Discrete Metric Design and Discrete Tangent Bundles: from Surfaces to 3-Manifolds

A Dissertation Presented

by

Xiaotian Yin

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2010

Stony Brook University

The Graduate School

Xiaotian Yin

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Xianfeng Gu – Dissertation Advisor
Associate Professor, Department of Computer Science

Joseph S.B. Mitchell – Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Jie Gao
Assistant Professor, Department of Computer Science

Xiangmin Jiao
Assistant Professor, Department of Applied Mathematics and Statistics

Feng Luo
Professor, Department of Mathematics
Rutgers University

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Discrete Metric Design and Discrete Tangent Bundles: from Surfaces to 3-Manifolds

by

Xiaotian Yin

Doctor of Philosophy

in

Computer Science

Stony Brook University

2010

One of the fundamental tasks in geometric modeling and computer graphics is to study shapes, such as surfaces and volumes, and differential objects associated with them, such as vector fields. For the study of shapes, the challenge in many cases comes from the need of a parameter domain that has a canonical and simple shape, which is equivalent to designing a metric of special properties. For differential objects, a fundamental problem is how to represent tangent bundles in a discrete setting, so that covariant differentiation and connections can be computed accordingly.

This dissertation aims to design rigorous and practical methods to deal with both tasks. For discrete metric design, a global parameterization method, called slit map, is designed for genus zero surfaces with multiple holes using discrete differential forms. A second method is designed to map volumetric handle bodies to direct product domains. A third method is designed to compute constant curvature metrics for hyperbolic 3-manifolds using a discrete curvature flow. For discrete tangent bundles, we propose discrete constructions using tetrahedral meshes to represent unit tangent bundles for various surfaces, including topological disks and closed orientable surfaces of arbitrary genus. All the proposed methods are based on solid results from topology and differential geometry, and are adapted with best efforts to engineering problems ranging from surfaces to 3-manifolds.

For my beloved family, advisors, friends and colleagues.

Contents

List of Figures

Acknowledgements

I would like to express my sincere gratitude to my thesis advisor, Professor Xianfeng Gu, for his invaluable guidance on research and his generous help at various stages of my Ph.D. studies. I cannot imagine myself completing this dissertation without his inspiration, discussion, and encouragement. He led me to the field of computational conformal geometry and computational topology, and showed such kindness, patience and deep understanding about geometry that no graduate student could expect more from their advisors.

I would like to thank Professor Fen Luo, Professor Hong Qin, Professor Jie Gao and Professor Shing-Tung Yau, for their inspiring discussions, advice and collaborations that I have substantially benefited from during my Ph.D. studies.

I would like to thank Professor Joseph Mitchell and Professor Xiangmin Jiao for their kind advice, as well as for serving on various committees.

I would also like to thank my colleagues who are and were in the center of visual computing and computer science department, Miao Jin, Feng Qiu, Zhe Fan, Yang Wang, Xiaohu Guo, Ying He, Kexiang Wang, Sen Wang, Hongyu Wang, Xin Li, Xiang Zeng, Ruirui Jiang, Wei Zeng, Min Zhang, Tingbo Hou, Ziyi Zheng, for delightful collaborations and discussions we had together, and thank my friends in Stony Brook, Bing Lai, Wenqing Fang, Meng Qu, Zhi Pan, Zhenguo Wang, Fen Zhang, Wei Hu, Yuanzhi Tang, Maohua Lu, Yang Yang, Zhen Liu, Marvin Hazan, Jane Hazan, and so many others for their help during the past several years. The members of the excellent support staff in the Computer Science Department were also of great assistance during these years.

Last but not least, I want to thank my parents, and my girl friend, for their endless love and support. Without their support, this dissertation would not have been possible.

This dissertation is dedicated to them.

Publication

[1] Xiaotian Yin, Miao Jin and Xianfeng Gu, Computing shortest cycles using universal covering space. *Visual Computer,* 2007, 23(12):999–1004.

[2] Jingyi Yu, Xiaotian Yin, Xianfeng Gu, Leonard McMillan and Steven Gortler, Focal surfaces of discrete geometry. *Proceedings of the fifth Eurographics symposium on Geometry processing,* 2007, 257:23–32.

[3] Xiaotian Yin, Junfei Dai, Shing-Tung Yau and Xianfeng Gu, Slit Map: Conformal Parameterization for Multiply Connected Surfaces. *Advances in Geometric Modeling and Processing,* 2008, 4975:410–422.

[4] Xiaotian Yin, Miao Jin, Feng Luo and Xianfeng David Gu, Computing Constant-Curvature Metric for Hyperbolic 3-manifolds with Boundaries using Truncated Tetrahedral Meshes. *International Journal of Shape Modeling,* 2008, 14(2):169.

[5] Wei Zeng, Yun Zeng, Yang Wang, Xiaotian Yin, Xianfeng Gu and Dimitris Samaras, 3D Non-rigid Surface Matching and Registration Based on Holomorphic Differentials. *Proceedings of the 10th European Conference on Computer Vision: Part III,* 2008, 5304:1–14.

[6] Yalin Wang, Xiaotian Yin, Jie Zhang, Xianfeng Gu, Tony F Chan, M. Thompson and Shing-tung Yau, Brain Mapping with the Ricci Flow Conformal Parameterization and Multivariate Statistics on Deformation Tensors. *2nd MICCAI Workshop on Mathematical Foundations of Computational Anatomy,* 2008, pp. 36–47.

[7] Xiaotian Yin, Miao Jin, Feng Luo and Xianfeng David Gu, Discrete Curvature Flows for Surfaces and 3-Manifolds. *Emerging Trends in Visual Computing: LIX Fall Colloquium,* 2009, 5416:38–74.

[8] Rik Sarkar, Xiaotian Yin, Jie Gao, Feng Luo and Xianfeng David Gu, Greedy routing with guaranteed delivery using Ricci flows. *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks,* 2009, 121–132.

[9] Wei Zeng, Xiaotian Yin, Min Zhang, Feng Luo and Xianfeng Gu, Generalized Koebe's method for conformal mapping multiply connected domains. *SIAM/ACM Joint Conference on Geometric and Physical Modeling,* 2009, 89–100.

[10] Guodong Rong, Yang Liu, Wenping Wang, Xiaotian Yin, Xianfeng David Gu and Xiaohu Guo, GPU-Assisted Computation of Centroidal Voronoi Tessellation. *IEEE Transactions on Visualization and Computer Graphics,* 2010, 99(2).

[11] Jiazhi Xia, Ying He, Xiaotian Yin, Shuchu Han and Xianfeng Gu, Direct-Product Volumetric Parameterization of Handlebodies via Harmonic Fields. *International Conference on Shape Modeling and Applications,* 2010, 3–12.

Chapter 1

Introduction

1.1 Discrete Metric Design

Surfaces encountered in computer graphics and geometric modeling are usually compact Riemannian surfaces, which are 2-manifolds with a *Riemannian metric*, which is intuitively a measure of length. In engineering fields, a surface is usually represented as a triangular mesh, which is a piecewise-linear approximation of a smooth surface. In such an approximation, a *discrete metric* can be represented by a set of edge length; that is, a function that assigns each edge with a real number so that the triangle inequality is satisfied within every triangle.

Another type of geometric objects that are widely used in computer graphics and geometric modeling are volumes, which are 3-manifolds. In engineering fields, a volume is usually represented as a tetrahedral mesh, which is a piecewise-linear approximation of a smooth 3-manifold. Similar to the surface case, a discrete metric for a tetrahedral mesh is also a function assigning each edge with a length.

No matter for surfaces or for volumes, a metric determines the shape of an object. Under certain circumstances one wants to change the shape of an object, which usually requires to replace the given metric with a new one that has special properties. We name this process as *metric design*. Such needs in the engineering fields give rise to a challenging problem:

Problem 1.1.1 (The Discrete Metric Design Problem)**.** *Given a surface represented as a triangular mesh, or a volume represented as a tetrehedral mesh, how to compute a discrete metric that meets specific requirements from a user?*

In mathematics there are many beautiful results that provide inspirations and tools for metric design. Among all of them, an important class of results are related to classification of surfaces and 3-manifolds. For example, the *uniformization theorem* states that, any simply connected Riemann surface is isomorphic to either the Riemann sphere, the complex plane, or the open unit disc. Intuitively it means that all surfaces in real life can be conformally mapped to one of three canonical shapes: the unit sphere (S^2), the Euclidean plane (E^2), and the hyperbolic space (H^2). This theorem implies that any of such surfaces admits a canonical Riemannian metric that has constant Gaussian curvature +1, 0 or -1 and is conformal to the original Riemannian metric.

For 3-manifolds, the classification is much more complicated. According to Thurston's *geometrization conjecture*, which is recently proved, any compact 3-manifold can be decomposed into pieces that have one of eight geometric structures: S^3 , E^3 , H^3 , $S^2 \times R$, $H^2 \times R$, universal cover of $SL_2(R)$, *Nil* and *Sol*.

In this work, we treat the discrete metric design problem by closely relying on those classical theories from topology and geometry, trying to apply them to real problems in engineering fields and bridging the gap between theories and applications. In particular, we investigate three different types of surfaces or volumes, and propose three methods of discrete metric designs accordingly.

[1] **Slit map**: for genus zero surfaces (2-manifolds) with multiple holes. It computes flat metrics with four different types of boundaries that all have regular shapes. The computation is based on discrete holomorphic one-forms.

[2] **Direct product parameterization**: for handle body volumes (3-manifolds). It computes Euclidean metrics with the boundary partitioned into flat bases and vertical walls. The computation is based on slit map and volumetric harmonic fields.

[3] **Discrete Curvature Flow**: for hyperbolic 3-manifold with boundaries. It computes a hyperbolic metric of uniform negative curvature with the boundary becoming geodesics. The computation is based on a discrete curvature flow for 3-manifolds.

In the following we give a brief introduction to each of these methods in section 1.1.1, 1.1.2 and 1.1.3. More details can be found in later chapters 2, 3 and 4).

1.1.1 Slit Map for Multiply Connected Surfaces

multiply connected surfaces, which are genus zero surfaces with multiple holes (figure 1.1a), are widely used in engineering fields, such as computer vision and medical imaging. For such surfaces, we propose a novel linear computational method, called *slit map*, which can conformally map the surface to four different flat domains:

[1] Cylindric slit domain (figure 1.1b): a cylinder with the top and bottom open, where two of the original boundaries become circles at the top and the bottom, while all the other boundaries (if any) become horizonal arcs in between;

[2] Parallel slit domain (figure 1.1c): a rectangle domain where two of the original boundaries are cut open and become two horizontal lines at the top and the bottom, while all the other original boundaries become horizontal slits in between;

[3] Concentric slit domain (figure 1.1d): an annular domain where two of the original boundaries are mapped to the outer and inner circle that concentric to each other, while all the other boundaries are mapped to concentric arcs in between.

[4] Circular slit domain (figure 1.1e): a round disk where one of the original boundary becomes the outer circle, while all the other boundaries are mapped to circles inside the disk.

The four domains have different boundary shapes, and can be applied to different practical problems. For example, the concentric domain can be used as fingerprints for shape analysis and classification purposes. The circular domain can be used to match scanned brain surfaces in medical imaging. The parallel domain and cylindric domain can be used for quad remeshing of a triangular surface.

Computing a slit map is equivalent to designing a flat metric on the given surface such that the boundary has certain regular shapes. Our numerical algorithm is based on a classical result on conformal invariants of such domains ([1]). According to this theory, such a metric can be induced from a holomorphic one-form that has special behavior on the boundary of the surface. To find such a holomorphic one-form, we first compute a set of basis of holomorphic one-forms on the given surface, and then compute a linear combination

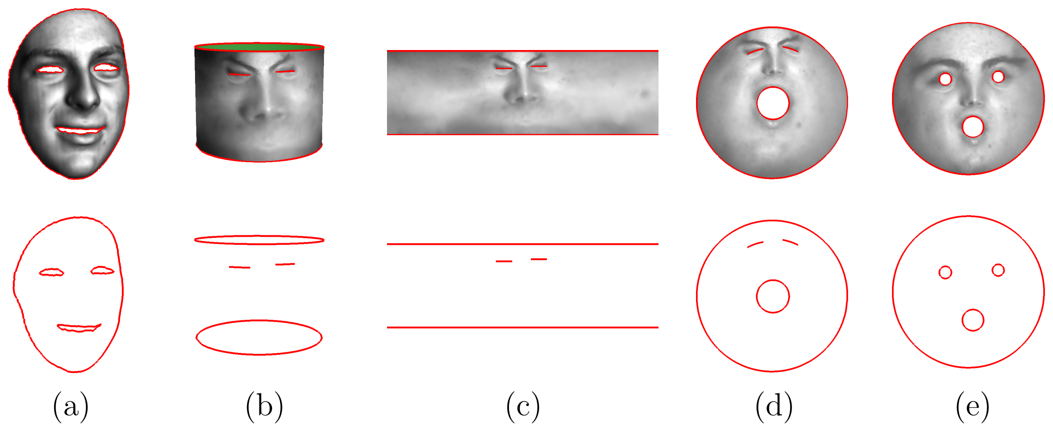


Figure 1.1: Slit map can map a multiply connected surface (a) to four different canonical domains: cylindric slit domain (b), parallel slit domain (c), concentric slit domain (d) and circular slit domain (e).

of the basis under certain constraints. The whole computation is linear, and the new metric is conformal (angle-preserving) to the original metric.

1.1.2 Direct Product Parameterization for Solid Handle Bodies

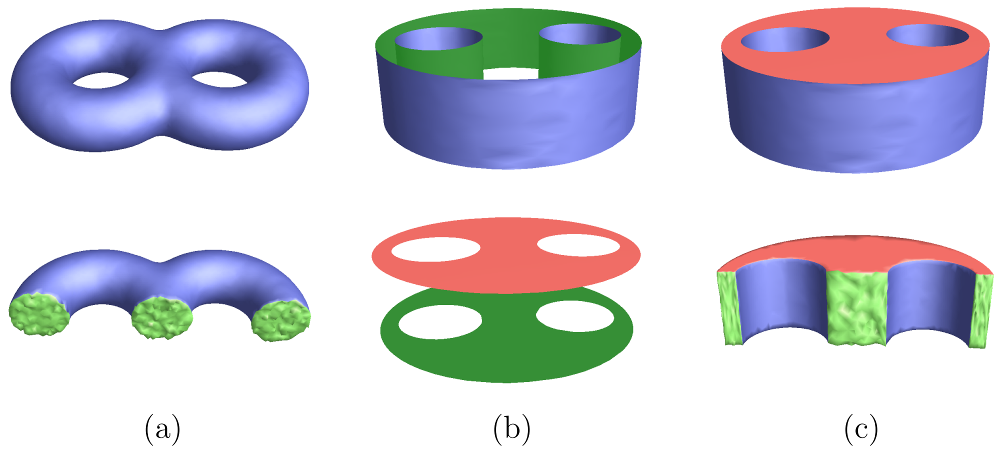


Figure 1.2: A direct product parameterization (c) for a handle body volume (a). It starts from a boundary parameterization (b).

This method targets at volumes that are handle bodies (figure 1.2), whose boundary surface is a torus with multiple handles. The shape of such a volume could be very complicated, while topologically they all have a simple nature; namely, any handle body is a direct product of a base and a fiber. The base here is a 2-dimensional disk with multiple holes, and the fiber is a 1-dimensional line segment.

From this observation, we propose a metric design method that starts from the boundary surface. We first partition the boundary surface into two bases, which are topological disks with multiple holes, and a set of walls, which are topological cylinders with both ends open. Then compute a flat metric on bases using either slit map or any alternative method, and map two bases back to back. Using these two flat bases as boundary condition, we compute a volumetric harmonic function to extend the metric of the base throughout the whole volume, which will give a parameter domain that is a direct product.

Note that for certain handle bodies, the above basic algorithm will generate huge distorsion between the given domain and the final domain. In order to relief such distorsion, we also propose an extended algorithm, which utilizes polycube map for the surface metric design, and build a map between the given volume and its polycube domain. This map can be further used to generate hexahedral meshes with high qualities.

1.1.3 Constant Curvature Metrics for Hyperbolic 3-Manifolds

This work is for hyperbolic 3-manifolds, which belongs to one of the eight basic geometric structures in Thurston's geometric conjecture, H^3. This is one of the most simple geometries in the family of eight. As an example, figure 1.3(a) shows such a 3-manifolds, called knotted Y-shape. It can be constructed by removing three knotted tunnels from a solid ball. The goal is to compute a metric that has constant negative curvature in the inside, while leaving the boundaries as geodesics.

Our numerical algorithm is directly based on a curvature flow for such 3-manifolds, proposed by Luo ([40]). It deforms a given metric according to the curvature, and the whole process is an analog of heat diffusion. At the end of the flow, the curvature will be uniformly distributed in the volume, and the curvature on the boundary becomes zero.

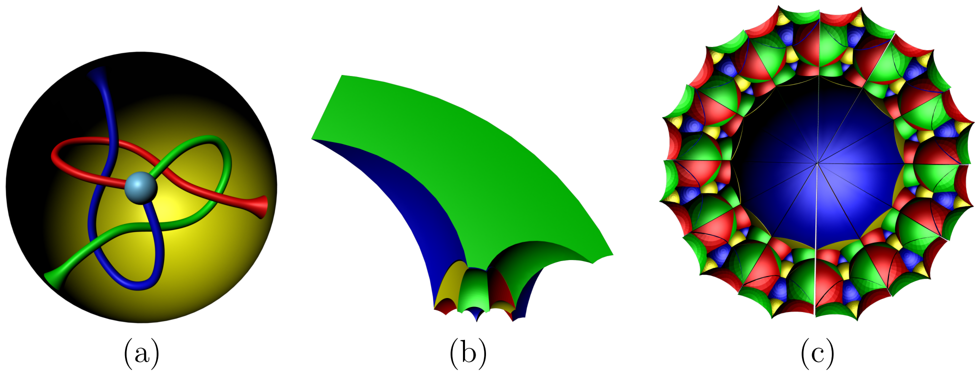


Figure 1.3: Computing constant curvature metric for knotted Y-shape (a), which is a hyperbolic 3-Manifold. With such a metric, the shape can be embedded in $\mathbb{H}^3$ in a single period (b) or multiple periods (c).

The curvature flow in [40] is designed for ideal triangulations, which is a mesh consisting of truncated tetrahedra. In order to compute such a flow, we propose an algorithm to convert a tetrahedral mesh to a truncated tetrahedral mesh, on which the constant curvature metric is computed then. We also provide algorithms to embed a given hyperbolic 3-manifold in a hyperbolic space using this hyperbolic metric.

1.2 Discrete Unit Tangent Bundles

A *unit tangent bundle* for a surface is a special fiber bundle where the fiber is a circle S^1. It is a natural representation for *unit tangent vector fields* on surfaces. Given an arbitrary point on a surface, all the unit tangent vectors there can be parameterized by a unit circle. Each unit tangent vector field can be represented as a section in the bundle, except for the singularity points (if any) where the section is not defined.

As a powerful tool in Riemannian geometry, unit tangent bundle contains important information about all the unit tangent vector fields on a surface, and therefore has a natural impact on some engineering problems, such as *vector field design* ([21], [70], [60], [53], [52], [46]) and *vector field topology* ([31], [50], [7], [8], [41], [38], [64]).

Although unit tangent bundles could theoretically play important roles in vector field study, they are unfortunately missing from the literature of those engineering fields. One of the fundamental reasons is that there is no appropriate discrete representation for such bundles, especially for those with non-trivial topology. This gives rise to the following problem that we are trying to tackle:

Problem 1.2.1 (The Discrete Unit Tangent Bundle Problem)**.** *Given a surface represented as a triangular mesh, how to generate a tetrahedral mesh that faithfully represents the unit tangent bundle of the given surface?*

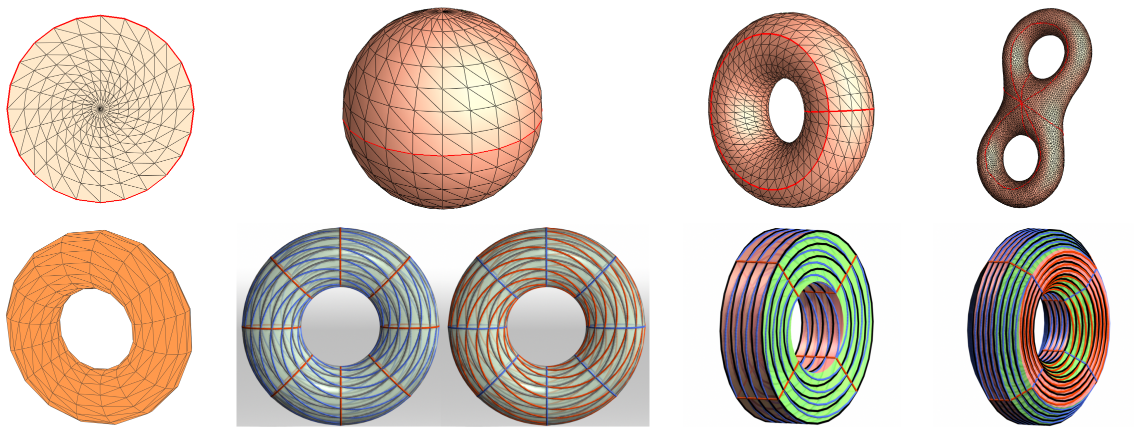

(a) (b) (c) (d)

Figure 1.4: Discrete unit tangent bundles for topological disks (a) and closed surfaces that have positive (b), zero (c) and negative (d) Eular characteristics.

In this work, we explore methods to represent unit tangent bundles for a wide range of surfaces, from topological disks to closed and oriented surfaces with arbitrary genus.

