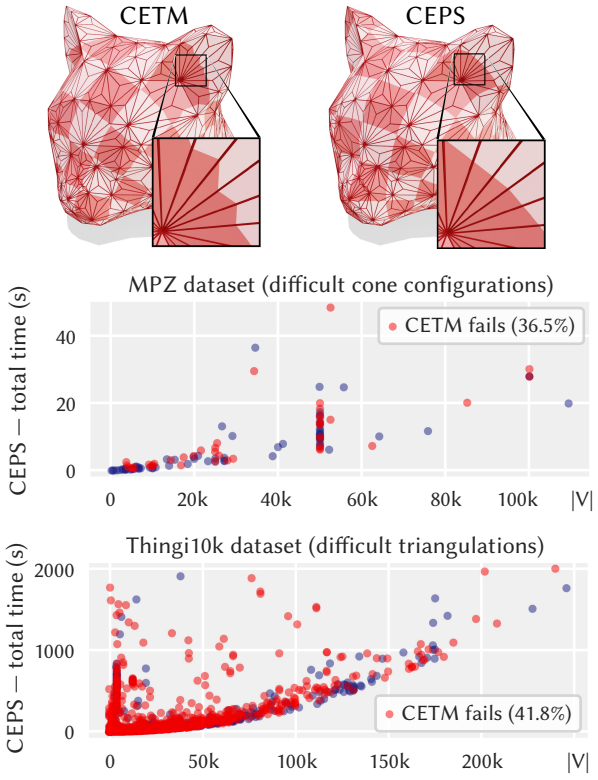


Figure 4.18: *Top*: Even when CETM succeeds, the quality of the map may be lower since it uses a different notion of conformal equivalence (based on the input rather than Delaunay triangulation). *Bottom*: Timings for our method (CEPS) on two datasets. Note that CETM fails on a large percentage of models where we succeed (highlighted in red).

Difficult Triangulations. As a stress test of floating-point behavior, we parameterized all manifold meshes from Thingi10k, splitting disconnected meshes into their connected components (32,744 examples in total), and using a time out of 2000 seconds. Note that previous work on cone parameterization does not even attempt this benchmark, which has dramatically worse element quality than MPZ. For these examples we apply the greedy cone placement strategy from Springborn et al. [2008, Section 5.1], stopping when all log scale factors u_i are in the range $[-5, 5]$ (*i.e.*, a max scale factor of about 150). Here CEPS successfully computes a parameterized mesh S for 98.6% of models, yielding an injective map on 97.7%.

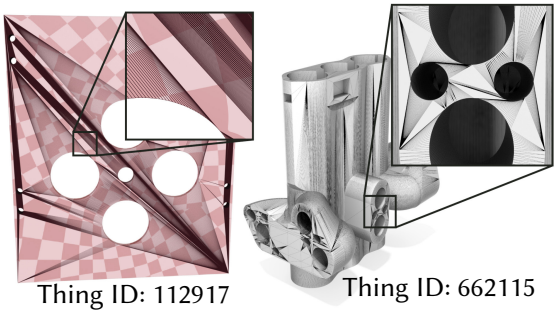


Figure 4.19: Our implementation robustly handles extremely poor triangulations (left) failing only on the most pathological inputs (right).

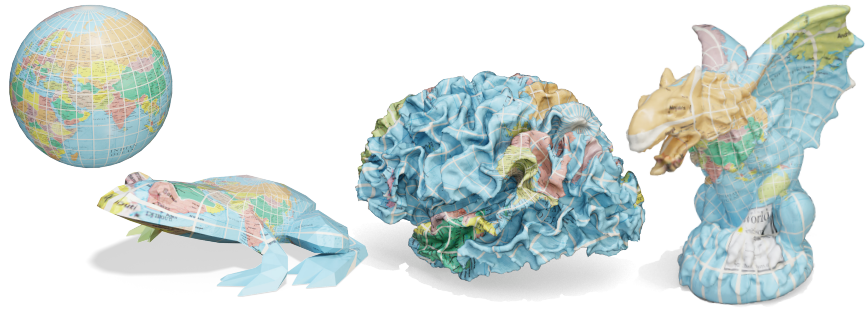


Figure 4.21: In the genus-0 case, our method guarantees a bijective discrete conformal map to a convex polyhedron with vertices on the sphere.

Examples where we fail are quite pathological (*e.g.*, Figure 4.19, *right*). Overall about 68% and 15% of failures were due to failure of iterative straightening or optimization (*resp.*) to converge within the time limit, and for about 13% Delaunay flipping failed due to floating point error. The worst cross ratio error was typically around 10^{-5}. CETM fails on almost half of these examples (Figure 4.18).

4.5.2 Spherical Parameterization Results

We ran our spherical algorithm on two other datasets: the *Spherical Demon* brain scan dataset of Yeo et al. [2009], and the anatomical surface dataset of Boyer et al. [2011] (Figure 4.21). On the brain dataset, where each model has about 230k faces, we obtained injective discrete conformal maps to the sphere on all 78 brain hemispheres, taking an average of 493 seconds per hemisphere. The anatomical surface models are topological disks, so we compute conformal maps to a hemisphere. We succeed in computing these maps on all 187 of the manifold meshes without handles in the dataset. One of our maps contains degenerate triangles, but none have flipped triangles. The models take an average of 14.4 seconds to uniformize.

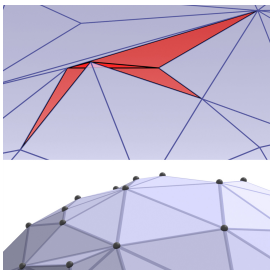


Figure 4.20: Existing spherical methods often exhibit foldover, and do not guarantee convexity.

Past methods for spherical conformal parameterization largely compute maps to the sphere that are harmonic with respect to piecewise linear Dirichlet energy [Gu et al. 2004]. However, unless the input mesh is already Delaunay, such maps can have flipped triangles (Figure 4.20, *right*). More fundamentally, it is impossible for any method that uses a fixed triangulation to guarantee convexity—no matter what algorithm is used, or where the vertices are placed—since not all combinatorial triangulations admit a convex embedding in the sphere [Rivin 1996]. In practice, flipped triangles and nonconvex edges are quite common in discrete harmonic maps: on the brain dataset we observed, on average, foldover at about 100 edges and nonconvexity at about 20k edges when using the method of Kazhdan et al. [2012] (see Figure 4.20). Other techniques for spherical conformal mapping gave very similar results [Crane et al. 2013b; Dym et al. 2019].

4.5.3 Performance & Complexity

Our implementation[2] runs in a matter of seconds on high-quality input meshes, and can take a few minutes to parameterize near-degenerate input triangulations (Figure 4.18). However, we optimized mostly for robustness rather than performance, so there are still several parts of the algorithm whose performance has room for asymptotic improvement.

On reasonably-triangulated meshes, the cost of computing discrete conformal parameterizations is dominated by the convex optimization problem used to compute the uniformizing scale factors u_i (Figure 4.22, *left*). Solving a convex optimization problem using Newton's method with line search almost always converges with five or six iterations, so the cost essentially reduces to the cost of a single step of optimization [Boyd & Vandenberghe 2004, Section 9.5]. Each step of optimization requires us to construct and solve a sparse $|V| \times |V|$ linear system. Since our matrix is a Laplacian, one can—in theory—solve such systems in time near $O(|E| \log^2 |V|)$ [Koutis et al. 2014], although in practice we use a sparse Cholesky factorization. The exact complexity of sparse Cholesky factorization on a general triangle mesh is hard to analyze, but it is known to require requires $O(|V|^{3/2})$ time and $O(|V| \log |V|)$ space on regular grids [George 1973].

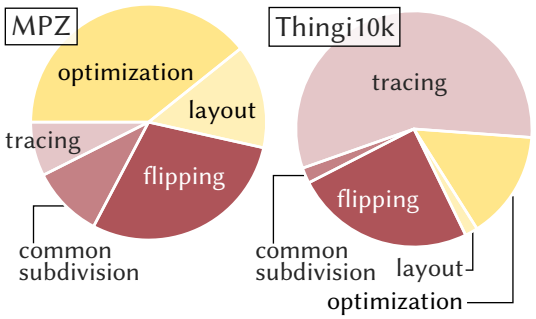
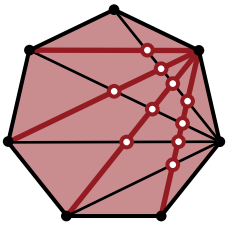


Figure 4.22: Average breakdown of costs in CEPS on different datasets; layout and optimization steps are shared by CETM.

On poorly-triangulated meshes, the cost of our algorithm is dominated by the cost of tracing geodesics (Figure 4.22, *right*). At the moment, we use a naïve scheme which scales linearly with the number of intersections between the two triangulations, which can become quadratic—or worse—in some cases, such as the polygon depicted in the inset. One could instead use asymptotically-faster tracing schemes, such as the normal-coordinate-based algorithm of Erickson & Nayyeri [2013], but we leave such explorations to future work. And more basically, even the current tracing and refinement steps of CEPS could be trivially parallelized over edges and faces, *resp.*



[2] timings measured on an Intel i9-9980XE with 32 GB of RAM, using a single thread.

Chapter 5

Surface Simplification

I would have written a shorter letter, but I did not have the time.
Blaise Pascal, *Les Provinciales* [1657]

We now turn our attention to another instance of time-evolving surfaces: surface simplification. We seek to take in a surface mesh, and produce progressively simpler approximations which capture its geometry. Whereas past simplification methods focus on visual appearance, our goal is to solve equations on the surface. Hence, rather than approximate the extrinsic geometry, we construct a coarse *intrinsic triangulation* which approximates the input domain. In the spirit of the *quadric error metric (QEM)* of Garland & Heckbert [1997], we perform greedy decimation while agglomerating global information about approximation error. In lieu of extrinsic quadrics, however, we store intrinsic tangent vectors that track how far curvature "drifts" as the surface evolves during simplification. This process also yields a bijective map between the fine and coarse mesh, and prolongation operators for both scalar- and vector-valued data. Moreover, we obtain hard guarantees on element quality via intrinsic retriangulation—a feature unique to the intrinsic setting. The overall payoff is a "black box" approach to geometry processing, which decouples mesh resolution from the size of matrices used to solve equations.

Problem Statement

Suppose we start with a high-resolution manifold triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ equipped with edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$. Our goal in this chapter is to construct a small intrinsic approximation to T. Explicitly, we seek to construct a second intrinsic triangulation $\tilde{\mathsf{T}} = \left(\tilde{\mathsf{V}}, \tilde{\mathsf{E}}, \tilde{\mathsf{F}}\right)$ and edge lengths $\tilde{\ell} : \tilde{\mathsf{E}} \to \mathbb{R}_{>0}$, along with a bijective correspondence function $f : \tilde{\mathsf{T}} \to \mathsf{T}$ mapping points from $\tilde{\mathsf{T}}$ to T (and vice versa). $\tilde{\mathsf{T}}$ should be small, in the sense that that $\left|\tilde{\mathsf{V}}\right| \ll |\mathsf{V}|$, but it should also be a good approximation, in the sense that f should have low distortion (*e.g.* not significantly changing the distances between points).

We break the problem of intrinsic simplification into two pieces: we have an atomic simplification operation (vertex removal), which we can use to reduce the size of the mesh, and we have an estimate of distortion (intrinsic curvature error), which we build up over time and use to decide where to simplify the mesh. We first describe our procedure for intrinsic vertex removal in Section 5.1, and discuss the work required to track the correspondence between this evolving surface and our original mesh in Section 5.2, before describing our intrinsic curvature error metric in Section 5.3 and showing some results in Section 5.5.

5.1 Intrinsic Vertex Removal

Extrinsic simplification methods reduce vertex count by making local changes to the mesh connectivity [Schroeder et al. 1992; Hoppe 1996; Garland & Heckbert 1997]. We extend local simplification to the intrinsic setting, using vertex removal as our atomic simplification operation.

In the extrinsic setting, simplification algorithms often use other atomic operations such as edge collapses instead of plain vertex removal. Edge collapses, which merge two adjacent vertices into a single new vertex, have the advantage of offering continuous degrees of freedom, which can be optimized to decrease distortion: one can often find a good location for the new vertex which well-approximates the original mesh. By contrast, there are very few degrees of freedom available when removing a vertex extrinsically: the only choice is how to triangulate the hole left behind by deleting the vertex.

But in the intrinsic setting, even vertex removal offers plenty of degrees of freedom. The key idea behind our intrinsic removal operation is that intrinsically, removing a vertex amounts to a small parameterization problem—we wish to take an intrinsically-curved vertex, and replace it with an intrinsically-flat patch instead. And we have good algorithms for computing parameterizations with low distortion .

Our method removes a vertex i in three steps:

1. Intrinsically flatten i (Section 5.1.1).
2. Remove i from the triangulation (Section 5.1.2).
3. Flip to an intrinsic Delaunay triangulation (Section 2.3.3).

Note that all changes to the geometry occur in the first step, redistributing the curvature at i to neighboring vertices j. The second step merely retriangulates a flat region, and the third step performs only intrinsic edge flips. Hence, when measuring distortion in Section 5.3 will need only consider the first (flattening) step to prioritize vertex removals. Maintaining a Delaunay triangulation at each step helps ensure numerical robustness throughout simplification.

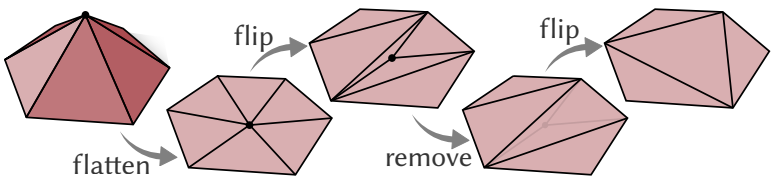


Figure 5.1: We remove an interior vertex by intrinsically flattening it, flipping to degree 3, removing it from the mesh, then flipping back to intrinsic Delaunay.

5.1.1 Vertex Flattening

We first eliminate all curvature at vertex i. For this operation to remain local and valid we must bijectively flatten the one-ring of vertex i, while keeping edge lengths along the boundary of this region fixed. Extrinsic flattening schemes can fix boundary vertices, but it is unclear how to construct the least-distorting boundary polygon with prescribed lengths. In contrast, the CETM algorithm of Springborn et al. [2008] supports edge length constraints, and operates directly on an intrinsic triangulation. More details on CETM can be found in Chapter 4.

For some vertices, this procedure may fail to find a valid parameterization. In this case, we simply skip removing this vertex and move on to remove another vertex instead.

