# Periodic Conformal Parameterization

Connor Zhizhen Lin

CMU-CS-19-123
July 2019

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Keenan Crane, Chair
James McCann

*Submitted in partial fulfillment of the requirements*
*for the degree of Master of Science.*

**Keywords:** computer graphics, geometry processing, quadrangulation, remeshing, parameterization, singularities

# Abstract

This thesis describes a method for computing near-conformal integer grid maps with very low area distortion; such maps can be used to directly extract a high-quality quad mesh. Unlike previous methods, this formulation involves no mixed integer problems or other forms of combinatorial optimization. Instead, it entails two easy problems (a convex program and a linear system with Dirichlet boundary conditions) that can be solved efficiently and optimally. The basic idea is to replace the difficult texture rectification step from discrete conformal mapping, which is formulated in the parameter domain, with a purely intrinsic variational procedure inspired by methods for periodic global parameterization. We apply this algorithm to generate quad meshes for all models in a representative database, achieving high quality results at significantly lower computational cost. We also show how this formulation provides a unified perspective on a long history of problems in field-aligned parameterization and global conformal parameterization.

## Acknowledgments

I would like to thank my advisor Keenan Crane for introducing me to the exciting intersection between geometry and computer science. Thank you for guiding the project in the right direction as well as all the research and life advice - I will keep these with me long after I have left CMU.

A huge shoutout to Nick Sharp, who was always happy to answer questions and help me better understand so many technical details throughout the project. Working with Nick was like having a second advisor throughout my thesis.

Thank you to Jim McCann, who agreed to be a member of my thesis committee and provided great advice and feedback on the project. Thanks to the entire Geometry Collective (p.s. we're on Instagram!) for answering my questions, providing feedback, and sharing research as well. Let's continue to keep in touch in the future.

Last but not least, I would like to thank all my friends and family for being so supportive. I would not be where I am today without all of your help and insights.

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

The study of geometry representation and encoding is a central problem in the fields of computer graphics and geometry processing, and each representation has its own advantages and disadvantages. Polygonal meshes are perhaps the most common explicit representations, as they can be encoded with various data structures that make them relatively easy to manipulate and store. Furthermore, meshes support and work well with many modern algorithms for applications such as ray tracing and simulation. Specifically, a polygonal mesh can be thought of as a discretization of a smooth surface; a mesh is composed of a set of vertices, edges, and faces. Each vertex is a point in 3D space, an edge is a line segment with two vertices as endpoints, and each face is a polygon bounded by a cycle of edges. Triangle meshes (meshes that only contain triangles for faces) are the most common form of polygon meshes, and they are typically much more efficient for performance-intensive graphics applications like ray tracing. Although significantly less prevalent, quad meshes also have advantages for certain graphics applications. Quads can be better aligned with principal curvature directions and local sharp features, which means they are excellent for designing objects and characters used for videogames or animation. Furthermore, quad meshes are well-suited for texturing, as groups of quads can be easily mapped to a rectan-

gular texture. Coarse quad meshes, which contain fewer elements to reduce storage space, are also equipped to portray higher-level shapes of objects for surface modeling and compression. Finally, quads appear in textile design for automated fabrication and manufacturing [12].

However, most models of objects that one can download off the internet today are represented as triangle rather than quad meshes. At a high level, the goal of this thesis is to tackle the quad-remeshing problem by constructing a pipeline that generates a high-quality quad mesh given a triangle mesh as input. This is no easy feat, as quad remeshing algorithms can depend on a wide assembly of geometry processing machinery: parameterization, singularity placement strategies, surface cutting, uniformization, branch coverings, mixed integer solvers, and more (these will be covered in more detail in later sections). Furthermore, the output quad mesh needs to satisfy several desired properties in order to be suitable for the above applications. Namely, the output mesh should be a quad mesh that is seamless, nearly conformal (low angle area distortion), and has very low area distortion. The output should contain predominantly quads, have few irregular vertices, and depend as little as possible on the quality of the input. Current state of the art approaches perform well on simpler meshes, but tend to introduce new singularities or incur higher length distortion on more complex meshes. Quad remeshing algorithms capable of handling more difficult meshes heavily rely upon mixed integer solvers, which can take a significant amount of computation time and provide no guarantees on the optimality of the output. This thesis seeks to build a robust quad remeshing pipeline that avoids integer solvers and can handle complex meshes, while still producing a quad mesh that satisfies the desired properties listed above.

## 1.2 Background and Related Work

### 1.2.1 Parameterization

Quad meshing and parameterization are deeply related problems. At its core, the goal of parameterization is to flatten a surface in 3D space into the 2D plane (See Figure 1.1).

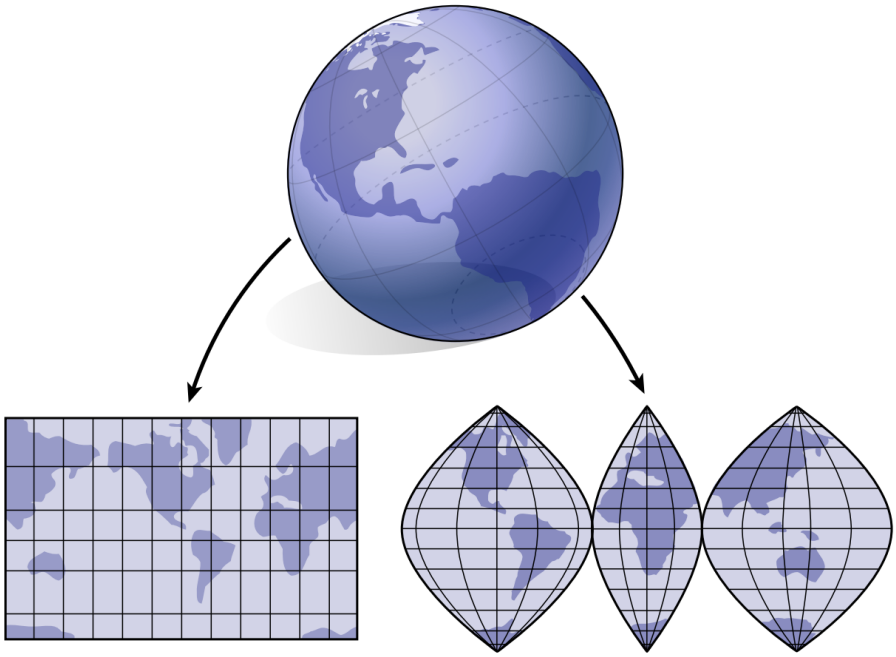


Figure 1.1: A classic example of surface parameterization is making a map of the Earth! [6]

More specifically, each vertex of the mesh should be mapped to $(u, v)$ 2D coordinates. Parameterization and quad meshing are closely related, as one can draw a Cartesian grid of integer isolines onto the resulting $uv$ map such that the grid stitches to a quad mesh on the surface. In order for the isolines to stitch together continuously (i.e. seamlessness), points $p_A, p_B$ on either side of the stitching should satisfy grid automorphism transition functions of the form:

$$p_A = R_{90}^i p_B + (j, k)^T \tag{1.1}$$

where $i, j, k \in \mathbb{Z}$ and $R_{90}$ is the 2D matrix excoding a 90 degree rotation. This condition ensures that $p_A$ and $p_B$ are similar with respect to the Cartesian grid of integer isolines such that no

distortion is introduced. An important detail for this parameterization approach to quad meshing is that in order to be able to stitch a mapping back together, one first has to cut the surface first into individual patches with disk topology. However, special singular points can exist where several seams intersect, breaking the regular isoline sampling pattern even if 1.1 is satisfied. The placement of these singularities is an important decision and will be discussed in further detail in the next section. Both the surface cutting and singularity selection steps significantly influence the resulting angle and area distortion - there exists a substantial amount of literature and work on these two problems alone. Fortunately, our pipeline formulation does not need to perform a surface cutting step.

Furthermore, parameterization in general is a difficult task because the mapping should be able to be computed quickly while also minimizing distortion. A mapping that preserves edge lengths (and therefore angles as well) is isometric, while a mapping that only preserves angles is conformal. Although it is not always possible to find an isometric map, Riemann's theorem states that in the smooth setting, there always exists a perfectly conformal map for any surface homeomorphic to a disk, which is achiveable via surface cutting. It is also important to note that perfectly conformal maps do not always exist in the discrete mesh setting, as there exist additional constraints that edges must be straight and the mapping must vary linearly within faces. However, as the mesh becomes smoother under additional refinement, conformal mapping algorithms will converge towards perfect conformality.

Conformal maps are our mapping of choice in this pipeline, since they can typically be computed quickly and there exist techniques for picking singularities that exactly minimize the area distortion of a conformal map. This allows the pipeline to essentially get the best of both worlds by minimizing both angle and area distortion. However, our pipeline does not necessarily demand optimal singularity placement to produce good quality results.

### 1.2.2 Singularities

Singular vertices are closely related to the output quality of the parameterization. Any parameterization that follows the constraints of 1.1 induces a metric (i.e. edge lengths) that is flat away from singular vertices. In other words, the Gaussian curvature is 0 at every non-singular point, where the Gaussian curvature $K_i$ of an interior vertex $i$ is defined as $2\pi$ minus the sum of the angles around it (and $\pi$ minus the angle sum for boundary vertices):

$$K_i = 2\pi - \sum_{ijk \in F} \phi_i^{jk} \tag{1.2}$$

The Gaussian curvature of a singular vertex resembles that of a cone point, and it is common to refer to singularities as cone points. One approach to surface flattening is to designate a few of the vertices as cone singularities such that the entire Gaussian curvature of the mesh is concentrated at those singularities, leaving the mesh flat everywhere else. This is useful for quad meshing techniques that cut the surface to a disk-like topology with singularities on the boundary, as it allows for a seamless stitching (see Figure 1.2). However, picking the location of the singular vertices is an important task, as their placement can greatly affect the resulting area and angle distortion. Singular vertices are also closely associated with the resulting quad mesh as well; every vertex with a surrounding angle sum of $k\frac{\pi}{2}$ is incident to $k$ edges. Therefore, points in flat regions will be regular with degree 4, whereas singular points will be irregular. For example, the vertices at the corners of a cube have an angle sum of $3\frac{\pi}{2}$, and therefore are incident to 3 edges in the resulting quad mesh.

Selecting the set of singular vertices is difficult, as the singularities are needed for the parameterization, but the parameterization is also needed to measure the distortion and determine the location of the singularities. However, greedy iterative processes like the ones taken in Conformal Flattening by Curvature Prescription and Metric Scaling [1] and Conformal Equivalence of Triangle Meshes [16] estimate the resulting distortion by computing a scale factor on each edge using the Yamabe equation. This scale factor indicates how much the edge length needs to change in order to obtain the desired Gaussian curvature, and points near the extrema of these

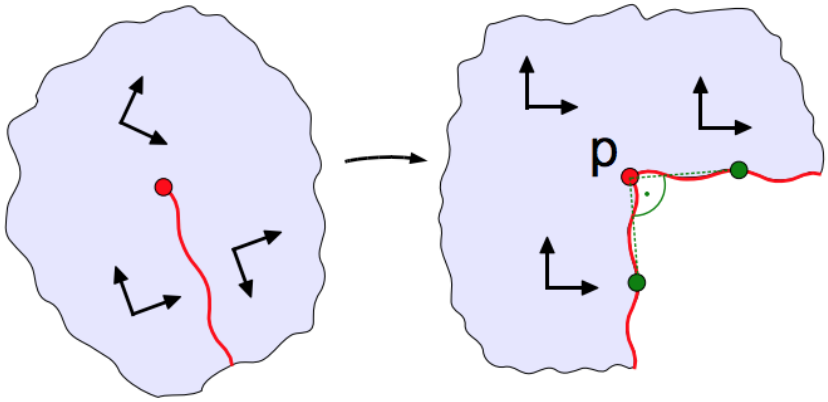



Figure 1.2: Seamless stitching along a cut to a cone singularity. [3]

scale factors are then chosen to be singularities. Direction field approaches like Globally Optimal Direction Fields [9] instead compute a smooth direction field on the surface and then extract the resulting singularities from the generate field. The intuition here is that a field with properties such as being globally smoothest produces a small number of singularities. Finally, the current state-of-the-art approach in a conformal map setting is Optimal Cone Singularities for Conformal Flattening [15], which formulates selecting singularities as a PDE-constrained optimization problem. Similar to the greedy approaches, the Yamabe equation for conformal scale factors reappears as a constraint in the optimization problem. A convex relaxation is applied, and the solution yields singularities that provably minimize the area distortion.

### 1.2.3 Non-Parameterization Methods

Before delving into quad meshing methods that use parameterization, it is worth briefly discussing a few approaches that do not involve parameterization. An intuitive and straightforward approach is tri-to-quad; these approaches primarly revolve around fusing two triangles of the original triangle mesh into a single quad or subdividing the surface. However, the resulting quad mesh will usually contain a large number of irregular vertices. Furthermore, the output quality strongly depends on the quality of the input (see Figure 1.3).

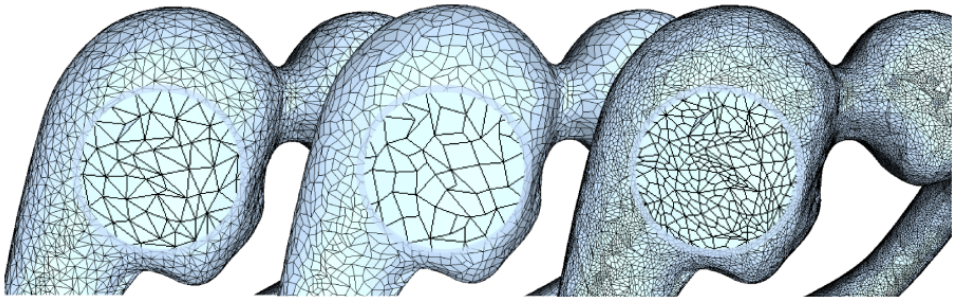


Figure 1.3: Left: input mesh. Center: Tri-to-Quad conversion. Right: Catmull-Clark subdivision [4]

Patch based strategies perform a 1-to-1 mapping of the original surface onto a set of square patches. The quad mesh is generated by sampling each patch in parametric space or with a regular grid. Again, the shape of the output quad mesh strongly depends on the input quality. There also exist other methods such as paving that generates quads row-by-row [2] and Spectral Surface Quandrangulation [7], which uses the Morse-Smale complex for quadrangulation.

### 1.2.4 Field-Guided Methods

Throughout the years, this is the class of methods that most current state-of-the-art quad meshing algorithms fall under, including our proposed pipeline. These methods depend less on the quality of the input triangle mesh, as they allow more control over the orientation and size of each quad by specifying a guiding cross field. Each cross can be thought of as a pair of intersecting line segments that encode a parallelogram 1.4. Cross fields contain the same types of singularities

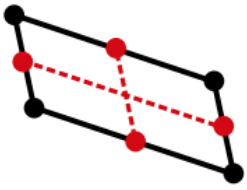


Figure 1.4: One way of encoding a parallelogram is by using a cross.

that can be observed in quad meshes, which means that a cross field with fewer singular points results in a more regular quad mesh.

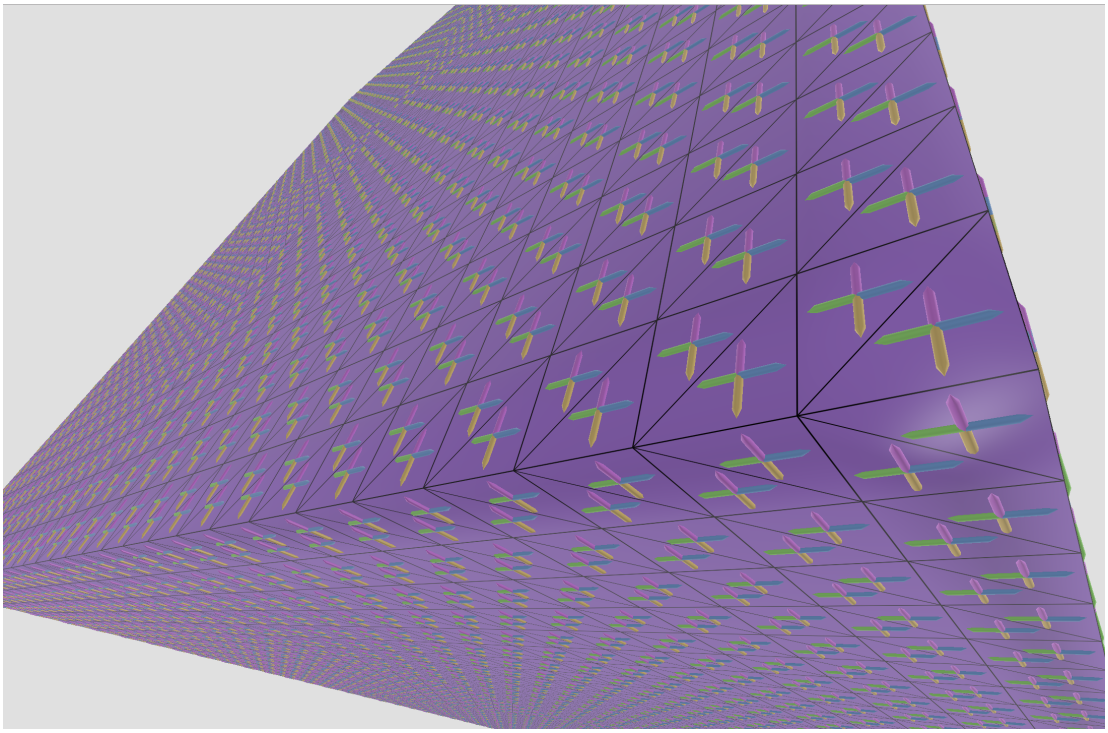


Figure 1.5: An example of a cross field on a cube.