For a topological disk, the unit tangent bundle is simply a direct product of the disk with a circle, which is a solid torus (figure 1.3a). It is easy to build a tetrahedral mesh to represent such a bundle. For closed surfaces, however, the unit tangent bundles are in general non-trivial (except for genus one surfaces), and there is no obvious way to discretize such bundles with tetrahedral meshes. Although generating tetrahedral meshes has a fruitful literature, to name a few, [56],[34],[18],[43],[45],[10],[9],[15],[19] and etc, none of them could nicely handle this problem. The challenges are multi-folded:

- A unit tangent bundle for a closed surface is a closed 3-manifold, there is no boundary surface that we can start from to tessellate the interior.
- A fiber bundle like this cannot be embedded in R^3; no traditional method could generate such a tetrahedral mesh directly.
- Topologically speaking this bundle is globally nontrivial (i.e. not direct product); special considerations need to be taken in order to maintain certain regular structures in the tetrahedral mesh as much as possible, at least locally.

In order to solve problem 1.2.1 for closed surfaces, we use a *local-to-global* framework. In a *local construction* stage, we first partition the given surface into one or multiple topological disks that cover the original surface, and build unit tangent bundles for each of them. These trivial bundles are called *local bundles*, which are solid tori. In a *global construction* stage, we combine local bundle(s) into a *global bundle*, i.e. the unit tangent bundle of the original surface. This is achieved by gluing solid tori carefully along their boundaries.

This framework gives rise to two specific requirements that our solution has to meet.

- In the global construction, in order to generate a discrete representation of unit tangent bundle that is faithful to its smooth counterpart, the gluing operation should satisfy certain topological properties; this requires that the triangulation on the boundary of each local bundle must allow such faithful gluing.
- In the local construction, the tetrahedral tessellation within each local bundle should reflect the direct product nature, under constraints on the boundary triangulation that are imposed by the global construction.

Taking all these challenges and requirements into consideration, we propose a family of faithful and efficient solutions to the discrete unit tangent bundle problem for different types of surfaces. In particular,

- For a topological disk, we build its discrete unit tangent bundle with regular structures both in the interior and on the boundary. We reduce this problem to an equivalent but simpler 2D problem, and provide efficient algorithms under various types of boundary conditions. These algorithms are proved to output a valid solution to the 2D problem if there is one and are proved to output the obstructions if there is no solution.

- For a closed surface of genus zero, we build its discrete unit tangent bundle from two trivial local bundles. We also define a discrete *gluing map* to glue two local bundles up to certain topological requirements, and design a *gluable triangulation* on the boundary of each local bundle, which allows the above faithful gluing.

- For a closed surface of genus greater than zero, we build its discrete unit tangent bundle from only one trivial local bundle. Similar to the genus zero case, we design specific gluable triangulations on the boundary of the local bundle, and also a discrete gluing map to glue the local bundle to itself along its own boundary.

To the best knowledge of the author, this is the first systematic solution to build discrete unit tangent bundles using tetrahedral meshes for a large set of surfaces.

1.3 Organizations and Notations

In the rest of this dissertation we will spend three chapters on discrete metric designs; namely, chapter 2 on discrete slit map for multiply-connected surfaces, chapter 3 on direct-product parameterization of handle bodies, chapter 4 on constant curvature metric design for hyperbolic 3-manifolds. Then another three chapters are devoted to discrete unit tangent bundle designs for various types of surfaces; namely, chapter 5 for topological disks, chapter 6 for $g = 0$ closed surfaces and chapter 7 for $g > 0$ closed surfaces. Finally we conclude briefly in chapter 8.

The following notations are commonly used in the rest of the dissertation. Other specific notations will be introduced in individual chapters.

∂M denotes the boundary of a manifold M. If M is a k-dimensional manifold ($k > 0$), ∂M will be a (k-1)-dimensional manifold.

S^k denotes a k-dimensional sphere. For example, S^1 denotes a circle (1-dimensional), S^2 denotes a 2-dimensional sphere, S^3 denotes a 3-dimensional sphere.

T^k denotes a k-dimensional torus. For example, T^2 denotes a 2-dimensional torus (with only one handle), which is topologically equivalent to $S^1 \times S^1$. T^3 denotes a 3-dimensional torus, which is topologically equivalent to $S^1 \times S^1 \times S^1$.

$T^2(g)$ denotes 2-dimensional torus with g handles, i.e. a closed oriented surface of genus-g. For example, $T^2(2)$ denotes a 2-dimensional torus with two handles. And as a special case $T^2(1)$ is exactly T^2, a 2-dimensional torus with only one handle.

D^k (or B^k) denotes a k-dimensional disk (or ball), whose boundary is a (k-1)-dimensional sphere, i.e. $\partial D^k = S^{k-1}$. For example, D^2 (or B^2) denotes a 2-dimensional topological disk, whose boundary is a circle S^1; D^3 (or B^3) denotes a solid ball (3-dimensional), whose boundary is a 2-sphere S^2.

Chapter 2

Discrete Metric Design via Slit Map

2.1 Motivations and Related Work

Surface conformal parameterization is a fundamental tool in geometric modeling and processing. It is an essential technique for many applications, such as texture mapping, surface matching, registration and tracking, re-meshing, mesh-spline conversion and so on.

Many methods have been proposed to parameterize surfaces varying from closed surfaces to surfaces with boundaries, from genus zero spheres to high genus surfaces. For a general overview of the literature, we refer the readers to the comprehensive survey papers [55] and [22].

Among all these surfaces, genus zero surfaces with multiple boundaries, or namely, *multiply connected surfaces*, are widely used in many engineering fields. For example, in certain computer vision applications, people punch holes in the human face mesh at the eyes and the mouth (figure 1.1a). In medical imaging field, people specify certain line landmarks on the brain surface for registration purposes, and each of the lines can be sliced open to be a hole. Our parameterization method is targeting at these multiply connected surfaces.

Also, we are looking for parameter domains that have regular shape, such as unit disk, rectangle or something alike. Such canonical domains are highly preferable in many applications, such as texture mapping, shape comparison, computation using finite element method (FEM) and etc.

In the literature there are some methods to parameterize multiply connected surfaces, such as [14], [37], [25] and etc. But all these methods leave the boundary free and thus cannot generate any canonical parameter domain in general. In fact, the regularity of the shape of the parameter domain is mainly reflected in the shape of its boundaries. In order to achieve canonical domains, the surface boundaries have to be fixed in most (if not all) cases.

There are some parameterization methods with fixed boundaries for multiply connected surfaces. One of such work is *discrete Ricci flow* [32], which is able to map the surface to a round disk with circular holes inside. However, that method is non-linear. Actually that method targets at general surfaces, not specifically tuned for the multiply connected surfaces.

In this work, we propose a novel linear computational method, called slit map, for global conformal parameterization of multiply connected surfaces with canonical domains (figure 1.1). Four different canonical domains can be computed: cylindric slit domain (figure 1.1b), which is an open cylinder with two circle boundaries at the top and the bottom, while other boundaries (if any) become arcs around the axis of the cylinder; parallel slit domain (figure 1.1c), which is a rectangle where all the original boundaries become parallel slits; concentric slit domain (figure 1.1d), where two boundaries are mapped to concentric circles, while others mapped to concentric arcs in between; circular slit domain (figure 1.1e), which is a round disk with all the boundaries being circles.

Computing a slit map is equivalent to finding a certain holomorphic one-form, which we called *slit holomorphic one-form*, with special behavior on the surface boundaries. In recent years, discrete one-form has been studied and applied in many applications, such as vector field design and decomposition ([21], [64]), surface parameterization ([27], [25]), quad mesh design ([63]) and etc. In this work, we explicitly compute the basis of holomorphic one-forms, by an approach which is similar yet slightly different to that in [27]), and then compute the slit holomorphic one-form from there up to certain special constraints.

In a nutshell, slit map proposed in this paper own the following merits.

[1] It can generate four different canonical domains for an arbitrary multiply connected surface;

[2] The mapping is conformal, one-to-one and global (i.e. do not need to partition the surface);

[3] The algorithm is linear.

2.2 Theoretic Background

The key of computing *slit map* is to find a so called *slit holomorphic one-form*, which is a pair of orthogonal *harmonic* one-forms with special behavior on the surface boundary. Every slit map is guaranteed to be conformal and essentially one-to-one. In this section, we briefly introduce the major concepts and theories that are directly relevant to our parameterization method. For further details, the interested readers are referred to [1] and [28].

Harmonic Function

Suppose S is a surface embedded in $\mathbb{R}^3$ with induced Euclidean metric $\mathbf{g}$. S is covered by an atlas $\{(U_\alpha, \phi_\alpha)\}$. Suppose (x_α, y_α) is the local parameter on the chart (U_α, ϕ_α). We say (x_α, y_α) is *isothermal*, if the metric has the representation

$$\mathbf{g} = e^{2\lambda(x_\alpha, y_\alpha)}(dx_\alpha^2 + dy_\alpha^2).$$

The *Laplace-Beltrami operator* is defined as

$$\Delta_{\mathbf{g}} = \frac{1}{e^{2\lambda(x_\alpha, y_\alpha)}}(\frac{\partial^2}{\partial x_\alpha^2} + \frac{\partial^2}{\partial y_\alpha^2}).$$

A function $f : S \to \mathbb{R}$ is *harmonic*, if $\Delta_{\mathbf{g}} f \equiv 0$.

2.2.1 Holomorphic One-form

Suppose η is a differential one-form with the representation $f_\alpha dx_\alpha + g_\alpha dy_\alpha$ in the local parameters (x_α, y_α), and $f_\beta dx_\beta + g_\beta dy_\beta$ in the local parameters (x_β, y_β). Then

$$\begin{pmatrix} \frac{\partial x_\alpha}{\partial x_\beta} & \frac{\partial y_\alpha}{\partial x_\beta} \\ \frac{\partial x_\alpha}{\partial y_\beta} & \frac{\partial y_\alpha}{\partial y_\beta} \end{pmatrix} \begin{pmatrix} f_\alpha \\ g_\alpha \end{pmatrix} = \begin{pmatrix} f_\beta \\ g_\beta \end{pmatrix}.$$

η is a *closed one-form* if on each chart (x_α, y_α)

$$\partial f / \partial y_\alpha - \partial g / \partial x_\alpha = 0$$

η is an *exact one-form* if it equals the gradient of some function. An exact one-form is also a closed one-form.

A closed one-form η is a *harmonic one-form* if it satisfies

$$\partial f / \partial x_\alpha + \partial g / \partial y_\alpha = 0$$

The gradient of a harmonic function is an exact harmonic one-form.

The so-called *Hodge star operator* turns a one-form η to its *conjugate*

$$^*\eta = -g_\alpha dx_\alpha + f_\alpha dy_\alpha$$

Definition 2.2.1 (Holomorphic One-form). *A holomorphic one-form is a complex differential form $\eta + \sqrt{-1}^*\eta$, where η is a harmonic one-form.*

The wedge product of two one-forms $\eta_k = f_k dx + g_k dy, k = 1, 2$ is a two-form

$$\eta_1 \wedge \eta_2 = (f_1 g_2 - f_2 g_1) dx \wedge dy.$$

2.2.2 Slit Holomorphic One-form

Suppose S is an open surface with n boundaries $\gamma_0, \cdots, \gamma_{n-1}$. One can uniquely find a holomorphic one-form $\omega = \eta + \sqrt{-1}^*\eta$ such that η is exact, $^*\eta$ is closed, and

$$Im\left(\int_{\gamma_k} \omega\right) = \begin{cases} 2\pi & k = 0 \\ -2\pi & k = 1 \\ 0 & otherwise \end{cases} \tag{2.1}$$

where Im takes the imaginary part of the complex valued integration. Such a holomorphic one-form ω is called *slit holomorphic one-form.* It is periodic in the imaginary part on two of the boundaries.

All the holomorphic one-forms form a linear space, which is a finite dimensional space for multiply connected surfaces with finite number of boundary curves. The slit holomorphic one-form is a point in this space, and therefore can be expressed as a linear combination of those basis holomorphic one-forms. Both the basis and the linear combination can be computed using linear methods (section 2.3).

2.2.3 Slit Map

With the slit holomorphic one-form ω, one is allowed to map the original surface to one of four parameter domains: the cylindric, parallel, concentric and circular domain, and the corresponding slit map can be defined accordingly.

The *cylindric slit map* uses an open cylinder as the target domain, with two circle boundaries at the top and the bottom, while with all the other boundaries (if any) as arcs around the axis of the cylinder. The formal definition is as follows:

Definition 2.2.2 (Cylindric Slit Map)**.** *Let $\bar{S}$ be the universal covering space of the surface S, $\pi : \bar{S} \to S$ be the projection and $\bar{\omega} = \pi^*\omega$ be the pull back of ω. Fix a point $\bar{p}_0$ on $\bar{S}$, for any point $p \in \bar{S}$, let $\bar{\gamma}$ be an arbitrary path connecting $\bar{p}_0$ and $\bar{p}$, then the cylindric slit map is defined as*

$$\bar{\phi}(\bar{p}) = \int_{\bar{\gamma}} \bar{\omega}$$

The *parallel slit map* is an immediate variation of the cylindric slit map, by cutting and unwrapping the cylinder into a rectangle. All the original boundaries become parallel slits, plus two cutting boundaries orthogonal to them. The parallel slit map is actually a single period representation of the periodic slit holomorphic one-form.

The *concentric slit map* is essentially the exponential of the cylindric slit map. The target domain is a flat disk, with two concentric circle boundaries and several (if any) concentric arc boundaries in between. It is formally defined as:

Definition 2.2.3 (Concentric Slit Map)**.** *Fix a point p_0 on the surface, for any point $p \in S$, let γ be an arbitrary path connecting p_0 and p, then the circular slit map is defined as*

$$\phi(p) = e^{\int_\gamma \omega}$$

The *circular slit map* computes a flat disk domain, where all the boundaries are circles. It can be computed from the concentric slit domain by expanding each (round-trip) arc boundary into a perfect circle. This can be achieved through a sequence of analytic functions. First, use certain Möbius transformation to put the target arc (round trip) as the positive x-axis, with one end point at the origin and the other one at the infinity. Then take the square root function to squeeze the domain into the upper half plane, where the target arc (single trip) covers the whole x-axis. Finally use another Möbius transformation to transform the x-axis to a circle. Repeating this process for each arc, one will transform the concentric slit domain into a circular slit domain.

2.3 Algorithm Pipeline

The goal of this work is to compute a conformal mapping of a multiply connected mesh to a certain slit domain. As explained in the previous section, the mapping is acquired by integrating a slit holomorphic one-form, which is in turn calculated from a set of holomorphic one-form basis. The whole algorithm can be outlined as follows:

[1] Compute the basis for all the holomorphic one-forms; (*section 2.3.1*)

[2] Compute the slit holomorphic one-forms; (*section 2.3.2*)

[3] Compute four kinds of slit domains. (*section 2.3.3*)

In this section we assume the boundary of the input mesh has $n + 1$ connected components $\partial M = \gamma_0 - \gamma_1 - \cdots - \gamma_n$, and without loss of generality, we map γ_0 and γ_1 to the outer and inner circle of the concentric slit domain, while all the others to the concentric slits.

2.3.1 Computing holomorphic one-form basis

In order to compute a set of basis holomorphic one-forms, we employ a method very similar to that introduced in [27]. The surfaces in that work are closed, while the ones we are studying have boundaries. But essentially, both the work share the same idea. We first compute a set of harmonic one-forms, then pair each of them with its conjugate through the hodge star operation.

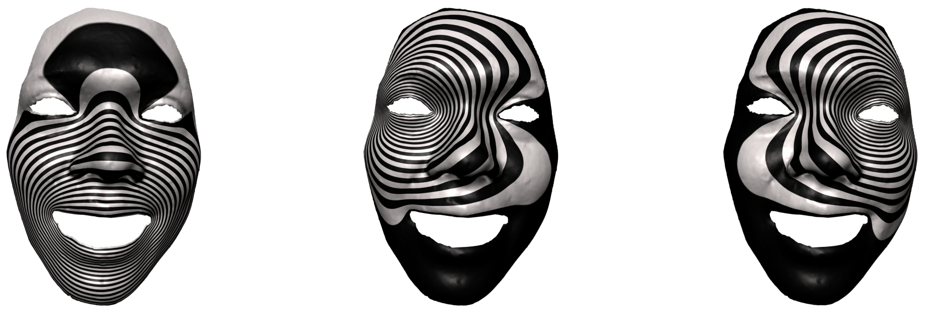

(a)η_1 (b) η_2 (c) η_3

Figure 2.1: Basis of exact harmonic one-forms.

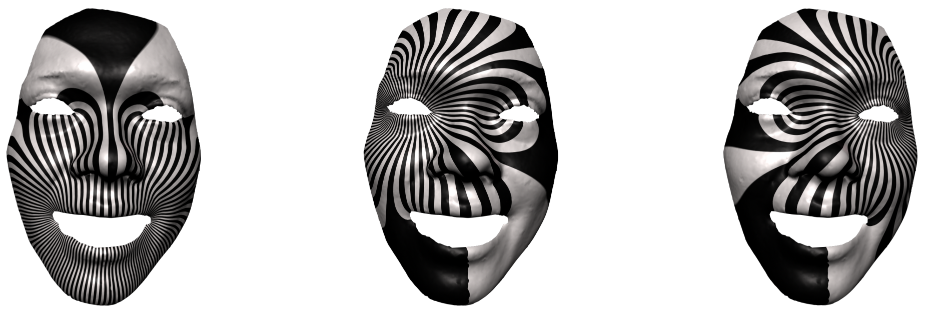

(a)τ_1 (b)τ_1 (c)τ_3

Figure 2.2: Basis of closed but not exact harmonic one-forms.

Let γ_k be an inner boundary, we compute a harmonic function $f_k : S \to \mathbb{R}$ by solving a Dirichlet problem on the mesh M:

$$\begin{cases} \Delta f_k \equiv 0 \\ f_k|_{\gamma_j} = \delta_{kj} \end{cases}$$

where δ_{kj} is the Kronecker function, Δ is the discrete Laplacian-Beltrami operator using the well-known co-tangent formula proposed in [51]. Taking the gradient of the harmonic function f_k, we get an exact harmonic one-form η_k $\eta_k = df_k$; in this way we can compute a set of exact harmonic one-forms

$$\{\eta_1, \eta_2, \cdots, \eta_n\}$$

which are visualized in figure 2.1.

The next step would be computing the conjugate for each η_k. A naive method would be rotating η_k by $90°$ about the surface normal. But in practice the resulting one-form (denoted as η'_k) is usually not accurate enough. In this work we take η'_k as an initial approximation, and then improve the accuracy utilizing the harmonic one-form basis, which can be represented as

$$\{\eta_1, \eta_2, \cdots, \eta_n, \tau_1, \tau_2, \cdots, \tau_n\}$$

where η_k comes from the previous step, while τ_k can be computed as follows.

For each inner boundary γ_k ($k > 0$), compute a path ζ_k bridging γ_k and γ_0; slice the mesh open along this path, denote the resulting mesh as M_k; ζ_k itself is split into two boundary segments ζ_k^+ and ζ_k^- in M_k. As done in [63], compute a function $g_k : M_k \to \mathbb{R}$ by solving a Dirichlet problem,

$$\begin{cases} \Delta g_k \equiv 0 \\ g_k|_{\zeta_k^+} = 1, g_k|_{\zeta_k^-} = 0. \end{cases}$$

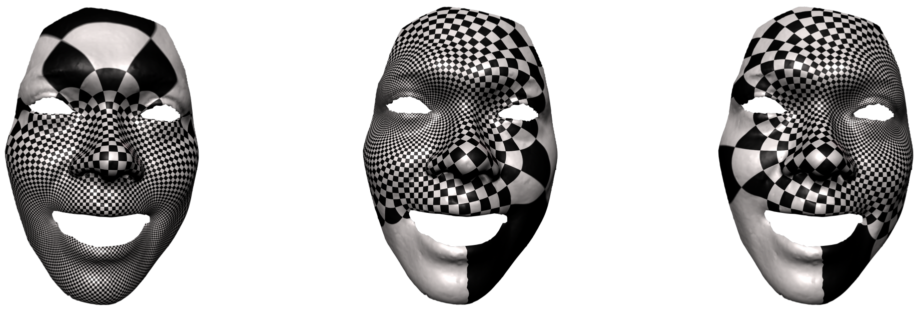
Compute the gradient of g_k and let $\tau_k = dg_k$, then map τ_k back to M, where τ_k becomes a closed one-form. In order to make it harmonic, we compute a function $h_k : M \to \mathbb{R}$ by solving a linear system $\Delta(\tau_k + dh_k) \equiv 0$, and update τ_k to be $\tau_k + dh_k$. Now we get a set of basis

$$\{\tau_1, \tau_2, \ldots, \tau_n\}$$

for all the closed but not exact harmonic one-forms (visualized in figure 2.2).

Come back to the question of computing the conjugate one-form for each η_k. Since η_k is harmonic, its conjugate ${}^*\eta_k$ should also be harmonic, and can therefore be represented as a linear combination of the base harmonic one-forms ${}^*\eta_k = \sum_{i=1}^n a_i\eta_i + \sum_{i=1}^n b_i\tau_i$. The coefficients a_i and b_i ($i = 1, 2, \cdots, n$) can be computed by solving the following linear system,

$$\int_M {}^*\eta_k \wedge \eta_i = \int_M \eta'_k \wedge \eta_i, \int_M {}^*\eta_k \wedge \tau_j = \int_M \eta'_k \wedge \tau_j.$$



$(\eta_1 + {}^*\eta_1)$ $(\eta_2 + {}^*\eta_2)$ $(\eta_3 + {}^*\eta_3)$

Figure 2.3: Basis of holomorphic one-forms

where $\wedge$ is the wedge product (section 2.2). Once each ${}^*\eta_k$ is computed this way, we can pair it with its conjugate

$$\{\eta_1 + \sqrt{-1}{}^*\eta_1, \cdots, \eta_n + \sqrt{-1}{}^*\eta_n\}$$

which is a set of basis for the holomorphic one-form group (see figure 2.3).