5.1.2 Flat Vertex Removal

To remove an intrinsically-flat vertex i from the mesh, we follow the strategy described in Section 3.5.2: we perform edge flips until i has degree three, at which point it can be removed from the mesh and replace by a new face. In the rare case where our flipping heuristic gets stuck, producing a vertex with $\deg(i) \neq 3$ while no flippable edges remain, we skip this vertex removal and revert the mesh to its previous state.

After removal, we must also update the angular coordinates φ_{jk} and corresponding edge vectors e_{jk} for each edge jk with endpoints adjacent to i (Section 2.3.2). We then flip the mesh to an intrinsic Delaunay triangulation, *à la* Section 2.3.3.

Special cases. We cannot remove vertices i incident on a boundary self-edge, since every boundary loop must contain at least one vertex. Likewise, vertices i of self-faces (*i.e.*, triangles with only a single distinct vertex) can cause trouble for flipping, and are skipped.

5.2 Correspondence Tracking

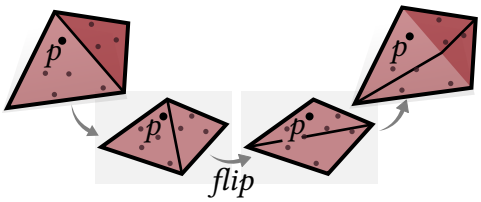
As usual, we often want to track the correspondence between our intrinsic triangulation and the input mesh while performing simplification. However, since we change the intrinsic geometry by removing intrinsically-curved vertices, we can no longer use any existing data structures to encode this correspondence. But rather than trying to these data structures to our new setting, we opt for a simpler strategy: we maintain a list of all of the local operations performed during simplification (namely: edge flips, vertex flattenings, and vertex removals), and describe how each operation maps points on one mesh to points on the other. Then, given any point p on the fine mesh, we can "re-play" these operations until we obtain the corresponding point $\tilde{p}$ on the simplified mesh, or vice versa, *à la* Liu et al. [2020]. Once we can map points back and forth, it is simple to extend the procedure to map edges or functions between the two triangulations.

5.2.1 Mapping Points

To map any point p on the fine mesh to a point $\tilde{p}$ on the coarse mesh, we track its barycentric coordinates through each local simplification operation. This map is trivially bijective, since at each step we simply re-write the given barycentric coordinates with respect to a different

triangulation of the same planar region. The only way to violate bijectivity would be to perform a non-bijective vertex flattening—which we explicitly forbid.

1. *Edge flips*: To track a point p through an intrinsic edge flip, we unfold the two adjacent triangles into the plane using the formulas provided by Sharp et al. [2021, Section 2.3.7], and compute the barycentric coordinates of p in the new triangle.

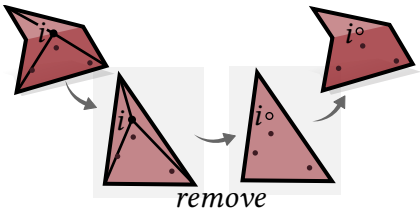


2. *Vertex flattening*: We must also compute new barycentric coordinates $\tilde{b}$ after each vertex flattening (Section 5.1.1). Here we use the projective interpolation scheme of Springborn et al. [2008, §3.4]. Since our parameterization is a discrete conformal equivalence, this scheme defines a continuous (C^0) bijective map. Let b_i, b_j, b_k be barycentric coordinates for a point in face ijk, and let u_i be the scale factor at i. Then

$$(\tilde{b}_i, \tilde{b}_j, \tilde{b}_k) = \frac{(e^{u_i} b_i, b_j, b_k)}{e^{u_i} b_i + b_j + b_k}, \tag{5.1}$$

   where the denominator ensures our updated values still sum to 1.

3. *Vertex removal*: Once vertex i is flattened and flipped to degree three, its neighborhood can be laid out in the plane without distortion. Here we apply standard formulas to compute barycentric coordinates for vertex i in the new triangle, along with coordinates for any points located in the three removed triangles.



5.2.2 Mapping Edges

In addition to mapping points between the fine and coarse meshes, we can also map an edge on one mesh to the corresponding polygonal curve on the other. The key is to determine how each local operation modifies a polygonal curve. In our case, edge flips and vertex insertion/removal may add vertices to the polygonal curve, but leave its geometry the same, while vertex flattening distortions the curve geometry but leaves its combinatorics the same. Mapping edges allows us to draw the intrinsic triangulation sitting atop the extrinsic mesh, and to compute the common subdivision of the two triangulations.

Note, however, that unlike the situation in Chapter 3, the mapping between our original and simplified triangulations is not necessarily linear on each face of the common subdivision. Since we are now allowed to change the underlying geometry of our surface, the correspondences that we obtain via Section 5.2.1 are often more complicated.

5.2.3 Mapping Functions

Algorithms such as multigrid often require not only pointwise correspondences, but also *prolongation operators* which transfer functions from a coarse mesh to a finer one. We define a

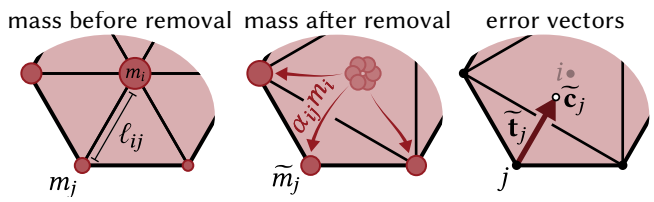


Figure 5.2: The local cost of removing any vertex i is the *optimal transport cost* of transporting its mass m_i to its neighbors j. We can also calculate this cost as the sum of new masses $\tilde{m}_j$ times the length of *error vectors* $\tilde{t}_j$, which point to the new centers of mass $\tilde{c}_j$.

prolongation operator via an approach similar to Lee et al. [1998a] and Liu et al. [2021a]. In particular, we track the barycentric coordinates of all fine vertices *à la* Section 5.2.1. A function on the coarse mesh is then mapped to the fine mesh via barycentric linear interpolation. Explicitly, suppose each fine vertex $i \in \mathsf{V}$ has barycentric coordinates b_1, b_2, b_3 in coarse triangle $n_1 n_2 n_3 \in \tilde{\mathsf{F}}$. We then build a sparse matrix $\mathsf{P} \in \mathbb{R}^{\mathsf{V} \times \tilde{\mathsf{V}}}$ where row i has three nonzeros $\mathsf{P}_{i,n_j} = b_{n_j}$, for $j = 1, 2, 3$. We can then prolong a function $\tilde{f} \in \mathbb{R}^{\tilde{\mathsf{V}}}$ on the coarse mesh to obtain a function $f \in \mathbb{R}^{\mathsf{V}}$ on the fine mesh by evaluating the matrix-vector product $f = \mathsf{P}\tilde{f}$.

5.3 Measuring Distortion

To prioritize vertex removals, we must quantify the cost of removing a vertex. Standard extrinsic metrics, such as QEM, are not appropriate: even if they could somehow be evaluated intrinsically, they would attempt to preserve irrelevant aspects of the geometry. Our method is however inspired by the remarkable effectiveness of greedy local error accumulation in QEM. Likewise, metrics that focus on finite element equality (*à la* [Shewchuk 2002]) are not appropriate, since the triangulation used to encode the intrinsic geometry is transient and subject to change.

Our ICE metric is instead based on two intrinsic and triangulation-independent concepts: *optimal transport* [Santambrogio 2015], and the *Karcher mean* [Karcher 2014]. Optimal transport helps quantify the effort of redistributing mass, providing the local cost for our metric. Karcher means encode the center of mass of all fine vertices contributing to a coarse vertex i, providing a way of accumulating information. These two pieces fit together in a natural way: after a single vertex removal, the mass-weighted norm of all error vectors t_i encoding Karcher means is exactly equal to the optimal transport cost. Hence, after many vertex removals this norm approximates the cost of transporting the initial fine mass distribution to the coarsened vertices. Vertex removals that keep cost small should hence be prioritized, since they better preserve the initial mass distribution. Just as in QEM, this information is captured by a fixed-size representation (masses and tangent vectors at each vertex) that is easily agglomerated during coarsening. For clarity of exposition we first define error metrics in 2D, before generalizing to surfaces and incorporating data like curvature or other attributes.

5.3.1 Flat Error Metric

Consider a mass distribution $m : V \to \mathbb{R}_{\geq 0}$ at mesh vertices, representing any nonnegative user-defined quantity (signed quantities will be addressed in Section 5.3.2). Suppose we remove vertex i, redistributing its mass m_i to its immediate neighbors j. In particular, let $\alpha_{ij} \in [0, 1]$ be

the fraction of m_i sent to vertex j (hence $\sum_{j\sim i} \alpha_{ij} = 1$), so that the new mass at j is

$$\tilde{m}_j = m_j + \alpha_{ij} m_i. \tag{5.2}$$

To measure how mass is transported across the surface, we need to track not only the mass distribution, but also where mass came from. Hence, at each vertex i we store an *error vector* t_i (initially set to zero) pointing to the center of mass c_i of all vertices that contributed to the current value of m_i. Explicitly, after removing i, the center of mass at vertex j is

$$\tilde{c}_j = \frac{\alpha_{ij} m_i x_i + m_j x_j}{\alpha_{ij} m_i + m_j}, \tag{5.3}$$

where $x_i \in \mathbb{R}^2$ denotes the location of vertex i. The vector pointing from x_j to $\tilde{c}_j$ is thus

$$\tilde{t}_j = \tilde{c}_j - x_j = \frac{\alpha_{ij} m_i e_{ji}}{\alpha_{ij} m_i + m_j}, \tag{5.4}$$

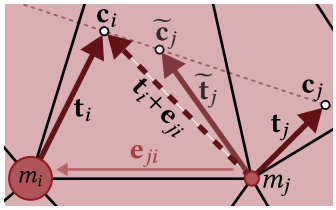
where $e_{ji} = x_i - x_j$ is the vector along edge ji. The total cost of removing i can then be measured by summing up the mass-weighted norms of these vectors. Noting that $\|e_{ji}\| = \ell_{ij}$, we get a cost

$$C_i = \sum_{j\sim i} \tilde{m}_j \|\tilde{t}_j\| = \sum_{j\sim i} \alpha_{ij} m_i \ell_{ij}. \tag{5.5}$$

This cost also coincides with the so-called *1-Wasserstein distance* between the old and new mass distribution [Santambrogio 2015, Chapter 5]. Intuitively, this distance measures the total "effort" of moving mass from i to neighbors j, penalizing not only the amount of mass moved, but also the distance traveled.

Rather than assign a cost to each vertex removal in isolation, we can accumulate information about how mass has been redistributed across all prior removals. At each step, we update the mass distribution via Equation (5.2), but also update vectors encoding the centers of mass via

$$\tilde{t}_j = \frac{\alpha_{ij} m_i (t_i + e_{ji}) + m_j t_j}{\alpha_{ij} m_i + m_j}. \tag{5.6}$$



In other words, we re-express t_i relative to x_j by adding the edge vector e_{ji}, then take the mass-weighted average of the old error vector t_j with this new vector. The overall cost is still evaluated via Equation (5.5), but now approximates the effort of moving the initial mass distribution to the current one—rather than just penalizing the most recent change. This cost is only approximate since the 1-Wasserstein distance to the center of mass is not in general equal to the distance to the original fine distribution—but it is usually quite close. Thus, our error metric favors decimation sequences which keep each coarse vertex close to the center of all fine vertices that contribute to its mass.

5.3.2 Intrinsic Curvature Error Metric

Due to the Gauss-Bonnet theorem, flattening a vertex i conservatively redistributes curvature to neighboring vertices j, making curvature a natural "mass" distribution to guide simplification. A

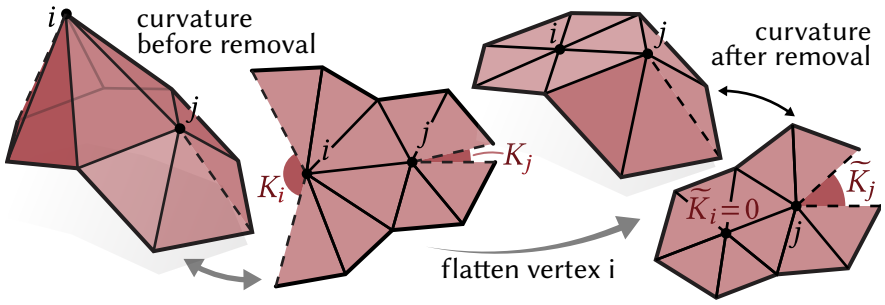


Figure 5.3: Flattening a vertex i changes the angle sums Θ at neighboring vertices j, effectively redistributing the discrete curvature $K = 2\pi - \Theta$. We use the change in curvature from K to $\tilde{K}$ to guide simplification.

challenge here is that the old and new curvatures K and $\tilde{K}$ are not in general positive quantities. One possibility might be to use a transport cost for signed measures such as [Mainini 2012], but doing so would require us to solve a small optimal transport problem for each vertex removal. We instead adopt a cheap alternative. In particular, we define convex weights

$$\alpha_{ij} := \frac{|\tilde{K}_j - K_j|}{\sum_{l \sim i} |\tilde{K}_l - K_l|}. \tag{5.7}$$

For boundary vertices we use the same formula, but replace Gaussian curvature K with geodesic curvature κ. If vertex i is already flat prior to removal, then there is no change in curvature and we simply distribute mass equally to all neighbors. We then split the initial fine curvature function K (or κ) into two positive mass functions $K_i^+ := \max(K_i, 0)$ and $K_i^- := -\min(K_i, 0)$. Each of these quantities is tracked throughout simplification like m_i above, using two separate vectors t_i^+ and t_i^- (respectively), and weights α from Equation (5.7):

$$\tilde{t}_j^{\pm} = \frac{\alpha_{ij} m_i (R_{ij} t_i^{\pm} + e_{ji}) + m_j t_j^{\pm}}{\alpha_{ij} m_i + m_j}. \tag{5.8}$$

The parallel transport term R_{ij} to account for the surface curvature. The overall error, which defines the ICE metric, is then the sum of the errors in the two curvature functions (*à la* Equation (5.5)). Note that if a vertex i cannot be flattened or removed, we assign it an infinite cost (which may later get updated to a finite value when its neighbors are removed.