Another advantage of field-guided methods is that they can split up the quad meshing problem into two simpler and often independent subproblems:

1. Orientation field generation
2. Quad mesh synthesis, usually involving parameterization guided by an orientation field

Since there exist many different methods for performing each of the above steps, there is a significant amount of room for experimentation with field-guided methods. Two quad meshing algorithms sharing similar machinery with our pipeline (QuadCover [8], PGP [13]) and a state-of-the-art quad meshing algorithm (MIQ [3]) will be briefly covered.

### 1.2.5 Orientation Field Generation

QuadCover views an orientation cross field as four interlinked unit vector fields $d_0, d_1, d_2, d_3$ which are pairwise anti-symmetric, i.e. $d_i = d_{(i+2) \bmod 4}$. Furthermore, $d_0$ and $d_1$ are orthogonal, which means that one can obtain the complete cross field by taking one branch $d_i$ and rotating by $\frac{\pi}{2}$ three times. To represent non-orientable regions near singular points of a cross field, QuadCover introduces the notion of matchings and branch coverings. A matching on a discrete mesh $M$ with edges $E$ is a mapping

$$r : \{e_{ij} \in E | T_i \cap T_j = e_{ij}\} \rightarrow \{0, 1, 2, 3\}$$

where $T_i, T_j$ are two adjacent faces that share the edge $e_{ij}$. The idea here is to directly encode the $k\frac{\pi}{2}$ degree rotation constraint of 1.1 into the mapping $r$. In QuadCover, the cross field and mapping $r$ are computed from the principal curvature directions with some additional smoothing.

A $n$-fold branch cover $\tilde{M}$ can be thought of as $n$ glued together copies of $M$. 4-fold branch covers share a natural relationship with cross fields, and $r$ can now be thought of as the sheet interchange function that determines how adjacent faces are glued together. For example, two faces $T_i, T_j \in M$ now correspond to $T_{i0}, T_{i1}, T_{i2}, T_{i3}$ and $T_{j0}, T_{j1}, T_{j2}, T_{j3} \in \tilde{M}$ . If $r_{ij} = 1$, then $T_{i0}$ would be connected to $T_{j1}$, $T_{i1}$ connected to $T_{j2}$, etc. (See Figure 1.6). Singular vertices are special in that there only exists a single copy of the vertex which is connected to elements on all other sheets. More details on branch cover construction and representation will be discussed in later sections. However, it is worth noting that in practice, $\tilde{M}$ never needs to be explicitly constructed, which saves memory and space.

Working with a 4-fold branch cover $\tilde{M}$ leaves one better equipped to handle singularities of fractional index on $M$. Traversing a small loop around the singularity simply rotates the cross field by $k\frac{\pi}{2}$, which is equivalent to jumping $k$ sheets of the branch cover (each sheet can be thought of as a 90 degree rotation from the previous sheet). This means that the orientation field is now continuous and orientable on $\tilde{M}$, which is important for the following quad mesh synthesis step.

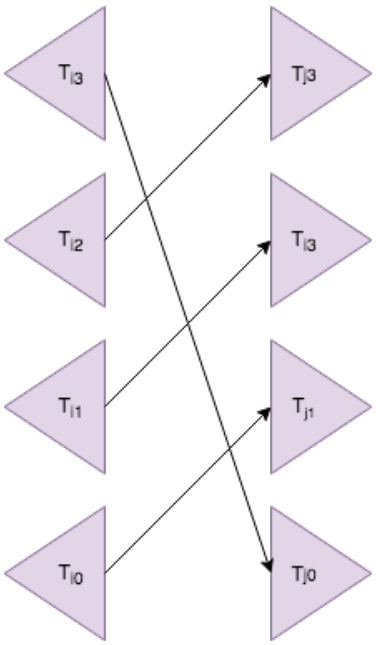



Figure 1.6: An example of a 4-fold branch cover with a sheet interchange of 1 between two adjacent faces $T_i$ and $T_j$.

PGP instead assumes that two orthogonal vector fields are provided as input. Similar to QuadCover, these are typically the estimated principal directions of curvature, but the only strict requirement is that they match the orthogonality constraint.

On the other hand, MIQ argues that using the principal curvature directions as a guiding field leads to suboptimal results. This is because near flat or umbilic points, the principal curvature directions are ill defined. Instead, MIQ searches for the smoothest cross field, using the $N$-Symmetry direction fields formulation [14] where a cross field ($N$ = 4) is defined by an angle field $\theta \in \mathbb{R}$ per face and a period-jump $p \in \mathbb{Z}$ per edge. By using a local coordinate frame for each face, one can use the angles $\theta$ to compute a unit length vector-field, which then extends to a symmetric cross field by applying three rotations of $\frac{\pi}{2}$. The period-jumps $p$ determine how crosses between two adjacent triangles line up, i.e. which vector of the first cross is associated

with which vector of the second cross, and is essentially the same idea as the sheet interchange function used by QuadCover. The smoothness energy is then defined as :

$$\sum_{e_{ij} \in E} (\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j)^2$$

where $\kappa_{ij}$ is the angle between the local frames of the two adjacent faces. The gradient of this energy can then be set to zero and formulated as a mixed-integer problem.

### 1.2.6 Quad Mesh Synthesis

Quad mesh synthesis is typically the trickier part of the two stages in field-aligned quad meshing. To synthesize a quad mesh, QuadCover computes a global parameterization function by first approximating the cross field by the gradient of a scalar function $\tilde{\phi}$. In other words, the following energy is minimized on the branch covering $\tilde{M}$ to find $\tilde{\phi}$ :

$$\int_{\tilde{M}} \|\nabla \tilde{\phi} - K\|^2 dA$$

where $K$ is the cross field lifted to the branch cover $\tilde{M}$, i.e. each cross is separated so that each of its four vectors belongs on the corresponding sheet. In order to compute a parameterization that minimizes the energy above, the mesh is cut to produce a surface topologically equivalent to a disk. $\tilde{\phi}$ is then modified to be globally continuous along the edges of the cut graph in order to output a function $\phi$ that maps from the mesh $M$ to $\mathbb{R}^2$. More specifically, a harmonic scalar function $\psi$ with minimal $L^2$ norm is added to $\tilde{\phi}$ to get the globally closed function $\phi$. To synthesize the quad mesh, $\phi$ can be used to produce a grid of integer iso-lines and mapped back to $M$.

PGP searches for a similar coordinate function $\phi$ with an additional term $\omega$:

$$\nabla \phi = \omega K$$

The goal is to find a periodic global parameterization in which translations are multiples of $2\pi$ and rotations are multiplies of $\frac{\pi}{2}$. Therefore, one can think of $\omega$ as controlling the period of $\phi$.

Essentially, PGP minimizes the same energy as QuadCover, except that the alignment energy between $\phi$ and the cross field $K$ is restated in terms of sine and cosine functions to handle the $2\pi$ periodicity. The result is globally continuous (except at singularities) by construction of minimizing a global energy functional.

Similar to QuadCover and PGP, MIQ minimizes the same alignment energy, but restated in terms of a mixed-integer problem. The $(u, v)$ parameter values on the two sides of a cut edge can be formulated as $1.1$, where $i, j, k$ are integer coefficients. After the cut, $i$ can be computed by propagating the orientation of each face in a breadth first search. Therefore, $j, k$ are the integer variables and the parameter coordinates $u, v$ are the non-integer variables in the mixed-integer problem. Singularities are then iteratively snapped to integer coordinates so that a pure quadrangulation is guaranteed.

The $(u, v)$ values are referred to as matchings, which are in principle essentially equivalent to branch covers when dealing with singular regions. Matchings determine a covering space, and optimization on a branch cover is no more costly than on the original surface due to conjugate symmetry. However, working with covering spaces allows for a meaningful formulation in the smooth setting, simplifies implementation, and helps clarify interpolation.

The above methods can be compared in terms of distortion and cost. The PGP method provides the best length distortion, but introduces additional singularities. The number of additional singularities can be reduced by a curl corrected sizing field. While QC and MIQ do not add new singularities, this comes at the cost of more length distortion. Furthermore, QC and MIQ share a trade-off between mapping distortion and runtime since they are based on the same function space construction. For coarser quad meshes, MIQ is superior because the integer estimation for the seamlessness condition $1.1$ has a high impact on the resulting distortion. Since QC employs a simpler heuristic for integer estimation, it is much faster with only two sparse linear systems, making it better suited for producing finer quad meshes. For a more detailed comparison of the three methods, see Figure 9 of [4].

## 1.3 Contributions

We present an algorithm for quad meshing that guarantees the following:

- The final parameterization is globally continuous in terms of rotation, scale, and translation, i.e., it is seamless and an integer grid map exhibiting the prescribed cones.
- We do not work with any mixed integer problems or other forms of combinatorial optimization, which makes the algorithm fast and easy to solve.
- The resulting quad mesh is almost conformal and produces very low area distortion, with very few to no new singularities.
- We also show how this formulation provides a unified perspective on a long history of problems in field-aligned parameterization and global conformal parameterization.

# Chapter 2

# Quad Meshing Pipeline

### 2.0.1 Overview

At a high level, this quad meshing pipeline consists of the following steps:

**(1) Singularity Selection**

We begin by selecting the location of singular vertices. This step affects the resulting distortion, but should not affect the robustness or other guarantees of our algorithm.

**(2) Uniformization**

Then, we uniformize the surface by switching to a cone metric, i.e. one that has zero Gaussian curvature away from singular vertices, which lets us find a constant harmonic cross field.

**(3) Branch Cover Construction**

We construct a branch covering of the uniformized surface to handle non-orientable regions, using the singular vertices to determine how sheets of the surface are connected to each other.

**(4) Global Parameterization**

Using the branch cover, we compute a global parameterization in order to generate a coordinate function for visualizing and extracting the resulting quad mesh.

**(5) Removing New Singularities**

Finally, we optimize to reduce the number of singularities appearing in the final quad mesh.

# Chapter 3

# Singularity Selection

### 3.0.1 Motivation

Because we are interested in a conformal mapping, we can take advantage of OCS [15], which places the optimal cones for conformal maps that minimize area distortion. Alternatively, one can also compute a smoothest cross field on the surface and extract the singular regions based on the field. The latter was chosen for testing purposes, as it was more straightforward to implement for experimentation. Furthermore, our quad meshing pipeline should not depend on having optimal cones, so this was a good check to perform. GOD [9] was adapted to compute a cross field on faces with singularities on vertices.

### 3.0.2 Cross Field Generation

By Euler's formula, a rotation of a complex number by $\theta$ can be represented by $e^{i\theta}$, which means that a cross field can be expressed as a collection of 4 complex functions:

$$\{e^{ik\pi/2}z, k = 0, 1, 2, 3\}$$

i.e. 4 copies of $z$, each rotated by some integer multiple of $\frac{\pi}{2}$. We can raise any of the above functions to the 4th power to get a single complex function:

$$u := z^4$$

and recover the individual vectors by taking the 4th roots. To compute the smoothest cross field, we compute a $u$ per face that minimizes the Dirchlet Energy 3.1.

$$E(u) = \frac{1}{2}\int_{\Omega} ||\nabla u(x)||^2 dx \tag{3.1}$$

The Dirichlet Energy measures how much $u$ changes across the domain $\Omega$. Therefore, we want to minimize this energy to obtain the smoothest cross field. We can discretize this energy by the following:

$$E(u) = \frac{1}{2}\sum_{T_i}\sum_{T_j \in N_{T_i}} |u_i - r_{ji}u_j|^2$$

where $N_{T_i}$ represents the neighbors of $T_i$ and $r_{ji}$ is the change of basis from the local coordinate frame of $T_j$ to the local coordinate frame of $T_i$. The change of bases $r$ can be computed by isometrically flattening each triangle $T_{ijk}$ and computing its 2D edge vectors $e_{ij}, e_{jk}, e_{ki}$. Then, the change of basis $r_{ij}$ from $T_{ijk}$ to $T_{jil}$ can be computed by:

$$r_{ij}e_{ij} = -e_{ji} \rightarrow r_{ij} = -\frac{e_{ij}}{e_{ji}}$$

Finally, the above energy can then be minimized using the inverse power method, which performs a linear solve at each step to obtain the smallest eigenvalue and its associated eigenvector.

### 3.0.3 Singularity Extraction

To determine where singular vertices arise in the resulting cross field, we look at the rotation in the cross field around each vertex:

$$\Theta_v = \sum_{T_i, T_{i+1} \in N_v} arg(\frac{u_{i+1}}{r_{i,i+1}u_i})$$

The singular index of a vertex is then determined by:

$$\psi_v = \frac{1}{2\pi}(\Theta_v + 4\Omega_v) \tag{3.2}$$

where $\Omega$ is the Gaussian curvature at vertex $v$. This quantity is guaranteed to be an integer satisfying Gauss-Bonnet and Poincare-Hopf, as explained in Appendix B of [9]. Essentially, the above equation computes whether $u$ jumps by an integer multiple of $2\pi$, i.e. if the cross field jumps by an integer multiple of $\frac{\pi}{2}$.

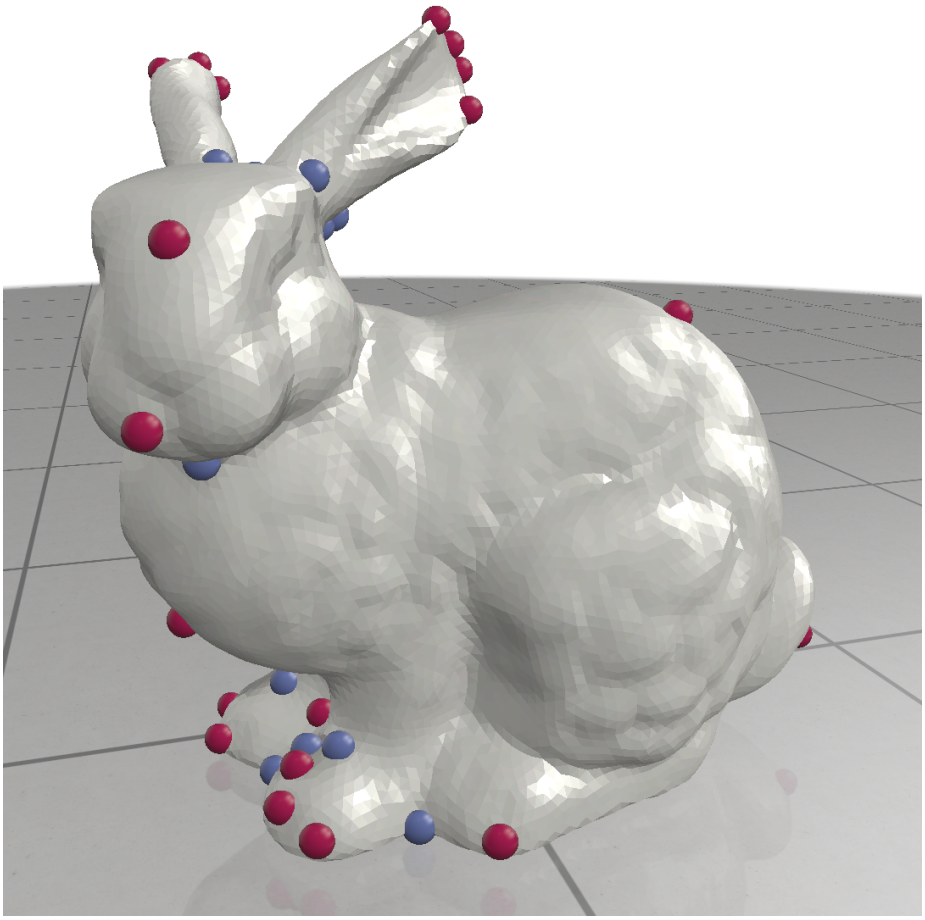


Figure 3.1: Example of computed singularities from the generated cross field. Red points are +1 index, and blue points are -1 index.

# Chapter 4

# Uniformization

### 4.0.1 Motivation

Ultimately, we want to compute a guiding vector field $\omega$ on the 4-fold branch cover of the surface $M$ that is locally integrable. In order for a vector field to be integrable, it needs to be curl-free. Since curl and divergence are related by 90 degree rotations of each other, being curl-free on one sheet means that the cross field is also divergence-free on the next sheet. This implies that $\omega$ is actually harmonic. On a uniformized surface, one can always find a constant cross field, and any constant cross field is also harmonic. The only interesting regions in the uniformized surface are around singularities - the constant cross field remains unchanged (rotating an orthogonal cross field by 90 degrees leaves it unchanged).

The idea behind uniformization is that we can change the metric $\ell$ of the surface to that of a cone metric:

$$\forall v \in V, \Omega_v = k\frac{\pi}{2}, k \in \mathbb{Z}$$

In other words, we want the resulting surface $\tilde{M}$ to be flat away from singularities. On such a polycube-like surface, one can always find a constant cross field by performing a BFS with two additional degrees of freedom in the initial scale and rotation of the cross field.

### 4.0.2 Uniformization Theory

Given a closed finite genus-$g$ discrete surface $(M, \ell)$ and a set of target curvatures $\Omega^*$, we want to compute a surface $(\tilde{M}, \tilde{\ell})$ that is discretely conformally equivalent and follows the prescribed curvatures. In order to define what it means to be discretely conformally equivalent, we can first take a look at what it means to be conformally equivalent in the smooth setting.