2.3.2 Computing slit holomorphic one-form

As explained in section 2.2, the slit holomorphic one-form ω is a special linear combination of these basis holomorphic one-forms

$$\omega = \sum_{i=1}^{n} \lambda_i(\eta_i + \sqrt{-1}{}^*\eta_i)$$

such that the imaginary part of its integration satisfies equation 2.1. This constraint can be formulated as the following linear system:

$$\begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix} = \begin{pmatrix} -2\pi \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where $\alpha_{kj} = \int_{\gamma_j} {}^*\eta_k$, and λ_i $(i = 1, \cdots, n)$ are the unknowns.

It can be proven that this linear system has a unique solution, which reflects the fact that γ_1 is mapped to the inner circle of the concentric slit domain. Further, the system implies another equation

$$\alpha_{01}\lambda_1 + \alpha_{02}\lambda_2 + \cdots + \alpha_{0n}\lambda_n = 2\pi,$$

which means that γ_0 is mapped to the outer circle in the concentric slit domain.

2.3.3 Computing slit domains

Once the slit holomorphic one-form ω has been computed, we can integrate it on the mesh. Starting from a base vertex v_0 (which can be chosen arbitrarily), build a spanning tree among all the vertices. This tree will induce a path ζ_i from v_0 to any other vertex v_i. Then we can assign each vertex with a complex number $h_i + \sqrt{-1}\theta_i$ by integrating ω along ζ_i:

$$\begin{cases} h_i = Re\left(\int_{v_0}^{v_i}\omega\right) \\ \theta_i = Im\left(\int_{v_0}^{v_i}\omega\right) \end{cases} \tag{2.2}$$

From this integration, we can directly compute the cylindric slit domain (2.3.3), and then , parallel (2.3.3) and concentric (2.3.3) slit domain accordingly. The construction of the circular slit domain (2.3.3) is more complicated than that of the other three domains, and need further treatments using complex analysis techniques.

The cylindric slit domain The cylindric slit domain (figure 1.1b) can be built from the integration of slit holomorphic one-form directly. In the cylindric slit domain, the (x, y, z) coordinate for each vertex v_i can be represented as $(\cos\theta_i, \sin\theta_i, h_i)$. Obviously, the radius of the sectional circle is 1.0. Please note that, although the range of θ could be beyond 2π, the cylindric slit domain is still well defined since the cos and sin will regulate θ by 2π period.

The parallel slit domain To compute the parallel slit domain (figure 1.1c), we can re-parameterize the cylindric domain by replacing $\theta \in \mathbb{R}$ with $\alpha \in [0..2\pi]$. Cutting the cylinder surface open along the vertical line $\alpha = 0$, and unrolling the surface into the XY plane, we get the parallel slit domain. The (x, y) coordinate for each vertex v_i is represented as (α_i, h_i). Note that the width of the domain is automatically normalized to 2π this way.

The concentric slit domain The concentric slit domain (figure 1.1d) is computed by taking the exponential of the integration 2.2. Namely, The complex coordinate for each vertex v_i is represented as

$$x_i + \sqrt{-1}y_i = e^{h_i+\sqrt{-1}\theta_i}$$

The circular slit domain The circular slit domain (figure 2.5d) can be computed from the concentric slit domain. As an initialization, we normalize the concentric slit domain to a unit disk (figure 2.5a); that is, the radius of the outer circle boundary is 1. For each concentric slit arc γ_i $(i > 1)$, let v_1^i and v_2^i be the vertices at the two ends of the arc, let z_1^i and z_2^i be their complex coordinates respectively. Repeating the following steps for $i = 2, \cdot, n$, one will expand each slit arc γ_i into a full circle.

Firstly, take the Möbius transformation $\mathrm{z} = (\mathrm{z} - \mathrm{z}_1^i)/(\mathrm{z} - \mathrm{z}_2^i)$ on the whole domain, so that v_1^i and v_2^i are mapped to the original and the infinity respectively. In order to handle the infinity point, we take a stereo graphic projection and map the flat domain to a sphere (figure 2.4a), where v_1^i and v_2^i appears at the south and north pole respectively. Due to the fact that γ_i is a round-trip arc in the concentric slit domain, now it is mapped to a half of a great circle (round-trip) along a longitude line on the sphere connecting the two poles.

Secondly, rotate the sphere around the z-axis until the half great circle γ_i falls into the $y = 0$ plane (figure 2.4b). If we map the sphere back to the complex plane by an inverse stereo graphic projection, γ_i will lie on the x-axis, with v_1^i and v_2^i at the original and infinity respectively.

Thirdly, take the complex square root $\mathrm{z} = \mathrm{z}^{1/2}$ operation to cut the complex plane along the positive x-axis and squeeze the whole plane into the $y \geq 0$ half plane, where γ_i spans the whole x-axis. Again, in order to accommodate the infinity point nicely, this step can be carried out on the sphere instead; it will squeeze the whole sphere into a half sphere embedded in the $y \geq 0$ half

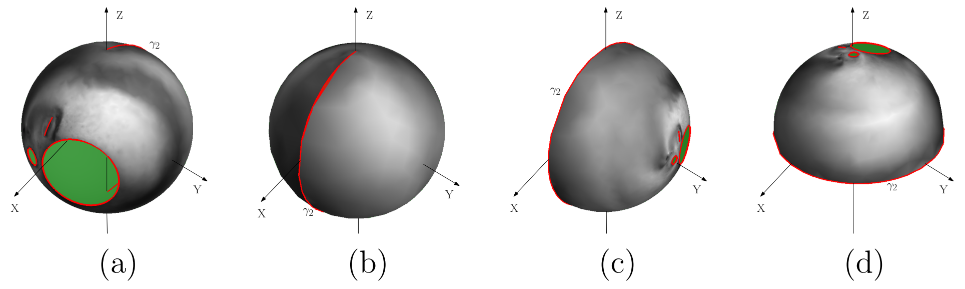


Figure 2.4: Expanding slit arc γ_2 to a full circle (section 2.3.3). (a) stereo graphic projection of the concentric slit domain int figure 2.5a. (b) the sphere is rotated around the z-axis to place γ_2 in the $y = 0$ plane. (c) the arc is expanded to a full circle by taking the complex square root. (d) the semi-sphere is rotated around the x-axis to place γ_2 in the $y = 0$ plane.

space (figure 2.4c). The key point is, γ_i is now expanded into a full circle on the $y = 0$ plane.

Finally, rotate the half sphere around the x-axis until the circle γ_i falls on the $z = 0$ plane (figure 2.4d), then take an inverse stereo graphic projection to map the half sphere to a unit disk on the complex plane (figure 2.5b). Now γ_i is turned into the outer boundary circle.

Note that, since all the above operations are angle-preserving, each circle before the iteration will still be a circle after the iteration, and each slit arc will remain an arc (i.e. part of a certain circle). Therefore the above process can be carried out on all the slit arcs iteratively until all of them are turned into circles (figure 2.5b, 2.5c). After this, an inversion operation will bring γ_0 back to the outer boundary (figure 2.5d).

2.4 Applications and Extensions

Slit map can parameterize a given surface with four different slit domains. Although all these domains are derived from the same holomorphic one-form, they are quite different from one another in terms of their boundary shapes, and thus can potentially facilitate a wide range of applications with different domain preference.

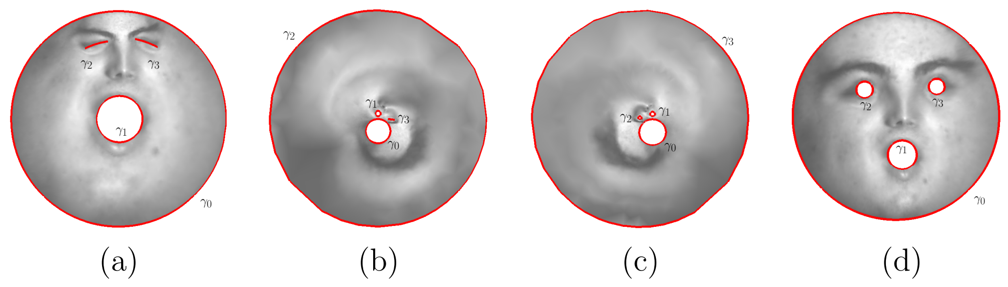


Figure 2.5: Construction of the circular slit domain (section 2.3.3). (a) is the concentric slit domain with two arcs γ_2 and γ_3. (b) and (c) turn slit arc γ_2 and γ_3 into full circles respectively. (d) is the final circular slit domain that enforces γ_0 to be the outer boundary.

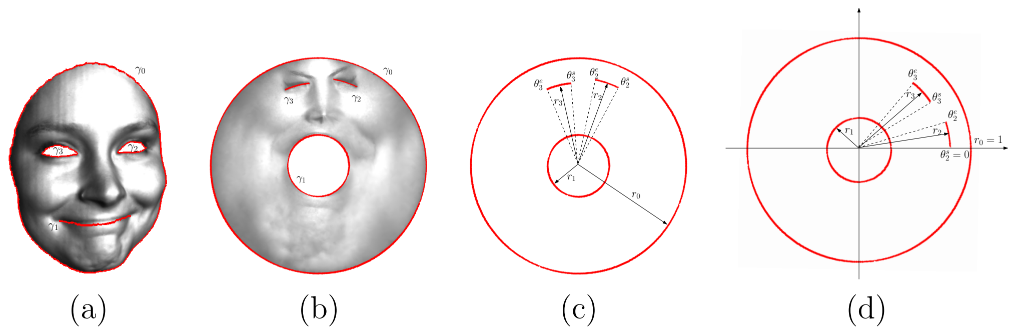


Figure 2.6: Finger-prints visualized by slit domains.

2.4.1 Surface Fingerprint

Table 2.1: Numerical fingerprints (normalized) for faces in figure 2.7.

Model	Finger-prints $[r_1, r_2, r_3, \theta_2^e, \theta_3^s, \theta_3^e]$
fig.2.7a	[0.231425, 0.815924, 0.763043, 0.264949, 0.553551, 0.860910]
fig.2.7b	[0.287076, 0.814576, 0.769996, 0.277660, 0.561342, 0.865693]
fig.2.7b	[0.243699, 0.776883, 0.744513, 0.247569, 0.554127, 0.869185]
fig.2.7b	[0.177832, 0.768475, 0.752875, 0.322658, 0.657879, 1.007431]

Slit map computes the conformal invariants of the surface. The shape parameters of the circular slit domain indicate the conformal equivalence class of the surface and can be treated as the fingerprints of the surface. We test our algorithm for several human faces from different persons with different expressions. The result is illustrated in figure 2.6. From this figure, it is very clear that the fingerprints of the three calm faces are very similar, whereas the

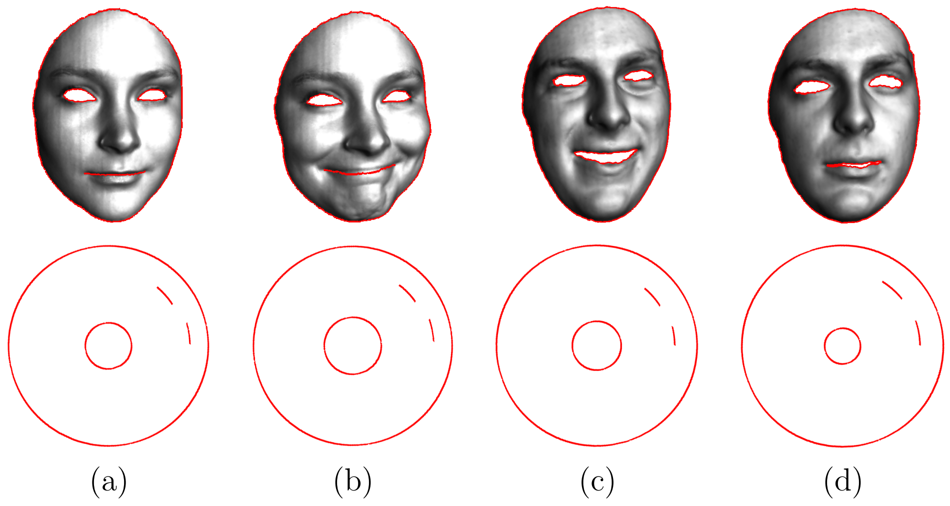


Figure 2.7: Fingerprints for 4 face models using concentric slit domains; this is a visualization of their numerical fingerprints in table 2.1.

fingerprint of the laughing face is quite different from others. This gives us a way to measure the expression quantitatively.

2.4.2 Brain Mapping

Slit map provides a valuable tool for conformal brain mapping with landmarks. As shown in figure 2.8, brain surfaces are highly convoluted. It is a great challenge to match two cortical surfaces directly in $\mathbb{R}^3$. Conformal brain mapping flattens the brain surface onto the canonical domains. Special landmarks are labeled on the surface, which are required to be registered across different brain surfaces. By using slit map, all the land marks are mapped to concentric or parallel slits, and the whole brain is mapped to annulus or rectangle. This makes the down stream registration and analysis much easier. In figure 2.8, the mapping result is shown on the circular slit domain and the parallel slit domain respectively.

2.4.3 Quad Remeshing

Slit map can be used to convert a given triangular mesh to a quadrilateral mesh. Both the parallel slit domain and the cylindric slit domain can be used

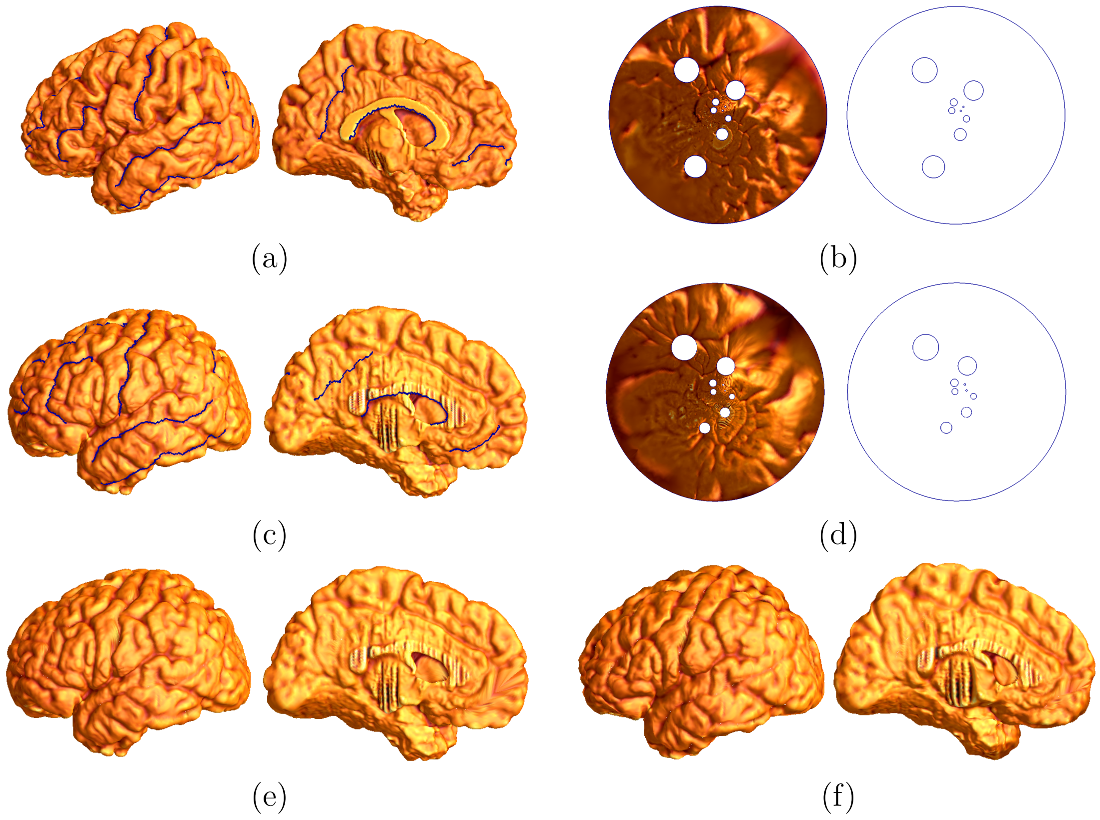


Figure 2.8: Brain mapping using circular slit domain (data courtesy by LONI, UCLA). (a) and (c) are two brain surfaces with 10 landmarks each. (b) and (d) are their circular slit domains respectively. (e) is the brain surface (c) resampled using (a)'s triangulation, (f) is the average shape between (a) and (e).

to do the task. For example, in a parallel slit domain, one just needs to trace a set of horizontal lines and a set of vertical lines. Note that for each horizontal slit, there should be one horizontal line placed at that height, and two vertical lines passing through two ends of the slit. Figure 2.9 shows a result of quad remeshing over a triangulated human face. Note how the iso lines are parallel or perpendicular to boundary lines.

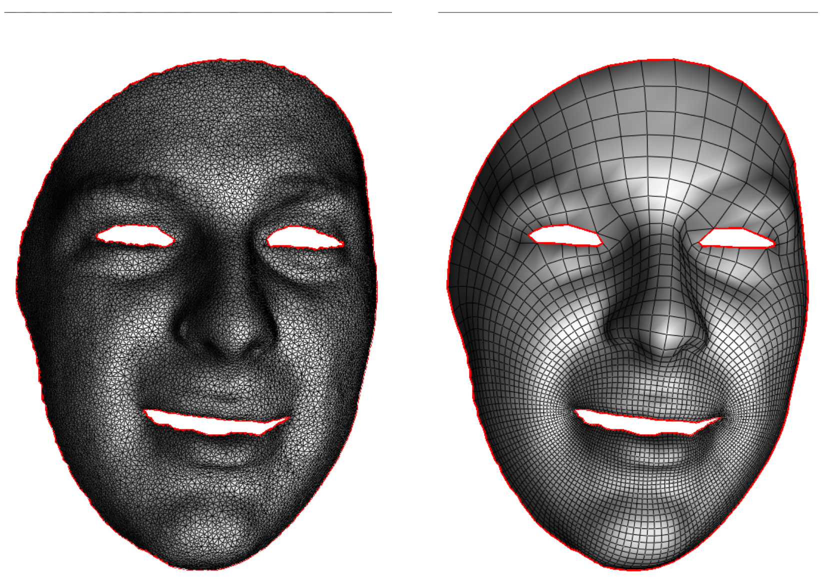


Figure 2.9: Slit map can convert a triangular mesh (on the left) to a quadrilateral mesh (on the right).

2.4.4 Extension to Closed Surfaces

Although this work focuses on multiply connected surfaces, in fact our algorithm can be easily generalized to handle high genus closed surfaces, such as the eight model in figure 2.10. The only extra requirement is to slice the surface open along certain cycles (see figure 2.10 c). Such cycles can be automatically computed using methods like that proposed by T. Dey et al in [16]. After turning the surface into a multiply connected one, we can carry out the slit algorithm thereafter directly.

2.5 Performance

We implemented the slit map algorithm in C++ on Windows platform. All the experiments were carried out on a IBM T-42 laptop with 1.80 GHz Intel Pentium M processor and 768 MB RAM. We used MATLAB as the sparse linear solver.

We tested the algorithm pipeline on multiple models with various number

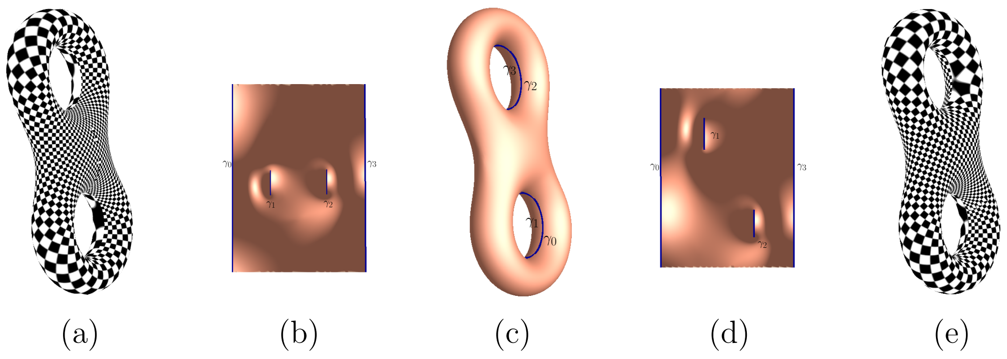


Figure 2.10: Slit map of closed mesh. The closed eight model (c) is sliced open into an annulus with 4 boundaries. (b) and (d) show the parallel slit domains with different prescribed outer boundaries. (a) and (e) show the texture mapping corresponding to domain (b) and (d) respectively.

of triangles and various number of boundary curves. The time consumed for each model is listed in table 2.2.

The time complexity of the slit map algorithm depends on two factors. One is the size of the mesh in terms of total number of triangle faces. For sophia face models with 3 boundary curves, the total computational time ranges from 2 to 20 seconds as the number of triangles varies from 4,000 to 30,000.

Another factor is the number of boundary curves. For david face models consisting of about 8,000 faces, the computational time is ranging from 4 to 25 seconds while the number of boundary curves varies from 3 to 9.

Both factors have much more impacts on the computation of holomorphic one-form basis than on the construction of the slit domains. Actually most of the computational time is taken by the basis construction. For the face model in figure 1.1, it takes 20 seconds to compute the basis, while only less than 3 seconds to compute the slit holomorphic one-form and the four slit domains.