Auxiliary Data Similar to Garland & Heckbert [1998], we can use other quantities at vertices (areas, colors, *etc.*) to drive simplification in an analogous fashion: each signed quantity is split into two positive mass functions, and a list of all "channels" $m^1, \ldots, m^k$ is tracked along with associated tangent vectors $t^1, \ldots, t^k$. The cost is then given by

$$C_i = \sum_{j \in \mathcal{N}_i} \sum_{p=1}^{k} w^p m_i^p \|t_j^p\|, \tag{5.9}$$

where a choice of weights $w^1, \ldots, w^k \in \mathbb{R}_{\geq 0}$ puts an emphasis on different features. For instance, Figure 5.4 shows the impact of different weightings on curvature versus area.

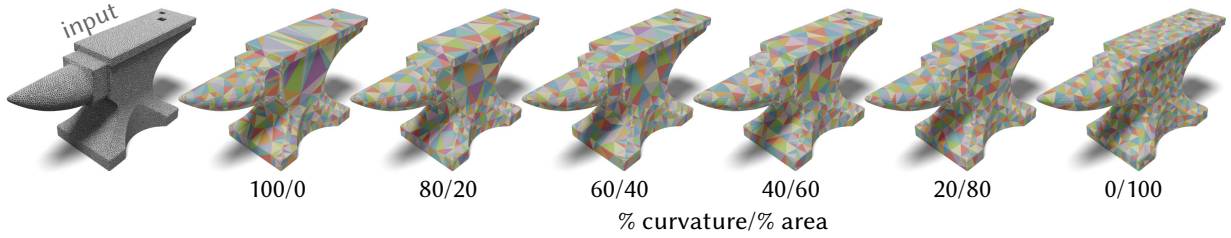


Figure 5.4: We can mix and match different quantities to guide coarsening. Here, for instance, strongly weighting Gaussian curvature emphasizes preservation of intrinsic geometry, whereas strongly weighting area prioritizes uniform triangle size.

Units. The assorted masses $m^1, \ldots, m^k$ may naturally be measured in different units, in which case the sum in Equation (5.9) will not have consistent units. For instance, if we naïvely add together curvature error and area error in Figure 5.4, the resulting error is neither scale-invariant (like discrete Gaussian curvature), nor scales quadratically (like area). Instead, the balance between curvature error and area error changes as the input mesh is scaled up and down. An easy solution is to work with unitless mass fractions $\tilde{m}_i := \frac{1}{\sum_j m_j} m_i$. In this case, Equation (5.9) always yields a cost C_i with units of distance.

5.4 Simplification Algorithm

We now have all the ingredients to perform intrinsic simplification. Just as in QEM, we first initialize a priority queue by evaluating the ICE metric at all vertices (Section 5.4), then greedily remove the lowest-cost vertex from this queue until we reach a target vertex count n, or until no more vertices can be removed. Algorithm 6 provides pseudocode.

Initialization For each vertex $i \in V$ we compute the initial masses m_i (*e.g.*, the curvature functions K^+ and K^- from Section 5.3.2) and an initial error vector $t_i = 0$. To compute the cost of removing i we perform a tentative flattening *à la* Section 5.1.1 and use the resulting weights α_{ij} from Equation (5.7) to evaluate the cost C_i via the second sum in Equation (5.5). If flattening yields invalid edge lengths, or vertex i cannot be flipped to degree 3 (or 2 on the boundary), we let $C_i = \infty$. After evaluating the cost function, we "undo" the tentative removal, *i.e.*, we restore the previous connectivity and revert any changes to edge lengths.

Algorithm 6 SimplifyIntrinsic$(\mathsf{T}, \ell, \varphi, n)$

Input: A triangulation $\mathsf{T} = (\mathsf{V}, \mathsf{E}, \mathsf{F})$ equipped with edge lengths $\ell : \mathsf{E} \to \mathbb{R}_{>0}$ and angular coordinates $\varphi : \mathsf{H} \to \mathbb{R}$, along with a target vertex count n

Output: A coarsened mesh $\tilde{\mathsf{T}}$ equipped with coarsened edge lengths $\tilde{\ell} : \tilde{\mathsf{E}} \to \mathbb{R}_{>0}$, and a list H of all operations done to simplify $\tilde{\mathsf{T}}$.

$Q \leftarrow$ EmptyPriorityQueue() ▷ *initialize vertex queue*
$H \leftarrow []$ ▷ *initialize operation history*
$\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, H \leftarrow$ FlipToDelaunay$(\mathsf{T}, \ell, \varphi, H)$ ▷ *§2.3.3*
for each vertex $i \in \tilde{\mathsf{V}}$ **do** ▷ *initialize mass & error*
  $m_i, t_i \leftarrow (K_i^-, K_i^+), 0$
for each vertex $i \in \tilde{\mathsf{V}}$ **do**
  $c_i \leftarrow$ ICError$(\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, m, t, i)$ ▷ *§5.3.2*
  $Q \leftarrow$ Enqueue(Q, i, c_i)
▷ *coarsening*
while VertexCount$(\tilde{\mathsf{T}}) > n$ **and not** Empty(Q) **do**
  $i \leftarrow$ Pop(Q) ▷ *extract minimum-cost vertex*
  $\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi} \leftarrow$ Flatten$(\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, i)$ ▷ *§5.1.1*
  $H \leftarrow$ Append$(H,$ "*flatten i*"$)$ ▷ *record operation*
  $\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, m, t \leftarrow$ RemoveVertex$(\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, i, m, t)$ ▷ *§3.5.2*
  $H \leftarrow$ Append$(H,$ "*remove i*"$)$ ▷ *record operation*
  $\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, H \leftarrow$ FlipToDelaunay$(\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, H)$
  for each adjacent vertex $j \sim i$ **do**
    $c_j \leftarrow$ ICError$(\tilde{\mathsf{T}}, \tilde{\ell}, \tilde{\varphi}, m, t, j)$
    $Q \leftarrow$ UpdatePriority(Q, j, c_j)
return $(\tilde{\mathsf{T}}, \tilde{\ell}, H)$

Simplification At each iteration, we pick the vertex i with the minimum cost C_i from our priority queue. If $C_i = \infty$, then no more vertices can be removed and we terminate. Otherwise, we apply the removal procedure from Section 3.5.2. The resulting weights α_{ij} (Equation (5.7)) are used to compute new masses $\tilde{m}_j$ (Equation (5.2)) and updated transport vectors $\tilde{t}_j$ (Equation (5.8)) for each neighbor $j \sim i$. We record this vertex flattening and removal in our history H listing the local mesh operations performed during simplification. We then flip the mesh back to intrinsic Delaunay *à la* Section 2.3.3—note that to initialize the greedy flipping algorithm we need only enqueue edges incident to vertex i, since the mesh was already Delaunay prior to removing i. We ensure that our Delaunay flipping routine also records all edge flips in the history list H. Finally, we must also update the priority queue with new costs c_j, computed by tentatively flattening each neighbor j and evaluating the first sum in Equation (5.5) (this time over neighbors $k \sim j$). Here, finite costs may become infinite (or vice versa), since vertices that were previously removable may no longer be removable.

5.5 Simplification Results

5.5.1 Comparison with Extrinsic Methods

The flexibility gained by working in the larger space of intrinsic triangulations leads to lower geometric distortion than extrinsic meshes exhibit on meshes of equivalent size. In Figure 5.5 we coarsen a 28k bunny mesh down to 200 vertices with both the method of Liu et al. [2021b] and our method. Even on this highly regular geometry we observe a modest reduction of both area distortion and anisotropic distortion. For more difficult triangulations, or surfaces with lower intrinsic curvature, we observe more significant gains.

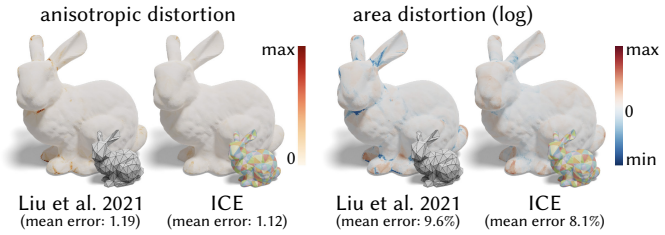


Figure 5.5: Even on an extremely nice triangulation of a highly regular surface we see a reduction in distortion relative to past methods—owing to the much larger space of intrinsic triangulations.

As an extreme case, Figure 5.6 coarsens a developable surface from [Verhoeven et al. 2022] via both QEM and ICE. Since coarse extrinsic edges are shortest paths in $\mathbb{R}^n$, they underestimate intrinsic distances (hence areas); in contrast, intrinsic edges are essentially embedded in the original surface, providing better approximation of the original geometry.

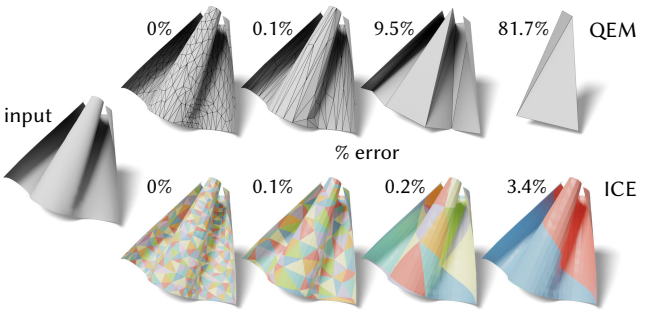


Figure 5.6: On surfaces with small extrinsic curvature, we achieve dramatically lower error in surface area compared to extrinsic methods like QEM.

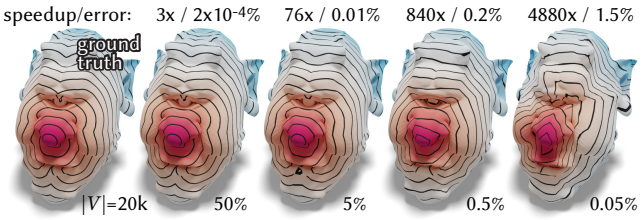


Figure 5.8: Intrinsic coarsening offers an attractive approach to approximating single-source geodesic distance, here providing a three orders of magnitude speedup for a fraction of a percent relative error.

5.5.2 Geometric Algorithms

Partial Differential Equations Better domain approximation in turn improves the quality of solutions computed on coarse meshes. For example, in Figure 5.7 we coarsen a cloth simulation mesh down to 500 vertices with an extrinsic method ([Liu et al. 2021b] using QEM simplification) and our intrinsic method. We then solve a Poisson problem on the coarse meshes and apply prolongation, yielding more accurate results in the intrinsic case.

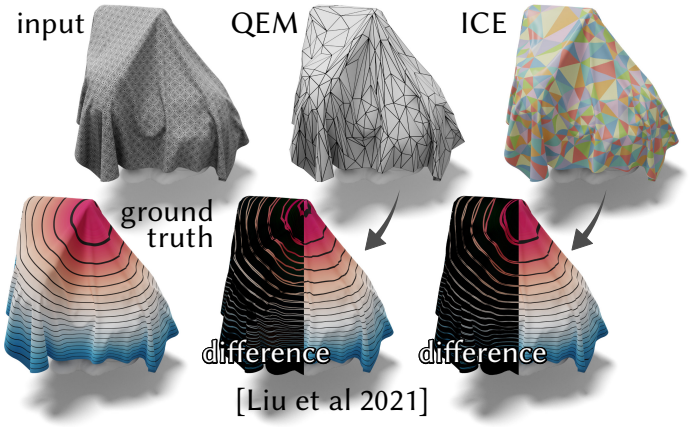


Figure 5.7: For the same vertex budget as extrinsic methods like QEM, ICE provides more accurate solutions for basic problems like solving a Poisson equation—seen here via smoother isolines that better approximate the ground truth.

Single-Source Geodesic Distance Geodesic distance is an intrinsic quantity, making it a natural fit for intrinsic coarsening. In Figure 5.9 we compare ICE to the extrinsic method of Lee et al. [1998b] by measuring the difference between the exact distance on the fine input, and prolongated distances from the coarse meshes (both computed via [Mitchell et al. 1987]); here ICE achieves a roughly 4x reduction in relative error. Figure 5.8 illustrates the speed-accuracy trade off of using ICE, here reducing cost by three orders of magnitude while introducing only ~ 1% relative approximation error.

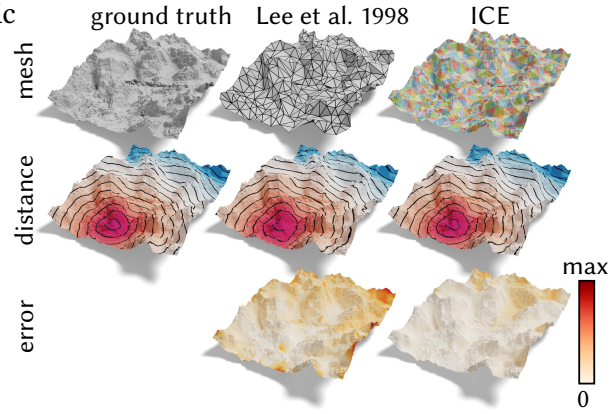


Figure 5.9: As geodesic distance is an intrinsic quantity, it is more accurately approximated via intrinsic coarsening—here providing a 4x reduction in relative error.

All-Pairs Geodesic Distance The benefits of an accurate intrinsic approximation become even more pronounced when approximating the dense matrix $D \in R^{|V|\times|V|}$ of all pairs of geodesic distances—a shape descriptor often used in correspondence and learning methods [Shamai & Kimmel 2017]. We can compute a low-rank approximation of D via

$$\widehat{D} := P\widetilde{D}P^{\top},$$

where $\widetilde{D}$ is the coarse all-pairs matrix (computed again via [Mitchell et al. 1987]). See for instance Figure 5.10—here again we achieve several orders of magnitude speedup, with only 1.4% relative error.

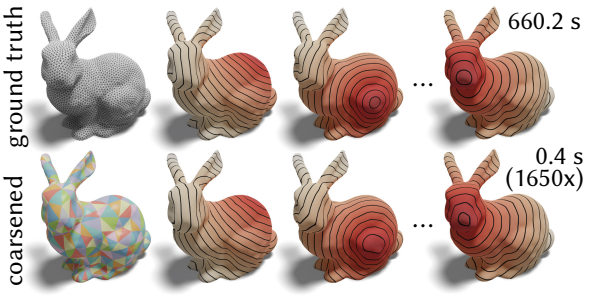


Figure 5.10: For a mesh with 6k vertices we obtain an all-pairs geodesic distance matrix 1650x faster, while incurring only 1.4% relative error.

5.5.3 Performance & Complexity

In practice, we find that the cost of performing intrinsic simplification and building the prolongation matrix (Section 5.2.3) scales approximately linearly as the size of the input mesh increases, or as the desired size of the output decreases (Figure 5.11). In this section, we discuss the computational complexity of the various steps involved and note what theoretical guarantees are available.