Two Riemannian metrics $\ell, \tilde{\ell}$ are conformally equivalent if they are related by a positive scaling, i.e., if $\tilde{\ell} = e^{2u}\ell$ for some function $u : M \to \mathbb{R}$ called the log conformal factor. In the discrete setting, we consider scale factors $u \in \mathbb{R}$ at each vertex and say that two discrete metrics $\ell, \tilde{\ell}$ are discretely conformally equivalent if for each edge $ij \in E$:

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

How do we know that such a $(\tilde{M}, \tilde{\ell})$ even exists? Fortunately, uniformization theory [5] states that for any closed finite genus-$g$ discrete surface $(M, \ell)$ with vertex set $V$, and any set of target cone angles $\Omega^*$ satisfying the Gauss-Bonnet condition

$$\sum_{v \in V} \Omega_v^* = 2\pi\chi$$

where $\chi$ is the Euler characteristic of the surface, there exists a discretely conformally equivalent surface $(\tilde{M}, \tilde{\ell})$ with $\tilde{\Omega} = \Omega^*$.

### 4.0.3 Target Curvatures

The desired target curvatures are directly related to the singular indices computed in section 3. If we sum 3.2 over the entire mesh, the rotation angles $\Theta$ cancel out and we are just left with the Gaussian curvatures, which again obey Gauss-Bonnet 4.1:

$$\sum_v \Omega_v = 2\pi\chi \tag{4.1}$$

$$\sum_v \psi_v = \frac{1}{2\pi}\sum_v \Theta_v + 4\Omega_v = \frac{1}{2\pi}4(2\pi\chi) = 4\chi$$

This is a discrete version of the Poincare-Hopf theorem on $n-$direction fields for $n = 4$, and it allows us to map a vertex with singular index $\psi_v$ to a target curvature of $\frac{\pi}{2}\psi_v$ for uniformization without violating Gauss-Bonnet:

$$\sum_v \Omega_v^* = \sum_v \frac{\pi}{2}\psi_v = \frac{\pi}{2}4\chi = 2\pi\chi$$

### 4.0.4 Uniformizing the Surface

In the smooth setting, one way of approaching uniformization is by following the Ricci or the closely related Yamabe flow in order to evolve the scale factors $u$ towards satisfying the target curvatures:

$$\frac{d}{dt}u(t) = \Omega(t) - \Omega^* \tag{4.2}$$

Differentiating 4.2 with respect to $u$ shows that this flow is also the gradient flow on a convex energy $E(u)$, and CETM [16] presents an explicit form for this energy. They show that the Hessian of $E(u)$ is equal to the cotangent discretization of the Laplace-Beltrami operator $\Delta$, and use this property to find the conformal scale factors $u$ that minimize $E(u)$ by using Newton's method. One can think of $\Delta$ as the second derivative, $\Omega - \Omega^*$ as the first derivative, and an initial guess of $u = 0$ for the scale factors. Explicitly, this can be implemented by solving the following linear system each iteration:

$$\Delta u = \Omega - \Omega^*$$

Then, the metric can be updated by the following discrete conformal rescaling:

$$\tilde{\ell_{ij_{t+1}}} = e^{(u_i+u_j)/2}\tilde{\ell_{ij_t}} \tag{4.3}$$

The Laplace-Beltrami operator $\Delta$ is a $|V| \times |V|$ matrix encoding the following expression for each vertex $v_i$, where $\alpha$ and $\beta$ are the two angles opposite edge $e_{ij}$.

$$\sum_{v_j \in N(v_i)} \frac{1}{2}(\cot(\alpha) + \cot(\beta))(v_i - v_j)$$

There is also a scaling ambiguity in the scale factors $u$, so at each iteration we subtract away the mean such that $\sum_v u_v = 0$. This iterative process is repeated until the L-infinity norm of $\Omega - \Omega^*$ is within some epsilon near zero, and converges very quickly (see Figure 4.1).

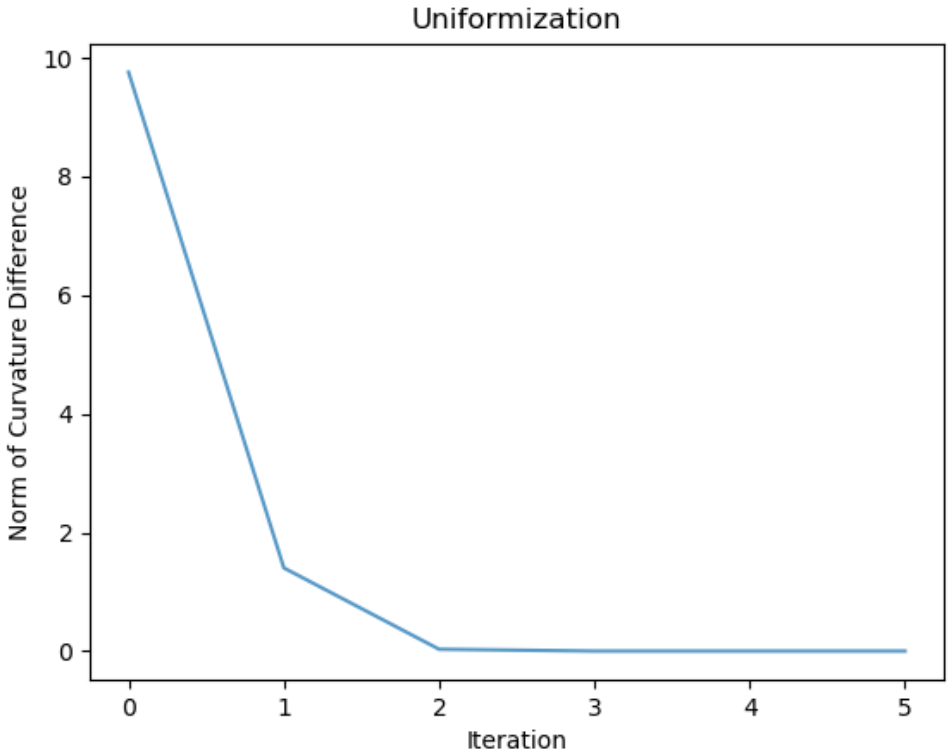



Figure 4.1: Norm of curvature difference per iteration of uniformization on a Bunny mesh.

### 4.0.5 Triangle Inequality Violations

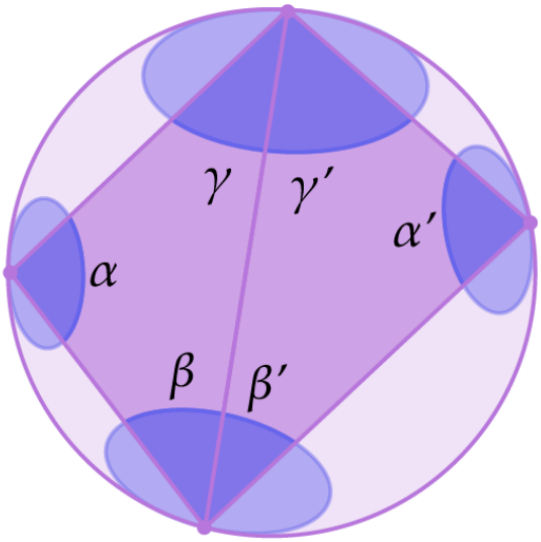
One important issue to consider when rescaling the metric using the algorithm above is that the triangle inequality can be violated, i.e. when one edge length exceeds the sum of the other two edge lengths in a triangle. Performing a Euclidean edge flip will modify the discrete conformal structure if the two adjacent triangles are not Delaunay (sum of angles opposite every interior edge is no greater than $\pi$) because the length cross ratio will be modified. However, we can avoid this by instead looking towards hyperbolic geometry and Ptolemy edge flips, which do not

require the intermediate triangulations to be Delaunay. These special edge flips also preserve the conformal structure of the surface by preserving length cross ratios. In other words, we can say that two triangulations are discretely conformally equivalent if they are related by a sequence of length scalings 4.3 and Ptolemy flips [5]. The exact Ptolemy flip condition and resulting edge lengths can be found in Figure 4.2. Because we are minimizing a convex energy and it has been proven that only a finite number of flips are needed to restore the condition, this uniformization algorithm always works and converges.

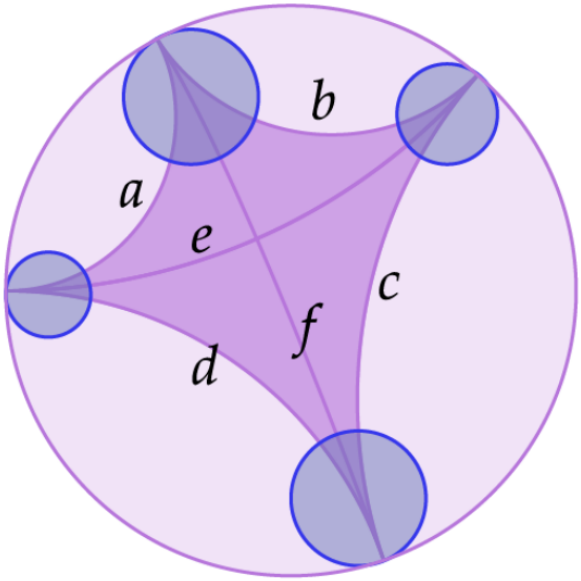
### 4.0.6 Guiding Harmonic Cross Field

Recall that uniformizing the surface $M$ to a cone metric allows one to find a constant cross field. Explicitly, this field can be computed by performing a breath first search on $M$. Starting at some arbitrary face $F_0$, we can lay down an initial vector $\omega_{F_0}$. Then, $\omega_{F_0}$ is transported trivially to its three neighboring faces using the same change of basis rotations from the branch cover construction. To ensure that $\omega$ is rotationally symmetric, the three remaining "spokes" of the cross are computed by rotating each vector by 90 degrees three times. However, it is important to note that this process only works perfectly ($\omega$ at face $i$ is equal to $\omega$ at an adjacent face $j$ transported to the basis of face $i$) for genus 0 surfaces. This is because our uniformization algorithm only ensures that there is going to be zero rotational monodromy around contractible loops, but gives no guarantees about non-contractible loops. Unfortunately, adding in additional constraints for zero rotational monodromy around non-contractible loops into the uniformization algorithm seems to overconstrain the system.

It is crucial that the surface be uniformized in order to find a harmonic vector field on $\tilde{M}$. Figure 4.3 shows what happens if we attempt to perform a breadth first search procedure on a non-uniformized surface.




$$\alpha + \alpha' \leq \beta + \beta' + \gamma + \gamma'$$




$$\ell_f = \frac{\ell_a \ell_c + \ell_b \ell_d}{\ell_e}$$

Figure 4.2: Ptolemy Edge Flips

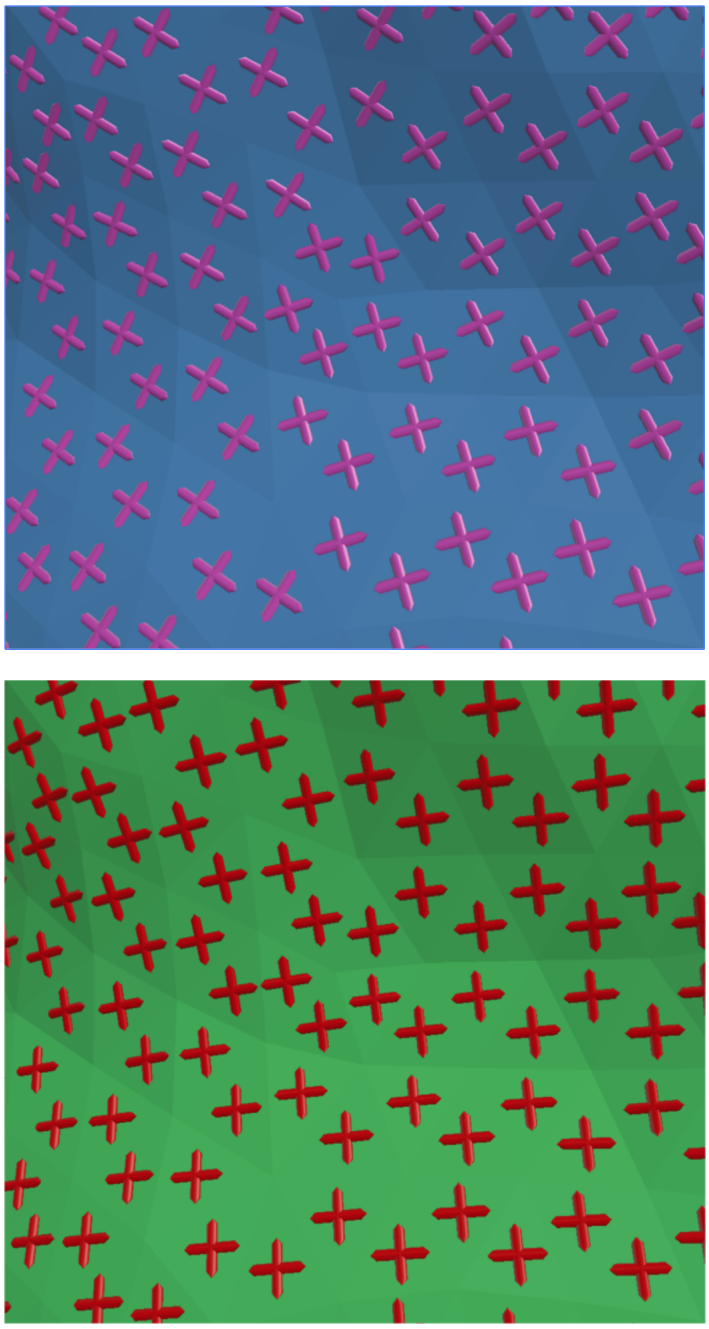


Figure 4.3: Top: BFS procedure without uniformizing the surface a priori, visualized as a cross field on the original surface $M$. Notice how the crosses do not all point in the same direction. Bottom: BFS procedure after uniformizing the surface.

# Chapter 5

# Branch Covering

### 5.0.1 Motivation

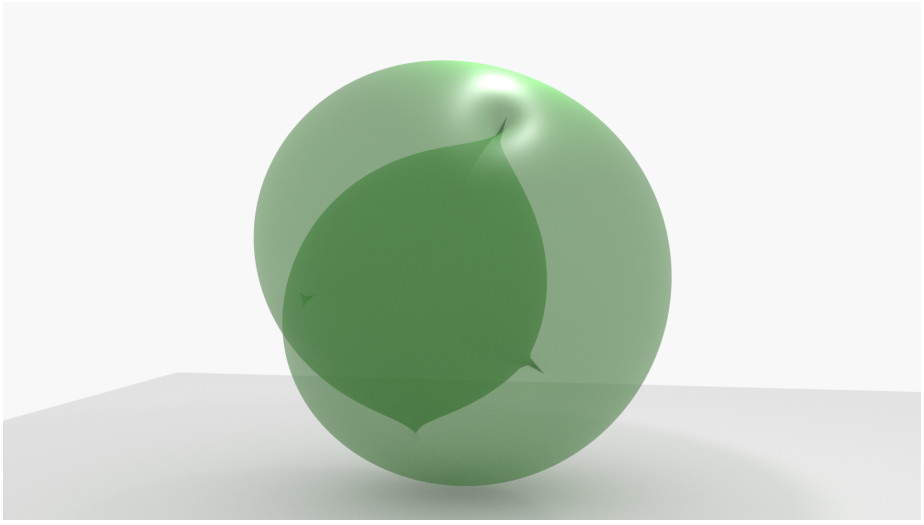
In order to handle non-orientable regions around singularities, we work with a 4-fold branch cover. 4-fold branch covers are no different from any other surface and can be thought of having 4 copies of the original surface, glued together. Although the 4-fold branch cover is somewhat difficult to visualize, the 2-fold branch cover is more intuitive to visualize, as given in Figure 5.1.

The harmonic cross field computed in the previous step can be lifted to a constant vector field $\tilde{\omega}$ on a 4-fold branch cover (see Figure 5.2) to attain non-ambiguous orientation. Cross fields share a natural relationship with 4-fold branch covers, as a 90 degree rotational "jump" around a singularity simply translates to going up a sheet in the branch cover.

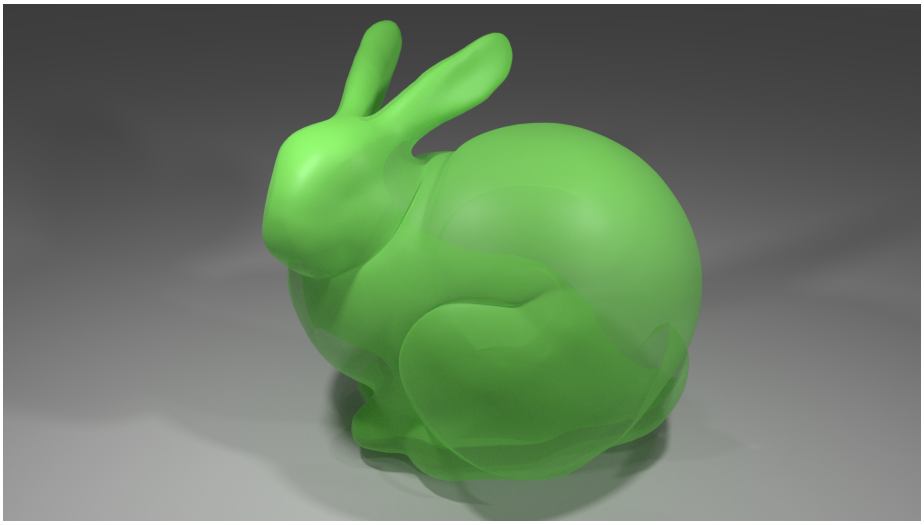
### 5.0.2 Branch Cover Description

After uniformizing the surface to have zero Gaussian curvature away from singularities, i.e. a cone metric, we now work in an intrinsic setting as the only information we have about the surface are its edge lengths.