Once the holomorphic one-form basis are acquired, one is allowed to choose different boundary curves as full circles. Figure 2.11 shows an example with different boundary configurations. All of them are computed from the same holomorphic one-form basis. From the discussion above, once the computationally heavy basis are computed, one is allowed to compute different slit domains with various boundary configurations at low prices.

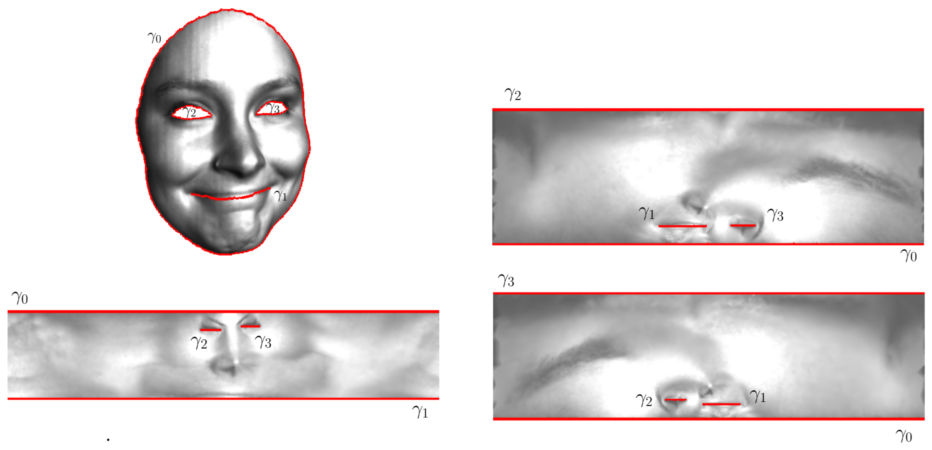


Figure 2.11: Slit map with different boundary configurations.

Table 2.2: Computational time

Model	number of boundaries	number of triangles	seconds to compute basis	seconds to compute 4 slit domains
sophia face	3	4,000	1	1
sophia face	3	10,000	4	1
sophia face	3	20,000	10	2
sophia face	3	30,000	17	3
david face	3	8,000	3	1
david face	5	8,000	7	1
david face	7	8,000	16	1
david face	9	8,000	24	1

Chapter 3

Discrete Metric Design via Direct Product Parameterizations

3.1 Motivations and Contributions

In this chapter we focus attention on discrete metric design for volumes with handles, namely *handle bodies*(3.3). In engineering fields such models are so common nowadays that the parameterization of them is becoming more and more prized and urgent. Due to their complicated topological nature, however, the task is quite challenging. But the idea behind our work is very straightforward and effective.

Note that many canonical domains have extremely simple structures. Some of them are just *direct product* of shapes from lower dimensions. Figure 3.1 shows examples including a solid cube, which is the direct product of three 1- dimensional line segments, and a solid torus, which is the direct product of a 2-dimensional disk and a 1-dimensional circle.

The intuition of our method is to compute a parameter domain that is the direct product of some sub-domains. Due to the shape characteristic of handle bodies, the sub-domains we choose here are 2D annulus with multiple holes and 1D line segment. The direct product nature is achieved by extending the parameterization of the 2D annulus through out the volume along a special harmonic field that is orthogonal to the annulus. The whole process is like building a *foliation* for the volume.

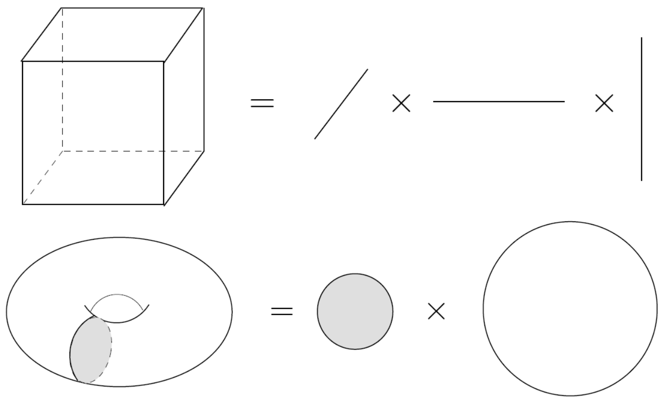


Figure 3.1: Volumetric domains that are direct products.

The reason we prefer direct product domains is multi-folded. Firstly, in a direct product domain there is no singularity, thus the parameterization would not degenerate at any point of the volume. Secondly, such a domain naturally allows a structure of orthogonality. In fact, at any point of the volume the three parameter lines are guaranteed to be orthogonal to one another. Thirdly and consequently, such structures will make many tasks easier, such as volumetric remeshing, volumetric registration, physical simulation and so on.

In a nutshell, this work distinguishes itself from other related works (section 3.2) with the following contributions:

- The algorithm is able to parameterize volumes with arbitrary number of handles; it can also be downgraded and applied to volumes with trivial topology (i.e. topological balls).
- The parameter domain is the direct product of a surface patch and a line segment; the mapping between the original volume and the parameter domain is homeomorphism, and there is no singularity.
- At any point in the volume, the three parameter lines are orthogonal to one another, which is a natural consequence of enforcing a conformal parameterization on the surface and a gradient field in the volume.
- It provides a way to extend polycube maps from the boundary surface into the volume for general handle bodies (see section 3.5), reducing the volumetric distortion tremendously.

The rest of the chapter is organized as follows. Section 3.2 gives an overview of related works. Section 3.3 explains the necessary notations and background knowledge. The details of the basic algorithm are presented in section 3.4; an extended algorithm dedicated to volumetric polycube map construction is in section 3.5, plus some experimental results.

3.2 Related Work

3.2.1 Volumetric Parameterization

Among the limited literature of parameterization for volumetric data sets, one of the earliest work is by Wang et al. [67]. They proposed a harmonic method that maps volumetric imaging data of human brains to solid balls. Their algorithm is only able to parameterize topological balls. Nevertheless, they provided a very simple version of volumetric harmonic maps for tetrahadron meshes that is adopted in this work.

Li et al. [39] used the method of fundamental solution (MFS) to build a mapping between volumes with the same topology. That method is essentially a simulation of electric fields over point clouds, and requires to place enough extra points (i.e. electric charges) off the boundary to enforce an approximated boundary condition. It is able to process handle bodies; but no direct product structure has been presented in there.

One of the methods that are most related to ours in spirits is by Martin et al. [42]. Their method starts from a parameterization of the boundary surface and extends it inwards to a one skeleton of the volume. The result is almost a direct product everywhere, except for the regions around the ends of the one skeleton, where singularities are introduced. What is more, that method is mainly targeting at volumes without handles.

As a comparison, the method presented in this work can parameterize arbitrary handle bodies with a direct-product parameter domain without any singularity. The input to the algorithm is a tetrahedral mesh $\mathbb{M}$ (see section 3.3), which can either be given directly, or be generated from a triangular mesh of its boundary surface using softwares like Tetgen ([57]).

3.2.2 Surface Parameterization

Our algorithm starts from a parameterization of a surface patch on the boundary of the volume. Surface parameterization has drawn huge attentions in the past decades, and the body of research dedicated to it is quite vast. The survey paper by Sheffer et al [55] and that by Floater et al. [22] are both good reference for general interests. Here we only review some works that are most relevant to ours; they should be able to handle genus zero surfaces with multiple holes, which are the *bases* in our surface partition (see section 3.3.1), and should be able to generate regular boundaries in the parameter domain. Furthermore, we require that the paramterization should be conformal, or angle preserving.

Many linear methods have been proposed for the conformal mapping. For example, *DCP* [14] and *LSCM* [37] are two of the earliest works that can handle multi-holed annuli. Both of them construct the mapping by solving certain linear systems with fixed or free boundary conditions. *Slit map* [68] is another linear method proposed to achieve regular boundaries of the annulus, where all the boundaries are mapped to parallel straight slits or concentric circles and arcs. All such methods could be extremely efficient in computation, but are lack of flexible control on the boundaries.

It turns out that finer control on the boundaries usually incurs much heavier computation. One of the major approaches is via *discrete curvature flow* [69], which uses curvature constraints to guide the metric deformation. Several non-linear computational methods have been proposed along this line, such as the *circle pattern* by Kharevych et al. [35], the *discrete Ricci flow* by Jin et al. [33] and the *conformal equivalence* by Springborn et al. [58]. To alleviate the computational burden, Ben-Chen et al. [3] proposed a linearized method with a trade-off that depends on applications. All these methods allow the user to design the boundary shape on his own will by prescribing the appropriate curvature on the boundary (as well as that inside).

3.2.3 Polycube Map

As a way to model the boundary surface, *polycube map* for surface meshes offers a rectangular structure which necessarily facilitates subsequent geometric computing and shape analysis. This concept was pioneered by Tarini et al. [59], who explicitly project the surface to an polycube domain. Wang et al. proposed an intrinsic method to construct the polycube map [65]. This method was improved later [66] to allow user controls over the placement of singularity points, and applied to the construction of manifold spline [26]. All these works focus on surface polycube maps, while our work could start from such a surface polycube map and extend it into the bounded volume.

3.3 Preliminaries and Notations

In this part we briefly introduce some necessary background knowledge that is underlying our algorithm, as well as the notations used in this work.

3.3.1 Handle Bodies and Boundary Partition

In general, an n-hole ($n \geq 1$) *handle body* is a 3-manifold whose boundary is a surface that can be continuously deformed to (i.e. *isotopic* to) some unknotted n-hole torus without tearing or self intersection. For such a volume ∂H, its boundary surface ∂H can be covered by a set of charts,

$$\partial H = B_0 \cup B_1 \cup \bigcup_{i=0}^{n} D_i$$

where B_0 and B_1 are called *bases*, or to be specific, B_0 is the *floor* and B_1 the *ceiling* respectively. They are two disjoint n-hole annuli, $B_0 \cap B_1 = \varnothing$. Each $D_i (i \in [0..n])$ is called a *wall*, which is a topological cylinder with both ends open. All the walls are pairwisely disjoint, $D_i \cap D_j = \varnothing (i \neq j)$. A base $B_k (k \in \{0, 1\})$ and a wall $D_l (l \in [0..n])$ intersect at a 1-dimensional simple loop, $\zeta_{kl} = B_k \cap D_l$.

As a special case, volumes without any handle (i.e. topological solid balls) can be considered as degenerate handle bodies with $n = 0$. The boundary surface for such bodies can also be partitioned into bases and walls, where each base is a topological disk and there is only one single wall. Our algorithm can be essentially applied to these topological balls, though the focus of the following presentation is mainly on general handle bodies with $n \geq 1$. We will make it clear in the context whenever it is necessary to distinguish them.

For the purpose of computation, a volume is usually modeled as point clouds or piecewise linear meshes. In this work, every handle body is represented as a tetrahedral mesh

$$\mathbb{M} = (\mathbb{T}, \mathbb{F}, \mathbb{E}, \mathbb{V}, \mathbb{C})$$

where $\mathbb{T}$, $\mathbb{F}$, $\mathbb{E}$ and $\mathbb{V}$ are the sets of tetrahedra, triangular faces, edges and vertices in the mesh, while $\mathbb{C}$ describes the connectivity among them. Furthermore, $\mathbb{F}_{B_i}$, $\mathbb{E}_{B_i}$, $\mathbb{V}_{B_i}$ and $\mathbb{C}_{B_i}$ denotes the set of faces, set of edges, set of vertices and connectivity for a base B_i ($i \in \{0, 1\}$); similarly, $\mathbb{F}_{D_i}$, $\mathbb{E}_{D_i}$, $\mathbb{V}_{D_i}$ and $\mathbb{C}_{D_i}$ denotes the counterparts for a wall D_i ($i \in [0..n]$);

3.3.2 Volumetric Parameterization

For surfaces, *parameterization* is the process of computing a mapping between the original surface mesh in $\mathbb{R}^3$ and a *parameter domain* that is usually a planar mesh in $\mathbb{R}^2$. This is equivalent to assigning a pair of real valued *coordinates* to every vertex in the mesh.

Parameterization for volume data can be defined similarly. Given a tetrahedral mesh $\mathbb{M}$ with vertex set $\mathbb{V}$, each vertex $v_i \in \mathbb{V}$ should be assigned with a triple coordinates (u_i, v_i, w_i). That amounts to computing three real valued *parameter functions*:

$$\{\mathbf{u}, \mathbf{v}, \mathbf{w}\} : \mathbb{V} \rightarrow \mathbb{R}$$

Note that although these functions are by definition restricted on vertices, it can be extended through out the whole tetrahedral mesh piecewisely. Namely, the function value for an arbitrary point in the volume is defined as the interpolation of the values on the four vertices of the enclosing tetrahedron. Here we use the same symbol to denote both the function restricted to vertices and that extended to the volume.

As a result, these parameter functions induce a piecewise-linear map from the original volumetric mesh $\mathbb{M}$ to a parameter domain $\mathbb{N}$. The domain $\mathbb{N}$ is a subset of $\mathbb{R}^3$, and should ideally have a very regular shape. In our case, the parameter domain is a direct product of a multi-holed annulus (parameterized by $\mathbf{u}$, $\mathbf{v}$) and a straight line segment (parameterized by $\mathbf{w}$).

3.3.3 Volumetric Harmonic Function

In general, a function $\mathbf{f}$ is *harmonic* if it satisfies the Laplace's equation $\triangle \mathbf{f} = 0$. If Dirichlet boundary condition is imposed on this partial differential equation, a harmonic function is the solution of the Dirichlet's problem. The same concepts can be well formulated on volumes in a discrete setting.

Given a tetrahedral mesh $\mathbb{M} = (\mathbb{T}, \mathbb{F}, \mathbb{E}, \mathbb{V}, \mathbb{C})$, let $\mathbf{w} : \mathbb{V} \to \mathbb{R}$ be a real valued function defined over the vertices, let $w_i = \mathbf{w}(v_i)$ $(v_i \in \mathbb{V})$. $\mathbf{w}$ is harmonic if and only if it satisfies the following discrete Laplace's equation:

$$\sum_{e_{ij} \in \mathbb{E}} k_{ij}(\mathbf{w}_j - \mathbf{w}_i) = 0$$

where e_{ij} is an edge connecting vertex v_i to v_j; k_{ij} is a real valued *weight* assigned with e_{ij} in the following way. Suppose edge e_{ij} is shared by t adjacent tetrahedra, it lies against t dihedral angles $\{\theta_k\}$, $k = 1, ..., t$. let l_{ij} be the length of edge e_{ij}. Then the edge weight for e_{ij} can be defined as

$$k_{ij} = \frac{1}{12} \sum_{k=1}^{t} l_{ij} \cot \theta_k$$

Same to that in the smooth setting, we can impose Dirichlet boundary conditions on the discrete volumetric harmonic function. Namely, we set the value of $\mathbf{w}$ fixed on certain vertices $v_i \in \mathbb{V}_c$, where $\mathbb{V}_c$ is the set of vertices that serve as constraint vertices.

Once a harmonic function $\mathbf{w}$ is computed over a tetrahedral mesh, one can compute its gradient $\nabla \mathbf{w}$, which is a vector field that is piecewise constant. Starting from an arbitrary point in the volume, one can trace an integral curve of the gradient field. In our work such a curve is called a *fiber*.

3.4 The Basic Algorithm

This section presents the algorithmic details of direct product volume parameterization. Given a tetrahedral mesh $\mathbb{M}$ of some handle body, our algorithm starts by partitioning the boundary surface $\partial\mathbb{M}$ into the floor, the ceiling and the walls. Then the floor is conformally parameterized using $\mathbf{u}$ and $\mathbf{v}$. Next, a special harmonic field $\nabla \mathbf{w}$ is computed throughout the volume. By tracing the integral lines (i.e. fibers), the $\mathbf{u}$, $\mathbf{v}$ parameters on the floor are smoothly extended through the volume all the way up to the ceiling; meanwhile, the $\mathbf{w}$

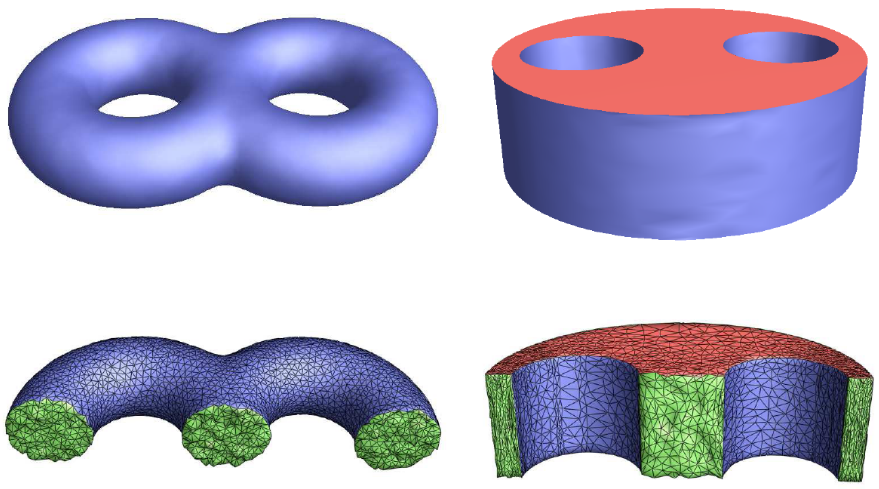


Figure 3.2: Tetrahedral mesh for a handle body with two handles (left) and the direct product domain (right).

parameter for each inner vertex is also determined by how far it is from the floor along the fiber.

The proposed algorithm contains the following steps:

- Input: A tetrahedral mesh $\mathbb{M}$ for a handle body.
- Output: A parameterization using three parameter functions $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$.
- Outline:

  [1] Partition the boundary surface $\partial\mathbb{M}$ into bases B_i and walls D_j (Section 3.4.1);

  [2] Parameterize the floor B_0 with $\mathbf{u}$ and $\mathbf{v}$ (Section 3.4.2);

  [3] Compute a harmonic function $\mathbf{w}$ over $\mathbb{M}$ (Section 3.4.3);

  [4] Tracing fibers and compute the $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ values for every vertex in the volume (Section 3.4.4).

3.4.1 Boundary Surface Partition

Our volumetric parameterization method starts from the manipulation of the boundary surface, since the shape of the final parameter domain is totally constrained by the boundary surface. In our case this amounts to a partition of the boundary surface into bases and walls (figure 3.4, 3.3 and 3.8).

The most straightforward one is to label each chart manually. To be specific, the user could specify a set of cycles on the surface, and split the surface into the desired patches along those cycles. This way one has the full control on the distortion and resolution of the final parameter domain. However, this may be very tedious for large-scale models.

In the following we introduce several automatic techniques that fit in our algorithm. One is called *normal-guided flooding*, which picks the bases directly and make the walls as left-overs. Another is called *tunnel cycle extension*, which starts from identifying the walls. A third method is by picking sharp edges in a *polycube map*.

Normal-Guided Flooding This technique traces a connected surface region where the variation of the surface normal is bounded from the above by a given threshold ϵ. The tracing is a Bread First Search (BFS) over the vertices of the surface mesh, starting from a seed vertex v_0 specified by the user. Let $\boldsymbol{n_0}$ be the normal at v_0; the search is halted at a vertex v_i if and only if one of the following two conditions is satisfied: first, all the vertices neighboring to v_i are already included in the region; second, the normal $\boldsymbol{n_i}$ there varies too much from $\boldsymbol{n_0}$, that is, $|1.0 - \boldsymbol{n_i} \cdot \boldsymbol{n_0}| > \epsilon$. Please note that all the normal vectors here are normalized to be unit vectors.

Using this technique, the user can trace the floor and the ceiling. The position and coverage of these patches can be optimized by adjusting the choice of seed vertices and the thresholds.

Tunnel Cycle Extension This technique traces a cylinder-shaped surface region. Similar to the normal-guided flooding, this one is also a BFS process, except for that the seed here is a cycle rather than a single vertex. The seed cycle should be placed around a tunnel, and extend to both side by a certain amount that can be specified by the user.

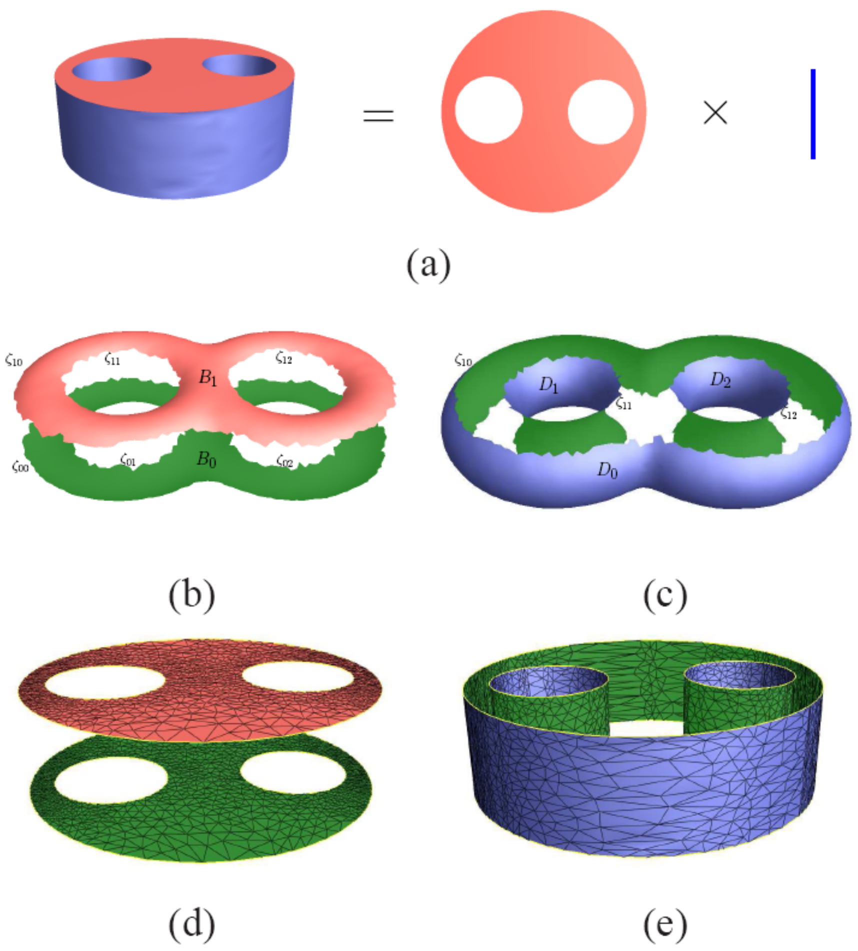


Figure 3.3: A direct-product domain for the Eight model.