Intrinsic Curvature Error. While performing simplification we maintain a priority queue of all remaining vertices, sorted by the cost required to remove them. Each time we remove a vertex we pop the cheapest vertex i from this queue, incurring a cost of $O(\log n)$. We then remove i and update the error vectors for all neighbors of i, which requires $O(\deg(i))$ arithmetic operations in addition to flipping to Delaunay. Finally, we compute the new cost of flattening those neighbors and update their keys in the priority queue, which has a cost of $O\left(\sum_{j\sim i} \deg(j)\right)$ to flatten them all, a cost of $O(\deg(i)\log(n))$ to update the priority queue, and required flipping to Delaunay $O(\deg(i))$ additional times.

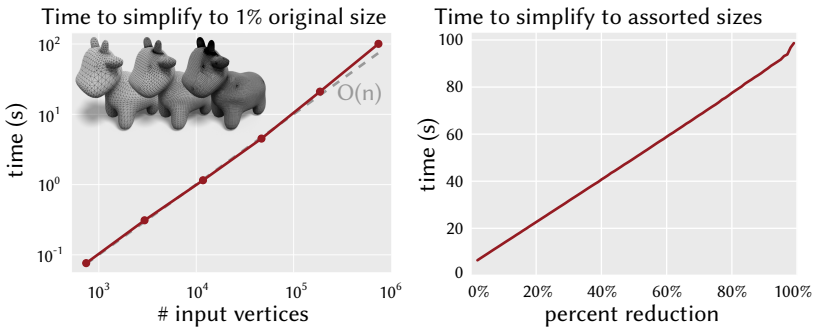


Figure 5.11: In practice, the total cost of simplification and building the prolongation matrix scales approximately linearly in both input mesh size and percent reduction. *Left:* increasingly high-resolution meshes are simplified. *Right:* a subdivision with 750k vertices is simplified to various resolutions.

Simplification. If we suppose that the maximum degree of any vertex encountered during simplification is some constant $D = O(\log n)$, then the total cost of removing k vertices is $O(kD \log n) = O(k \log^2 n)$ operations to manage the priority queue and perform the vertex removals, tracking correspondence through $O(kD)$ vertex flattening and removal operations, and flipping to Delaunay $O(kD)$ times. . In practice, flipping to Delaunay almost always requires $O(1)$ edge flips, since we start with a triangulation which is "close to" Delaunay. One might be able to perform a careful analysis similar to the work of Guibas et al. [1992] on planar Delaunay triangulations, but we leave such questions to future work.

Correspondence Tracking. Mapping a point through any of the simplification operations described in Section 5.2.1 requires a constant amount of work—edge flips and vertex removals both act on a fixed-sized neighborhood, so one can work out explicit formulas describing how a point should be mapped, and vertex flattening simply modifies a point's barycentric coordinates by the formula given in Equation (5.1).

In order to build the prolongation matrix P to map functions between the coarse and fine triangulations, we must map each vertex of the fine mesh onto the coarse mesh (Section 5.2.3). In theory, this operation requires at least quadratic complexity—after removing k vertices, we must map k points through $\Omega(k)$ simplification operations. In practice, our performance scaling seems closer to linear, as building the prolongation matrix is not the bottleneck cost (Figure 5.11). But it could be interesting to investigate constructions which provide better asymptotic guarantees in future.

Storage Cost The data structures used to perform intrinsic simplification require $O(|\mathsf{F}|)$ storage space. On top of the underlying mesh data structure, which requires $O(|\mathsf{F}|)$ space, our intrinsic triangulation data structure stores a constant amount of information per edge of the mesh (edge lengths and angular coordinates). In order to perform simplification, we store a constant amount of information at each vertex to track the intrinsic curvature error, and we also maintain a priority queue of mesh vertices sorted by the cost required to remove them, which also requires $O(|\mathsf{V}|)$ space. Note that in a triangle mesh, we know that $|\mathsf{E}| \leq 3|\mathsf{F}|$ and also that $|\mathsf{V}| \leq 3|\mathsf{F}|$, so all of this information can be stored in $O(|\mathsf{F}|)$ space.

If we also maintain correspondence by storing the list of simplification operations which were performed, we require an additional array whose size is linear in the number of operations performed. Each operation can be recorded in a fixed amount of space (*e.g.* "Flip edge ij, which lies between faces ijk and jil", or "Scale vertex i by $u_i = 3$"). However, as discussed above, we do not have an asymptotic bound on the number of local operations performed during simplification—such a bound would require, for instance, a bound on the complexity of flipping to Delaunay, which is currently unknown. Nonetheless, we find in practice that the number of operations required to simplify a mesh seems to scale linearly with the input (Figure 5.11), and thus the space required to track correspondence scales linearly as well.

Chapter 6

Open Questions

There are still many open questions about intrinsic triangulations. We conclude by listing several questions related to the work in this thesis.

Evolving both triangulations. In most existing work on intrinsic triangulations, one considers the correspondence between two triangulations T_1 and T_2 which share the same geometry. In this thesis, we considered several problems where the geometry of T_2 varies while the original triangulation T_1 stays fixed. But are there cases where we want to vary the geometry of both triangulations simultaneously? In such cases, one could try to track the correspondence between T_1 and T_2 with some fixed reference triangulation T_0. But can we track the correspondence directly while allowing both triangulations to evolve?

Discrete conformal mesh simplification. The surface simplifications which we compute in Chapter 5 are constructed via of a sequence of parameterizations which are each exactly discretely conformal in the sense of Section 4.1. But the composition is not a discrete conformal equivalence, because the theory of discrete conformal equivalence only applies to meshes with the same number of vertices. Is there a more general theory which allows us the change the number of vertices?

Hyperbolic parameterization. In Chapter 4, we describe how to compute discrete conformal maps to the Euclidean plane, and to the sphere. Bobenko et al. [2015, Section 6] also provide an analogous procedure for computing discrete conformal maps to the hyperbolic plane (in the fixed-triangulation case), but to our knowledge it has never been implemented, and the algorithm has not even been described in the variable-triangulation case.

Simplifying the topological class. In Chapter 5, we simplify the *geometry* of an intrinsic triangulation, but we always preserve its topological class. Our simplified surface is always *homeomorphic* to the original surface. However, meshes arising from 3d scans often have *toplogical* noise, such as spurious gaps or handles. How can you intrinsically simplify the topological class of a surface in addition to its geometry?

Asymptotic complexity of Delaunay flipping. In theory, one can construct intrinsic triangulations with a fixed number of vertices which nonetheless take an arbitrarily long time to flip flip to Delaunay. Starting from real-world extrinsic triangulations, however,

Delaunay flipping seems to run in essentially linear time [Sharp et al. 2019, Figure 10]. Can one bound the work required to flip an extrinsic triangulation to Delaunay? 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?

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 [Devillers & Pion 2003], 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?

Analysis of intrinsic Delaunay refinement. In Section 3.6 we showed that intrinsic Delaunay refinement must successfully on meshes without boundary, but did not consider meshes with boundary, or more fine-grained analysis of the spatial grading. Can one adapt more sophisticated analysis of planar Delaunay refinement (*e.g.* Section 3.4.2 of Shewchuk [1997]) to work in the intrinsic setting ?

Other algorithms for Delaunay triangulation. The flip algorithm is rarely used to construct Delaunay triangulations in the plane. There are a variety of techniques with faster worst-case complexity: for instance, there are incremental constructions which iteratively add vertices into the triangulation [Bowyer 1981; Watson 1981], reductions to convex hull construction [Brown 1979], and divide and conquer algorithms which recursively compute Delaunay triangulations of subsets of the vertices [Shamos & Hoey 1975; Guibas & Stolfi 1985]. Do any of these generalize to the intrinsic setting? Is it any easier if one wants to construct the intrinsic Delaunay triangulation of a given extrinsic triangle mesh?

Bibliography

Abikoff, W. (Oct. 1981). "The uniformization theorem". *The American Mathematical Monthly* 88.8, pp. 574–492. DOI: 10.2307/2320507.

Ahlfors, L. V. (1973). *Conformal invariants: topics in geometric function theory*. Vol. 371. American Mathematical Society.

Aigerman, N., Poranne, R., and Lipman, Y. (July 2014). "Lifted bijections for low distortion surface mappings". *ACM Transactions on Graphics* 33.4, pp. 1–12. DOI: 10.1145/2601097.2601158.

Alekseevskij, D., Vinberg, E. B., and Solodovnikov, A. (June 1993). "Geometry of spaces of constant curvature". *Geometry II*. Vol. 29. Encyclopaedia of Mathematical Sciences. Springer. DOI: 10.1007/978-3-662-02901-5_1.

Alexandrov, A. D. (1942). "Existence of a convex polyhedron and of a convex surface with a given metric". *Matematicheskii Sbornik* 53.11, pp. 15–65.

Alexandrov, A. D. (1948). *Intrinsic Geometry of Convex Surfaces*. Vol. 2. OGIZ, Moscow-Leningrad. DOI: 10.1201/9780203643846.

Baden, A., Crane, K., and Kazhdan, M. (Aug. 2018). "Möbius registration". *Computer Graphics Forum* 37.5. DOI: 10.1111/cgf.13503.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Karpeyev, D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2019). *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 3.11. Argonne National Laboratory.

Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. (1997). "Efficient management of parallelism in object oriented numerical software libraries". *Modern Software Tools in Scientific Computing*. Ed. by Arge, E., Bruaset, A. M., and Langtangen, H. P. Birkhäuser Press.

Baumgart, B. G. (May 1975). "A polyhedron representation for computer vision". *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, pp. 589–596. DOI: 10.1145/1499949.1500071.

Bell, M. (2013). *flipper (Computer Software)*. https://pypi.python.org/pypi/flipper.

Bell, M. (2015). "Recognising mapping classes". PhD thesis. University of Warwick.

Benson, S., McInnes, L. C., More, J. J., and Sarich, J. (2003). *TAO Users Manual*. Tech. rep. Argonne National Lab., IL (US).

Bobenko, A. and Springborn, B. (2004). "Variational principles for circle patterns and Koebe's theorem". *Transactions of the American Mathematical Society* 356.2, pp. 659–689. DOI: 10.1090/S0002-9947-03-03239-2.

Bobenko, A. I. and Izmestiev, I. (2008). "Alexandrov's theorem, weighted Delaunay triangulations, and mixed volumes". *Annales de l'Institut Fourier*. Vol. 58. 2, pp. 447–505. DOI: 10.5802/aif.2358.

Bobenko, A. I. and Lutz, C. O. (2023). "Decorated discrete conformal maps and convex polyhedral cusps". *arXiv preprint*. URL: https://arxiv.org/abs/2310.17529.

Bobenko, A. I., Pinkall, U., and Springborn, B. A. (2015). "Discrete conformal maps and ideal hyperbolic polyhedra". *Geometry & Topology* 19.4, pp. 2155–2215. DOI: 10.2140/gt.2015.19.2155.

Bobenko, A. I. and Springborn, B. A. (Sept. 2007). "A discrete Laplace–Beltrami operator for simplicial surfaces". *Discrete & Computational Geometry* 38.4, pp. 740–756. DOI: 10.1007/s00454-007-9006-1.

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Lévy, B. (2010). *Polygon Mesh Processing*. DOI: 10.1201/b10688.

Bowyer, A. (Jan. 1981). "Computing Dirichlet tessellations". *The Computer Journal* 24.2, pp. 162–166. DOI: 10.1093/comjnl/24.2.162.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press. DOI: 10.1017/CBO9780511804441.

Boyer, D. M., Lipman, Y., Clair, E. S., Puente, J., Patel, B. A., Funkhouser, T., Jernvall, J., and Daubechies, I. (2011). "Algorithms to automatically quantify the geometric similarity of anatomical surfaces". *Proceedings of the National Academy of Sciences* 108.45. DOI: 10.1073/pnas.1112822108.

Bright, A., Chien, E., and Weber, O. (July 2017). "Harmonic global parametrization with rational holonomy". *ACM Transactions on Graphics* 36.4, pp. 1–15. DOI: 10.1145/3072959.3073646.

Brown, K. Q. (1979). "Voronoi diagrams from convex hulls". *Information processing letters* 9.5. DOI: 10.1016/0020-0190(79)90074-7.

Bücking, U. (2016). "Approximation of conformal mappings using conformally equivalent triangular lattices". *Advances in Discrete Differential Geometry*. Springer. DOI: 10.1007/978-3-662-50447-5_3.

Bücking, U. (2018). "C^∞-convergence of conformal mappings for conformally equivalent triangular lattices". *Results in Mathematics* 73.2. DOI: 10.1007/s00025-018-0845-2.

Burago, Y. D. and Zalgaller, V. A. (1960). "Polyhedral embedding of a flat metric with conical singularities". *Vestnik Leningrad Univ.* 15.7, pp. 66–80.

Burago, Y. D. and Zalgaller, V. A. (1995). "Isometric piecewise-linear embeddings of two-dimensional manifolds with a polyhedral metric into $\mathbb{R}^3$". *Algebra i Analiz* 7.3, pp. 76–95.

Campen, M., Capouellez, R., Shen, H., Zhu, L., Panozzo, D., and Zorin, D. (Dec. 2021). "Efficient and robust discrete conformal equivalence with boundary". *ACM Transactions on Graphics* 40.6. ISSN: 0730-0301. DOI: 10.1145/3478513.3480557.

Campen, M., Shen, H., Zhou, J., and Zorin, D. (2019). "Seamless parametrization with arbitrary cones for arbitrary genus". *ACM Transactions on Graphics* 39.1. DOI: 10.1145/3360511.

Campen, M. and Zorin, D. (2017a). "On discrete conformal seamless similarity maps". *arXiv preprint*. URL: https://arxiv.org/abs/1705.02422.

Campen, M. and Zorin, D. (2017b). "Similarity maps and field-guided T-splines: a perfect couple". *ACM Transactions on Graphics* 36.4. DOI: 10.1145/3072959.3073647.

Cannon, J. W., Floyd, W. J., Kenyon, R., Parry, W. R., et al. (1997). *Hyperbolic Geometry*. Vol. 31. MSRI Publications. ISBN: 0-521-62048-1.

do Carmo, M. P. (1992). *Riemannian Geometry*. 2nd ed. Translated by Francis Flaherty. Springer. ISBN: 978-0-8176-3490-2.

Chebyshev, P. L. (1899). *Œuvres de P.L. Tchebychef*. Vol. 1. Commissionaires de l'Académie Impériale des Sciences.