One can think of a 4-fold branch cover as just another surface $\tilde{M}$, except that it contains



(a) Double Cover of a Sphere



(b) Double Cover of a Bunny

Figure 5.1: Two Visualizations of Double Covers

roughly 4 copies of the original surface $M$. There are 4 copies of each face, edge, and non-singular vertex with each living on one of 4 sheets, thus composing the 4 "sheets" of the branch cover. There is only 1 copy of every singular vertex, as singular vertices correspond to branch points that live on all 4 sheets of the covering. Away from these points the 4-fold cover has four points $x_1, x_2, x_3, x_4 \in \tilde{M}$ "above" each point $x \in M$ which are mapped back "down" to the base by the projection map $\pi(x_1) = \pi(x_2) = \pi(x_3) = \pi(x_4) = x$. The sheet interchange function

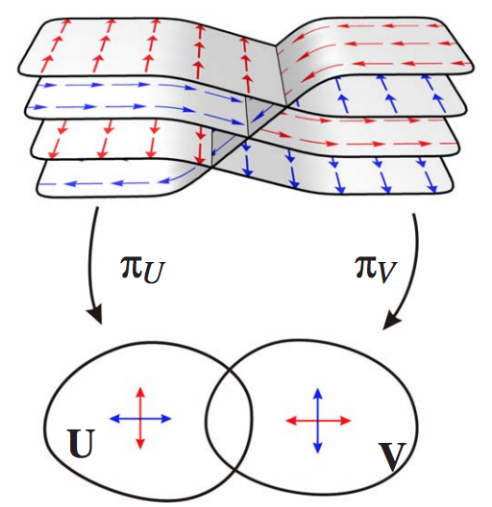



Figure 5.2: Frame field lifted to a vector field on the covering. [8]

$\tau(x_1) = x_2, \tau(x_2) = x_3, \tau(x_3) = x_4, \tau(x_4) = x_1$ describes how these points are connected.

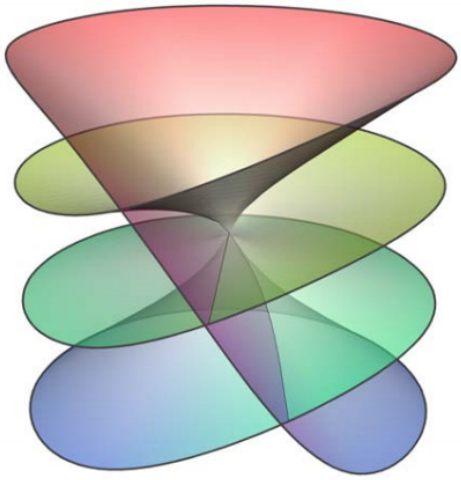


Figure 5.3: 4-fold branch covering around a branch point. [8]

### 5.0.3 Sheet Interchange Function from Singularities

One can also compute the sheet interchange function using only knowledge of the singularities, as the singularities themselves induce a unique branch covering. This is advantageous as we can

now construct a branch cover without making any assumptions on how the singularities were computed.

We first construct a spanning tree $T$ over the edges of the surface $M$. Then, $\eta$ is set to 0 for any edge not contained in $T$. Let $T_{leaf}$ be the collection of leaf nodes in $T$. Note that each node $p \in T_{leaf}$ has a single degree of freedom in $\eta$ for the edge $e_{pq} \in T$ connecting $p$ to its parent $q$. We then consider the sum $s$ of the $\eta$ values around $p$ (which is 0 initially), and set $\eta(e_{pq})$ such that $s$ is consistent with the singular index of $p$. This process is propagated up the tree, with new vertices $q$ being considered once all of $q$'s children have been processed. Because the sum of $\eta$ around each vertex $p$ is equal to the singular index of $p$ and the singular indices satisfy Gauss-Bonnet and Poincare-Hopf as explained in Chapter 4, this algorithm is guaranteed to give a consistent result at the root vertex.

### 5.0.4 Branch Cover Construction

After computing the sheet interchange function $\eta$, we can construct the branch cover. Because the 4-fold cover is essentially just another surface, it can be traversed in a similar fashion by tagging each halfedge, edge, vertex, and face with its sheet $s \in \{0, 1, 2, 3\}$. Therefore, we implemented a boilerplate branch cover halfedgemesh 5.4 class that obeys the following rules:

1. Each face has a sheet $s \in \{0, 1, 2, 3\}$.
2. The sheet of a halfedge is equal to the sheet of the face that the halfedge belongs to.
3. The sheet of a vertex is equal to the sheet of its canonical halfedge.
4. The sheet of an edge is equal to the sheet of its canonical halfedge.
5. Each singular vertex (i.e. branch point) is special in that it exists on all 4 sheets.

We can expose the exact same API for working with this augmented halfedgemesh data structure. The functions are mostly similar and just require some additional bookkeeping of the sheet value for each element. Some example code snippets are provided:

```
BHalfedge BHalfedge::next() {
        return BHalfedge{ he.next(), sheet };
}

BHalfedge BHalfedge::twin() {
        return BHalfedge{ he.twin(), (sheet + tau[he]) % 4 };
}

BVertex BHalfedge::vertex() {
        if (he == he.vertex().halfedge()) {
                int s = (isSingular(he.vertex())) ? 0 : sheet;
                return BVertex{ he.vertex(), s };
        }
        return twin().next().vertex();
}
```

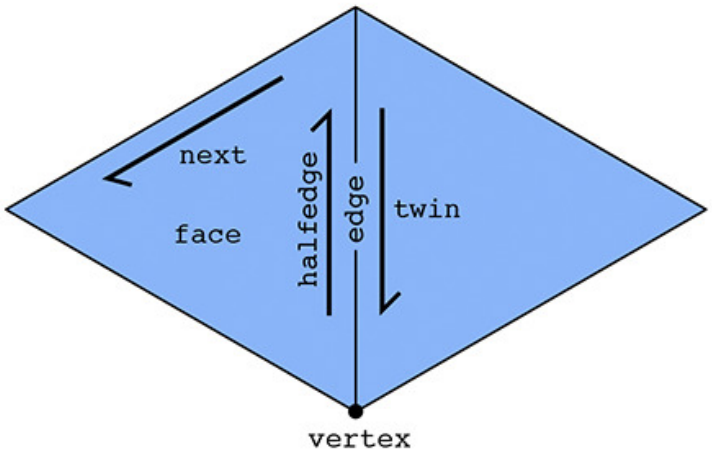



Figure 5.4: The halfedgemesh data structure encapsulates the connectivity of a polygon mesh.

# Chapter 6

# Global Parameterization

### 6.0.1 Smooth Formulation

Given a surface $M$, our goal is to compute a globally continuous parameterization, which means that we need to choose a representation for a coordinate function. We adapt the approach taken in PGP [13] and Stripe Patterns for Surfaces [10] and first consider a smooth unit tangent field $X$ on the unit circle $\theta \to (\cos\theta, \sin\theta)$ (Fig 6.1). Note that globally, $X$ cannot be expressed as the derivative of a smooth function because any such function would have to increase each time we walk around the circle. We can circumvent this by working instead with a complex coordinate function $\psi = e^{\imath\theta}$ that serves as a proxy for the real coordinate function $\theta$. $\psi$ is a periodic function with a well-defined derivative when viewed as a map to the unit circle, which means we have a representation of $\theta$ that is globally continuous and differentiable away from new singularities in the result. Unlike other approaches like QuadCover, we also no longer need to cut the mesh because coordinates are now globally continuous by construction.

We also want this parameterization to be as conformal as possible, which we can express as a form of the Cauchy-Riemann equation:

$$d\theta \circ \tau = \star d\theta \tag{6.1}$$

where $\tau^4 = id$ is the sheet interchange function on the 4-fold cover $\tilde{M}$ and $\star$ is the Hodge

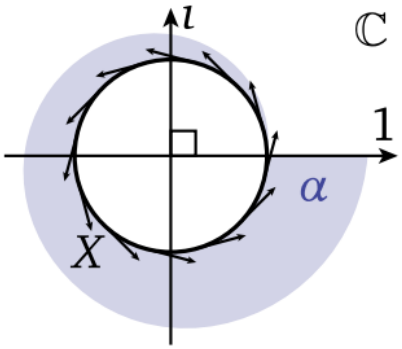



Figure 6.1: Locally, $X$ can be viewed as the derivative of the angle $\alpha$, but globally it cannot be expressed as the derivative of a smooth function [10]

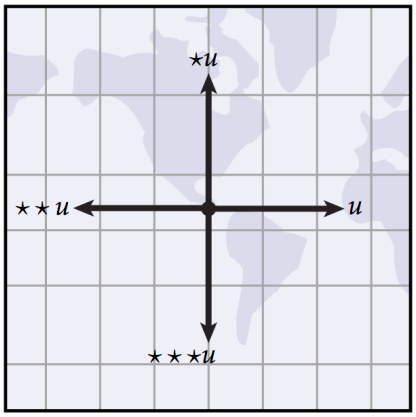



Figure 6.2: The Hodge star operator in 2D corresponds to a 90 degree CCW rotation. [6]

star operator, which corresponds to a 90 degree rotation in 2D (see Figure 6.2). Essentially, rotating the gradient of $\theta$ by 90 degrees on one sheet should give the same result as the gradient of $\theta$ on the next sheet above (recall that sheets encode 90 degree rotations). We also want the parameterization to be guided by our cross field $\omega$, i.e $\omega = d\theta$. Then,

$$\psi = e^{\imath\theta}$$

$$d\psi = \imath d\theta e^{\imath\theta} = \imath\omega\psi \tag{6.2}$$

We then consider its $L^2$ residual, which can be viewed as a Dirichlet energy on a complex

function $\psi$ with the constraint that $||\psi|| = 1$ to avoid the trivial solution $\psi = 0$:

$$\mathcal{E}(\psi) = \min_{||\psi||=1} \int_M |(d - \imath\omega)\psi|^2 dA \tag{6.3}$$

Note that if $\omega = d\theta$, then 6.1 is satisfied if $\omega \circ \tau = \star\omega$. So, we need to find an $\omega$ that is curl-free (since $\omega$ describes the gradient of a scalar function) and rotationally symmetric. Since curl and divergence are 90 degree rotations of each other, being curl-free on one sheet means that $\omega$ is also divergence-free on the sheet above. Therefore, $\omega$ is actually harmonic and can be computed as described in the previous chapter.

An important thing to note, however, is that perfectly conformal and periodic maps do not usually exist. For example, the flat torus with irrational edge lengths can be conformally mapped, but cannot be aligned such that its vertices sit in integer positions of an isoline grid (see Figure 6.3). Therefore, our periodic parameterization will seek to be as conformal as possible.

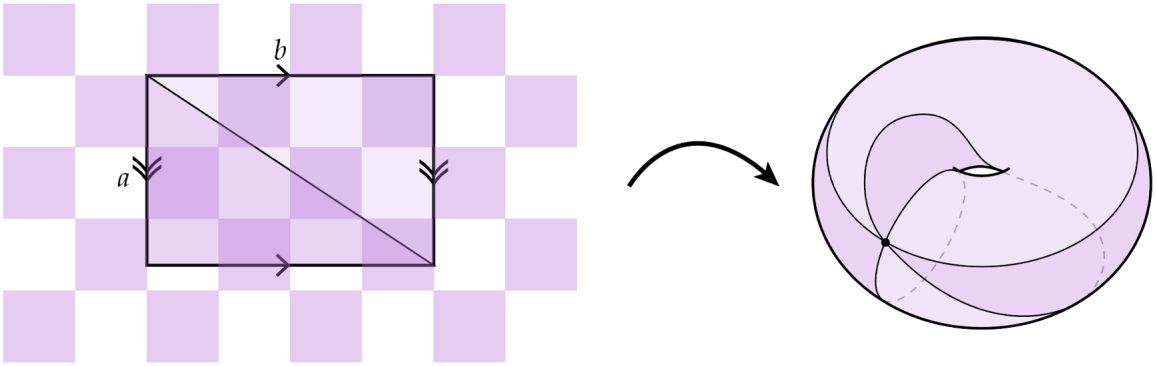



Figure 6.3: In general, perfectly conformal and periodic parameterizations do not usually exist.

### 6.0.2 Discretization

Give a surface $M = (V, E, F)$, we want to discretize and minimize 6.3 such that $d\theta = \omega$. The harmonic vector field computed in the previous step is a per face quantity. Let this guiding vector field on faces be $Z$. Following a similar approach to Stripe Patterns [10], we linearly interpolate

$Z$ and integrate along edges $e_{ij}$ to compute a discrete 1-form $\omega_{ij} = -\omega_{ji}$ [6].

$$\theta_j - \theta_i = \int_{ij} d\theta = \int_{ij} \omega = \frac{1}{2}(\langle e_{ij}, Z_i \rangle + \langle e_{ij}, Z_j \rangle) = \omega_{ij} \tag{6.4}$$

Since $\psi$ serves as a proxy function for the real coordinate $\theta$, we follow the Stripes discretization that asks the rotated value of $\psi_i$ to agree exactly with $\psi_j$:

$$e^{\imath\omega_{ij}}\psi_i = \psi_j \tag{6.5}$$

However, we cannot expect to satisfy this constraint exactly due to global nonintegrability. Although $\omega$ is locally integrable, it is not globally integrable because any coordinate function $\theta$ would have to increase each time it loops around the surface. Since $\omega$ cannot be expressed as exactly the derivative of $\theta$, we minimize the $L^2$ residual of 6.5:

$$\mathcal{E} = \sum_{ij \in E} w_{ij} |\psi_j - e^{\imath\omega_{ij}}\psi_i|^2 \tag{6.6}$$

where $w_{ij} = (\cot\alpha + \cot\beta)/2$ is the cotan weight along edge $e_{ij}$ that helps describe the shape of mesh elements [11].

### 6.0.3 Conjugate Symmetry

It is important to note that we are working with a 4-fold branch cover $\tilde{M}$, but we ultimately want texture coordinates on the original surface $M$. As explained in the Stripe Patterns paper, $\psi$ is conjugate symmetric:

$$\psi \circ \tau^2 = \overline{\psi}$$

where $\tau$ is the sheet interchange function that goes up a sheet (recall that sheets are related by 90 degree rotations). This is because we want the coordinate function $\theta$ to be antisymmetric with respect to sheet interchange, i.e. $\theta \circ \tau^2 = -\theta$.

$$\overline{\psi_i} = \overline{e^{\imath\theta_i}} = \cos(\theta_i) - \imath\sin(\theta_i) = \cos(-\theta_i) + \imath\sin(-\theta_i) = e^{-i\theta_i}$$

Stripe Patterns explains that the minimization of 6.6 always produces a $\psi$ that is conjugate symmetric, which means that we only need to work with two sheets of $\tilde{M}$ in the minimization and cleverly bake in the conjugate symmetry constraint. The energy defined by 6.6 can be built as a real matrix in order to represent conjugation of the stored values of $\psi$ - see Section 4.1 of Stripe Patterns for how this matrix is constructed. However, there are a couple key differences; we want to output texture coordinates, so the system now has twice the degrees of freedom (values of $\psi$ on both sheets 0 and 1). Furthermore, since we work with a 4-fold branch cover rather than a double cover, we cannot assign signs to edges the same way. Instead, each vertex $v_i$ has a sign $s_i$ that determines whether its imaginary component should be negated for conjugate symmetry:

$$s_i = \begin{cases} 1 & \text{if sheet}(v_i) \in \{0, 1\} \ , \\ -1 & \text{otherwise} \end{cases}$$

When building the Stripe Patterns energy matrix, operations of the form $\langle zu_i, v_j \rangle = \begin{bmatrix} a & b \end{bmatrix} [z] \begin{bmatrix} c & d \end{bmatrix}^T$ for complex numbers $u_i = a + b\imath$, $v_j = c + d\imath$, $z = x + y\imath$ now take the form:

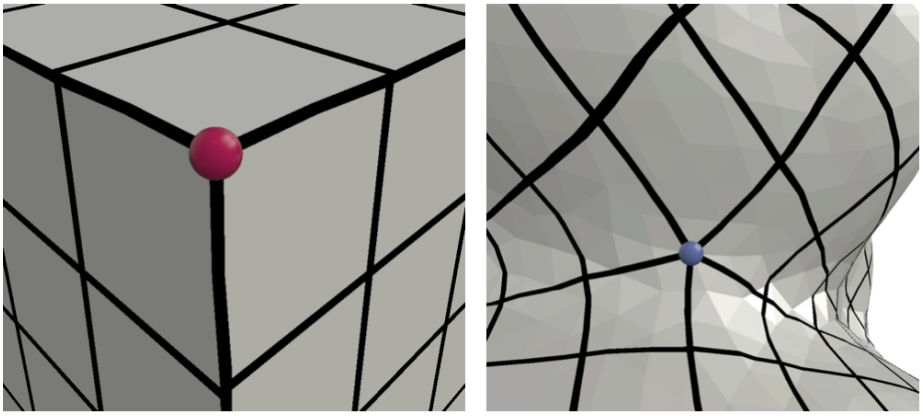
$$[z] := \begin{bmatrix} x & s_j y \\ -s_i y & s_i s_j x \end{bmatrix}$$

### 6.0.4 Boundary Conditions

For boundary edges, the unknown cotan weight is set to zero, which corresponds to zero-Neumann boundary conditions, i.e. setting the derivative of the solution to 0 on the boundary. Furthermore, branch points are handled by setting the weights of their neighborhood of faces to zero. Essentially, we cut out the neighborhood of each branch vertex from the surface, leaving only its link (See Figure 6.5).