Note that among all the $n + 1$ walls, there is usually a wider one, say D_0, that encloses the other n thinner ones. The tunnel cycles for those thin walls actually belong to a set of the homology group generators for the original boundary surface. Such generators can be traced automatically by the method of Erickson and Whittlesey [20] or that of Dey et al. [17]. The later one can even generate the shortest loop around a tunnel. For the wide wall, the tunnel cycle should be specified separately as a topological combination of the other cycles.

Figure 3.3 shows a boundary partition for an Eight model, where the floor and ceiling are traced using the above methods.

Partition via Polycubes Polycube map gives a parameter domain that roughly resembles the given mesh. A polycube domain is a union of multiple cubes. In such a domain it is very easy to define loops that goes along those sharp edges. By careful selection of such sharp loops, the polycube domain can be partitioned into multi-holed bases and cylindric walls. This will induce a partition in the original domain that serves the purpose for our algorithm. Actually, if one is able to compute a polycube map for the boundary surface, the surface partition and surface parameterization will come as a side-product of the map. Figure 3.9 and 3.8 show two examples of partition using polycube maps.

3.4.2 Boundary Surface Parameterization

After the boundary surface is partitioned, one of the bases needs to be parameterized conformally; without loss of generality, we choose the floor in the experiments. Recall that the bases are in general annuli with multiple holes; any algorithm that can conformally parameterize such domains will fit here. For example, *slit map* ([68]) can generate four different types of canonical domains. The shape of each domain is determined by the intrinsic invariants of the given surface. Another method is called *discrete Ricci flow* ([33]), which provides the user with much more flexibility to prescribe the boundary shape.

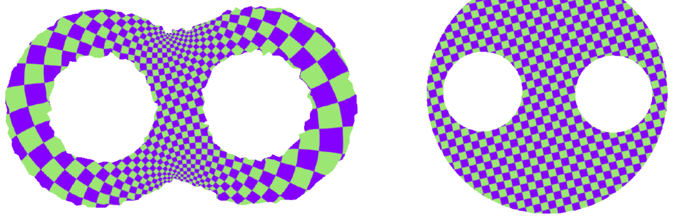


Figure 3.4: Conformal parameterization by discrete Ricci flow. (a) is the input surface, (b) is the parameter domain.

The parameter domain could have various shapes with different prescription of the boundary curvature. For the base surface in the Eight model, for example, one of the choices is to map to a planar round disk with two circular holes inside, as shown in figure 3.3; the angle-preserving nature of the conformal map is shown in 3.4.

Another choice is to map the same surface to a rectangle with two rectangle disks removed (figure 3.5). Here the curvature is set to be $\pi/2$ for corners on the outer boundary curve, $-\pi/2$ for corners on the inner boundary curve.

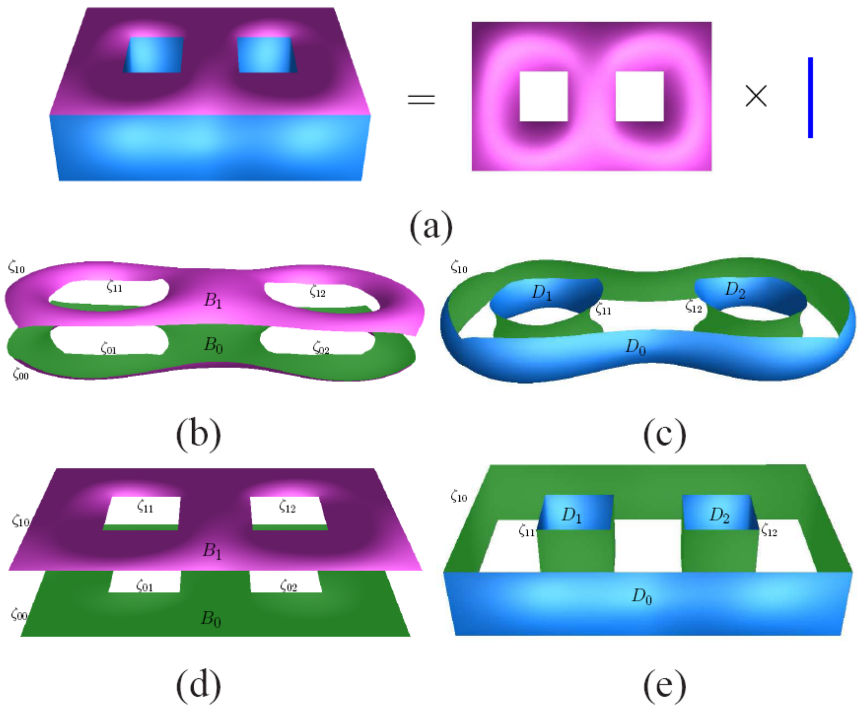


Figure 3.5: Another direct-product domain for the Eight model.

3.4.3 Computing The Harmonic Function

In section 3.4.2 the floor has been conformally parameterized, meaning that we have the $\mathbf{u}$ and $\mathbf{v}$ coordinates on one end of the volume. A natural following step is to extend them through the volume to the other end (the ceiling) smoothly. Meanwhile, we need a third coordinate along a direction orthogonal to the $\mathbf{u}$ $\mathbf{v}$ domain. Both goals are achieved via a vector field that is the gradient of a volumetric harmonic function. This section is devoted to the computation of such a harmonic function $\mathbf{w}$, which serves as the third coordinate in the parameter domain.

Based on the discussion in section 3.3.3, the volumetric harmonic function $\mathbf{w} : \mathbb{V} \to \mathbb{R}$ is the solution of the discrete Laplacian's equation with the following Dirichlet boundary condition 0.0 on the floor, and λ ($\lambda \in \mathbb{R}^+$) on the ceiling. Here λ specifies the height of the walls in the parameter domain. It can either be defined by the user, or be estimated automatically to match the width-height ratio in the original model. In the later case, one can compute a diameter of the floor d before the parameterization and that d' after the parameterization, plus the average height of the walls h before the parameterization. Then *lambda* can be simply set as $\lambda = hd/d'$.

Note that the function $\mathbf{w}$ computed from above can be extended to the whole volume piecewise-linearly (section 3.3.2), not necessary being restricted on vertices only. The extended function (also denoted as $\mathbf{w}$) provides the third parameter for any point in the tetrahedral mesh.

3.4.4 Tracing The Fibers

Besides serving as a third parameter function, $\mathbf{w}$ also provides an approach to bringing the ($\mathbf{u}$, $\mathbf{v}$) coordinates into the volume in a smooth manner. This is achieved by tracing the integral curves (i.e. fibers) of the gradient field $\nabla\mathbf{w}$ in the volume, either along or against the gradient direction.

To be specific, from a vertex $v_i \in \mathbb{V} - \mathbb{V}_{B_0}$ one can trace a fiber along $-\nabla\mathbf{w}$ until reaching the floor B_0. Suppose the fiber hits the floor at point p_i with surface parameter $(u,\ v)$; the same tuple $(u,\ v)$ is assigned to vertex v_i; plus the value of $w = \mathbf{w}(v_i)$, a complete set of coordinates $(u,\ v,\ w)$ in the parameter domain is determined for that vertex.

Or vice versa, one can start from a vertex on the floor $v_j \in \mathbb{V}_{B_0}$ and trace a fiber along $\nabla\mathbf{w}$ until hitting the ceiling. All the points on that fiber carry the same $(u,\ v)$ values as v_j does.

An important fact is that, due to the nature of the harmonic function and the way we set the boundary conditions, fibers should be disjoint pairwisely. That is, given two vertices with different $(u,\ v)$ coordinates, the fibers traced from there should not merge or intersect, provided that the discrete approximation is accurate enough. Secondly, a fiber will start from one base and end

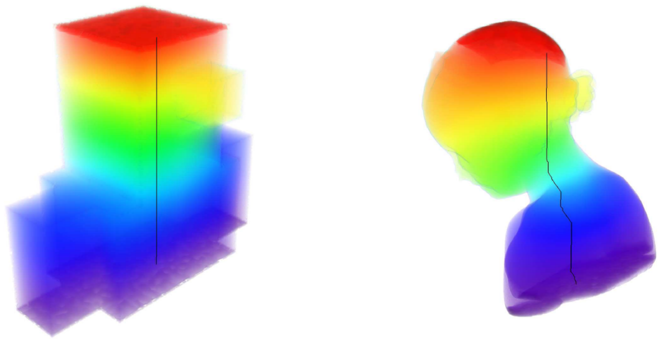

(a) (b)

Figure 3.6: Tracing a fiber.

at the other, no self-intersection will be present along it. Furthermore, a fiber starting from a vertex on a wall will stay on that wall, while one starting from somewhere off the walls will never touch any wall. The tracing procedure is slightly different for these two cases.

Tracing off the walls Starting from a point p_0 that is not on any wall, the fiber will penetrate a set of tetrahedra. Since the gradient field is constant within a tetrahedron, the fiber will consist of a set of consecutive straight line segments. Given the starting point p_i of the i'th segment ($i = 0, 1, ...$), how to find the ending point p_{i+1} depends on the position of p_i. Without loss of generality, let $\boldsymbol{r}(p_i, t_k)$ denote the ray starting from point p_i along the tracing direction assigned to tetrahedron t_k, which follows or against the gradient $\nabla \mathbf{w}$.

- If p_i is at a vertex v_{j_0}, check all the surrounding tetrahedra for one $t_k = (v_{j_0} v_{j_1} v_{j_2} v_{j_3})$ such that $\boldsymbol{r}(v_{j_0}, t_k)$ intersects the face $(v_{j_1} v_{j_2} v_{j_3})$ at a point p_{i+1}.

- If p_i is on an edge $e = (v_{j_0} v_{j_1})$, check all the tetrahedra sharing this edge, locate the one $t_k = (v_{j_0} v_{j_1} v_{j_2} v_{j_3})$ such that $\boldsymbol{r}(p_i, t_k)$ intersects either the face $(v_{j_0} v_{j_1} v_{j_2})$ or the face $(v_{j_1} v_{j_0} v_{j_3})$.

- If p_i is within a face $f = (v_{j_0} v_{j_1} v_{j_2})$, then do the same check for the tetrahedra on both sides of f, and find one t_k where $\boldsymbol{r}(p_i, t_k)$ hits one of the other three faces in t_k.

Tracing on a wall For a point p_0 that is on one of the walls, the tracing is restricted on that surface; the gradient field $\nabla \mathbf{w}$ should also be projected onto that surface. This way, the fiber starting from v_0 is a sequence of straight line segments crossing a set of triangle faces on that wall. Similarly, let p_i, p_{i+1} be the starting and ending points of the i'th segment; let $\boldsymbol{s}(p_i, f_k)$ be the ray along the tracing direction assigned to face f_k, starting from point p_i.

- If p_i is at a vertex v_{j_0}, check all the surrounding faces for one $f_k = (v_{j_0}v_{j_1}v_{j_2})$ such that $\boldsymbol{r}(p_i, f_k)$ intersects the edge $(v_{j_1}v_{j_2})$ at a point p_{i+1}.
- If p_i falls on an edge $e = (v_{j_0}v_{j_1})$, check the adjacent faces on both side of e, locate the one $f_k = (v_{j_0}v_{j_1}v_{j_2})$ such that $\boldsymbol{r}(p_i, f_k)$ intersects either the edge $(v_{j_0}v_{j_2})$ or the edge $(v_{j_0}v_{j_3})$.

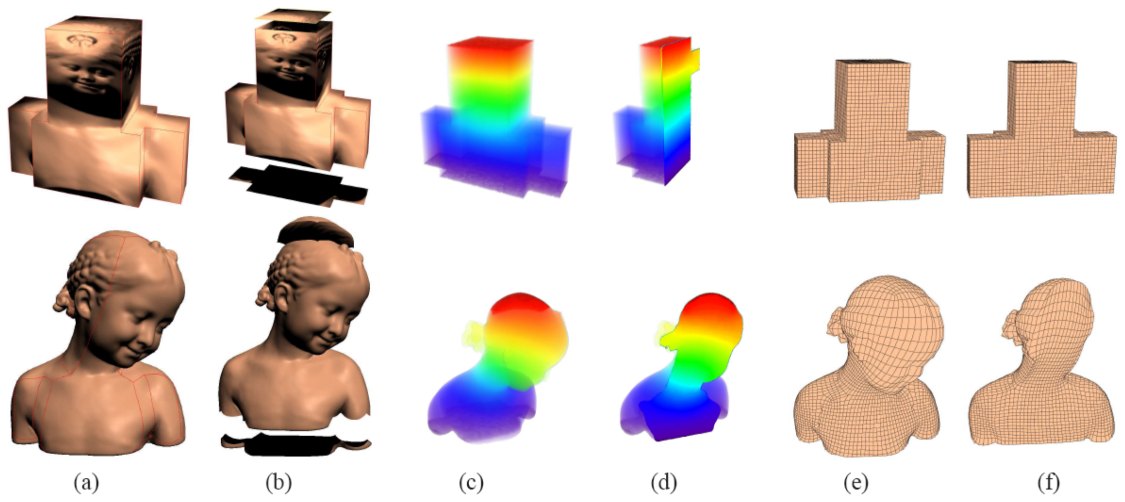


Figure 3.7: A genus-0 Bimba model (lower row) and its polycube parameter domain (upper row). (a) and (b) show the boundary manipulation; (c) and (d) show the color mapped harmonic fields; (e) and (f) show the hexahedral remeshing of the original volume based on the polycube parameterization.

3.5 Applications and Extensions

The basic algorithm of direct product parameterization could be adapted to further applications. One of such applications is to build the *hexahedral mesh* for a volume given as a tetrahedral mesh. It requires mapping the given volumetric mesh $\mathbb{M}$ to a polycube domain $\mathbb{M}^P$ using the direct product algorithm.

In order to build a hexahedral mesh, the polycube domain has several advantages over other parametric domains. First, polycube has a regular structure that can be used to construct all-hexahedral meshes. Second, there are well-developed techniques to construct polycube maps that can be used to parameterize the boundary surface of a given volume. Third, due to the geometric simplicity, it is usually easier to partition the boundary surface of a polycube domain than to partition that of the input model.

The adapted algorithm for hexahedral mesh construction will work the following way.

Firstly, map the boundary surface $\partial\mathbb{M}$ to a surface polycube domain $\partial\mathbb{M}^P$ using algorithms from [65]. Then we can tessellate the volume bounded by $\partial\mathbb{M}^P$ with cubes, resulting in a regular hexahedral mesh $\mathbb{M}^P$, which will serve as the parameter domain for the original volume $\mathbb{M}$. The mapping between $\mathbb{M}^P$ and $\mathbb{M}$ will be computed in consequential steps below.

Secondly, partition the polycube surface $\partial\mathbb{M}^P$ into bases $\{B_0^P, B_1^P\}$ and walls $\{D_0^P, \cdots, D_n^P\}$. Note that here the floor B_0^P and the ceiling B_1^P are in general not flat, but compositions of multiple flat pieces, which also applies to the walls. Furthermore, the partition of the polycube surface $\partial\mathbb{M}^P$ also induces a partition of the original boundary surface $\partial\mathbb{M}$: $\{B_0, B_1, D_0, \cdots, D_n\}$.

Thirdly, compute the harmonic field and trace the fibers on $\mathbb{M}^P$ and $\mathbb{M}$ separately, which will assign ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$) coordinates to each vertex of $\mathbb{M}^P$ and each vertex of $\mathbb{M}$. Then a map ϕ can be easily built between the original volume $\mathbb{M}$ and the polycube domain $\mathbb{M}^P$ by corresponding points with the same ($\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$) coordinates.

Finally, re-tessellate the polycube domain $\mathbb{M}^P$ with hexahedra. This is an easy task due to the geometric nature of polycube domain. Then with the map ϕ computed in the previous step, this tessellation can be directly transferred to the original volume $\mathbb{M}$, which results in a hexahedral mesh for the given volume.

The extended algorithm for constructing polycube domains and hexahedral meshes can be applied on models with different topology. Figure 3.9, 3.8 and 3.7 show the results on volumes with a boundary surface of genes four, two

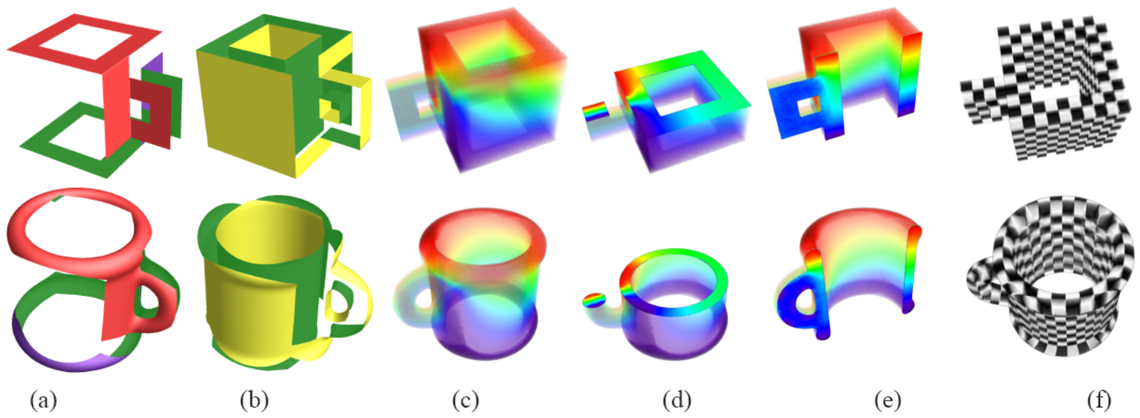


Figure 3.8: A genus-2 cup model (lower row) and its polycube parameter domain (upper row). (a) and (b) show the bases and walls in the boundary partition respectively; (c) to (e) show the color mapped harmonic field in the volume; (f) shows the conformality and distortion of the mapping using texture map.

and zero respectively. In particular, the algorithm can apply to degenerated handle bodies, i.e. topological balls (see figure 3.7).

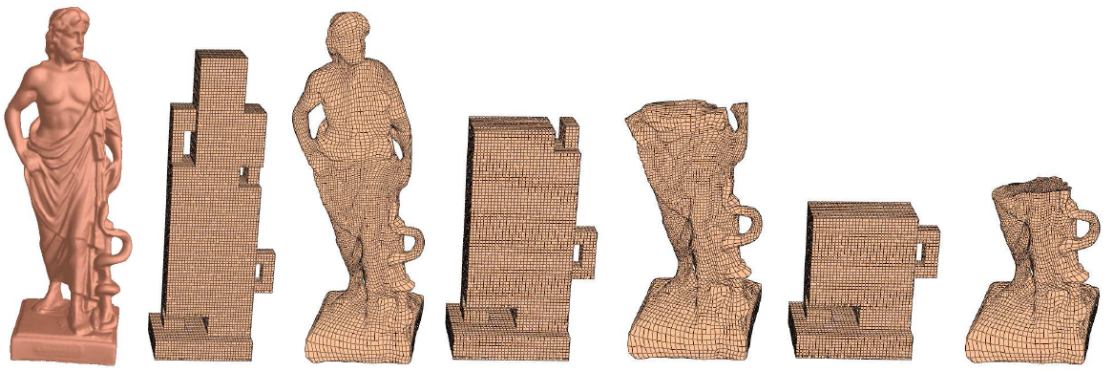


Figure 3.9: A genus-4 Greek model parameterized with a polycube domain and remeshed with hexahedra, which uses the direct-product algorithm.

Chapter 4

Discrete Metric Design via Discrete Curvature Flow

4.1 Motivations and Contributions

In geometric modeling, many problems can be reduced to designing *metrics* with *constant curvature*. For example, a common task in surface parameterization ([22], [55]) aims to find a flat 2D domain for the given surface, which is equivalent to computing a metric with zero curvature in its interior. According to the famous *Uniformization Theorem*, every compact 2-manifold (i.e. surface) admits a constant curvature metric, and the constant is one of +1, 0 and -1, which induces the *spherical*, *Euclidean* and *hyperbolic* geometry on surfaces respectively.

3-manifolds, according to Thurston's geometrization conjecture, can be decomposed into pieces that admit canonical geometries. There are eight canonical geometries, and three of them have constant sectional curvatures +1, 0 or -1. How to compute constant curvature metrics for such 3-manifolds are therefore becoming a natural question to ask.

An effective way to design such metrics is by *curvature flows*; such flows deform a given Riemannian metric according to its curvature. In certain circumstances, a curvature flow can lead to a constant curvature metric. In the mathematical society, there is some work along this line. For example, in a seminal paper [29] Hamilton introduced *the Ricci flow* for Riemannian manifolds of any dimension. And this flow has been applied in the proof of the Poincaré conjecture ([47], [49], [48]).

In particular, some works are dedicated to surfaces, such as [61], [62], [30], [54], [13], [12], [11], [5] and [4], either in smooth settings or in discrete settings. Some theoretical results in there have also been successfully introduced into the computer graphics and geometric modeling society in [35], [33], [3] and etc.