Chen, W., Zhang, M., Lei, N., and Gu, X. D. (2016). "Dynamic unified surface Ricci flow". *Geometry, Imaging and Computing* 3.1. DOI: 10.4310/GIC.2016.v3.n1.a2.

Chern, A., Knöppel, F., Pinkall, U., and Schröder, P. (July 2018). "Shape from metric". *ACM Transactions on Graphics* 37.4. DOI: 10.1145/3197517.3201276.

Chew, L. P. (1993). "Guaranteed-quality mesh generation for curved surfaces". *Proceedings of the ninth annual symposium on Computational geometry*, pp. 274–280. DOI: 10.1145/160985.161150.

Chien, E., Levi, Z., and Weber, O. (Dec. 2016). "Bounded distortion parametrization in the space of metrics". *ACM Transactions on Graphics* 35.6, pp. 1–16. DOI: 10.1145/2980179.2982426.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). "Meshlab: an open-source mesh processing tool." *Eurographics Italian chapter conference*. Vol. 2008. Salerno, Italy, pp. 129–136. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136.

Crane, K. (2020). "Conformal Geometry of Simplicial Surfaces". *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society.

Crane, K., de Goes, F., Desbrun, M., and Schröder, P. (2013a). "Digital geometry processing with discrete exterior calculus". *ACM SIGGRAPH 2013 Courses*. New York, NY, USA: ACM. DOI: 10.1145/2504435.2504442.

Crane, K., Pinkall, U., and Schröder, P. (2013b). "Robust Fairing via Conformal Curvature Flow". *ACM Transactions on Graphics* 32.4. DOI: 10.1145/2461912.2461986.

Devillers, O. and Pion, S. (2003). "Efficient Exact Geometric Predicates for Delauny Triangulations." *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*.

Dym, N., Slutsky, R., and Lipman, Y. (2019). "Linear variational principle for Riemann mappings and discrete conformality". *Proceedings of the National Academy of Sciences* 116.3. DOI: 10.1073/pnas.1809731116.

L'Engle, M. (1962). *A Wrinkle in Time*. Ariel Books.

Erickson, J. and Nayyeri, A. (2013). "Tracing compressed curves in triangulated surfaces". *Discrete & Computational Geometry* 49.4, pp. 823–863. DOI: 10.1007/s00454-013-9515-z.

Escher, M. C. (1958). *Letter to Coxeter*. Attested by Coxeter, H. S. M. (1979). *The Non-Euclidean Symmetry of Escher's Picture "Circle Limit III."* *Leonardo*, 12(1), 19–25.

Farb, B. and Margalit, D. (2011). *A primer on mapping class groups*. Princeton University Press.

Farrell, P. E., Piggott, M. D., Pain, C. C., Gorman, G. J., and Wilson, C. R. (2009). "Conservative interpolation between unstructured meshes via supermesh construction". *Computer methods in applied mechanics and engineering* 198.33-36, pp. 2632–2642. DOI: 10.1016/j.cma.2009.03.004.

Fillastre, F. (2008). "Polyhedral hyperbolic metrics on surfaces". *Geometriae Dedicata* 134.1. DOI: 10.1007/s10711-008-9305-6..

Finnendahl, U., Schwartz, M., and Alexa, M. (Apr. 2023). "Arap revisited discretizing the elastic energy using intrinsic voronoi cells". *Computer Graphics Forum (SGP)*. DOI: 10.1111/cgf.14790.

Fisher, M., Springborn, B., Bobenko, A. I., and Schröder, P. (2006). "An algorithm for the construction of intrinsic Delaunay triangulations with applications to digital geometry processing". *ACM SIGGRAPH 2006*, pp. 69–74. DOI: 10.1145/1185657.1185668.

Fisher, M., Springborn, B., Schröder, P., and Bobenko, A. (Aug. 2007). "An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing". *Computing* 81.2, pp. 199–213. DOI: 10.1007/s00607-007-0249-8.

Fumero, M., Möller, M., and Rodolà, E. (Nov. 2020). "Nonlinear spectral geometry processing via the tv transform". *ACM Transactions on Graphics* 39.6, pp. 1–16. DOI: 10.1145/3414685.3417849.

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Alken, P., Booth, M., Rossi, F., and Ulerich, R. (1994). *GNU Scientific Library*. Vol. 20. ACM.

Garland, M. and Heckbert, P. S. (Aug. 1997). "Surface simplification using quadric error metrics". *SIGGRAPH 1997*. New York, NY, USA: ACM, pp. 209–216. DOI: 10.1145/258734.258849.

Garland, M. and Heckbert, P. S. (1998). "Simplifying surfaces with color and texture using quadric error metrics". *Proceedings Visualization '98*. IEEE and ACM. DOI: 10.1109/VISUAL.1998.745312.

Gauß, C. F. (1825). *General Investigations of Curved Surfaces*.

George, A. (1973). "Nested dissection of a regular finite element mesh". *SIAM Journal on Numerical Analysis* 10.2, pp. 345–363. DOI: 10.1137/0710032.

Gillespie, M., Sharp, N., and Crane, K. (Dec. 2021a). "Integer coordinates for intrinsic geometry processing". *ACM Transactions on Graphics* 40.6, pp. 1–13. DOI: 10.1145/3478513.3480522.

Gillespie, M., Springborn, B., and Crane, K. (July 2021b). "Discrete conformal equivalence of polyhedral surfaces". *ACM Transactions on Graphics* 40.4, pp. 1–20. DOI: 10.1145/3592401.

Glickenstein, D. (2005). "Geometric triangulations and discrete Laplacians on manifolds". *arXiv preprint*. URL: https://arxiv.org/abs/math/0508188.

Glickenstein, D. (2023). "Geometric triangulations and discrete Laplacians on manifolds: an update". *Computational Geometry*. DOI: 10.1016/j.comgeo.2023.102063.

de Goes, F., Memari, P., Mullen, P., and Desbrun, M. (June 2014). "Weighted triangulations for geometry processing". *ACM Transactions on Graphics* 33.3, pp. 1–13. DOI: 10.1145/2602143.

Gu, X. D., Guo, R., Luo, F., Sun, J., and Wu, T. (July 2018a). "A discrete uniformization theorem for polyhedral surfaces II". *Journal of Differential Geometry* 109.3, pp. 431–466. DOI: 10.4310/jdg/1531188190.

Gu, X. D., Luo, F., Sun, J., and Wu, T. (June 2018b). "A discrete uniformization theorem for polyhedral surfaces". *Journal of Differential Geometry* 109.2, pp. 223–256. DOI: 10.4310/jdg/1527040872.

Gu, X. D., Luo, F., and Wu, T. (2019). "Convergence of discrete conformal geometry and computation of uniformization maps". *Asian Journal of Mathematics* 23.1. DOI: 10.4310/AJM.2019.v23.n1.a2.

Gu, X. D., Wang, Y., Chan, T. F., Thompson, P. M., and Yau, S.-T. (2004). "Genus zero surface conformal mapping and its application to brain surface mapping". *IEEE transactions on medical imaging* 23.8. DOI: 10.1109/TMI.2004.831226.

Guibas, L. and Stolfi, J. (Apr. 1985). "Primitives for the manipulation of general subdivisions and the computation of Voronoi". *ACM Transactions on Graphics* 4.2, pp. 74–123. DOI: 10.1145/282918.282923.

Guibas, L. J., Knuth, D. E., and Sharir, M. (1992). "Randomized incremental construction of delaunay and voronoi diagrams". *Algorithmica* 7.1, pp. 381–413. DOI: 10.1007/BF01758770.

Haken, W. (1961). "Theorie Der Normalflächen: Ein Isotopiekriterium Für Den Kreisknoten". *Acta Math.* 105.3-4.

Hass, J. and Trnkova, M. (Aug. 2020). "Approximating isosurfaces by guaranteed-quality triangular meshes". *Computer Graphics Forum (SGP)*. DOI: 10.1111/cgf.14066.

Hatcher, A. (2002). *Algebraic Topology*. ISBN: 978-0-521-79540-1.

Hilbert, D. (Jan. 1901). "Ueber flächen von constanter Gaussscher krümmung". *Transactions of the American Mathematical Society* 2.1, pp. 87–99. DOI: 10.2307/1986308.

Hirsch, M. W. and Mazur, B. (1974). *Smoothings of Piecewise Linear Manifolds*. Annals of Mathematics Studies 80. Princeton University Press. DOI: 10.1515/9781400881680.

Hoppe, H. (Aug. 1996). "Progressive meshes". *SIGGRAPH 1996*. New York, NY, USA: ACM, pp. 99–108. DOI: 10.1145/237170.237216.

Indermitte, C., Liebling, T. M., Troyanov, M., and Clémençon, H. (2001). "Voronoi diagrams on piecewise flat surfaces and an application to biological growth". *Theoretical Computer Science* 263.1, pp. 263–274. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(00)00248-6.

Jin, M., Wang, Y., Yau, S.-T., and Gu, X. D. (2004). "Optimal global conformal surface parameterization". *IEEE Visualization 2004*, pp. 267–274. DOI: 10.1109/VISUAL.2004.75.

Karcher, H. (2014). "Riemannian center of mass and so called Karcher mean". *arXiv preprint*. URL: https://arxiv.org/abs/1407.2087.

Kazhdan, M., Solomon, J., and Ben-Chen, M. (2012). "Can mean-curvature flow be modified to be non-singular?" *Computer Graphics Forum*. Vol. 31. DOI: 10.1111/j.1467-8659.2012.03179.x.

Kettner, L. (May 1999). "Using generic programming for designing a data structure for polyhedral surfaces". *Computational Geometry* 13.1. DOI: 10.1016/S0925-7721(99)00007-3.

Kharevych, L., Springborn, B., and Schröder, P. (Apr. 2006). "Discrete conformal mappings via circle patterns". *ACM Transactions on Graphics* 25.2, pp. 412–438. DOI: 10.1145/1138450.1138461.

Kirby, R. C. and Siebenmann, L. C. (July 1969). "On the triangulation of manifolds and the hauptvermutung". *Bulletin of the American Mathematical Society* 75.4, pp. 742–749. DOI: 10.1090/S0002-9904-1969-12271-8.

Kneser, H. (1929). "Geschlossene Flächen in Dreidimensionalen Mannigfaltigkeiten." *Jahresber. Dtsch. Math.-Ver.* 38.

Knöppel, F., Crane, K., Pinkall, U., and Schröder, P. (July 2013). "Globally optimal direction fields". *ACM Transactions on Graphics* 32.4. DOI: 10.1145/2461912.2462005.

Koutis, I., Miller, G. L., and Peng, R. (2014). "Approaching optimality for solving sdd linear systems". *SIAM Journal on Computing* 43.1, pp. 337–354. DOI: 10.1137/110845914.

Kronecker, L. (1886). *Lecture to the Berliner Naturforschung-Versammlung*. Attested by Weber, H. (1893), *Jahresbericht der Deutsche Mathematiker Vereinigung* (p.19).

Kuiper, N. H. (1955). "On C^1-isometric imbeddings. I & II". *Indagationes Mathematicae (Proceedings)* 58, pp. 545–556, 683–689. DOI: 10.1016/S1385-7258(55)50075-8.

Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L. C., and Dobkin, D. P. (1998a). "MAPS: multiresolution adaptive parameterization of surfaces". *SIGGRAPH 1998*. New York, NY, USA: ACM. DOI: 10.1145/280814.280828.

Lee, A. W. F., Sweldens, W., Schröder, P., Cowsar, L. C., and Dobkin, D. P. (July 1998b). "MAPS: multiresolution adaptive parameterization of surfaces". *SIGGRAPH 1998*. New York, NY, USA: ACM, pp. 95–104. DOI: 10.1145/280814.280828.

Lee, J. M. (2012). *Introduction to Smooth Manifolds*. 2nd ed. Vol. 218. Graduate Texts in Mathematics. Springer. ISBN: 978-1-4419-9981-8. DOI: 10.1007/978-1-4419-9982-5.

Levi, Z. and Zorin, D. (Nov. 2014). "Strict minimizers for geometric optimization". *ACM Transactions on Graphics* 33.6, pp. 1–14. DOI: 10.1145/2661229.2661258.

Lipman, Y. (July 2012). "Bounded distortion mapping spaces for triangular meshes". *ACM Transactions on Graphics* 31.4, pp. 1–13. DOI: 10.1145/2185520.2185604.

Liu, H.-T. D., Gillespie, M., Chislett, B., Sharp, N., Jacobson, A., and Crane, K. (July 2023). "Surface simplification using intrinsic error metrics". *ACM Transactions on Graphics* 42.4, pp. 1–17. ISSN: 0730-0301. DOI: 10.1145/3592403.

Liu, H. D., Kim, V. G., Chaudhuri, S., Aigerman, N., and Jacobson, A. (Aug. 2020). "Neural subdivision". *ACM Transactions on Graphics* 39.4, pp. 1–16. DOI: 10.1145/3386569.3392418.

Liu, H. D., Zhang, J. E., Ben-Chen, M., and Jacobson, A. (2021a). "Surface multigrid via intrinsic prolongation". *ACM Transactions on Graphics* 40.4. DOI: 10.1145/3450626.3459768.

Liu, H. D., Zhang, J. E., Ben-Chen, M., and Jacobson, A. (July 2021b). "Surface multigrid via intrinsic prolongation". *ACM Transactions on Graphics* 40.4, pp. 1–13. DOI: 10.1145/3450626.3459768.

Luo, F. (2004). "Combinatorial Yamabe flow on surfaces". *Communications in Contemporary Mathematics* 6.5, pp. 765–780. DOI: 10.1142/S0219199704001501.

MacNeal, R. H. (1949). "The Solution of Partial Differential Equations by Means of Electrical Networks". PhD thesis. California Institute of Technology. DOI: 10.7907/PZ04-5290.

Mainini, E. (2012). "A description of transport cost for signed measures". *Journal of Mathematical Sciences* 181, pp. 837–855. DOI: 10.1007/s10958-012-0718-2.

Milnor, J. (Sept. 1956). "On manifolds homeomorphic to the 7-sphere". *Annals of Mathematics* 64.2, pp. 399–405. DOI: 10.2307/1969983.

Mitchell, J. S., Mount, D. M., and Papadimitriou, C. H. (1987). "The discrete geodesic problem". *SIAM Journal on Computing* 16.4, pp. 647–668. DOI: 10.1137/0216045.

Moise, E. E. (July 1952). "Affine structures in 3-manifolds: V. the triangulation theorem and hauptvermutung". *Annals of Mathematics* 56.1, pp. 96–114. DOI: 10.2307/1969769.