Up to this point, the discretization has mostly followed the process outlined in Stripe Patterns. However, there are additional boundary conditions that fix the solution around branch points. As a result, the energy can be minimized by solving a single linear system with Dirichlet boundary conditions rather than a generalized eigenvalue problem with inverse power iterations.

Around a valence four vertex $v$, we want to have four isolines meeting at v. For a branch vertex with singular index +1 or -1, we want to have three isolines and five isolines, respectively (see Figure 6.4).



(a) Left: +1 singularity, Right: -1 singularity.

Figure 6.4: Desired Isoline Structure around Branch Points.

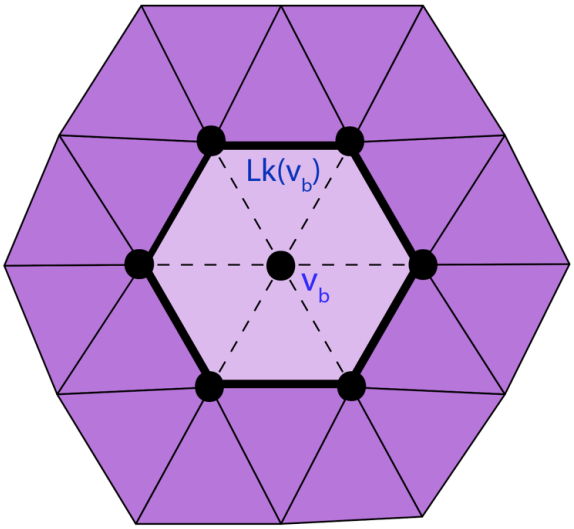
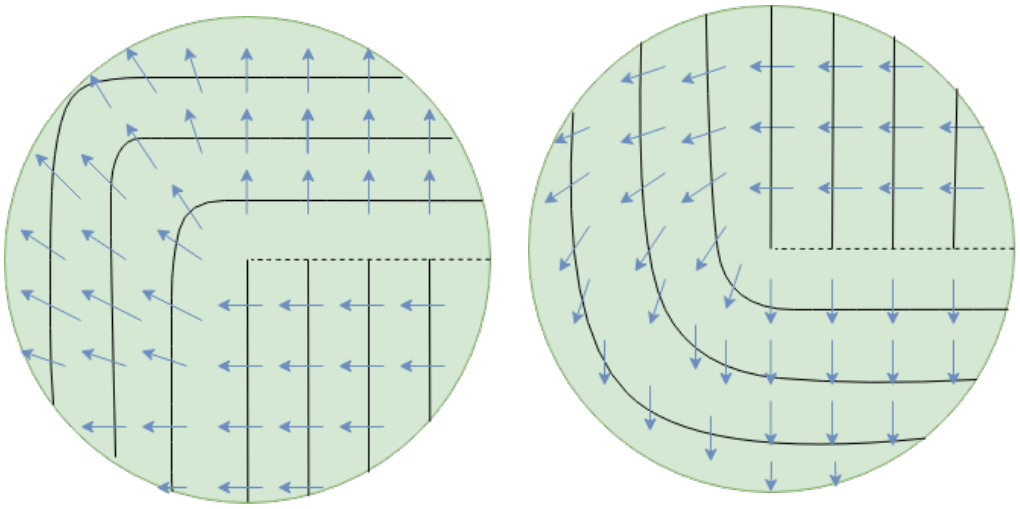



Figure 6.5: Note that to fully traverse the link of a branch vertex on the 4-fold branch cover, one needs to traverse each edge 4 times: once per sheet.

To achieve this, we want $\omega = d\theta$ to be satisfied exactly around the link 6.5 of a branch vertex. Furthermore, we ask that $\omega$ satisfy certain maxima minima properties. Consider a branch vertex



(a) Sheet 0 (b) Sheet 1

Figure 6.6: $\omega$ (blue arrows) and $\theta$ (black curves) around a branch vertex with singular index +1. Notice that if we were to superimpose the two figures, we would have 3 isolines meeting at $v$.

$v_b$ with a +1 singularity index - we define the half-link of $v_b$ as looping around two sheets of the link. Figure 6.6 describes the desired behavior of $\omega$ and $\theta$ around the half-link of $v_b$ when $\omega = d\theta$.

Let $W(\phi)$ be the integral of $\omega$ along the half-link of $v_b$, i.e. from $\phi$ to $\phi + 4\pi$. We want to observe the maxima and minima behavior of $W(\phi)$. On sheet 0, $W(\phi)$ roughly increases from $[0, \frac{2\pi}{3}]$, then decreases from $(\frac{2\pi}{3}, 2\pi]$ (1 maxima). On sheet 1, $W(\phi)$ now increases from $(2\pi, \frac{4\pi}{3}]$ (1 minima) and then decreases from $(\frac{4\pi}{3}, 4\pi]$ (1 maxima). The ordering of sheets here is arbitrary (sheet 0 and sheet 1 could have been swapped), so we could also have expected a total of 1 maxima and 2 minima. For a branch vertex with singular index $-1$, one can make a similar observation that $W(\phi)$ should have 3 maxima and 2 minima (or vice versa).

To detect the maxima minima structure of $W(\phi)$, we loop around the half-link of $v_b$ and count how many times $\omega$ changes sign. Let $i$ be the current vertex on the half-link of $v_b$ and $\delta_i$ represent

if we detect a maxima or minima. Then,

$$\delta_i = \begin{cases} Max & \text{if } \omega_{i-1,i} > 0, \omega_{i,i+1} < 0 \ , \\ \\ Min & \text{if } \omega_{i-1,i} < 0, \omega_{i,i+1} > 0 \ . \end{cases}$$

If $\omega$ does not satisfy the desired number of maxima and minima, one can perturb the values of $\omega$. In practice, we observed that $\omega$ always seems to follow this structure already, as the harmonic $\omega$ computed is typically smooth.

Since we want $\omega = d\theta$ exactly around each branch vertex $v_b$ so that the isoline structure is correct, it is possible to fix the solution $\psi$ on the vertices around $v_b$ so this condition holds. Recall that $\theta = arg(\psi)$. Then, $\theta \circ \tau^2 = -\theta$ due to conjugate symmetry. Going up a sheet involves traversing an angle sum of $2\pi$ around $v_b$, which means we can restate this as:

$$\theta(\phi) = -\theta(\phi + 4\pi)$$

Then, we can fix the value of $\theta(\phi)$ to $-\frac{W(\phi)}{2}$:

$$\theta(\phi) = -\theta(\phi + 4\pi) = -\theta(\phi) - \int_{\phi}^{\phi+4\pi} \frac{d\theta}{d\phi}$$

$$-\frac{W(\phi)}{2} = \frac{W(\phi)}{2} - \int_{\phi}^{\phi+4\pi} \frac{d\theta}{d\phi}$$

$$W(\phi) = \int_{\phi}^{\phi+4\pi} \frac{d\theta}{d\phi}$$

Recall that $W(\phi)$ is the integral of $\omega$ from $\phi$ to $\phi + 4\pi$, which means the above will hold if and only if $d\theta = \omega$ exactly. Furthermore, this fixes the value of $\psi_i$ at each vertex $v_i$ along the link of $v_b$. We can compute $W_i$ by looping through a half-link that begins at $v_i$ and summing the values of $\omega$. Then, $\theta_i = -\frac{W_i}{2}$ and $\psi_i = e^{\imath\theta_i}$.

Crucially, this scheme only works if the links of branch points are disjoint. Furthermore, each vertex must be adjacent to at most one branch point to avoid imposing multiple boundary conditions on the same vertex. In practice, we can take care of this easily by performing edge split operations on such edges prior to the branch cover construction (see Figure 6.7).

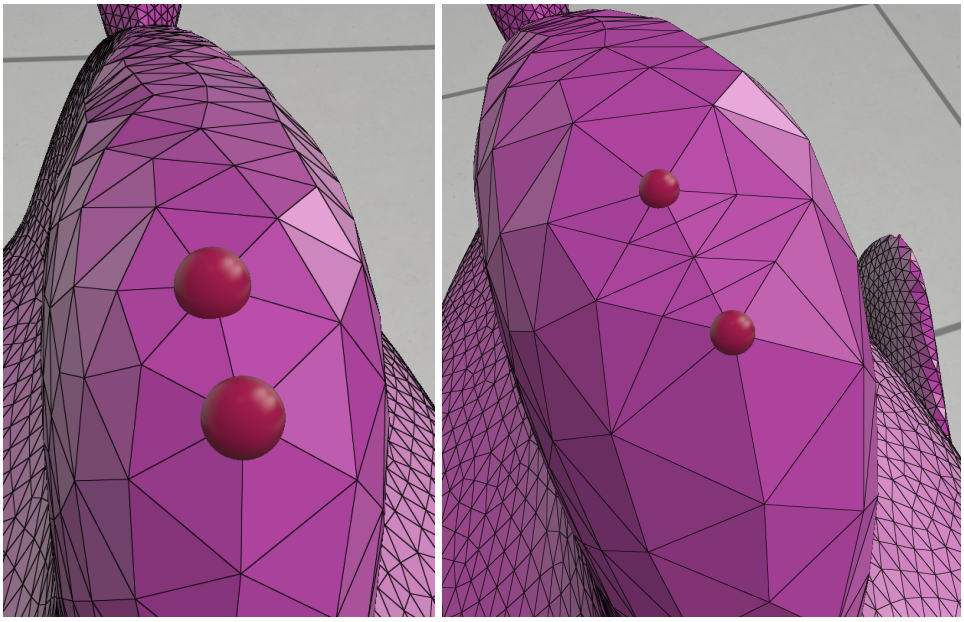


Figure 6.7: Left: two branch vertices adjacent to each other pre edge split. Right: post-edge split - note how the links of branch vertices are now completely disjoint, and the link of any vertex in the mesh contains at most 1 branch vertex.

Since the values of $\psi$ are now fixed along the link of each branch vertex, we can minimize 6.6 by solving a linear system with Dirchlet boundary conditions. Let $A$ be the matrix representing the Stripes energy in 6.6. We can partition $A$ into terms involving $\psi$ on the boundary and interior:

$$\begin{bmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{bmatrix} \begin{bmatrix} \psi_I \\ \psi_B \end{bmatrix} = 0$$

Since we know the values of $\psi_B$, this just boils down to solving:

$$A_{II}\psi_I = -A_{IB}\psi_B$$

$A$ is symmetric positive-definite with the same structure as the usual cotan-Laplace matrix, which makes solving the system very fast.

### 6.0.5 Texture Coordinates

We now have a value of $\psi$ at each non-branch vertex on sheets 0 and 1 of $\tilde{M}$ and wish to extract out texture coordinates. If we just take the argument of $\psi$, we will experience visual aliasing if the target frequency of $\omega$ is greater than the mesh spacing and discontinuities along zeros or branch points from linear interpolation (see Figure 6.8).

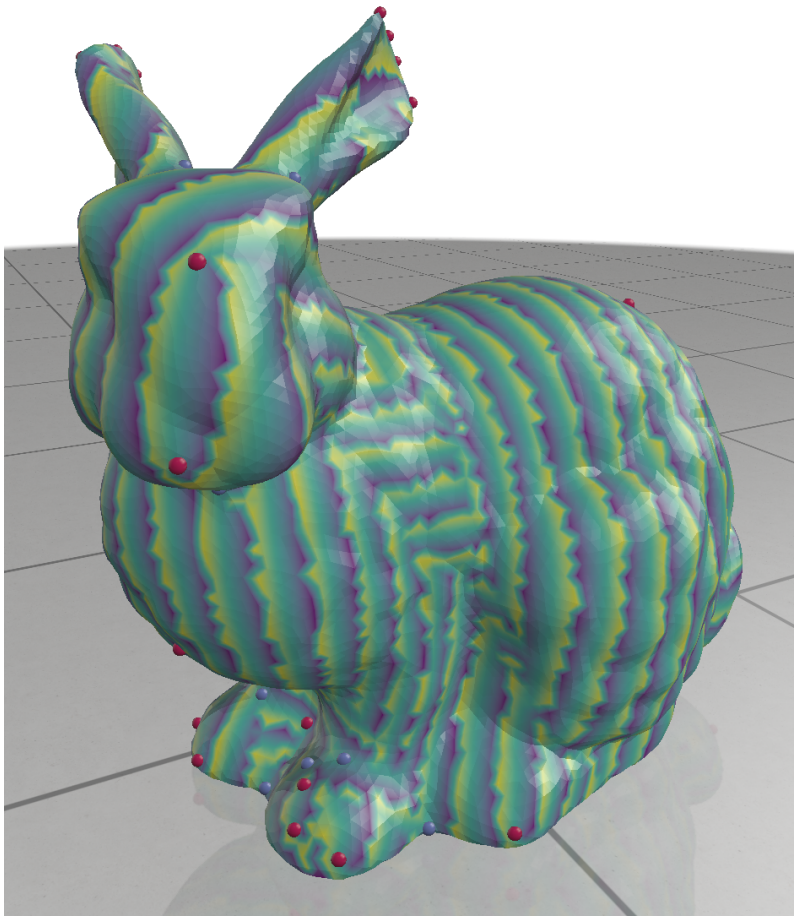


Figure 6.8: Visualizing $\theta$ on one of the two sheets directly - note the discontinuities near branch points. The stripes can also shift orientation due to sheet interchange.

Since we cannot achieve $\psi_j = e^{\imath\omega_{ij}}\psi_i$ exactly due to global nonintegrability, we follow the Stripe Patterns approach of computing a new angular displacement $\sigma$ closest to $\omega$ for which $\psi_j = e^{\imath\sigma_{ij}}\psi_i$ exactly. Essentially, we compute the angle difference $\delta_{ij} = arg(\frac{e^{i\omega_{ij}}\psi_i}{\psi_j})$ and let

$\sigma_{ij} = \omega_{ij} - \delta_{ij}$ (see Section 4.3 of the Stripe Patterns paper for more details). Then, the texture coordinates $(\theta_i, \theta_j, \theta_k)$ of every triangle $ijk$ that does not contain a branch vertex can be extracted by integrating $\sigma$ along its edges:

$$\theta_i = arg(\psi_i)$$

$$\theta_j = \theta_i + \sigma_{ij}$$

$$\theta_k = \theta_j + \sigma_{jk}$$

We can compute $(\theta_{i_0}, \theta_{i_1}), (\theta_{j_0}, \theta_{j_1}), (\theta_{k_0}, \theta_{k_1})$ using the values of $\psi_i$ on sheets 0 and 1 to get our desired texture coordinates. Note that $\theta_\ell = \theta_k + \sigma_{ki}$ should take us back to $\theta_i$ for a non-singular triangle. However, if $\frac{\theta_\ell - \theta_i}{2\pi}$ is non-zero, then $\theta$ has jumped by a multiple of $2\pi$ and cannot be interpolated by any continuous function. We refer to these as new zeros in the solution; these can be drawn as triangles instead of quads, but our alternating optimization scheme works to remove new zeros. See Chapter 7 for what these new zeros look like and how we handle them.

To handle triangles that contain a branch vertex $v_b$, we just need to compute values of $\sigma$ on the edges that were cut out from the surface, i.e. the edges that connect $v_b$ to vertices in its link. For each triangle $t_i$ adjacent to $v_b$ we ask that $\sigma_i + \sigma_{i+1} + \sigma_{i+2} = 0$, i.e. that a new zero does not appear next to a $v_b$ (see Figure 6.9).

Since we have values of $\sigma$ for edges along the link of $v_b$, there is only one degree of freedom remaining. Let $d$ be the degree of $v_b$:

$$\sigma_2 = -\sigma_1 - \sigma_0$$

$$\sigma_4 = -\sigma_3 - \sigma_2$$

$$\vdots$$

$$\sigma_{2d} = -\sigma_{2d-1} - \sigma_{2d-2}$$

Substituting each equation into the next and taking into account the orientation of $\sigma$, we get:

$$\sigma_{2d} = \sigma_0 + \sigma_1 + ... + \sigma_{2d-1}$$

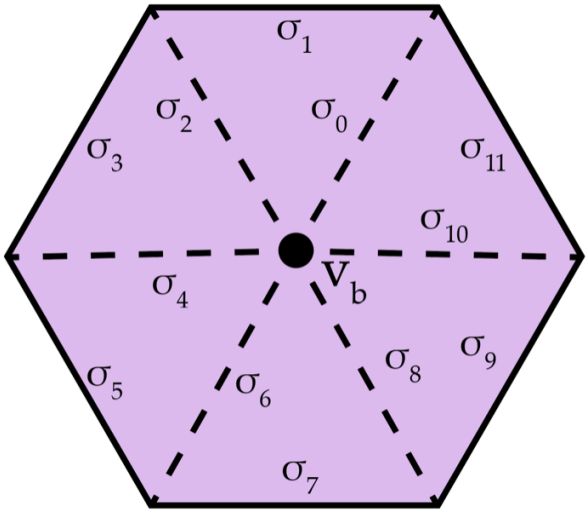



Figure 6.9: No new zeros should appear next to $v_b$: the sum of $\sigma$ around any triangle in its neighborhood must be zero.