For 3-manifolds, there is a series of work ([24], [23], [40] and etc) on hyperbolic 3-manifolds with complete geodesic boundaries. They provided some important theoretical results on the geometry of such manifolds in a combinatorial setting. In engineering fields, however, to the best knowledge of the authors there has been no work to compute or apply the constant curvature metric for 3-manifolds yet. Although there are some works on volumetric parameterizations (such as [67], [39] and [42]), they do not impose any requirement on the curvature.

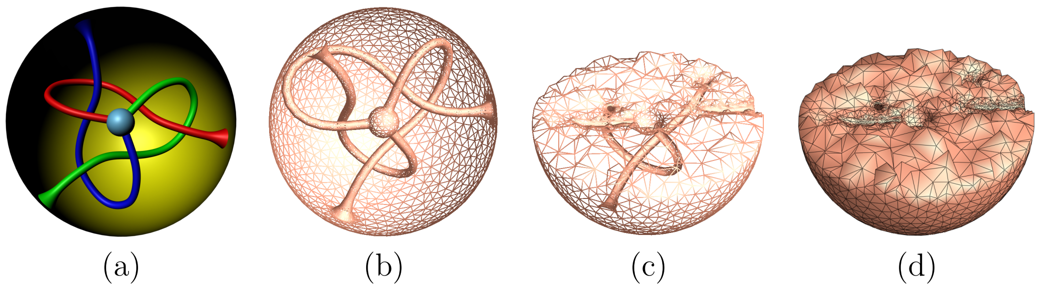


Figure 4.1: An example of hyperbolic 3-manifold, Thurston's knotted Y shape, constructed from a solid ball with three entangled tunnels removed. (a) and (b) show the boundary surface, (c) and (d) show the internal tessellation with tetrahedra.

Motivated by the above situation in engineering fields, this work serves as an initial attempt to extend the application of constant curvature metrics from surfaces to 3-manifolds. In particular we focus on a special type of 3-manifolds that have boundaries and satisfy the following two conditions:

[1] The boundary is a closed surface of genus g ($g > 1$).

[2] For any loop on the boundary surface, if it cannot shrink to a point on the boundary, then it cannot shrink to a point through the interior of the volume.

Note that such 3-manifolds admit hyperbolic metrics, which have constant sectional curvature of -1, and could therefore be called *hyperbolic 3-manifolds*. One of such examples is Thurston's *knotted Y-shape* (figure 4.1) constructed from a solid ball with three entangled tunnels removed.

We believe that such techniques will benefit various applications such as shape classification and etc. On the other hand, it also provides to geometers and topologists a numerical approach that would potentially facilitate their study on 3-manifolds at least in a discrete setting.

Since we are working in a discrete setting, any given 3-manifold should be represented by a discrete approximation, such as a volumetric mesh. Our curvature flow algorithm requires a special mesh consisting of a set of *truncated tetrahedra* (figure 4.2). This representation has been widely used in the study of 3-manifold since its introduction by Thurston [61]. It can model 3-manifolds with boundary surfaces of arbitrary genus, including but not limited to those mentioned above that have high genus boundary surfaces.

However such a representation hardly found its place in engineering applications, where one of the most commonly used representations is the tetrahedral mesh. The later can be built from a triangular mesh of the boundary surface easily by tools like tetgen [57], and there are various ways to remesh it, such as [36],[2].

To bridge this gap, we proposed an algorithm to build a truncated tetrahedral mesh from a tetrahedral mesh that is much easier to acquire. It can be used to model 3-manifolds with various boundary surfaces, including those with high genus boundary surfaces that are not easy to handle using other tools such as *SnapPea* or *Regina*. The process preserves topological invariants, and is totally automatic. With such a tool, one is able to construct simple representations for complicated 3-manifolds (with boundaries) easily, which is otherwise a much more difficult task that usually requires strong intuitions or long time training.

Once a truncated tetrahedral mesh is built, we use a numerical method to compute the constant curvature metric based on a discrete curvature flow proposed by Luo ([40]). This flow deforms edge lengths of the truncated tetrahedra by simulating a heat diffusion process; the convergence of the flow and the uniqueness of the solution is guaranteed by [40] for the hyperbolic 3-manifolds that we are handling. We did experiments on over 150 3-manifolds whose canonical metrics are known, and the results from our algorithm conformed very well to those obtained by algebraic approaches, with a numerical error no more than $1e^{-6}$.

Furthermore, in order to visualize the constant curvature metric, a third algorithm is provided to realize it in the hyperbolic space $\mathbb{H}^3$. Using this algorithm the metric can be visualized either by a single period (i.e. the fundamental domain) or by multiple periods (i.e. a finite portion of the universal covering).

As a brief summary, this work has the following contributions:

[1] We proposed an combinatorial algorithm to convert a tetrahedral mesh

to a truncated tetrahedral mesh for general 3-manifolds with various kinds of boundaries, plus a common data structure that guarantees a smooth conversion. This algorithm can not only generates the inputs for the discrete curvature flow algorithm, but also provides a computational tool for researchers to facilitate other tasks that may require such a construction.

[2] We proposed a numerical algorithm to compute constant curvature metrics for a special kind of hyperbolic 3-manifolds based on a discrete curvature flow. Experiments are carried out to validate the correctness and effectiveness of the algorithm.

[3] We proposed an algorithm to realize the input 3-manifold in the hyperbolic space, which paved a way to visualize the constant curvature metric that is just computed.

The rest of this chapter is organized as follows. We first present the algorithm of constructing truncated tetrahedral meshes in section 4.2, together with a discussion about the underlying data structure. Section 4.3 is dedicated to the algorithm on discrete curvature flow, including some necessary backgrounds and experimental results. The visualization method is shown in section 4.4.

4.2 Truncated Tetrahedral Meshes

This section is dedicated to an algorithm that generates a truncated tetrahedral mesh for a 3-manifold (with boundaries) that is given as a tetrahedral mesh. Before getting there, we first present the data structure used to represent a (truncated) tetrahedral mesh in computer.

4.2.1 Data Structures

In general, a tetrahedral mesh consists of a set of tetrahedra (figure 4.2(a)) and the connectivity (adjacency) among them. The constitution of a truncated tetrahedral mesh is similar, except for that it uses truncated tetrahedra (figure

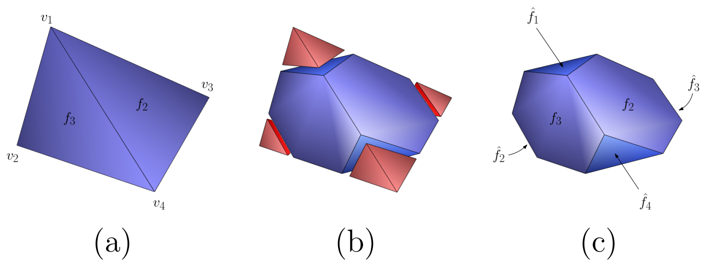


Figure 4.2: A truncated tetrahedron (c) resulting from a tetrahedron (a) with four corners trimmed off (b). The truncated tetrahedron has 4 hexagonal faces f_1 - f_4 and 4 triangular faces $\hat{f}_1$ - $\hat{f}_4$; in a truncated tetrahedral mesh, all f_j should be in the interior while all $\hat{f}_j$ should be on the boundary.

4.2(c)) as building blocks. Actually with careful design, a representation for the former can be converted to the later smoothly with just minor modifications. In the follows we show such a consistent design for both tetrahedral meshes (section 4.2.1) and truncated tetrahedral meshes (section 4.2.1).

Tetrahedral Meshes To represent a tetrahedral mesh, one of the data structures that are commonly used in geometric modeling is *vertex-oriented*. Namely, it starts from a given set of vertices $\{v_1, v_2, \cdots\}$ plus their coordinates (*embedding*) in the Euclidean space $\mathbb{R}^3$, and then defines each tetrahedron by four vertices *different* to one another. In such a data structure, two tetrahedra can only be glued along at most one face. Consequently it cannot model certain complicated inner structures that would occur in our algorithm.

In this work, we need a data structure that is more flexible. It should, for example, allow two tetrahedra to be identified at more than one face, and therefore allow multiple vertices (edges) of a single tetrahedron to be identified. For this purpose we designed a tetrahedral mesh that is *tetrahedron-oriented*, where we start with a set of tetrahedra and then define everything else from them. This construction is more general than the vertex-oriented one; actually the later is only a special case of the former, meaning that any vertex-oriented mesh can be converted into a tetrahedron-oriented one smoothly, but not vice versa.

Let *nt* be the number of tetrahedra in the mesh; the set of tetrahedra is

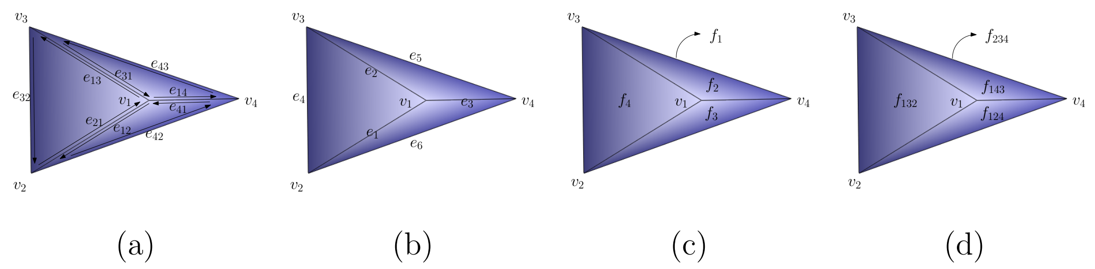


Figure 4.3: Local entities for a tetrahedron: t-vertices v_j (a - d), t-edges e_j (a), directed t-edges $e_{j_1 j_2}$ (b), t-faces f_j (c), oriented t-faces $f_{j_1 j_2 j_3}$ (d).

denoted as $T = \{t_1, \cdots, t_{nt}\}$. Comparatively, faces, edges and vertices can be defined both *locally* and *globally*, depending on how we treat each tetrahedron in the mesh.

The *local* definition arises when we treat the tetrahedra in the mesh as separate entities. For an arbitrary tetrahedron t_i, its vertices can be denoted as $TV_i = \{v_j^i | 1 \leqslant j \leqslant 4\}$, where each v_j^i is called a *t-vertex*.

Similarly, we can define the set of *t-edges* as $TE_i = \{e_j^i | 1 \leqslant j \leqslant 6\}$. A *directed t-edge* e_j^i can be denoted as $e_{j_1 j_2}^i$ ($1 \leqslant j_1 \leqslant 4$, $1 \leqslant j_2 \leqslant 4$, $j_1 \neq j_2$) if it starts from t-vertex $v_{j_1}^i$ and ends at $v_{j_2}^i$. The set of *t-faces* can be denoted as $TF_i = \{f_j^i | 1 \leqslant j \leqslant 4\}$. An *oriented t-face* f_j^i can be denoted as $e_{j_1 j_2 j_3}^i$ ($1 \leqslant j_1 \leqslant 4$, $1 \leqslant j_2 \leqslant 4$, $1 \leqslant j_3 \leqslant 4$, $j_1 \neq j_2$, $j_1 \neq j_3$, $j_2 \neq j_3$) if it covers t-vertices $v_{j_1}^i$, $v_{j_2}^i$ and $v_{j_3}^i$ in a counter-clockwise manner.

In a *global* sense, on the other hand, all the tetrahedra are glued together by identifying certain t-faces. To be specific, if two tetrahedra t_{i_1} and t_{i_2} are glued along t-faces $f_{j_1}^{i_1}$ and $f_{j_2}^{i_2}$, it will give rise to a *global face* (or *face* for short) f_k consisting of these two t-faces. And all the global faces constitute the face set $F = \{f_1, \cdots, f_{nf}\}$, where nf is the total number of global faces in the mesh. The gluing among t-faces is be represented by an *identifying function* (or *gluing function*)

$$\varphi_f : \bigcup_{i=1}^{nt} TF_i \to F$$

Given a face $f_k \in F$, its *pre-images* are defined as $\varphi_f^{-1}(f_k) = \{f_j^i | \varphi_f(f_j^i) =$

$f_k\}$. Note that the number of pre-images $|\varphi_f^{-1}(f_k)|$ for a face should be no less than 1 and no more than 2.

Furthermore, the gluing function on t-faces induces a gluing function on t-edges:

$$\varphi_e : \bigcup_{i=1}^{nt} TE_i \to E$$

where $E = \{e_1, \cdots, e_{ne}\}$ is the set of *global edges* (or *edges* for short). Again, given an edge $e_k \in E$, its pre-images are denoted as $\varphi_e^{-1}(e_k)$, and the number of pre-images satisfies $|\varphi_e^{-1}(e_k)| \geqslant 1$.

Similarly, we can define the set of *global vertices* (or *vertices* for short) $V = \{v_1, \cdots, v_{nv}\}$ and the vertex gluing function

$$\varphi_v : \bigcup_{i=1}^{nt} TV_i \to V$$

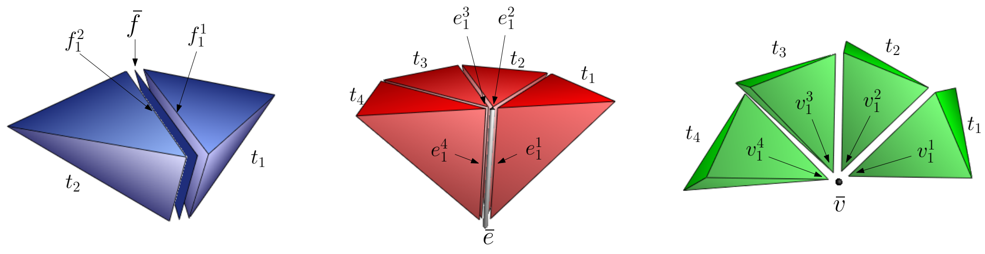
.

under which the number of pre-images for an arbitrary vertex $v_k \in V$ satisfies $|\varphi_v^{-1}(v_k)| \geqslant 1$.

Note that the local entities (t-vertices, t-edges and t-faces) can be looked as part of a tetrahedron. For example, we say a tetrahedron t_i *contains* a t-vertex v_j^i if $v_j^i \in TV_i$; the containment of t-edges or t-faces is defined similarly. Meanwhile, these local entities are also part of the corresponding global entities. For example, a vertex v_k *contains* a t-vertex v_j^i if $\varphi_v(v_j^i) = v_k$.

The global entities, however, are not belonging to any tetrahedron nor local entity. In this sense, we will say a tetrahedron t_i is *incident* to a vertex v_k if t_i contains a t-vertex t_j^i whose image is v_k under map φ_v. The incident relation between tetrahedra and edges or faces are defined similarly. Two vertices are *adjacent* if each of them contains a t-vertex, such that these two t-vertices are in the same tetrahedron and are therefore connected by some t-edge.

Truncated Tetrahedral Meshes Given a tetrahedron, if one trims four smaller tetrahedra from its four vertex corners (figure 4.2(b)), the remaining object is called a *truncated tetrahedron* (figure 4.2(c)), spanning 12 vertices,



(a) $\varphi_f(f_j^i) = \bar{f}$ (b) $\varphi_e(e_j^i) = \bar{e}$ (c) $\varphi_v(v_j^i) = \bar{v}$

Figure 4.4: Global entities (face $\bar{f}$, edge $\bar{e}$, vertex $\bar{v}$) by identifying local entities (t-faces f_j^i, t-edges e_j^i, t-vertices v_j^i) from a couple of tetrahedra $\{t_i | i = 1, 2, \cdots\}$ using gluing functions (φ_f, φ_e, φ_v). Note that each face $\bar{f}$ has at most 2 pre-images, while an edge $\bar{e}$ or a vertex $\bar{v}$ could have more than 2 pre-images.

bounded by 4 triangular faces and 4 hexagonal faces. Multiple truncated tetrahedra can be glued together along hexagonal faces to form a truncated tetrahedral mesh.

The data structure we used to represent a truncated tetrahedral mesh is similar to that for a tetrahedral mesh, with slight changes. It contains a set of truncated tetrahedra $T = \{t_1, \cdots, t_{nt}\}$. Each t_i has 4 *hexagonal t-faces* $TF_i = \{f_j^i | 1 \leqslant j \leqslant 4\}$, 6 *t-edges* $TE_i = \{e_j^i | 1 \leqslant j \leqslant 6\}$; instead of having 4 t-vertices, it has 4 *triangular t-faces* $\hat{TF}_i = \{\hat{f}_j^i | 1 \leqslant j \leqslant 4\}$. The gluing functions are defined as:

$$\varphi_f : \bigcup_{i=1}^{nt} TF_i \to F$$

$$\varphi_e : \bigcup_{i=1}^{nt} TE_i \to E$$

where F, E are the sets of *(global) hexagonal faces* and *(global) edges*. One thing different to tetrahedral meshes is that, the number of pre-images for an arbitrary face $f_k \in F$ should be exactly 2, and that for an arbitrary edge $e_k \in E$ should be no less than 2. This fact implies that every hexagonal t-face should be identified with some one else and is therefore in the interior of the volume, and the same to each t-edges. Another difference is that there is no gluing function defined for triangular t-faces, which means that such faces should not be identified with any one else and should therefore be on the boundary.

The data structure introduced above is purely *topological* in its own right, but allows *geometric* information to be assigned later; for example, in the discrete curvature flow (see section 4.3), certain geometric information will be assigned, such as dihedral angle, edge length, edge curvature and etc. However, the (truncated) tetrahedral mesh itself does not necessarily rely on geometry.

4.2.2 Algorithms

In this section we introduce the algorithm that takes a tetrahedral mesh as input and produces a truncated tetrahedral mesh of the given 3-manifold. Note that both meshes should be represented using the data structure discussed in above.

A basic operation in this conversion is called *vertex merge* (figure 4.6), which merges one vertex into another one that is adjacent to the first one; during the merge, some tetrahedra around these two vertices need to be removed or adjusted. To be specific, suppose we want to merge vertex v_2 into vertex v_1, or make a *v_2-to-v_1 merge*, four sets of tetrahedra will be affected (figure 4.5).

One set is T_{II}, consisting of all the tetrahedra that are incident to both v_1 and v_2. All such tetrahedra will be removed from the mesh after this merge.

Another set is T_I, consisting of all the tetrahedra $\{t_i\}$ such that t_i is incident to vertex v_1 but not incident to v_2, and has a t-face f^i_j that was identified with some t-face from T_{II}. After the merge, the t-face f^i_j will be identified by another t-face from T_{III} (see below), but its image under φ_f remains unchanged.

A third set is T_{III}, consisting of all the tetrahedra $\{t_i\}$ such that t_i is incident to vertex v_2 but not incident to v_1, and has a t-face f^i_j that was identified with some t-face from T_{II}. After the merge, the t-face f^i_j will be identified to another t-face from T_I, and its image under φ_f will change accordingly. Consequently, the gluing image of some t-edges and t-vertices in t_i will also be

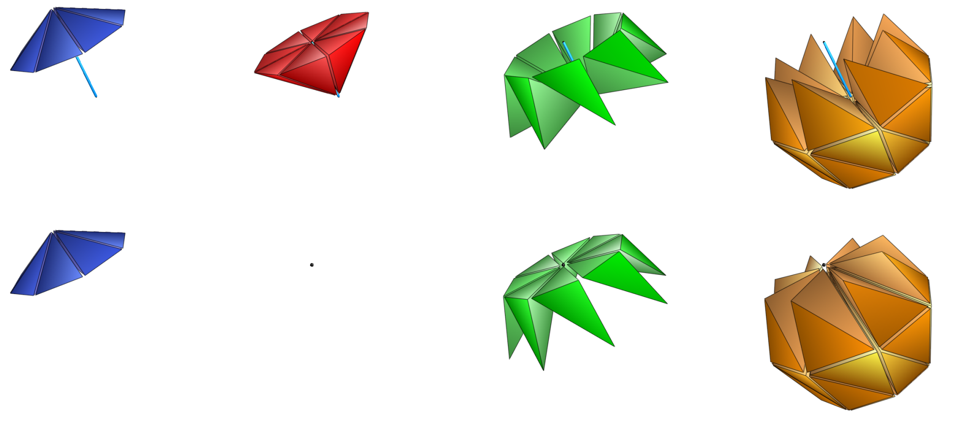


Figure 4.5: Four types of tetrahedra affected by a vertex merge, shown in the top row (before the shrinking) and the bottom row (after the shrinking). The type-II tetrahedra will be deleted, the type-III and type-IV tetrahedra will be deformed to meet the type-I tetrahedra by faces (and therefore by edges and vertices) and by vertices only respectively.

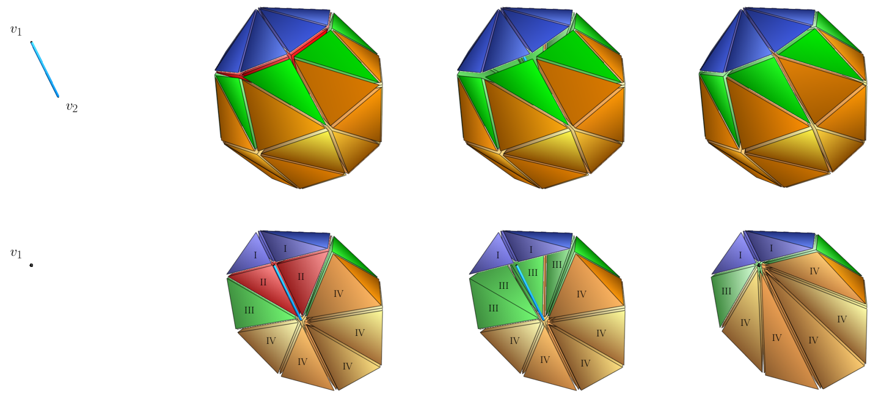
changed.