Moise, E. E. (2013). *Geometric Topology in Dimensions 2 and 3*. 1st ed. Vol. 47. Springer. DOI: 10.1007/978-1-4612-9906-6.

Mosher, L. (Mar. 1988). "Tiling the projective foliation space of a punctured surface". *Transactions of the American Mathematical Society* 306.1, pp. 1–70. DOI: 10.2307/2000830.

Munson, T., Sarich, J., Wild, S., Benson, S., and McInnes, L. C. (2014). *Toolkit for Advanced Optimization (TAO) Users Manual*. Tech. rep. ANL/MCS-TM-322 - Revision 3.5. Argonne National Laboratory.

Myles, A., Pietroni, N., and Zorin, D. (July 2014). "Robust field-aligned global parametrization". *ACM Transactions on Graphics* 33.4, pp. 1–14. DOI: 10.1145/2601097.2601154.

Nash, J. (1954). "C^1 isometric imbeddings". *Annals of Mathematics* 60.3, pp. 383–396. DOI: 10.2307/1969840.

Nash, J. (1956). "The Imbedding Problem for Riemannian Manifolds". *Annals of Mathematics* 63.1, pp. 20–63. DOI: 10.2307/1969989.

Pascal, B. (1657). *Les Provinciales*. Bibliothèque nationale.

Penner, R. C. (Jan. 2012). *Decorated Teichmüller Theory*. QGM Master Class Series. European Mathematical Society, Zürich. DOI: 10.4171/075.

Prosanov, R. (2020). "Ideal polyhedral surfaces in Fuchsian manifolds". *Geometriae Dedicata* 206. DOI: https://doi.org/10.1007/s10711-019-00480-y.

Radó, T. (1925). "Über den Begriff der Riemannschen Fläche". *Acta Litt. Sci. Univ. Szeged* 2, pp. 101–121.

Rand, A. (2011). "Where and How Chew's Second Delaunay Refinement Algorithm Works." *CCCG*.

Regge, T. (1961). "General relativity without coordinates". *Il Nuovo Cimento (1955-1965)* 19.3. DOI: 10.1007/BF02733251.

Riemann, B. (1854). *On the Hypotheses which lie at the Bases of Geometry*. DOI: 10.1007/978-3-319-26042-6.

Rivin, I. (May 1994a). "Euclidean structures on simplicial surfaces and hyperbolic volume". *Annals of Mathematics* 139.3, pp. 553–580. DOI: 10.2307/2118572.

Rivin, I. (1994b). "Intrinsic Geometry of Convex Ideal Polyhedra in Hyperbolic 3-Space". *Analysis, Algebra, and Computers in Mathematical Research (Luleå, 1992)*. Vol. 156. Lecture Notes in Pure and Applied Math.

Rivin, I. (Jan. 1996). "A characterization of ideal polyhedra in hyperbolic 3-space". *Annals of Mathematics. Second Series* 143.1, pp. 51–70. DOI: 10.2307/2118652.

Roček, M. and Williams, R. M. (1984). "The quantization of Regge calculus". *Zeitschrift für Physik C Particles and Fields* 21.4, pp. 371–381. DOI: 10.1007/BF01581603.

Santambrogio, F. (2015). *Optimal Transport for Applied Mathematicians*. Vol. 87. Progress in Nonlinear Differential Equations and Their Applications. Birkäuser Cham. DOI: 10.1007/978-3-319-20828-2.

Sawhney, R. and Crane, K. (2017). "Boundary first flattening". *ACM Transactions on Graphics* 37.5, pp. 1–14. DOI: 10.1145/3132705.

Schaefer, M., Sedgwick, E., and Stefankovic, D. (2008). "Computing Dehn Twists and Geometric Intersection Numbers in Polynomial Time." *CCCG*. Vol. 20, pp. 111–114.

Schaefer, M., Sedgwick, E., and Štefankovič, D. (2002). "Algorithms for normal curves and surfaces". *International Computing and Combinatorics Conference*. Springer, pp. 370–380. DOI: doi.org/10.1007/3-540-45655-4_40.

Schindler, M. and Chen, E. (2012). "Barycentric Coordinates in Olympiad Geometry". *Olympiad Articles*, pp. 1–40.

Schroeder, W. J., Zarge, J. A., and Lorensen, W. E. (July 1992). "Decimation of triangle meshes". *SIGGRAPH 1992*. New York, NY, USA: ACM, pp. 65–70. DOI: 10.1145/142920.134010.

Shamai, G. and Kimmel, R. (July 2017). "Geodesic distance descriptors". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. DOI: 10.1109/CVPR.2017.386.

Shamos, M. I. and Hoey, D. (1975). "Closest-point problems". *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. IEEE, pp. 151–162. DOI: 10.1109/SFCS.1975.8.

Sharp, N. and Crane, K. (Aug. 2020a). "A Laplacian for nonmanifold triangle meshes". *Computer Graphics Forum (SGP)* 39.5. DOI: 10.1111/cgf.14069.

Sharp, N. and Crane, K. (2020b). "You can find geodesic paths in triangle meshes by just flipping edges". *ACM Transactions on Graphics* 39.6, pp. 1–15. DOI: 10.1145/3414685.3417839.

Sharp, N. and Crane, K. (Nov. 2020c). "You can find geodesic paths in triangle meshes by just flipping edges". *ACM Transactions on Graphics* 39.6. DOI: 10.1145/3414685.3417839.

Sharp, N., Gillespie, M., and Crane, K. (July 2021). "Geometry processing with intrinsic triangulations". ACM SIGGRAPH 2021 Courses. DOI: 10.1145/3450508.3464592.

Sharp, N., Soliman, Y., and Crane, K. (July 2019). "Navigating intrinsic triangulations". *ACM Transactions on Graphics* 38.4. DOI: 10.1145/3306346.3322979.

Shewchuk, J. R. (1997). "Delaunay refinement mesh generation". PhD thesis. Carnegie-Mellon Univ School of Computer Science.

Shewchuk, J. R. (2002). "What Is a Good Linear Finite Element? Interpolation, Conditioning, Anisotropy, and Quality Measures".

Springborn, B., Schröder, P., and Pinkall, U. (Aug. 2008). "Conformal equivalence of triangle meshes". *ACM Transactions on Graphics* 27.3, pp. 1–11. DOI: 10.1145/1360612.1360676.

Springborn, B. A. (Sept. 2019). "Ideal hyperbolic polyhedra and discrete uniformization". *Discrete & Computational Geometry* 64, pp. 63–108. DOI: 10.1007/s00454-019-00132-8.

Steinitz, E. (1908). "Beiträge zur Analysis situs". *Sitzungsberichte der Berliner Mathematische Gesellschaft* 7, pp. 29–49.

Sun, J., Wu, T., Gu, X. D., and Luo, F. (2015). "Discrete conformal deformation: algorithm and experiments". *SIAM Journal on Imaging Sciences* 8.3. DOI: 10.1137/141001986.

Thurston, D. and Yuan, Q. (2012). "Notes on Curves on Surfaces".

Thurston, W. P. (1997). "Hyperbolic Geometry and Its Friends". *Three-Dimensional Geometry and Topology*. Vol. 1. Princeton University Press, pp. 43–108. DOI: 10.1515/9781400865321-004.

Tietze, H. (Dec. 1908). "Über die topologischen invarianten mehrdimensionaler mannigfaltigkeiten". *Monatshefte für Mathematik und Physik* 19, pp. 1–118. DOI: 10.1007/BF01736688.

Troyanov, M. (1986). "Les surfaces Euclidiennes à singularités coniques". *L'Enseignement Mathématique* 32, pp. 79–94. DOI: 10.5169/seals-55079.

Troyanov, M. (1991). "Prescribing curvature on compact surfaces with conical singularities". *Transactions of the American Mathematical Society* 324. DOI: 10.2307/2001742.

Verhoeven, F., Vaxman, A., Hoffmann, T., and Sorkine-Hornung, O. (Mar. 2022). "Dev2pq: planar quadrilateral strip remeshing of developable surfaces". *ACM Transactions on Graphics* 41.3, pp. 1–18. DOI: 10.1145/3510002.

Watson, D. F. (Jan. 1981). "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes". *The Computer Journal* 24.2, pp. 167–172. DOI: 10.1093/comjnl/24.2.167.

Weiler, K. (Jan. 1985). "Edge-based data structures for solid modeling in curved-surface environments". *IEEE Computer Graphics and Applications* 5.1, pp. 21–40. DOI: 10.1109/MCG.1985.276271.

Whitehead, J. H. C. (Oct. 1940). "On C^1-complexes". *Annals of Mathematics* 41.4, pp. 809–824. DOI: 10.2307/1968861.

Wu, T. (2014). "Finiteness of Switches in Discrete Yamabe Flow". PhD thesis. Master's Thesis, Tsinghua University.

Xia, G. (2013). "The stretch factor of the Delaunay triangulation is less than 1.998". *SIAM Journal on Computing* 42.4, pp. 1620–1659. DOI: 10.1137/110832458.

Yeo, B. T., Sabuncu, M. R., Vercauteren, T., Ayache, N., Fischl, B., and Golland, P. (2009). "Spherical demons: fast diffeomorphic landmark-free surface registration". *IEEE transactions on medical imaging* 29.3. DOI: 10.1109/TMI.2009.2030797.

Yu, X., Lei, N., Wang, Y., and Gu, X. D. (2017). "Intrinsic 3D dynamic surface tracking based on dynamic ricci flow and teichmuller map". *IEEE International Conference on Computer Vision (ICCV)*. DOI: 10.1109/ICCV.2017.576.

Zhou, J., Tu, C., Zorin, D., and Campen, M. (2020). "Combinatorial construction of seamless parameter domains". *Computer Graphics Forum*. Vol. 39, pp. 179–190. DOI: 10.1111/cgf.13922.

Zhou, Q. and Jacobson, A. (2016). "Thingi10K: A Dataset of 10,000 3D-Printing Models". *arXiv preprint*. URL: https://arxiv.org/abs/1605.04797.

Appendix A

A Brief Introduction to Hyperbolic Geometry

Outside there is "absolute nothingness". And yet this round world cannot exist without the emptiness around it, not only because "inside" presupposes "outside", but also because it is out there in the "nothingness" that the scaffolding lies, determining with geometric precision the centres of the circular arcs which form the skeleton.

M.C. Escher [1958]

A.1 Models of Hyperbolic Geometry

Just as the sphere S^2 is a surface of constant curvature $K = +1$, the *hyperbolic plane* H^2 is a surface of constant negative curvature $K = -1$. Unlike S^2, there is no way to smoothly embed H^2 in Euclidean $\mathbb{R}^3$ *isometrically*, *i.e.*, without distorting its geometry [Hilbert 1901]. Instead, we must visualize it through one of several *models*, each of which faithfully represents only some of its geometric features. A good analogy is the Mercator projection of the globe, which

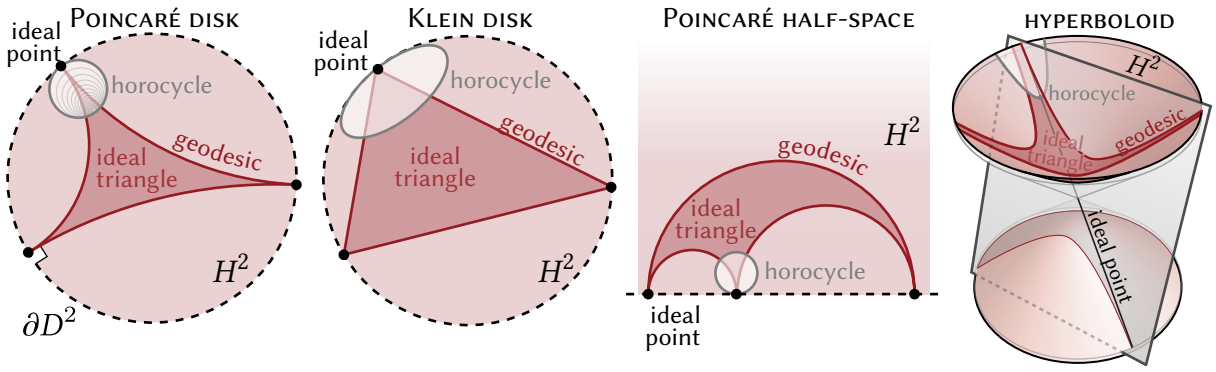


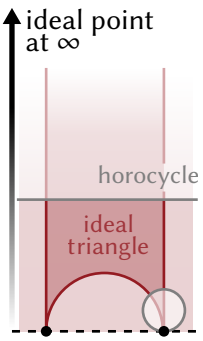
Figure A.1: Since the hyperbolic plane H^2 cannot be isometrically embedded in $\mathbb{R}^3$, it must be understood through the use of several "models"—here we illustrate how several key quantities are realized in each model.

preserves angles but distorts the size of land masses. Figure A.1 depicts three models that are useful for our purposes. For further background on hyperbolic geometry, see Cannon et al. [1997], Alekseevskij et al. [1993], and Thurston [1997].

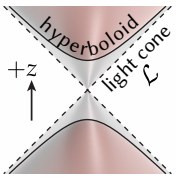
In the *Poincaré disk model*, points in H^2 are identified with points in the open unit disk $D^2 := \{p \in \mathbb{R}^2 : |p| < 1\}$. Although this disk looks like a finite piece of the Euclidean plane, lengths at a point $p \in D^2$ get scaled by $2/(1-|p|^2)$ so that short distances near the boundary ∂D^2 represent large distances in H^2. One can hence travel any distance along a straightest curve or *geodesic* without ever reaching the boundary—limit points on ∂D^2 are called *ideal points*. Though geodesics are straight in H^2, in the Poincaré model they appear as circular arcs orthogonal to ∂D^2. The Poincaré model is conformal: angles between circular arcs give the true angle between geodesics in H^2. Finally, just as a straight line in $\mathbb{R}^2$ can be viewed as a circle of "infinite radius," a *horocycle* is the limit of a family of increasingly large circles tangent at a common point—drawn in the Poincaré model as a circle tangent to the boundary.

The *Beltrami-Klein model* is much like the Poincaré model, but with a different metric. Geodesics appear as straight lines, but Euclidean angles no longer give the true angles in H^2, *i.e.*, the Beltrami-Klein model is not conformal. Horocycles in the Beltrami-Klein model appear as ellipses. This model helps explain the relationship between Euclidean and hyperbolic polyhedra (Appendix A.2.1).