Due to conjugate symmetry, we must also have:

$$\sigma_{2d} = -\sigma_0$$

Combining the two, we therefore get:

$$\sigma_0 = -(\sigma_1 + \sigma_2 + ... + \sigma_{2d-1})/2 = -\frac{W}{2}$$

If we fix $\sigma_0$ to the above, then the remaining values of $\sigma$ on edges containing $v_b$ can be computed by propagating around the faces containing $v_b$.

Now that we have values of $\sigma$ on every edge of $\tilde{M}$, we can extract texture coordinates the same way we handle any other face: by integrating $\sigma$ around $ijk$. However, one has to take the extra precaution to not start integrating at $v_b$, since branch vertices do not have a value of $\psi$ associated with them.

### 6.0.6 Isoline Visualization

Visualizing the quad mesh isolines is mostly straightforward after computing texture coordinates for each face of $M$. Because of conjugate symmetry, the isolines on sheets $s$ and $(s+2)\%4$ should

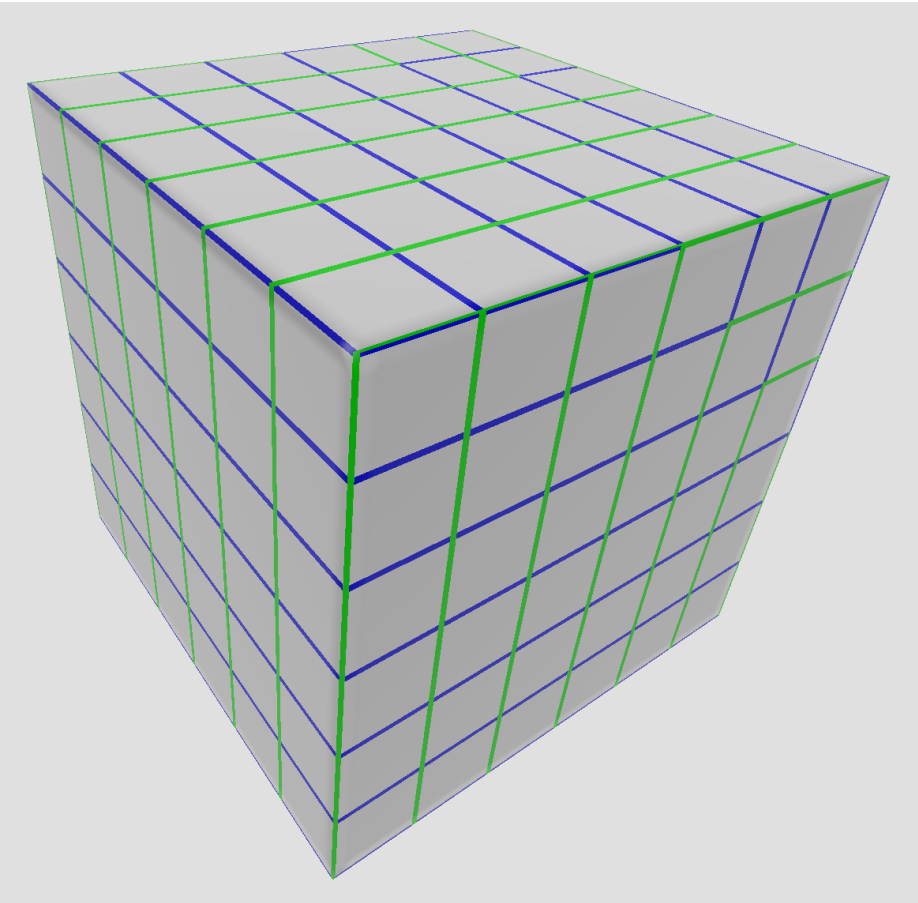


Figure 6.10: Isolines of a Quad Mesh on a Cube

look the same. This means that the isolines must be symmetric across the real axis. Drawing the $\pi$ isolines captures whenever the coordinate function reaches 0 and therefore produces the desired isoline structure. If $(u, v)$ are the texture coordinates passed into the fragment shader, then we draw an isoline if $mod(u, \pi)$ or $mod(v, \pi)$ is within $w$ of 0, where $w$ is the width of the isoline (see Figure 6.10).

# Chapter 7

# Removing New Singularities

### 7.0.1 Motivation

One of the primary objectives of this quad meshing pipeline was to ensure that no new singularities appear in the result (See Figure 7.1). Our first approach is an alternating optimization scheme that reduces the number of singularities in the output each iteration. Although we do not show a strict upper bound on the number of iterations, our experiments suggest that this scheme converges in a reasonable number of iterations on a representative dataset.

### 7.0.2 Alternating Optimization Scheme

We propose the following algorithm to remove new singularities in the result, given an initial vector field $\omega_0$ on the branch cover:

1. Compute $\psi_t$ guided by $\omega_t$ (as explained in Chapter 6).
2. Evaluate $\tilde{\omega_{t+1}} = d\theta$, where $\theta = arg(\psi_t)$.
3. Find a harmonic $\omega_{t+1}$ closest to $\tilde{\omega_{i+1}}$.
4. Repeat until convergence.

The most interesting step in this scheme is step 2, where we find a harmonic field that is

closest to the one produced by the paramterization. The intuition behind this is that if we feed in a field closer to the actual output, the energy will be less stressed and avoid placing new singularities. Since we work with the L2 energy, it would make sense for the parameterization to spread the deviation across the surface rather than concentrate it at a single vertex.

### 7.0.3 Helmholtz-Hodge Decomposition

In order to find a harmonic vector field $\tilde{\omega}$ closest to any input vector field $\omega$, we perform a Helmholtz-Hodge decomposition. Essentially, any vector field on a surface can be split up into a curl-free component, a divergence-free component, and a harmonic component that is both curl-free and divergence-free.

The procedure is fast and efficient to compute, only involving two linear systems with a positive-definite Laplace Beltrami operator $\Delta$ and a symmetric system. The former can be solved quickly with a sparse Cholesky factorization, and the latter can be computed using a LU factorization (roughly twice as costly as Cholesky). Because the branch cover is just like any other surface, the Helmholtz-Hodge decomposition algorithm does not need to be modified at all. The resulting harmonic field $\gamma$ is then input as a guiding field for the computation of $\psi$. One thing to note is that $\gamma$ is no longer a constant vector field, which seems to greatly help reduce the number of new singularities. However, this process does not seem to fully solve the problem, as for some meshes (see the results section for more details), the optimization converges at a handful of new singularities. Helmholtz-Hodge decomposition is a good intuitive approach, but the optimization scheme is still a work in progress!

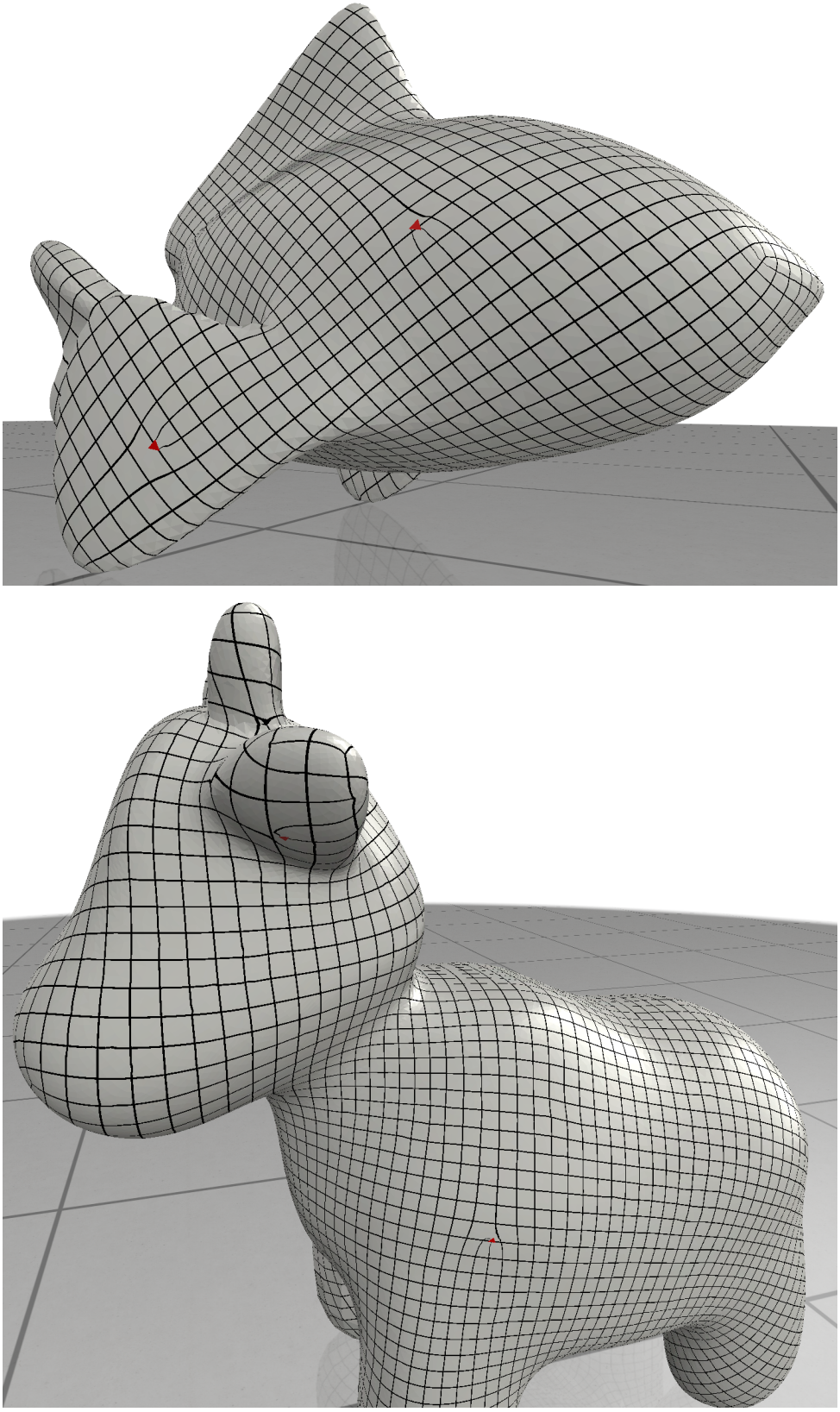


Figure 7.1: New zeros in the resulting quad mesh.

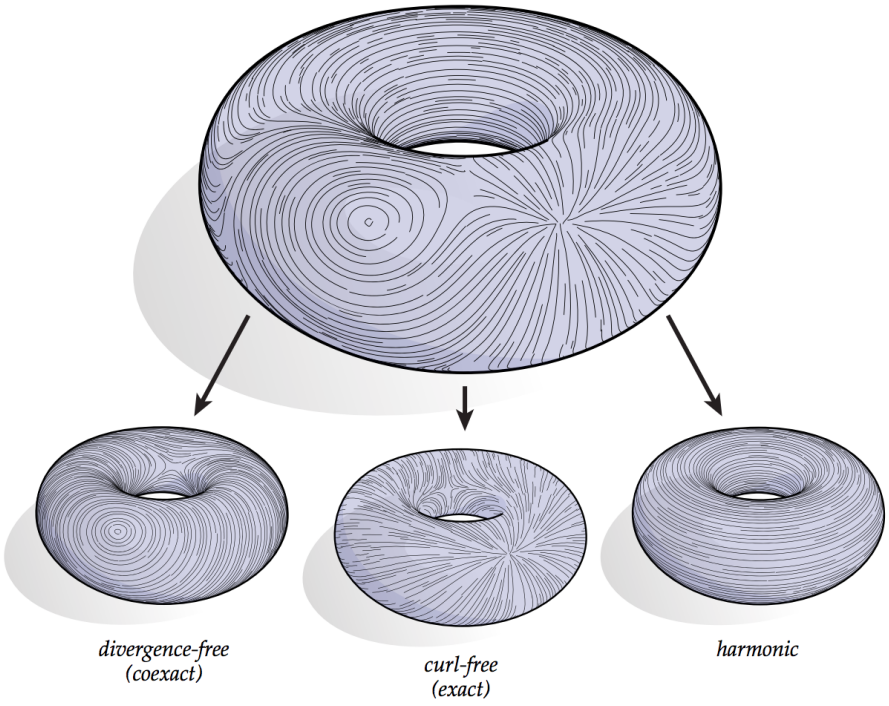



Figure 7.2: Any vector field on a surface can be expressed as the sum of a curl-free, divergence-free, and most importantly a harmonic component. [6]

# Chapter 8

# Results

### 8.0.1 Output Quad Mesh Visualizations

Figures 8.1 and 8.2 show isoline visualizations of some resulting quad meshes. We can see that the output quads are high quality - distortion plots are presented later. The correct isoline behavior is exhibited near irregular vertices corresponding to singularities, and quads in these regions are smaller due to the conformal constraint. Some of the quads (e.g. the cube) may look somewhat twisted along the edges of the original surface. This is because we have not yet implemented sharp feature alignment such that our guiding field preserves sharp edges of the original surface.

### 8.0.2 Quad Mesh Extraction

There are various ways of extracting out the quad mesh. The approach we took was to compute a graph that defines the connectivity of the resulting quad mesh. To do so, we find the isoline crossings at every edge, face, and branch point. Then, edges of degree 2 are collapsed until only regular vertices of degree 4 and singular vertices of degree 3 or 5 remain. Quads can then be read off by finding all cycles of length 4. See figures 8.3 and 8.4 for a visual representation.

### 8.0.3 Experimentation

We tested our pipeline on a representative database of genus-0 surfaces and recorded the number of iterations it took for the alternative optimization scheme to converge. As shown in Figure 8.5, each mesh converged within 12 iterations. Furthermore, at most 4 new singularities remained after 20 iterations, even for meshes that began with a large number of new singularities. Some meshes also began with 0 new singularities as well, which is great for computation as no additional optimization needs to be done for those outputs.

Although our pipeline is guaranteed to be intrinsically conformal, the area distortion is significantly influenced by the initial placement of singularities. However, on simpler meshes like the cube, even cones computed from a smooth cross field will be the optimal ones, i.e. singular points at the corners. Therefore, we can get a better sense of what the distortion will look like for the cube (see Figure 8.6). As expected, the vast majority of corner angles are around 90 degrees - we cannot expect to get 90 degrees exactly everywhere, as edges are straight and cannot wrap around or bend in the extrinsic setting. The areas of each quad are also clumped together, which is a good sign that the elements of the quad mesh are mostly uniform throughout. Opposite edges are approximately the same length, which means most elements are rectangular as well. Plots concerning distortion data are also included for several other extracted meshes; each output quad mesh has low angle distortion and great edge length ratios. The latter two have good but slightly worse area distortion - this is due to non-optimal cone point placement.

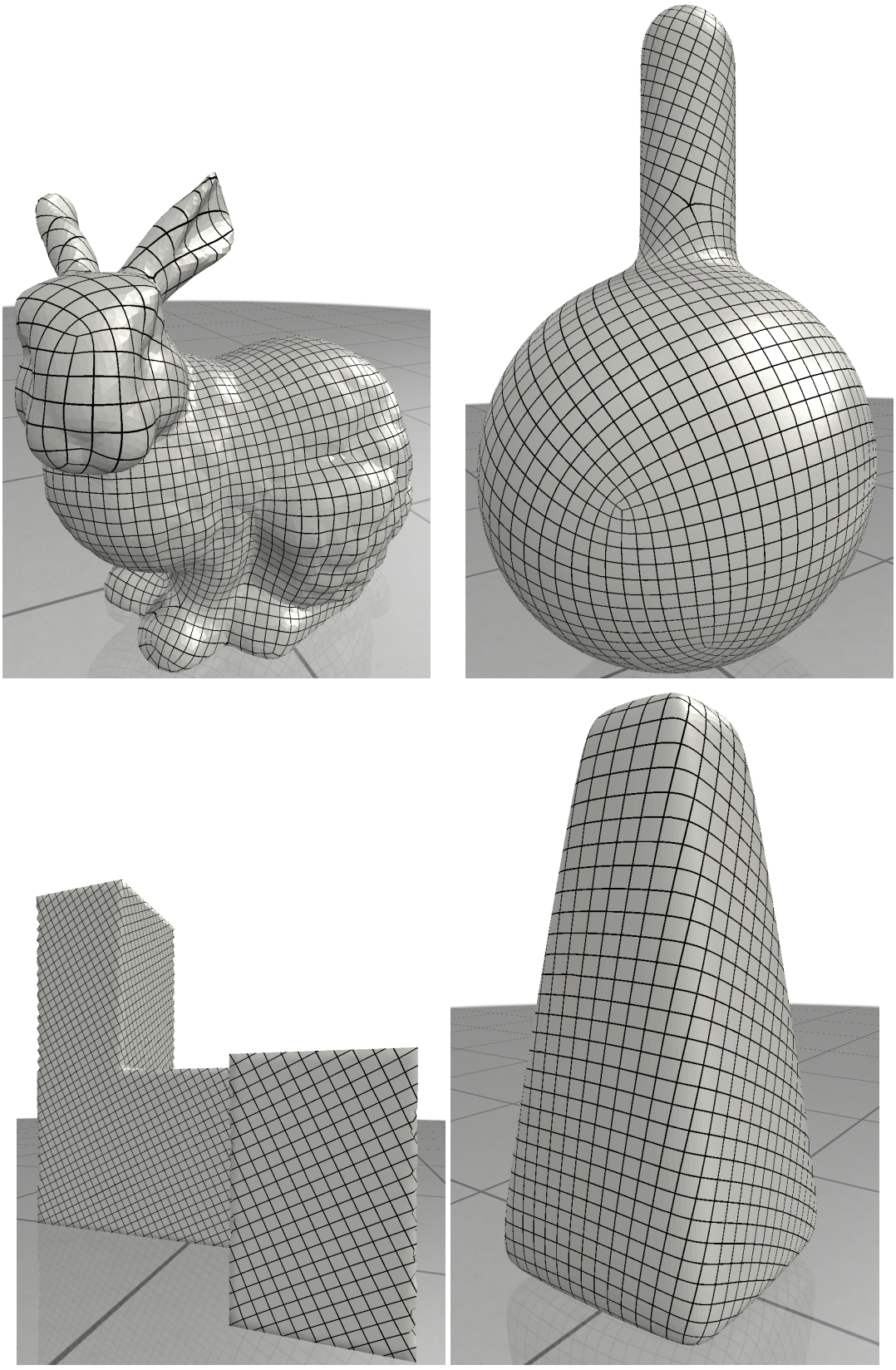


Figure 8.1: Example visualizations of output quad mesh on various inputs.