The last set is T_{IV}, consisting of all the tetrahedra $\{t_i\}$ such that t_i is incident to vertex v_2 but not incident to v_1, and does not have any t-face that was identified with any t-face from T_{II}. After the merge, one of its t-vertices that was belonging to v_2 will now belong to v_1, and the images of the surrounding t-edges will also be changed accordingly.

The conversion algorithm could be divided into several steps.

[1] Using vertex merge, remove all the vertices that are in the interior of the mesh, together with all the type-II tetrahedra getting involved in each merge. After this step, the mesh is reduced to a smaller set of tetrahedra, with all the vertices on the boundary.

[2] For each face f_i on the mesh boundary, create a tetrahedron t_i that identifies one of its t-faces f^i_j with f_i and leaves only one t-vertex v^i_j out side of the original mesh. All the newly created tetrahedra $\{t_i\}$ are glued together nicely according to the connectivity between their underlying faces $\{f_i\}$. Also, for all the tetrahedra that sit on the same component of



(a) (b) (c) (d)

Figure 4.6: Vertex merge (v_2-to-v_1). The process is shown in (b) to (d) from both global views (top row) and cutting views (bottom row). (b) shows all the tetrahedra getting involved, among which the type-II tetrahedra are deleted in (c) and the type-III and type-IV tetrahedra are then deformed in (d) to meet the type-I tetrahedra.

the boundary surface, identify their only t-vertices that stick out of the original volume. This will give rise to a set of new vertices, the number of them equals to the number of boundary components of the original volume. Now the mesh is converted to a closed tetrahedral mesh.

[3] Use vertex merge again to remove all the vertices in the closed mesh except for those that are newly created in the previous step. Now all the vertices from the original tetrahedral mesh have been removed.

[4] For each tetrahedra remaining in the mesh, turn it into a truncated one by trimming off a small tetrahedron from each of its t-vertices. Consequently all the hexagonal faces are kept inside the resulting mesh, while all the triangular faces are exposed on the boundary. The original tetrahedral mesh is now converted to a truncated tetrahedral mesh.

The above algorithm is topology-preserving; in another word, it does not change the fundamental group of the input 3-manifold. Actually this is a very important property that is required for the computation of constant curvature metric on 3-manifolds. Due to the so called *Mostow rigidity* ([44]), the geometry of a finite volume hyperbolic 3-manifold is totally determined by its fundamental group. Different 3-manifolds have equivalent constant curvature metrics if they have the same topology. As a consequence, different triangulations and different ways to transform the mesh will not affect the computational results of the discrete curvature flow, so long as the fundamental group of the 3-manifold is preserved during the transformation.

4.2.3 Experiments

The algorithm constructing truncated tetrahedral meshes has been tested and validated by experiments on various 3-manifolds with different kinds of boundary surfaces, including those with a boundary of sphere (genus 0), 1-hole torus (genus 1) and multi-hole torus (with genus greater than 1).

One of the examples is the knotted Y-shape (figure 4.1) that is given as a tetrahedral mesh with 16,374 vertices and 83,622 tetrahedra. It can be converted to a truncated tetrahedral mesh with only two truncated tetrahedra $\{t_1, t_2\}$ (figure 4.7a) glued together by the pattern shown in figure 4.7(b). To change the notations, for each t_i, let $\{A_i, B_i, C_i, D_i\}$ be its four hexagon faces, let $\{a_i, b_i, c_i, d_i\}$ be the truncated vertices. The gluing pattern is given as follows, where the arrow $\rightarrow$ means to identify the identity on the left and that on the right:

$$\begin{array}{ll}
A_1 \rightarrow B_2 & \{b_1 \rightarrow c_2, d_1 \rightarrow a_2, c_1 \rightarrow d_2\} \\
B_1 \rightarrow A_2 & \{c_1 \rightarrow b_2, d_1 \rightarrow c_2, a_1 \rightarrow d_2\} \\
C_1 \rightarrow C_2 & \{a_1 \rightarrow a_2, d_1 \rightarrow b_2, b_1 \rightarrow d_2\} \\
D_1 \rightarrow D_2 & \{a_1 \rightarrow a_2, b_1 \rightarrow c_2, c_1 \rightarrow b_2\}
\end{array}$$

4.3 Computing Hyperbolic Metrics

In this section we introduce the algorithm to compute hyperbolic metrics that have constant curvature -1 for the hyperbolic 3-manifolds defined in section 4.1. The input to the algorithm is a truncated tetrahedral mesh computed from section 4.2, but assigned with a hyperbolic metric (section 4.3.1). On such a mesh with geometric information, one can define *discrete curvatures* (section 4.3.2) and then simulate a discrete curvature flow on the mesh (section 4.3.3).

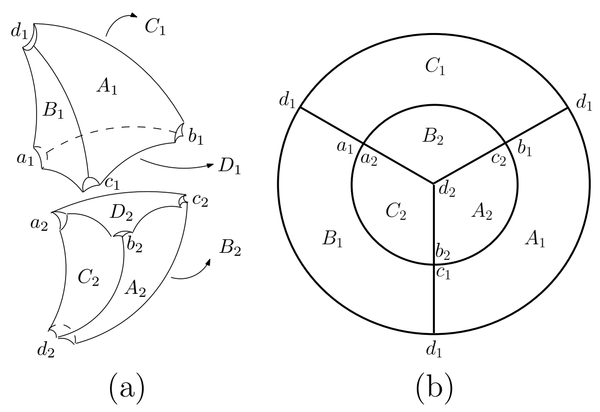


Figure 4.7: The truncated tetrahedral mesh and gluing pattern for the knotted Y shape

The algorithm has been validated by experiments (section 4.3.4).

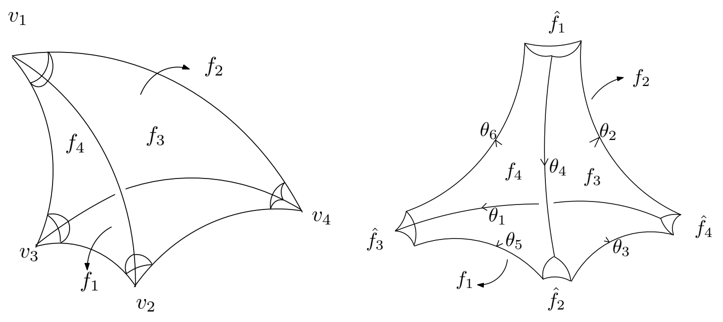


Figure 4.8: Hyperbolic tetrahedron and truncated hyperbolic tetrahedron.

4.3.1 Hyperbolic Truncated Tetrahedra

A (truncated) tetrahedron assigned with a hyperbolic metric is called a *hyperbolic (truncated) tetrahedron.* In figure 4.8, for example, the left frame shows a hyperbolic tetrahedron, where each face f_i is a hyperbolic triangle, each edge e_{ij} is a hyperbolic line segment. The right frame shows a hyperbolic truncated tetrahedron, where each triangular face $\hat{f}_i$ is a hyperbolic triangle that is perpendicular to edges e_{ij}, e_{ik}, e_{il}, each hexagonal face f_i is a right-angled hyperbolic hexagon.

Hyperbolic triangles and right angled hyperbolic hexagons satisfy special cosine laws (figure 4.9). Given a hyperbolic triangle with edge length $\{y_i, y_j, y_k\}$ and inner angles $\{\theta_i, \theta_j, \theta_k\}$, where θ_i is against y_i, we have the following hyperbolic cosine laws

$$\cosh y_i = \frac{\cos\theta_i + \cos\theta_j \cos\theta_k}{\sin\theta_j \sin\theta_k} \tag{4.1}$$

$$\cos\theta_i = \frac{-\cosh y_i + \cosh y_j \cosh y_k}{\sinh y_j \sinh y_k} \tag{4.2}$$

Given a hyperbolic hexagon as shown in figure 4.9 with right inner angles and edge length $\{x_i, x_j, x_k\} \cup \{y_i, y_j, y_k\}$, where x_i is against y_i, the following cosine law holds:

$$\cosh y_i = \frac{\cosh x_i + \cosh x_j \cosh x_k}{\sinh x_j \sinh x_k} \tag{4.3}$$

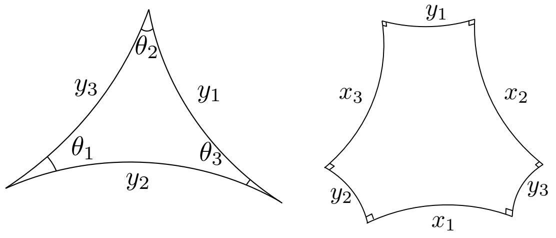


Figure 4.9: Hyperbolic Cosine laws for triangle and right-angled hexagon.

The geometry of the truncated tetrahedron is determined by the dihedral angles $\{\theta_1, \theta_2, \cdots, \theta_6\}$ as shown in Figure 4.8. For example, the hyperbolic triangle at v_2 has inner angles $\theta_3, \theta_4, \theta_5$, its edge lengths can be determined using formula 4.1. For face f_4, the edge length e_{12}, e_{23}, e_{31} can be determined by the hyperbolic triangles at v_1, v_2, v_3 using the right-angled hyperbolic hexagon cosine law 4.3.

On the other hand, the geometry of a hyperbolic truncated tetrahedron is determined by the length of edges $e_{12}, e_{13}, e_{14}, e_{23}, e_{34}, e_{42}$. Due to the fact that each face f_i is a right angled hexagon, the above six edge lengths will determine the edge length of each hyperbolic triangular face $\hat{f}_j$, and therefore determines its three inner angles, which equal to the corresponding dihedral angles of the hyperbolic truncated tetrahedron.

4.3.2 The Discrete Curvature

For surface meshes, the discrete (vertex) curvature is represented as the angle deficit. For an interior vertex, the curvature is 2π minus the surrounding corner angles

$$K(v_i) = 2\pi - \sum_{jk} \alpha_i^{jk}.$$

for a boundary vertex, the curvature is π minus the surrounding corner angles.

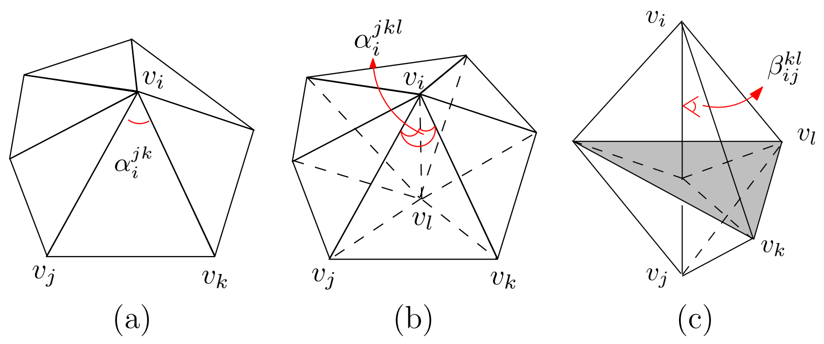


Figure 4.10: Discrete curvatures: vertex curvature (a) for 2-manifolds, vertex curvature (b) and edge curvature (c) for 3-manifolds.

In a tetrahedral mesh for a 3-manifold, one can also define *vertex curvature*; as shown in figure 4.10, each tetrahedron $[v_i, v_j, v_k, v_l]$ has four solid angles at their vertices, $\{\alpha_i^{jkl}, \alpha_j^{kli}, \alpha_k^{lij}, \alpha_l^{ijk}\}$; for an interior vertex, the vertex curvature is defined as 4π minus the surrounding solid angles,

$$K(v_i) = 4\pi - \sum_{jkl} \alpha_i^{jkl}.$$

if the vertex is on the boundary, the vertex curvature is defined as

$$K(v_i) = 2\pi - \sum_{jkl} \alpha_i^{jkl}.$$

Besides vertex curvature, tetrahedral meshes also present another type of discrete curvature, *edge curvature.* Suppose $[v_i, v_j, v_k, v_l]$ is a tetrahedron, the dihedral angle on edge e_{ij} is β_{ij}^{kl}. If edge e_{ij} is an interior edge (i.e. e_{ij} is not on the boundary), the edge curvature is defined as

$$K(e_{ij}) = 2\pi - \sum_{kl} \beta_{ij}^{kl}.$$

If e_{ij} is on the boundary, the edge curvature is defined as

$$K(e_{ij}) = \pi - \sum_{kl} \beta_{ij}^{kl}.$$

The two types of discrete curvatures are actually closely related. Edge curvature determines vertex curvature, $\sum_j K(e_{ij}) = K(v_i)$, and is therefore more essential than the later.

4.3.3 Discrete Curvature Flow

Given a hyperbolic tetrahedron with edge lengths l_{ij} and dihedral angles θ_{ij}, the volume of the tetrahedron V is a function of the dihedral angles $V = V(\theta_{12}, \theta_{13}, \theta_{14}, \theta_{23}, \theta_{24}, \theta_{34})$, and the Schlaefli formula can be expressed as

$$\frac{\partial V}{\partial \theta_{ij}} = \frac{-l_{ij}}{2},$$

namely, the differential 1-form dV is $\frac{-1}{2} \sum_{ij} l_{ij} d\theta_{ij}$. It is proved in [40] that the volume of a hyperbolic truncated tetrahedron is a strictly concave function of the dihedral angles.

Given a 3-manifold represented as truncated tetrahedral mesh, we can define the discrete metric function as $x : E \to \mathbb{R}^+$, where E is the set of edges that are along the hexagonal faces but not along the triangular faces. The discrete curvature function can be defined as $K : E \to R$. Given an edge $e_{ij} \in E$, its edge length and edge curvature can be represented as x_{ij} and K_{ij} respectively. From the discussion in above, the edge curvature can be determined by the dihedral angles, which in turn is a function of edge length. Therefore, the set $\{K_{ij}\}$ can be calculated from $\{x_{ij}\}$. The discrete curvature flow is then defined as

$$\frac{dx_{ij}}{dt} = K_{ij}, \tag{4.4}$$

From this differential equation, the deformation of the metric is driven by the edge curvature, and the whole process is like a heat diffusion. Any numerical method for solving the discrete heat diffusion problem can be applied to solve the curvature flow equation. And in the current implementation, we simply use a gradient descent method, with initial edge length set to $x_{ij} = 1$.

During the flow, the total edge curvature $\sum_{ij} K_{ij}^2$ is strictly decreasing. When the flow reaches the equilibrium state, both the edge curvature and the vertex curvature vanish (see proof in [40]). The boundary surface will become a hyperbolic geodesic, while all the curvature (which is negative) is uniformly distributed within each truncated hyperbolic tetrahedron. Due to the fact the total curvature is negative, the resulting metric is a *hyperbolic* one.

4.3.4 Experimental Results

The algorithm has been tested on 151 hyperbolic 3-manifolds that can be constructed by three truncated tetrahedra glued with different patterns. The constant curvature metrics (in terms of dihedral angles) for these manifolds have been reported in [6]; we compared our results with theirs, finding that the difference in between was never above $1e^{-6}$. The running time is less than a second for each 3-manifold.

We also tested the algorithm on the knotted Y-shape, which is represented as a mesh of two truncated tetrahedra. The resulting dihedral angles are $\{0.523599, 0.523599, 0.523599, 0.523599, 0.523599, 0.523599\}$ for both truncated tetrahedra.

4.4 Visualization by Hyperbolic Embedding

Once the canonical hyperbolic metric is computed, one is ready to realize it in the hyperbolic space $\mathbb{H}^3$. There are two ways to realize the metric. The first one is a single period representation (figure 4.13), which is a union of multiple truncated hyperbolic tetrahedra. The second is a multiple period representation (figure 4.14), which consists of multiple copies of the single period representation. We will first introduce the hyperbolic space models we used here (section 4.4.1), then the embedding algorithm for one truncated tetrahedron (section 4.4.2), a single period representation (section 4.4.3) and a multiple period representation (section 4.4.4) respectively.

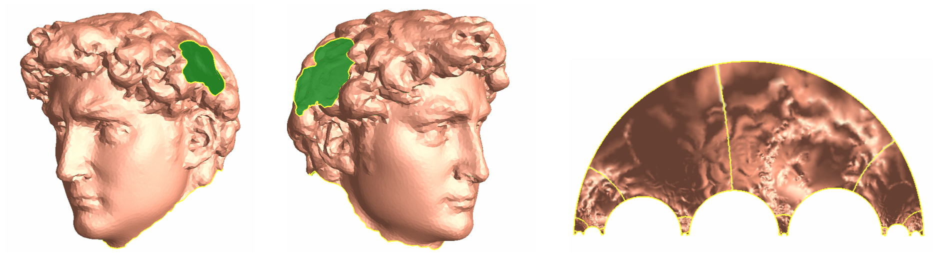

(a)Left view (b) Right view (c) Periodic embedding

Figure 4.11: The upper half plane model of $\mathbb{H}^2$, with a hyperbolic surface embedded in it.

4.4.1 Hyperbolic Space Model

During the hyperbolic embedding, we will work in both 2D and 3D hyperbolic spaces. For 2D hyperbolic space $\mathbb{H}^2$, we use the upper half plane model $\mathbb{H}^2 = \{(x,y) \in \mathbb{R}^2 | y > 0\}$ (figure 4.11), with Riemannian metric

$$ds^2 = \frac{dx^2 + dy^2}{y^2}$$

.

In $\mathbb{H}^2$, hyperbolic lines are circular arcs perpendicular to and centered on the x-axis or straight lines orthogonal to and ending at x-axis. The rigid motion is given by the so-called *Möbius transformation*

$$\frac{az+b}{cz+d}, ac - bd = 1, a, b, c, d \in \mathbb{R},$$

where $z = x + iy$ is the complex number to be transformed.

For 3D hyperbolic space $\mathbb{H}^3$, we use the upper half space model $\mathbb{H}^3 =$

$\{(x, y, z) \in \mathbb{R}^3 | z > 0\}$ (figure 4.12), with Riemannian metric

$$ds^2 = \frac{dx^2 + dy^2 + dz^2}{z^2}.$$

In $\mathbb{H}^3$, the hyperbolic planes are hemispheres whose equators are on the xy-plane or vertical planes perpendicular to and ending at the xy-plane. The xy-plane represents all the infinity points in $\mathbb{H}^3$. The rigid motion in H^3 is determined by its restriction on the xy-plane, which is a Möbius transformation on the complex plane in the form of

$$\frac{az + b}{cz + d}, ac - bd = 1, a, b, c, d \in \mathbb{C}.$$

4.4.2 Embedding One Truncated Tetrahedron

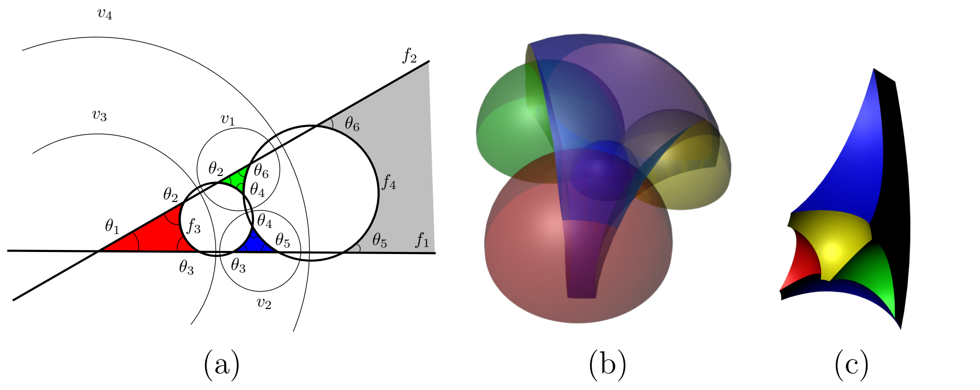


Figure 4.12: Embedding one hyperbolic truncated tetrahedron (c) in $\mathbb{H}^3$ by taking intersections among hemispheres (b) based on their intersection circles with the infinity plane $z = 0$ (a).

Given the edge length of a hyperbolic truncated tetrahedron, its dihedral angles are uniquely determined so that the truncated tetrahedron can be embedded in $\mathbb{H}^3$ uniquely up to rigid motion. To be specific, its embedding is determined by the position of its four right-angled hexagonal faces $\{f_1, f_2, f_3, f_4\}$ and that of its four triangular faces $\{\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{f}_4\}$. Each of these faces is a hyperbolic plane (i.e. semi-sphere shown in figure 4.12(b)), separating $\mathbb{H}^3$ into two half spaces. By choosing the right half space for each face and taking the intersection of all these chosen half spaces, one will get an embedding of the hyperbolic truncated tetrahedron (figure 4.12(c)).

To compute the position of each hyperbolic plane, let's consider its intersection with the infinity plane $z = 0$, which is a Euclidean circle (figure 4.12(a)). Here we reuse the symbol f_i and $\hat{f}_j$ to represent the intersection circle by the hyperbolic plane f_i and $\hat{f}_j$ respectively. As shown in figure 4.12(a), all the circles can be computed explicitly, such that circle f_i and circle f_j intersect at the given dihedral angle θ_k, while circle $\hat{f}_i$ is orthogonal to circles $\{f_j, f_k, f_l\}$. In order to remove the ambiguity caused by rigid motion, we fix circle f_1 to be line $y = 0$, f_2 to be line $y = \tan\theta_1 x$, and normalize the circle f_3 to have radius 1.

Once the intersection circles are computed, we can directly construct hemispheres (i.e. hyperbolic planes) whose equators are those circles. By choosing the right half space for each hemisphere and using CSG operations to compute the intersection of these half spaces, we get an embedding of a single hyperbolic truncated tetrahedron as shown in figure 4.12(c).