In the *Poincaré half-space model*, the hyperbolic plane H^2 is identified with the upper half-space $\{(x, y) \in \mathbb{R}^2 : y > 0\}$. As in the disk models, distances get increasingly distorted as you approach the boundary—lengths around the point (x, y) get scaled by $\frac{1}{y}$. And just like in the Poincaré disk, geodesics appear as circular arcs orthogonal to the boundary, and horocycles appear as circles tangent to the boundary. The resemblance to the Poincaré model is no coincidence: the conformal map $f(z) = \frac{z-i}{z+i}$ is an isometry between the Poincaré half space and disk models (viewed as subsets of the complex plane $\mathbb{C}$). This mapping identifies the point $p = 1$ of the Poincaré disk model with a point at infinity in the half-space model, which we also identify as an ideal point of H^2. Ideal triangles with a vertex at this ideal point at infinity appear as vertical strips, and horocycles touching this ideal point appear as horizontal lines (see inset).

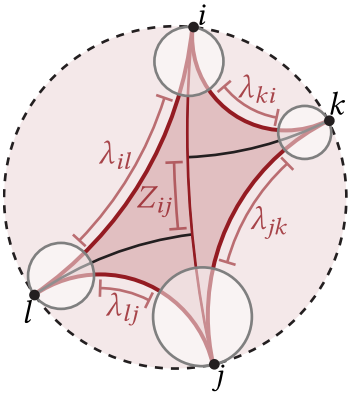


The *hyperboloid model* represents H^2 as the upper sheet of the two-sheeted hyperboloid. Just as the sphere is the set of all points $p \in \mathbb{R}^3$ such that $\langle p, p\rangle = 1$, this hyperboloid is the set of all points satisfying $\langle p, p\rangle_{2,1} = -1$, where $\langle p, q\rangle_{2,1} := p_x q_x + p_y q_y - p_z q_z$ is the *Lorentz inner product*; this inner product is also used to measure the angles and lengths of vectors tangent to the hyperboloid. Geodesics in H^2 correspond to intersections of the hyperboloid with planes through the origin, and ideal points are identified with lines in the *light cone* $\mathcal{L} := \{p \in \mathbb{R}^3 : \langle p, p\rangle_{2,1} = 0\}$. Horocycles are obtained by taking a plane tangent to $\mathcal{L}$, shifting it in the positive z-direction, and intersecting with the hyperboloid. Thus, we can identify horocycles with points in the *positive light cone* $\mathcal{L}^+ := \{p \in \mathcal{L} : p_z > 0\}$; each point $p \in \mathcal{L}^+$ also corresponds to the plane $\{q \in \mathbb{R}^3 : \langle p, q\rangle_{2,1} = -1\}$. The hyperboloid model is essential for developing our interpolation scheme—see Section 4.2.3.



A.2 Ideal Polyhedra

An *ideal hyperbolic polyhedron* is a surface of constant negative curvature, and a finite collection of *cusps* analogous to Euclidean cone points (Figure 4.9, *right*). We can construct ideal polyhedra by gluing together *ideal triangles*: regions of H^2 bounded by three geodesics approaching three ideal points at infinity (Figure A.1). A strange fact about ideal triangles is that they are all congruent, *i.e.*, they are identical up to isometries of H^2. Hence, the geometry of an ideal polyhedron is determined entirely by how neighboring triangles ijk, jil are glued together—namely, how far we slide them along the shared geodesic ij. One way to quantify gluings is to use *shear coordinates*, which for each edge ij give the distance $Z_{ij} \in \mathbb{R}$ between the altitudes dropped from opposite vertices k and l (see inset). Alternatively, we can pick an arbitrary horocycle at each vertex, yielding a *decorated ideal polyhedron*. Though edges of an ideal triangle do not have finite length, there is now a finite distance $\lambda_{ij} \in \mathbb{R}$ between the horocycles at i and j—these values are called the *Penner coordinates*. Shear and Penner coordinates are related by



$$Z_{ij} = \tfrac{1}{2}(\lambda_{il} - \lambda_{lj} + \lambda_{jk} - \lambda_{ki}) \tag{A.1}$$

(see [Penner 2012, Corollary 4.16, p. 40]). Note that if the horocycles at i and j overlap, λ_{ij} will be negative. Yet unlike negative Euclidean lengths, negative Penner coordinates will cause no trouble for discrete uniformization. Likewise, whereas Euclidean lengths must satisfy the triangle inequality, any three Penner coordinates $\lambda_{ij}, \lambda_{jk}, \lambda_{ki} \in \mathbb{R}$ (whether positive or negative) can be realized by some choice of horocycles.

A.2.1 Euclidean-Ideal Correspondence

Every Euclidean polyhedron gives rise to an ideal polyhedron, in the following way. Any triangle $ijk \in \mathsf{F}$ drawn in its Euclidean circumdisk can be interpreted as an ideal triangle in the Beltrami-Klein model. To glue two ideal triangles ijk, jil together along an edge ij, we simply identify the same points as in the Euclidean polyhedron. An ideal polyhedron constructed this way will have shear coordinates $Z_{ij} = \log \mathfrak{c}_{ij}$, and if we assign Penner coordinates

$$\lambda_{ij} = 2 \log \ell_{ij} \tag{A.2}$$

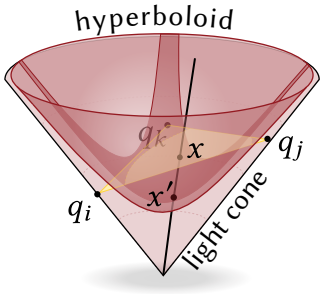
we get a decorated version of the same polyhedron. In general, we can move from Euclidean to hyperbolic polyhedra by "taking a logarithm"—for example, Equation (A.1) now just becomes the logarithm of the length cross ratio. More importantly, for a *fixed* triangulation, a conformal scaling of edge lengths corresponds to a shift in horocycles of the form

$$\tilde{\lambda}_{ij} = \lambda_{ij} + u_i + u_j. \tag{A.3}$$

In other words, conformally equivalent edge lengths $\ell, \tilde{\ell}$ describe the same ideal polyhedron, just decorated with different horocycles.

We can also view this correspondence through other models of hyperbolic space, which can be useful when performing calculations with Euclidean and ideal polyhedra.

Correspondence in the Hyperboloid Model In the hyperboloid model of H^2, three points q_i, q_j, q_k on different rays in the positive light cone $\mathcal{L}^+$ determine an ideal hyperbolic triangle decorated with horocycles at the vertices (Appendix A.1, Figure A.1). If we connect q_i, q_j, q_k by straight lines in $\mathbb{R}^3$, we obtain a secant triangle in the affine plane spanned by the three points, whose sides are chords of the light cone (see inset). This secant triangle can be identified with an ideal hyperbolic triangle on the hyperboloid model by central projection: given any point on the secant triangle, we can scale it to lie on the unit hyperboloid, and the collection of all such points defines a corresponding ideal hyperbolic triangle.



There is one subtlety involved here: we should not view the secant triangle as a Euclidean triangle in $\mathbb{R}^3$, but as a triangle in *Lorentz space* $\mathbb{R}^{2,1}$, equipped with the Lorentz inner product $\langle \cdot, \cdot \rangle_{2,1}$. The restriction of the Lorentz inner product to the affine plane spanned by q_i, q_j, q_K is positive definite so long as the Euclidean slope is less than 45°. In this case, the affine plane is called *spacelike* because the Lorentz inner product provides a Euclidean metric, turning the chord triangle in Lorentz space into a genuine Euclidean triangle with side lengths

$$\ell_{ij} = \tfrac{1}{2}\sqrt{\langle q_i - q_j, q_i - q_j \rangle_{2,1}}. \tag{A.4}$$

(See the Remark below for an explanation of the factor $\frac{1}{2}$). Since q_i, q_j are light-like (*i.e.*, $\langle q, q \rangle_{2,1} = 0$), we have

$$\langle q_i - q_j, q_i - q_j \rangle_{2,1} = -2\langle q_i, q_j \rangle_{2,1},$$

and therefore

$$\langle q_i, q_j \rangle_{2,1} = -2\ell_{ij}^2. \tag{A.5}$$

The affine plane spanned by q_i, q_j, q_k is spacelike if and only if the chord lengths $\ell_{ij}, \ell_{jk}, \ell_{ki}$ obtained from Equation (A.4) satisfy the triangle inequalities. This gives us a direct mapping between any Euclidean triangle and its ideal hyperbolic counterpart: take the Euclidean triangle, and find points q_i on the light cone such that $\|q_i - q_j\|_{2,1} = 2\ell_{ij}$, yielding a copy of the Euclidean triangle sitting in $\mathbb{R}^{2,1}$. Each point x in the Euclidean triangle (except the vertices, which are light-like) can be normalized to obtain a point $x' = x/\sqrt{-\langle x, x \rangle_{2,1}}$ on the unit hyperboloid, which we identify with the hyperbolic plane (see the inset at the beginning of this section).

We can express the determinant of the three light-like vectors q_i, q_j, q_k up to sign using terms of the Euclidean edge lengths as follows:

$$|\det(q_i\ q_j\ q_k)| = 4\,\ell_{ij}\ell_{jk}\ell_{ik}. \tag{A.6}$$

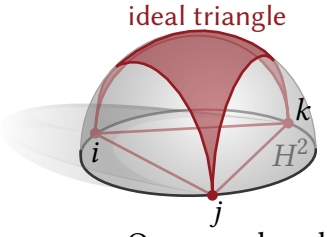
To derive this equation, note that

$$-2\begin{pmatrix} 0 & \ell_{ij}^2 & \ell_{ik}^2 \\ \ell_{ij}^2 & 0 & \ell_{jk}^2 \\ \ell_{ik}^2 & \ell_{jk} & 0 \end{pmatrix} = \begin{pmatrix} \langle q_i, q_i \rangle_{2,1} & \langle q_i, q_j \rangle_{2,1} & \langle q_i, q_k \rangle_{2,1} \\ \langle q_j, q_i \rangle_{2,1} & \langle q_j, q_j \rangle_{2,1} & \langle q_j, q_k \rangle_{2,1} \\ \langle q_k, q_i \rangle_{2,1} & \langle q_k, q_j \rangle_{2,1} & \langle q_k, q_k \rangle_{2,1} \end{pmatrix} = (q_i\ q_j\ q_k)^T \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} (q_i\ q_j\ q_k) \tag{A.7}$$

and take determinants.

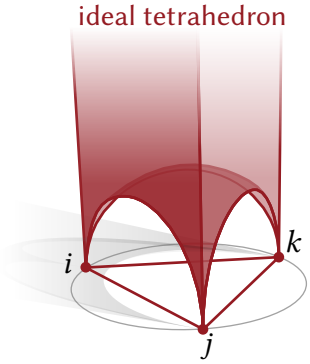
Remark. In Equation (A.4) measuring length in Lorentz space, we insert the global scale factor $\frac{1}{2}$ to be consistent with Equation (A.2) describing the relation between truncated hyperbolic lengths λ and Euclidean lengths ℓ. This relation was originally derived by Bobenko et al. [2015] (and used in this form by Springborn et al. [2008]) via the construction involving the half-space model described below. Both constructions provide the same correspondence between Euclidean and decorated ideal triangles, up to scale. The natural scale for Euclidean lengths in the construction of Bobenko et al. [2015] happens to differ from the natural scale in the light cone by a factor of 2.

Correspondence in the Half-space Model We can also understand the correspondence between Euclidean and ideal hyperbolic polyhedra by working in the half-space model of three-dimensional hyperbolic space H^3. This model identifies H^3 with the half-space $\{(x, y, z) : z > 0\}$. Again, distances about point (x, y, z) are scaled by a factor of $\frac{1}{z}$, and geodesics appear as circles orthogonal to the ideal boundary. Hemispheres orthogonal to the boundary provide copies of the two-dimensional hyperbolic plane H^2 embedded in three-dimensional hyperbolic space[1].



Any three ideal points i, j, k on the boundary plane $\{(x, y, z) : z = 0\}$ can be connected to form an ideal triangle embedded in H^3. If we take the circumcircle of the points in the plane, the hemisphere which orthogonally intersects the boundary along this circle is then a copy of the two-dimensional hyperbolic plane containing these three points in its ideal boundary, so the three points define an ideal triangle in this copy of H^2.

One can also obtain the horocycles decorating the vertices of this ideal triangle by considering the ideal hyperbolic tetrahedron with vertices i, j, k and the point at infinity. The key idea is that a horosphere at the vertex at infinity looks like a horizontal plane in the half space model, and the intersection of this plane with the ideal tetrahedron turns out to be a Euclidean triangle. We can pick our horosphere so that this triangle is congruent to the Euclidean triangle that we started with, and then we take horospheres at the other vertices tangent to this distinguished horosphere at the point at infinity—see Springborn [2019, Section 2] for more details.



A.2.2 Ptolemy Flip

As mentioned in Section 4.1.2, we can update an ideal polyhedron's Penner coordinates after an edge flip using *Ptolemy's relation* [Penner 2012, Corollary 4.16, p. 40]. It turns out that the formula is easiest to express in terms of the *Euclidean* edge lengths $\ell_{ij} = e^{\lambda_{ij}/2}$. We first evaluate

$$\ell_{kl} = (\ell_{ki}\ell_{lj} + \ell_{jk}\ell_{li})/\ell_{ij}, \tag{A.8}$$

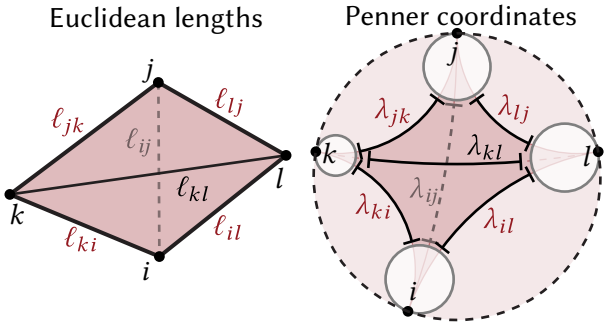
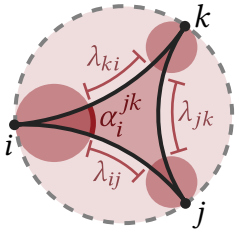


Figure A.2: An edge flip in an ideal hyperbolic polyhedron can be viewed in terms of Euclidean edge lengths *(left)*, or Penner coordinates *(right)*.

[1] This *hemisphere model* of the hyperbolic plane H^2 is another Klein-type model, but we will not go into the details here.

and then the new Penner coordinate is given by $\lambda_{kl} = 2\log(\ell_{kl})$ (Figure A.2, *top right*).