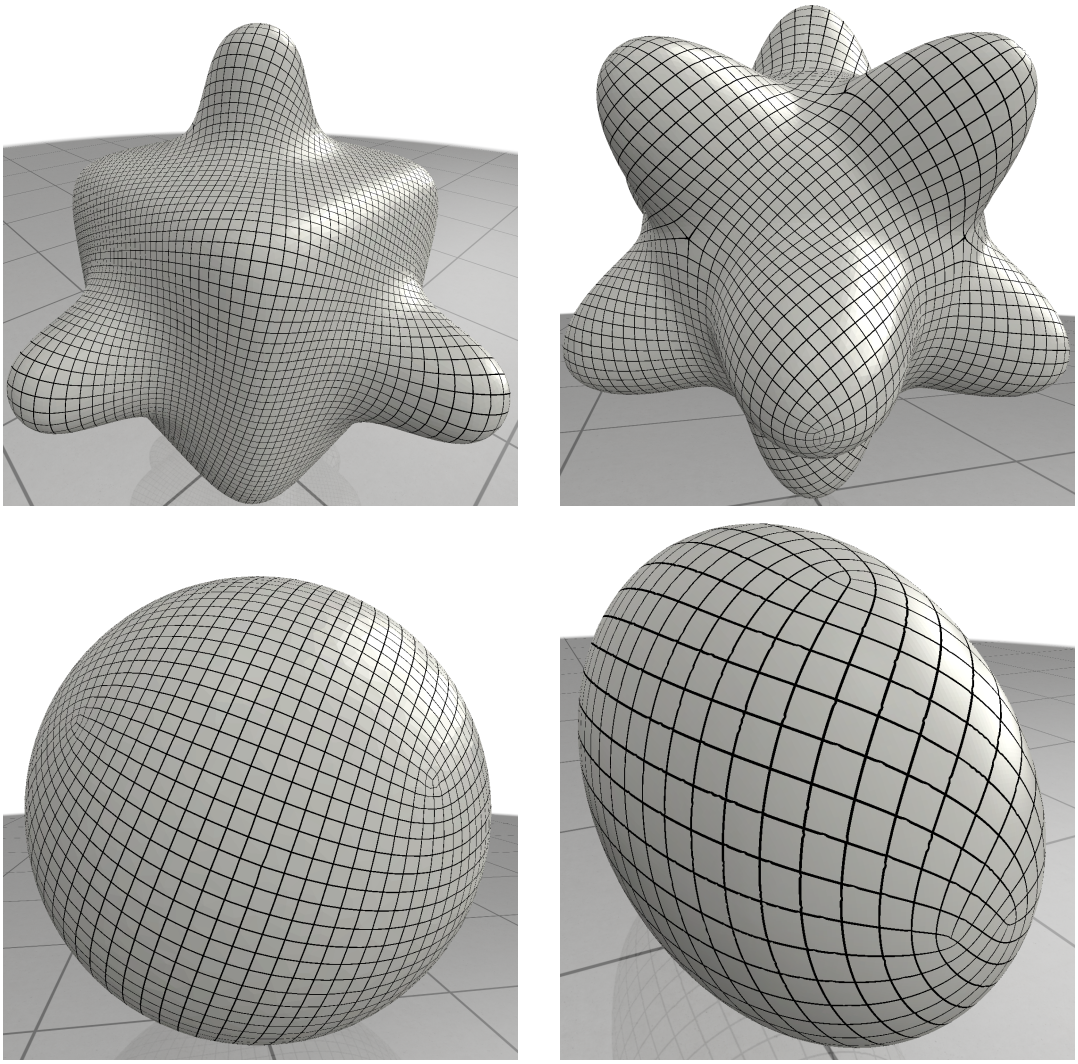


Figure 8.2: Example visualizations of output quad mesh on various inputs.

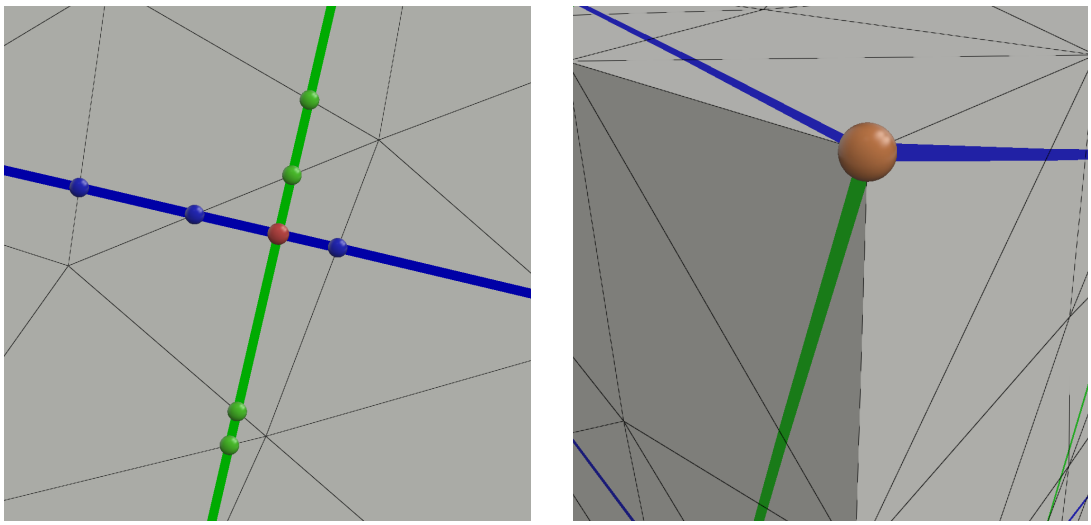


Figure 8.3: Left: isoline crossings on edges and faces. Right: isoline crossings on branch points.

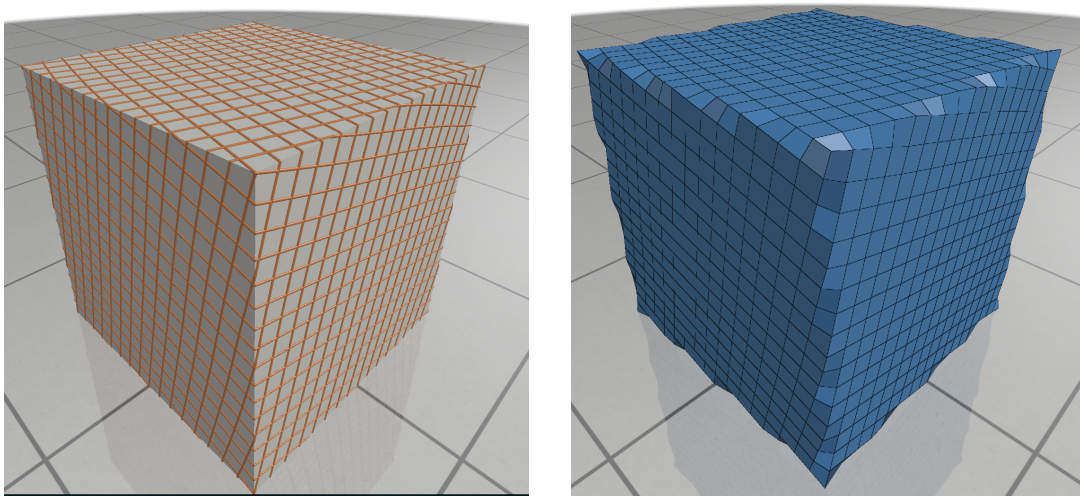


Figure 8.4: Left: resulting graph network. Right: extracted quad mesh.

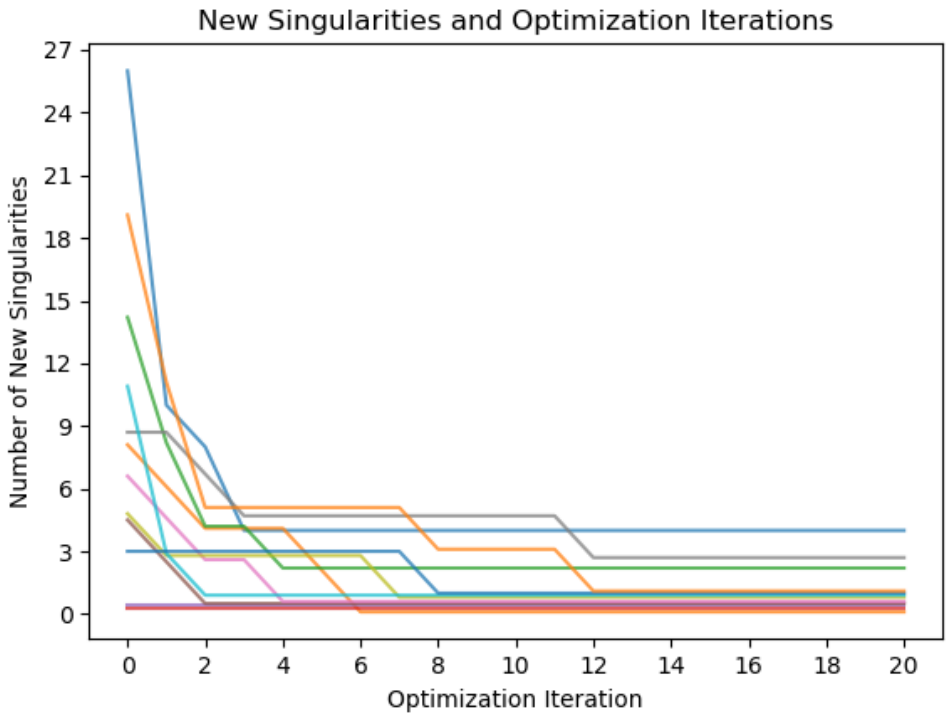



Figure 8.5: Plot of the number of new singularities and optimization iterations.

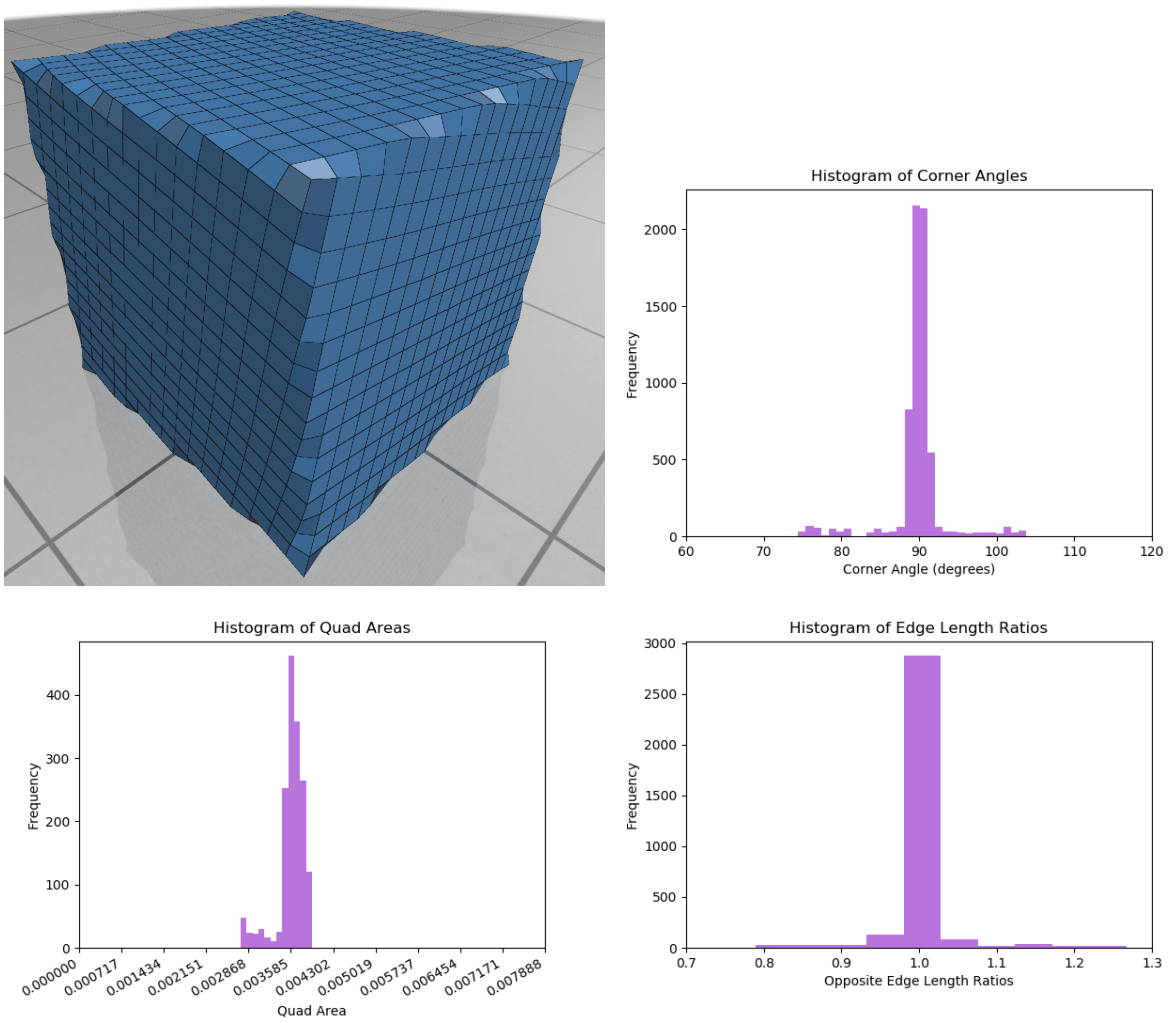
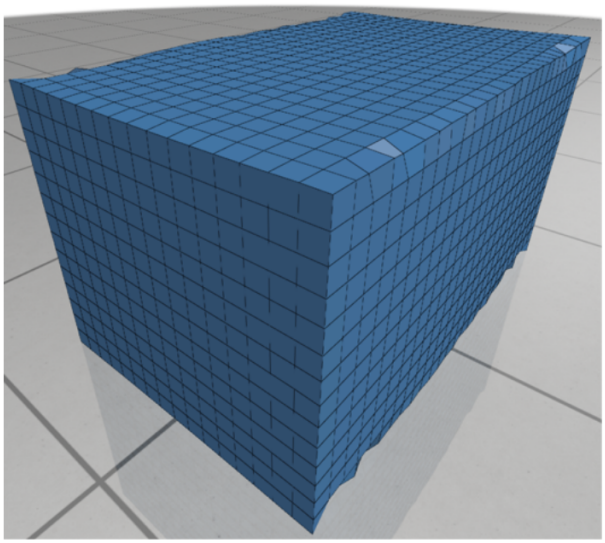
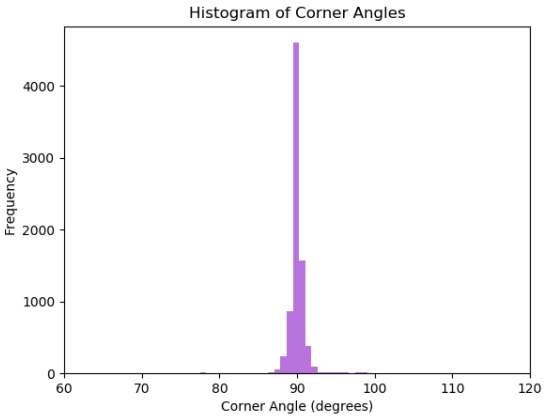
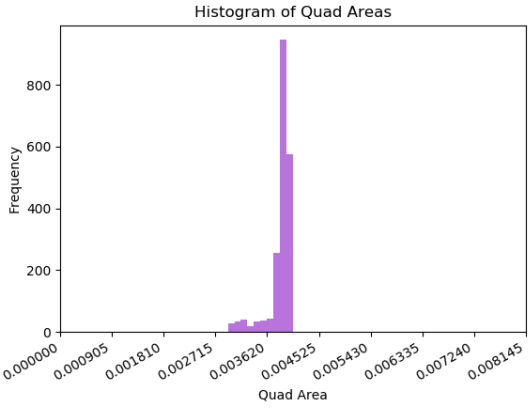
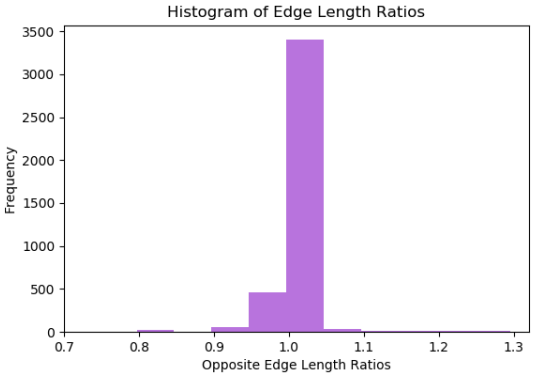


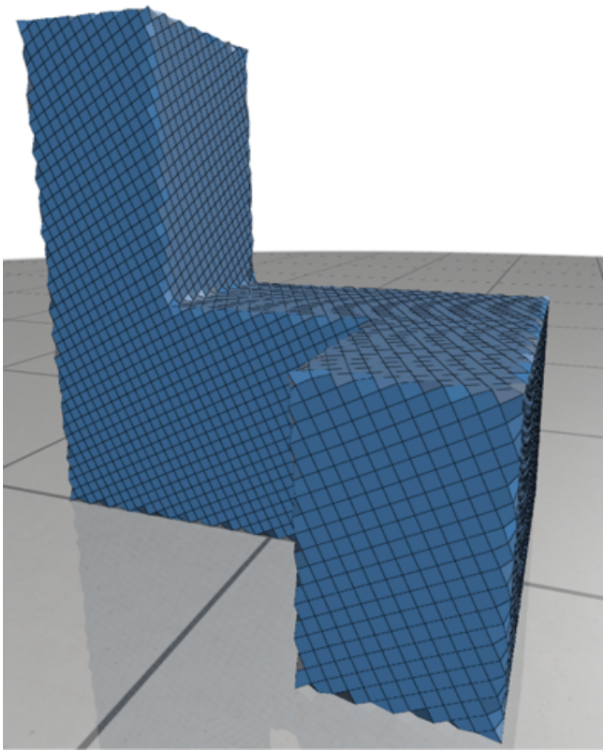

Figure 8.6: Plot of the resulting area and opposite edge length ratio of the quad meshed cube.