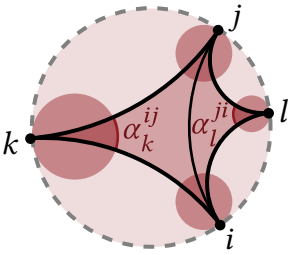
A.2.3 Ideal Delaunay Triangulations



Ideal Delaunay triangulation provide the natural hyperbolic analogue of Euclidean Delaunay triangulations[Springborn 2019, Section 4]. Ideal Delaunay triangulations can be characterized by a local condition almost identical to the Euclidean Delaunay condition given in Section 2.3.3, except we replace the angles with *horocyclic arc lengths*. Given a horocycle at vertex i of an ideal triangle ijk, we can measure the length α_i^{jk} of the segment of the horocycle contained inside of the ideal triangle ijk. One can show that the length is given by

$$\alpha_i^{jk} = e^{2\lambda_{jk}-\lambda_{ki}-\lambda_{ij}} = \frac{\ell_{jk}}{\ell_{ki}\ell_{ij}}. \tag{A.9}$$

Although this formula uses the values λ_{ij} and λ_{ki}—which depend on the horocycles at j and k in addition to the horocycle at i—the final result depends only on the choice of horocycle at i. If we modify the Euclidean edge lengths by any discrete conformal scale factor $u : \mathsf{V} \to \mathbb{R}$ (*i.e.* change the choice of horocycles at all vertices), then this horocyclic arc length simply gets scaled by a factor of e^{-u_i}, which depends only on the change of horocycle at vertex i itself.



We say that an edge ij then satisfies the local ideal Delaunay condition if

$$\alpha_k^{ji} + \alpha_l^{ij} < \alpha_i^{jk} + \alpha_i^{lj} + \alpha_j^{ik} + \alpha_j^{li}. \tag{A.10}$$

Note that if we replace the horocyclic arc lengths α_i^{jk} with the Euclidean corner angles θ_i^{jk} of a Euclidean triangle pair, then Equation (A.10) reduces to the standard Euclidean Delaunay condition. We can also substitute Equation (A.9) into Equation (A.10) to obtain the expression given in Equation (4.5) for the ideal Delaunay condition in terms of the Euclidean lengths ℓ_{ij}:

$$\ell_{ij}^2(\ell_{jk}\ell_{ki} + \ell_{il}\ell_{lj}) < (\ell_{il}\ell_{ki} + \ell_{jk}\ell_{lj})(\ell_{il}\ell_{jk} + \ell_{ki}\ell_{lj}). \tag{A.11}$$

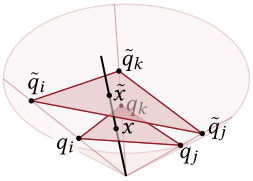
Just as we can find Euclidean Delaunay triangulations by repeatedly flipping any edge violating the local Euclidean condition, we can find ideal Delaunay triangulations by repeatedly flipping any edge violating the ideal Delaunay condition. And as discussed in Section 4.1.2, the lengths $\ell_{ij} = e^{\lambda_{ij}/2}$ arising from an ideal Delaunay triangulation are guaranteed to describe a valid Euclidean intrinsic Delaunay triangulation [Springborn 2019, p. 4.14].

Weighted Delaunay triangulations. At first glance, ideal Delaunay triangulations sound related to weighted Delaunay triangulations, as both provide Delaunay-like triangulations which depend on scalar values associated to the vertices of the triangulation. However, the two are distinct—the

ideal Delaunay triangulation associated to a set of scale factors $u : V \to \mathbb{R}$ is generally not the same as the weighted Delaunay triangulation associated to those weights. Weighted Delaunay triangulations are instead closely related to triangulations of more general objects *hyperideal hyperbolic polyhedra*: for more exploration of hyperideal polyhedra and discrete conformal maps, see the work of Bobenko & Lutz [2023] or Chen et al. [2016]

A.3 Light Cone Formulas

A.3.1 Vertex Scaling and Projective Interpolation



Now consider the points $\tilde{q}_i = e^{u_i} q_i$, $\tilde{q}_j = e^{u_j} q_j$, $\tilde{q}_k = e^{u_k} q_k$ on the same rays in the light cone, describing the same ideal triangle but decorated with different horocycles. By Equation (A.5), their chordal distances $\tilde{\ell}$ are related to the chordal distances ℓ by Equation (4.1). Moreover, if the scaled lengths $\tilde{\ell}$ satisfy the triangle inequalities, then the second triangle is also Euclidean and the circumcircle preserving projective map between them [Springborn et al. 2008, Section 3.4] is just central projection mapping a point x to the point $\tilde{x}$ in the same ray from the origin (see inset).

More explicitly, suppose we have a linear function on triangle $\widetilde{ijk}$ defined by values $\tilde{f}_i, \tilde{f}_j, \tilde{f}_k$ at the vertices. We want to pull this function back to the lower triangle ijk by defining $f(x) = \tilde{f}(\tilde{x})$. Suppose

$$x = \alpha_i q_i + \alpha_j q_j + \alpha_k q_k.$$

Since $\tilde{q}_i = e^{u_i} q_i$, we can also write

$$x = \alpha_i e^{-u_i} \tilde{q}_i + \alpha_j e^{-u_j} \tilde{q}_j + \alpha_k e^{-u_k} \tilde{q}_k.$$

To scale x to lie in the triangle spanned by $\tilde{q}_i, \tilde{q}_j, \tilde{q}_k$, we just have to normalize its coefficients to sum to 1:

$$\tilde{x} = \frac{\alpha_i e^{-u_i} \tilde{q}_i + \alpha_j e^{-u_j} \tilde{q}_j + \alpha_k e^{-u_k} \tilde{q}_k}{\alpha_i e^{-u_i} + \alpha_j e^{-u_j} + \alpha_k e^{-u_k}}.$$

Finally, we can evaluate our function f. Since $\tilde{f}$ is linear, we find that

$$f(x) = \tilde{f}(\tilde{x}) = \frac{\alpha_i e^{-u_i} \tilde{f}_i + \alpha_j e^{-u_j} \tilde{f}_j + \alpha_k e^{-u_k} \tilde{f}_k}{\alpha_i e^{-u_i} + \alpha_j e^{-u_j} + \alpha_k e^{-u_k}}.$$

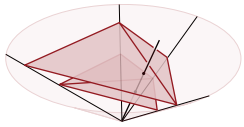
This is precisely the circumcircle-preserving projective map between our two triangles. We can write it more compactly by introducing *homogeneous coordinates*.

$$h(x) = \alpha_i e^{-u_i} (\tilde{f}_i, 1) + \alpha_j e^{-u_j} (\tilde{f}_j, 1) + \alpha_k e^{-u_k} (\tilde{f}_k, 1), \tag{A.12}$$

and we obtain $f(x)$ by dividing the first component of $h(x)$ by the second component. In the end, our interpolation amounts to linearly interpolating the values $e^{-u_i}(\tilde{f}_i, 1)$ and then performing this homogeneous divide.

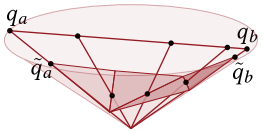
A.3.2 Edge Flips

The real power in the hyperboloid is that it allows us to interpolate between different triangulations of the same vertex set using exactly the same procedure. Consider for example a pair of triangles which have been flipped (by a Ptolemy flip) and rescaled.



In the hyperboloid model, the Ptolemy flip really does correspond to an extrinsic flip, since the extrinsic Lorentz distance corresponds to the hyperbolic distance between horocycles. So if we take two triangles, perform a Ptolemy flip, and then rescale the edge lengths, we end up with two pairs of triangles with one hanging above the other. We can map between the two triangle pairs by rescaling, exactly as in the 1-triangle case. The rescaling map is a piecewise-projective map on the common refinement of the two meshes. Since the map is piecewise-projective, we can specify the whole map by computing how much it scales by at each vertex, and at each projective intersection of edges. In this case, we can find the intersection point and the map's scale factor at the intersection by applying Equation (4.9).

A.3.3 Piecewise-Projective Interpolation



Now, suppose the triangulations differ by more than just an edge flip. As we observed above, the piecewise projective map depends only on scale factors at vertices, and at the intersections between edges of the two triangulations. We know the scale factors at vertices, so in order to compute the piecewise-projective map, we need to determine where the edges of the two triangulations intersect, and what the appropriate scale factor is at each intersection. In our algorithm, we need to trace edges of T^B over T^C. For each edge $ab \in \mathsf{E}^B$, this amounts to laying out the strip of triangles from T^C which ab crosses in the light cone, drawing the edge from q_a to q_b above it, and computing barycentric coordinates and scale factors for each intersection. Since $\tilde{q}_a = e^{u_a} q_a$, we simply place q_a and q_b by rescaling the triangle strip's endpoints by e^{-u_a} and e^{-u_b} respectively in Section 4.2.1. Once we have computed these scale factors, we do projective interpolation using Equation (A.12) on each triangle of the common refinement. The final expression appears as Equation (4.10).

A.3.4 Discrete Uniformization: Hyperboloid Model POV

To understand mapping between the intrinsic Delaunay triangulation (T^B, ℓ) and the discretely conformally equivalent intrinsic Delauny triangulation $(\mathsf{T}^C, \tilde{\ell})$ that is obtained by vertex scaling with logarithmic factors u and Ptolemy flips, it is useful to picture this process in the hyperboloid model as follows.

First, imagine laying out the triangulation T^B in the light cone. We will provide more detail in the following two sections, but the idea is straightforward: Place the vertices q_i, q_j, q_k of a first triangle ijk on arbitrary rays in $\mathcal{L}^+$ so that the chordal distances are ℓ_{ij}, ℓ_{jk}, ℓ_{ki}. Then for a neighboring triangle, say jil, the position $q_l \in \mathcal{L}^+$ of the third vertex is determined by the side lengths ℓ_{il}, ℓ_{jl}. Note that as you layout the triangles around one vertex, this will never close up. Instead, if you keep laying out, each triangle of T^B will correspond to infinitely many chordal

triangles. The result is a polyhedral surface P_1 with vertices in the light cone, all of which have infinite degree. Yet, every ray from the origin contained in the light cone will intersect this polyhedral surface exactly once. Moreover, the ideal Delaunay condition on (T^B, ℓ^B) is precisely the condition that this polyhedral surface is convex [Penner 2012, Lemma 1.7, p. 128].

The next step, corresponding to the vertex scaling (Equation (4.1)), is to slide all the laid out vertices along their rays in the light cone by applying the scale factors e^u as in Appendix A.3.1. The resulting polyhedral surface P_2 will in general not be convex, nor will all its triangles span spacelike planes.

The process of applying Ptolemy flips to obtain the ideal Delaunay triangulation $(\mathsf{T}^C, \tilde{\ell})$ corresponds, in the hyperboloid model, to applying extrinsic edge flips to the polyhedral surface P_2 to obtain a convex surface P_3. All of its triangles will then automatically be in spacelike planes. Finally, the map from P_1 to P_3 is just central projection from the origin.

A.3.5 Layout in the Light Cone I: Placing the First Triangle

We will now derive some practical equations for laying out a Euclidean triangulation in the light cone. Note that in practice we only ever lay out triangle strips of one triangulation that are crossed by an edge of another triangulation.

To lay out the first triangle ijk we place the vertices at the points

$$\begin{aligned} q_i &= w_i\,(1, 0, 1), \\ q_j &= w_j\,(\cos(2\pi/3), \sin(2\pi/3), 1), \\ q_k &= w_k\,(\cos(4\pi/3), \sin(4\pi/3), 1), \end{aligned}$$

in $\mathcal{L}^+$, where the positive scalar factors w_i, w_i, w_i are determined by the edge lengths via Equation (A.5):

$$\begin{aligned} \ell_{ij}^2 &= -\tfrac{1}{2}\langle q_i, q_j\rangle_{2,1} = \tfrac{3}{4} w_i w_j \\ \ell_{jk}^2 &= -\tfrac{1}{2}\langle q_j, q_k\rangle_{2,1} = \tfrac{3}{4} w_j w_k \\ \ell_{ki}^2 &= -\tfrac{1}{2}\langle q_k, q_i\rangle_{2,1} = \tfrac{3}{4} w_k w_i \end{aligned}$$

The solution of this system of equations is

$$w_i = \frac{2\,\ell_{ij}\ell_{ki}}{\sqrt{3}\,\ell_{jk}}, \qquad w_j = \frac{2\,\ell_{jk}\ell_{ij}}{\sqrt{3}\,\ell_{ki}}, \qquad w_k = \frac{2\,\ell_{ki}\ell_{jk}}{\sqrt{3}\,\ell_{ij}}\,.$$

A.3.6 Layout in the Light Cone II: Placing the Next Triangle

Suppose we have already determined the vertex positions $q_i, q_j, q_k \in \mathcal{L}^+$ of the triangle ijk, and we want to determine the position q_l of third vertex in the adjacent triangle jil. Note that q_i, q_j, q_k form a basis of $\mathbb{R}^3$ so and we can write the unknown vertex position q_l as a linear combination. To obtain more symmetric expressions, we will determine coefficients c_i, c_j, c_k, c_l for which

$$c_i q_i + c_j q_j + c_k q_k + c_l q_l = 0. \tag{A.13}$$

By taking the inner product of Equation (A.13) with each of the four vertex positions q and using Equation (A.5), we get a system of linear equations

$$\begin{bmatrix} 0 & \ell_{ij}^2 & \ell_{ik}^2 & \ell_{il}^2 \\ \ell_{ij}^2 & 0 & \ell_{jk}^2 & \ell_{jl}^2 \\ \ell_{ik}^2 & \ell_{jk}^2 & 0 & \ell_{kl}^2 \\ \ell_{il}^2 & \ell_{jl}^2 & \ell_{kl}^2 & 0 \end{bmatrix} \begin{bmatrix} c_i \\ c_j \\ c_k \\ c_l \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where $\ell_{ij}, \ell_{jk}, \ell_{ki}, \ell_{il}, \ell_{lj}$ are the sides lengths of the triangles ijk and jil, and ℓ_{kl} is determined by Ptolemy's formula (Equation (4.4)). A solution of this system is given by

$$c_i = \frac{\ell_{jl}\ell_{kl}}{\ell_{il}}, \quad c_j = -\frac{\ell_{ik}\ell_{kl}}{\ell_{jk}}, \quad c_k = -\frac{\ell_{jl}\ell_{ij}}{\ell_{jk}}, \quad c_l = \frac{\ell_{ik}\ell_{ij}}{\ell_{il}}.$$

We can use these coefficients in Equation (A.13) to get the next vertex position q_l.