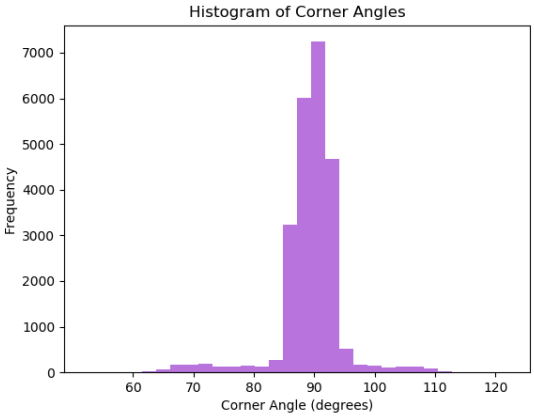


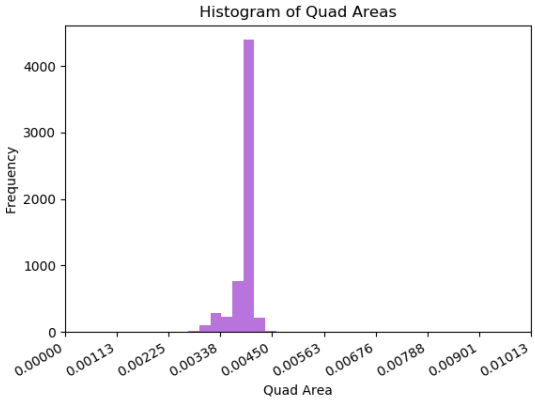

Histogram of Corner Angles
Frequency
Corner Angle (degrees)
0
1000
2000
3000
4000
60
70
80
90
100
110
120

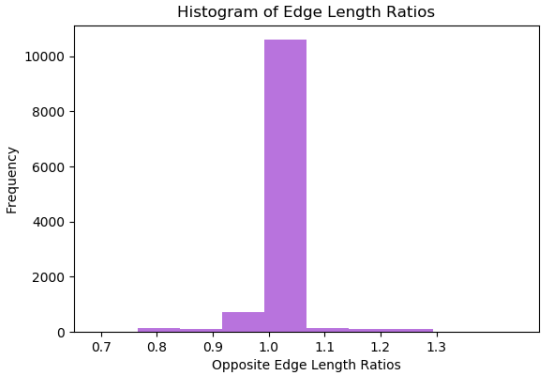

Histogram of Quad Areas
Frequency
Quad Area
0
200
400
600
800
0.000000
0.000905
0.001810
0.002715
0.003620
0.004525
0.005430
0.006335
0.007240
0.008145

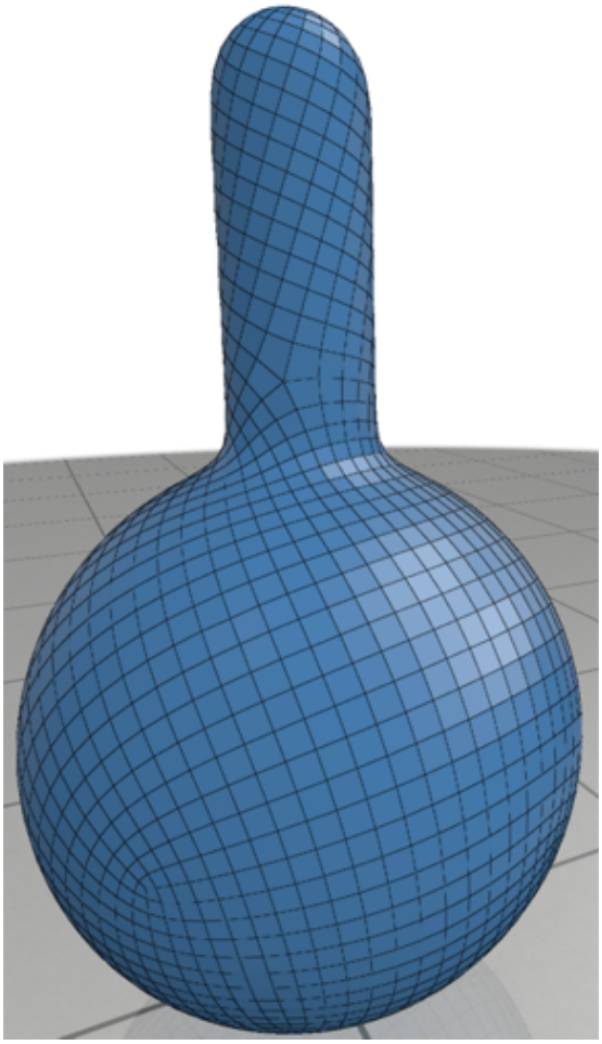

Histogram of Edge Length Ratios
Frequency
Opposite Edge Length Ratios
0
500
1000
1500
2000
2500
3000
3500
0.7
0.8
0.9
1.0
1.1
1.2
1.3

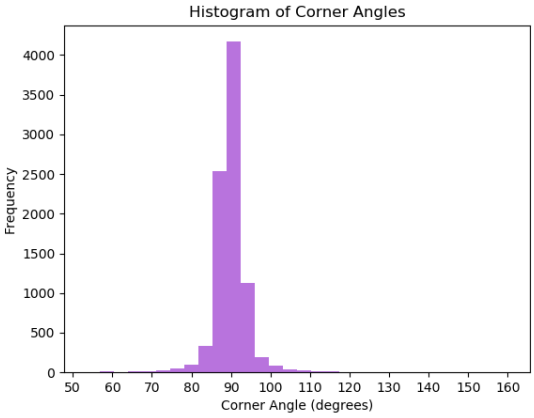


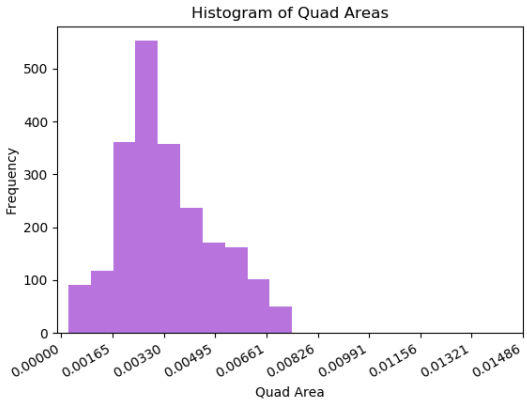

Histogram of Corner Angles
Frequency
Corner Angle (degrees)
0
1000
2000
3000
4000
5000
6000
7000
60
70
80
90
100
110
120

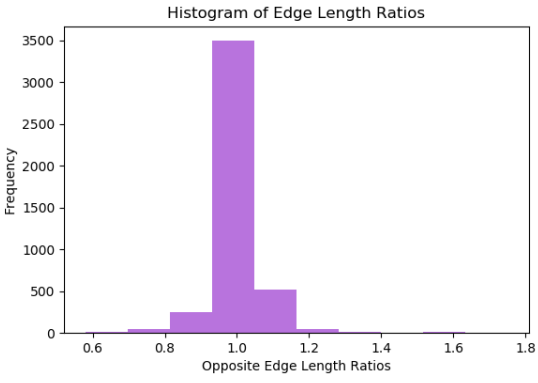

Histogram of Quad Areas
Frequency
Quad Area
0
1000
2000
3000
4000
0.00000
0.00113
0.00225
0.00338
0.00450
0.00563
0.00676
0.00788
0.00901
0.01013

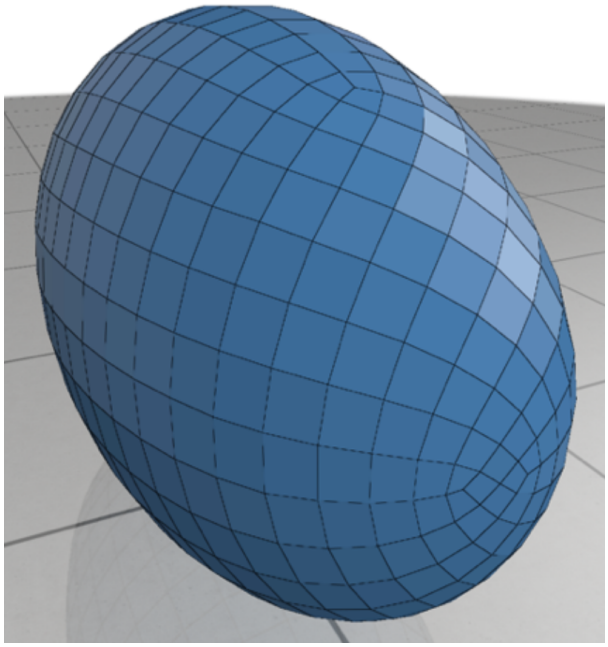

Histogram of Edge Length Ratios
Frequency
Opposite Edge Length Ratios
0
2000
4000
6000
8000
10000
0.7
0.8
0.9
1.0
1.1
1.2
1.3

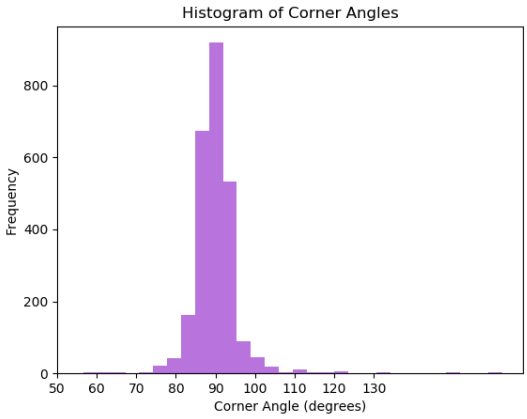


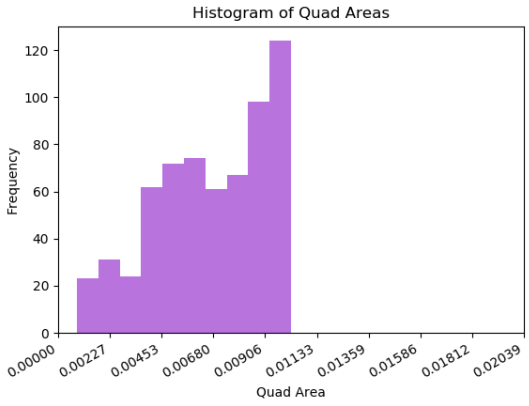

Histogram of Corner Angles
Frequency
Corner Angle (degrees)
0
500
1000
1500
2000
2500
3000
3500
4000
50
60
70
80
90
100
110
120
130
140
150
160

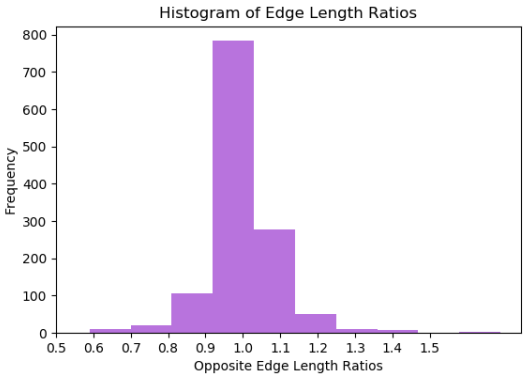

Histogram of Quad Areas
Frequency
Quad Area
0
100
200
300
400
500
0.00000
0.00165
0.00330
0.00495
0.00661
0.00826
0.00991
0.01156
0.01321
0.01486


Histogram of Edge Length Ratios
Frequency
Opposite Edge Length Ratios
0
500
1000
1500
2000
2500
3000
3500
0.6
0.8
1.0
1.2
1.4
1.6
1.8




Histogram of Corner Angles
Frequency
0
200
400
600
800
50
60
70
80
90
100
110
120
130
Corner Angle (degrees)


Histogram of Quad Areas
Frequency
0
20
40
60
80
100
120
0.00000
0.00227
0.00453
0.00680
0.00906
0.01133
0.01359
0.01586
0.01812
0.02039
Quad Area


Histogram of Edge Length Ratios
Frequency
0
100
200
300
400
500
600
700
800
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
Opposite Edge Length Ratios

# Chapter 9

# Future Work

The current pipeline produces a seamless and nearly conformal quad mesh with low area distortion given proper cone placements. No mixed integer solvers are employed, and the only systems solved are sparse linear systems for uniformization, energy minimization, and optimization (if necessary), which means that the pipeline is fast and efficient. We also show how this formulation nicely ties together field-aligned parameterization and global conformal parameterization.

However, there remains a significant amount of ongoing work in the optimization step to fully remove the presence of new singularities in the solution. We are currently looking into a strategy similar to the curl correction approach from PGP (i.e., the smallest change that makes a given vector field curl-free), taking into account the global periods around non-contractable generator loops to produce a field that is both globally and locally integrable. To better support non-genus 0 surfaces, we can replace our simple BFS procedure with a better field generation algorithm. Furthermore, more experimentation needs to be done by trying different cone placement algorithms and extracting more meshes. Finally, a common desired property of quad meshing algorithms is to support feature alignment such that sharp feature lines of the input mesh are preserved. Many quad meshing papers address this already, and we can take advantage of the existing literature to follow a similar approach.

# Bibliography


[1] Mirela Ben-Chen, Craig Gotsman, and Guy Bunin. Conformal flattening by curvature prescription and metric scaling. In *Computer Graphics Forum*, volume 27, pages 449–458. Wiley Online Library, 2008. 1.2.2

[2] Ted D Blacker and Michael B Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32 (4):811–847, 1991. 1.2.3

[3] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)*, 28(3):77, 2009. 1.2, 1.2.4

[4] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, volume 32, pages 51–76. Wiley Online Library, 2013. 1.3, 1.2.6

[5] Keenan Crane. Discrete Conformal Geometry. In *Proceedings of Symposia in Applied Mathematics*. American Mathematical Society, 2018. 4.0.2, 4.0.5

[6] Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*, page 7. ACM, 2013. 1.1, 6.2, 6.0.2, 7.2

[7] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C Hart. Spectral surface quadrangulation. *Acm transactions on graphics (tog)*, 25(3):1057–1066, 2006. 1.2.3

[8] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover-surface parameterization using branched coverings. In *Computer graphics forum*, volume 26, pages 375–384. Wiley Online Library, 2007. 1.2.4, 5.2, 5.3

[9] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), 2013. 1.2.2, 3.0.1, 3.0.3

[10] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Stripe patterns on surfaces. *ACM Trans. Graph.*, 34, 2015. 6.0.1, 6.1, 6.0.2

[11] Richard H MacNeal. *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology, 1949. 6.0.2

[12] Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. Automatic machine knitting of 3d meshes. *ACM Trans. Graph.*, 37(3):35:1–35:15, August 2018. ISSN 0730-0301. doi: 10.1145/3186265. URL `http://doi.acm.org/10.1145/3186265`. 1.1

[13] Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. Periodic global parameterization. *ACM Transactions on Graphics (TOG)*, 25(4):1460–1485, 2006. 1.2.4, 6.0.1

[14] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Transactions on Graphics (TOG)*, 27(2):10, 2008. 1.2.5

[15] Yousuf Soliman, Dejan Slepčev, and Keenan Crane. Optimal cone singularities for conformal flattening. *ACM Trans. Graph.*, 37(4), 2018. 1.2.2, 3.0.1

[16] Boris Springborn, Peter Schröder, and Ulrich Pinkall. Conformal equivalence of triangle meshes. *ACM Transactions on Graphics (TOG)*, 27(3):77, 2008. 1.2.2, 4.0.4