4.4.3 Embedding a Single Period

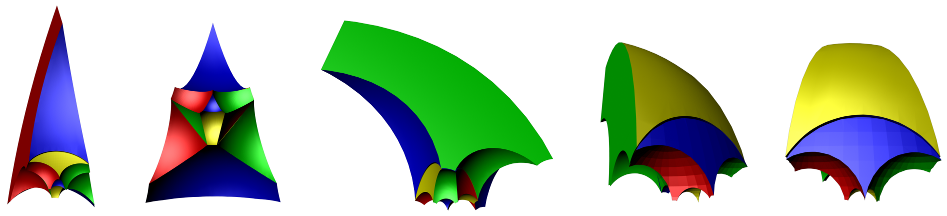


Figure 4.13: Embedding the fundamental domain of the knotted Y-shape; i.e. a single period realization of the hyperbolic metric in $\mathbb{H}^3$. The embedding is shown from 5 different views.

A single period representation of the given 3-manifold with canonical hyperbolic metric is a union of all the constituting hyperbolic truncated tetrahedra. It is constructed as follows. First, embed one truncated tetrahedron t_0 as explained above. Then pick another not-embedded truncated tetrahedron t_1, which is neighboring to t_0 by identifying hexagonal t-face $f_1 \in t_1$ with $f_0 \in T_0$. Compute a Möbius transformation in $\mathbb{H}^3$ that rigidly moves t_1 to a position such that $f_1 \in t_1$ can be perfectly glued to $f_0 \in t_0$. Now we get a partially embedded volume. Repeat the process of picking, moving and gluing for other truncated tetrahedra until the whole volume is embedded.

The above algorithm is essentially a bread-first-search (BFS) in the given 3-manifold; it results in a tree spanning all the truncated tetrahedra in the mesh. Due to the nature of the constant curvature metric, such gluing (or spanning) operation can be carried out seamlessly, until finally all the truncated tetrahedra are glued together nicely into a simply connected domain, which is a topological ball. Such a single period representation is usually called the *fundamental domain* for the original volume (see [28]). Figure 4.13 visualizes the embedded fundamental domain for the knotted Y-shape.

4.4.4 Embedding Multiple Periods

A multiple period representation of the hyperbolic metric is a union of multiple copies of the fundamental domain, which is essentially a *universal covering space* (UCS) of the original 3-manifolds (see [28]). UCS is also a simply connected topological ball. Similar to the embedding of a fundamental domain, the UCS can also be constructed through a sequence of gluing operations; the difference is, the primitive construction blocks are copies of the embedded fundamental domain rather than the truncated tetrahedra.

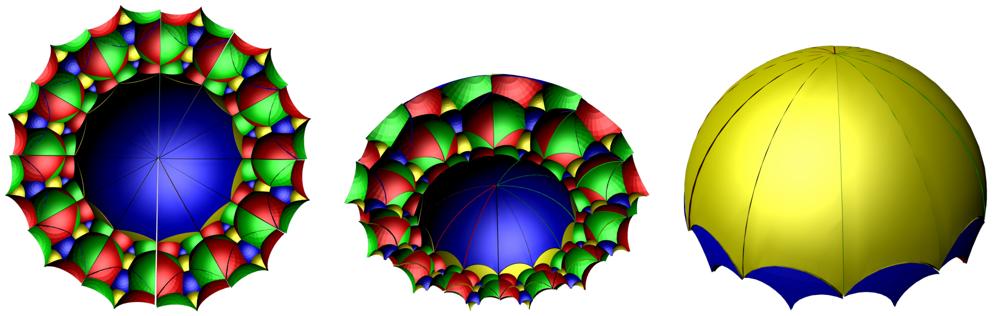


Figure 4.14: Embedding a finite portion of the UCS of the knotted Y-shape, i.e. a multiple period realization of the hyperbolic metric in $\mathbb{H}^3$. The embedding is shown from 3 different views.

Recall the algorithm for embedding a fundamental domain, any two hyperbolic truncated tetrahedra are glued to each other via a pair of hexagonal t-faces. Besides those t-faces, there will be some other hexagonal t-faces left open, glued to nothing else. This is natural because otherwise the fundamental domain will not be simply connected. All the open hexagonal t-faces are grouped into several connected components, each component constitutes a *gluable face* for the whole fundamental domain. All the gluable faces can be coupled nicely as follows. For each gluable face, there exists another gluable face uniquely in the same fundamental domain such that they are able to glue to each other nicely. Actually, the fundamental domain can be viewed as a result of cutting the original mesh open along these gluable faces that can be coupled by pairs. And two copies of the fundamental domain can be glued together along a pair of gluable faces.

Different to the construction of a single fundamental domain, the gluing operation among multiple copies of the fundamental domain can be repeated infinitely, having infinitely many copies involved in the UCS. In practice, we only construct a finite portion of the UCS, as the one visualized in figure 4.14.

Chapter 5

Discrete Unit Tangent Bundles for Disks

Starting from this chapter, we spend a couple of chapters on the construction of discrete unit tangent bundles for various surfaces.

This chapter is devoted to simply connected 2-dimensional disks D^2. We first give intuitions behind our construction (section 5.1), where we convert this 3D problem to an equivalent 2D problem. Then for the 2D problem, we propose a basic algorithm for a restricted boundary condition (section 5.2), and generalize it to more flexible boundary conditions (section 5.3 and 5.4). We also analyze the convergence of each algorithms and existence of solutions under each boundary condition.

5.1 Intuitions

The unit tangent bundle over a topological disk D^2 is a direct product of the disk D^2 and a circle S^1, $UT(D^2) = D^2 \times S^1$. In another word, it is a trivial S^1 bundle that can be represented as a solid torus with only one handle (figure 5.1). Given a topological disk represented as a triangular mesh D, our goal is to generate a tetrahedral mesh T that represents the bundle $UT(D)$, and the tetrahedral mesh should reflect the direct product nature of the bundle. In particular, we require that:

Definition 5.1.1. *Regularity of Discrete $UT(D^2)$*

[1] A set of copies of the base (i.e. the given disk mesh) should be explicitly represented, disjoint to one another along the fiber;

[2] A set of fibers (i.e. S^1), one per vertex of the disk mesh, should be explicitly represented, disjoint to one another;

[3] All the fibers should have the same resolution in terms of number of sampling points along the fiber.

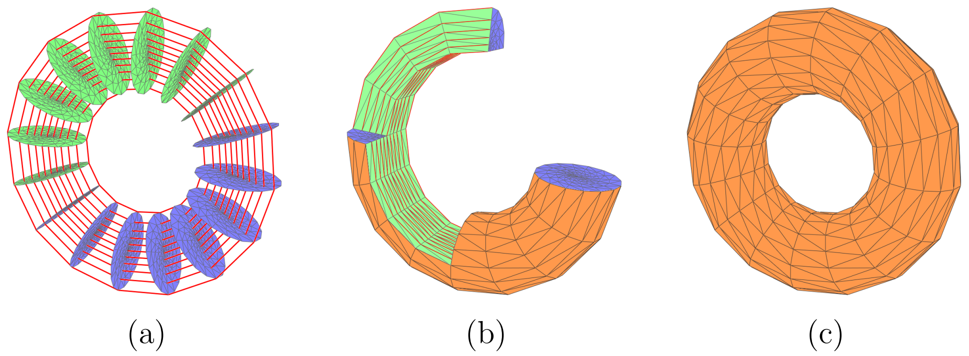


Figure 5.1: The unit tangent bundle for a topological disk is a solid torus (c), which is a direct product of a disk and a circle (a) and can be represented as a tetrahedral mesh with regular structure (b).

5.1.1 Construction by Sweeping

Towards this goal, we sweep the given triangular mesh D along a circle S^1 that is orthogonal to D, make stops at certain points on the way (figure 5.1), and finally get back to meet its original position. The resulting volume represents the unit tangent bundle T.

Let us take a closer look at the volume between two consecutive stops. Here every vertex $v \in D$ sweeps a line segment I. Every triangle $f \in D$ sweeps a volume that is a direct product of f and a straight line segment I. Such a volume is called a *prism* (figure 5.2), which is bounded by two triangles from top and bottom, and three quadrangular walls from the side. Therefore the volume between two consecutive stops is a union of prisms, we call this volume a *segment* of T. In fact, each segment is a direct product $D \times I$, and the union of all the segments is the target bundle $T = D \times S^1$.

The above sweeping construction gives a mesh of T consisting of prisms that are grouped into segments. It is easy to observe that

[1] At every stop there is a base copy D_i ($1 \leqslant i \leqslant N$), and all the base copies are disjoint from one another.

[2] Every vertex $v \in D$ sweeps a line segment between two stops, all the line segments joins into a piecewise linear circle that represents a fiber over v. And these fibers are disjoint to one another from vertex to vertex.

[3] Suppose we make N stops in the sweeping. Then every fiber is a union of N straight line segments; that is, the discretizing resolution is uniformly equal to N.

It means that the resulting prismatic mesh from the above sweeping construction already presents the regularity stated in definition 5.1.1. Recall that our goal is a tetrahedral mesh for the unit tangent bundle. So the rest of the task is to cut each prism into tetrahedra in a consistent way. What is more, it turns out that in certain cases there are certain extra constraints on the boundary that we need to consider. As one will see below, finding such a cutting is equivalent to a simpler 2D problem, to which we can always find a solution under most circumstances.

5.1.2 Conversion to A 2D Problem

Note that each prism can be cut into three tetrahedra by planes passing through certain corners of the prism (figure 5.2b-d). To distinguish different options of cutting, we assign +1 or -1 to each (directed) edge B_iB_{i+1} of the bottom triangle, if the wall above it is sliced along diagonal B_iT_{i+1} or $B_{i+1}T_i$ respectively. Then every cutting option can be encoded as a 3-tuple of ± 1 on the bottom triangle.

In order to tessellate the solid torus with tetrahedra, all the prisms have to be cut in a consistent way. In our construction this can be done individually in each segment between any two consecutive base copies, which boils down, as discussed above, to assigning a 3-tuple of ± 1 to every triangle in each base copy.

Several issues need to be considered here:

- Not every combination of three ± 1 represents a valid cutting option. Figure 5.2(e) gives a counterexample, where the 3-tuple carries a unique sign. A valid cutting pattern should have both +1 and -1 in each triangle.
- For any pair of prisms adjacent to each other by a quadrangular wall, their cutting lines on that wall should coincide. Therefore the cutting pattern in the bottom triangles should have opposite signs across the common edge.

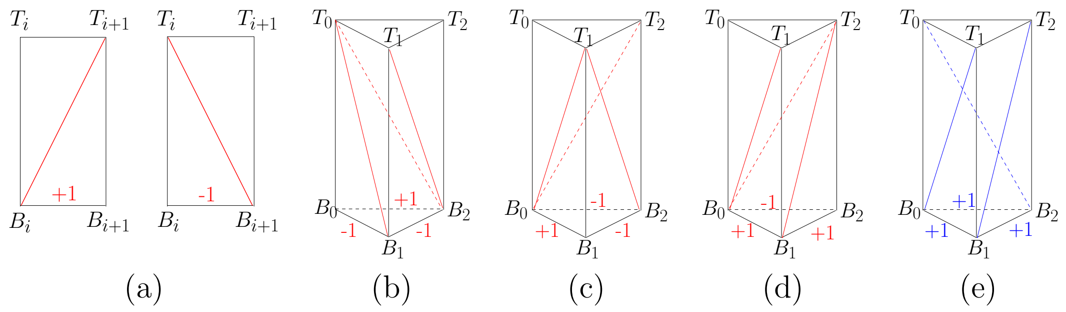


Figure 5.2: There are multiple patterns to cut a prism into three tetrahedra, each of them can be encoded by assigning +1 or -1 to each edge of its bottom triangle $B_0B_1B_2$ (a). A valid cutting pattern consists of both signs (b, c and d), while a bad one consists of only one sign (e).

With these considerations, the tetrahedral tessellation of a local bundle is converted to the following 2D problem:

Definition 5.1.2 (The Cutting Pattern Problem). *Given a triangular mesh of a topological disk, assign each triangle with a 3-tuple of* ± 1*, such that:*

- *Every triangle should have both* $+1$ *and* -1 *in its 3-tuple;*
- *Every edge shared by two adjacent triangles should be assigned with opposite values in these two triangles;*
- *The values on boundary edges could either be prescribed as inputs (fixed boundary) or be determined in the algorithm (free boundary).*

5.1.3 Boundary Conditions

The cutting pattern problem in definition 5.1.2 is defined over simply connected topological disks represented as triangular meshes. The solution to this problem relies on the boundary condition, which involves two issues: whether or not the cutting pattern is prescribed on boundary edges, and if yes, how the cutting pattern is initialized on boundary edges. Considering various situations, we can classify the boundary conditions into three categories.

Definition 5.1.3 (Boundary Conditions for Cutting Pattern Problem)**.** *In the cutting pattern problem (definition 5.1.2), the boundary condition for an input mesh D can be classified into the following categories:*

[1] Restricted boundary: Every boundary edge of D has a prescribed cutting pattern. In particular,

- *For any triangle with only one edge exposed on the boundary of D, the prescribed cutting pattern for this boundary edge can take an arbitrary value of ± 1;*
- *For any triangle with at least two edges exposed on the boundary of D, two of such boundary edges have prescribed cutting patterns of opposite values (i.e. $+1$ and -1).*

[2] General boundary: Every boundary edge of D has a prescribed cutting pattern that can take an arbitrary value of ± 1;

[3] Free boundary: None of the boundary edges has a prescribed cutting pattern.

The first and second conditions are both fixed boundary conditions, where the cutting pattern is prescribed on every boundary edge and should keep fixed in the algorithm. On the other hand the last condition does not prescribe any value on boundary edges, thus the algorithm has the freedom to determine the boundary cutting patterns.

Under different types of boundary conditions, a solution to the cutting pattern problem may or may not exist, and if existing, it may require different treatments. In the rest of this chapter, we first present a basic algorithm for restricted boundary conditions in section 5.2; then generalize it to general boundary conditions in section 5.3 and free boundary conditions in section 5.4.

5.2 An Algorithm for Restricted Boundary Conditions

In this section we present an algorithm to solve the cutting pattern problem under restricted boundary conditions (see definition 5.1.3). We first define some concepts that are necessary for later discussions, then introduce the pipeline of our algorithm and the details for specific steps.

5.2.1 Some Definitions

First of all, let us define several concepts for a given component C (or a sub-mesh in general).

Given a triangular mesh D consisting of a set of triangles $F = \{f_1, f_2, \cdots, f_n\}$, a *sub-mesh* of D consists of a subset of triangles $\{f_{i_1}, f_{i_2}, \cdots, f_{i_m}\}$, where $f_{i_k} \in F$ for $1 \leqslant k \leqslant m$ and $m \leqslant n$.

The following concepts are defined within a given sub-mesh C.

For a triangle $f \in C$, its *edge-valence* (or *valence* in short) is the number of triangles adjacent to f across some edges. It should be an non-negative integer that is at most 3.

A triangle f is call a *dangling triangle* if its valence is 1. A dangling triangle has exactly two edges exposed on the boundary of C.

A triangle f is call a *dangling triangle* if its valence is 0. In a singular triangle, all three edges are on the boundary of C.

For an edge e shared by two triangles $f_1, f_2 \in C$, we say it is of (i, j)-*type* if f_1 and f_2 have valence i and j respectively $(1 \leqslant i \leqslant j \leqslant 3)$.

An edge e is called a *separating edge* if it is shared by two triangles and the end vertices of e are both on the boundary of C (figure 5.3a).

A edge e is called a *breaking edge* if it is a separating edge and at least one of its adjacent triangles has valence 3 (figure 5.3b and 5.3c).

5.2.2 Algorithm Pipeline

The goal of the algorithm is to assign a value $+1$ or -1 to every edge in every triangle. In our algorithm, for any edge e belonging to a triangle f, we assign it with an integer

$$a(e, f) \in \{+1, -1, 0\}$$

meaning that the edge e in triangle f is either assigned with a determined cutting pattern ($+1$ or -1) or is still open (0). An edge e is *completely solved* (or *solved* for short) if and only if it has a determined cutting pattern in every triangle that encloses e. An triangle f is *completely solved* (or *solved* for short) if and only if all three edges in f have determined cutting patterns. A sub-mesh C is *completely solved* (or *solved* for short) if and only if all the triangles in C are solved.

Upon input, all the boundary edges have prescribed assignments ± 1 while all the other edges are initialized with assignment 0. Then the input mesh D is processed in a divide-and-conquer manner. Namely, mesh D is partitioned into a set of sub-components that have special properties, and each sub-component is completely solved or partially solved separately. After this, the remaining sub-mesh consisting of all the unsolved triangles will be brought into another iteration of divide-and-conquer like the above. Repeat this process until the algorithm terminates.

Algorithm 5.2.1 (Cutting Pattern Algorithm - I)**.**

- *Input: A simply connected topological disk represented as a triangular mesh D with restricted boundary conditions (definition 5.1.3).*
- *Output: A set of valid cutting patterns for the whole mesh D.*
- *Procedures:*

  [1] Initialize all the inner edges with assignment 0, and all the triangles as unsolved.

[2] Repeat the following procedures on the sub-mesh consisting of all the unsolved triangles, until no unsolved triangle left.

(a) Partition the unsolved sub-mesh into a minimal set of sub-components of basic types, and solve the newly exposed boundary edges (section 5.2.4).

(b) Solve each sub-component completely or partially according to its type (section 5.2.5, 5.2.6 and *5.2.7).*

(c) Check the unsolved sub-mesh. If it is empty, exit; otherwise, go back to step 2a.

5.2.3 Algorithm Invariants

In order to guaranteed that the algorithm will terminate and generate the desired outputs, we need define several invariants that the algorithm should preserve.

The first set of invariants arise from the definition of the cutting pattern problem 5.1.2. First, every triangle should have both $+1$ and -1 in its 3-tuple of cutting patterns. Second, every pair of adjacent triangles sharing a common edge should have opposite cutting patterns on that edge.

Another important invariant is related to the boundary cutting patterns for certain sub-meshes. In the process of the algorithm, various kinds of sub-meshes could appear in different steps for various purposes; such as a sub-component resulting from step 2a, the unsolved sub-mesh in step 2c, or even the mesh D which is the input to the algorithm. Given a sub-mesh C, a *boundary assignment* $\mathbb{A}$ is a set of values ($+1$, -1 or 0) assigned to the boundary edges of C, one value per edge, where $+1$ or -1 represents a determined cutting pattern while 0 means to be determined. It turns out that the boundary assignments certain sub-meshes should satisfy certain requirement, which is captured in the following definition.

Definition 5.2.1 (Safe Boundary Assignments)**.** *For a sub-mesh C in a given triangular mesh D, a boundary assignment $\mathbb{A}$ is a safe boundary assignment* *for C if and only if for any dangling or singular triangle $f \in C$, at least two of the boundary edges are assigned with opposite non-zero values ($+1$ and -1).*

In our algorithm, we will enforce that the sub-components and the unsolved sub-mesh should have safe boundary assignments at certain points in the process.

In a nutshell, we propose the following invariants for the basic algorithm:

Definition 5.2.2 (Invariants of the Basic Algorithm)**.** *In the basic algorithm (5.2.1) for the cutting pattern problem under restricted boundary conditions, there are several invariants that need to be preserved:*

- *Completeness: For every completely solved triangle, the 3-tuple of cutting patterns should consist of both $+1$ and -1;*
- *Consistency: For every completely solved edge that is shared by two triangles in the original mesh, its cutting pattern in these two triangles should have opposite signs;*
- *Safeness: In any iteration within step 2, the unsolved sub-mesh (or mesh) at the beginning of step 2a should have a safe boundary assignment, and every sub-component at the end of step 2a should also have a safe boundary assignment.*

As verified in later discussions, all these invariants are preserved by the basic algorithm. That means this algorithm will not get stuck at any point in the process until finally the whole mesh is successfully solved.

5.2.4 Constructing Sub-components

This algorithm is a divide-and-conquer one. In step 2a, the unsolved sub-mesh (i.e. the union of all the unsolved triangles) will be partitioned into smaller pieces. This partition happens in two stages, where the second stage refines the partition from the first stage. Then the boundary edges that are newly exposed in the partition are solved.

The First Stage Partition

In the first stage of partition, all the unsolved triangles will be grouped into a set of *components*. A component is a sub-mesh of the original mesh D that is *edge-connected* (or *connected* in short), meaning that every triangle in the component is adjacent to at least one other triangle in the component across a common edge. We require the set of components here satisfies the following conditions:

- They make a *covering* of the original mesh D, meaning that the union of all the components is D;
- They are *triangle-disjoint* (or *disjoint* in short), meaning that there is no triangle shared by any pair of components;
- Each component is *maximal*, meaning that any union of two or more components is no longer edge-connected and thus no longer a component.

This stage of partition can be implemented as a Breadth-First-Search among all the unsolved triangles.

The Second Stage Partition

In the second stage, each component from the first stage is further partitioned into *sub-components*, which are edge-connected sub-meshes of C. And we require that every sub-component should have one of the following basic types.

Definition 5.2.3 (Types of Sub-components)**.**

- *Aggregated sub-component (figure 5.7): a sub-component that does not have any separating edge, and therefore every triangle has edge-valence at least 2 within this sub-component;*
- *Linear sub-component (figure 5.6): a sub-component consisting of a sequential strip of triangles, where two triangles at the ends have edge-valence 1 (i.e. dangling triangles) and all the other triangles have edge-valence 2 within this sub-component.*

- *Singular sub-component (figure 5.5): a sub-component consisting of a singular triangle.*

Recall the definition of separating edges and breaking edges from section 5.2.1. If a given component is a simply connected topological disk, which is always true in the process to solve the cutting pattern problem using our algorithms, slicing along any separating edge or breaking edge will both partition the component into two smaller pieces.

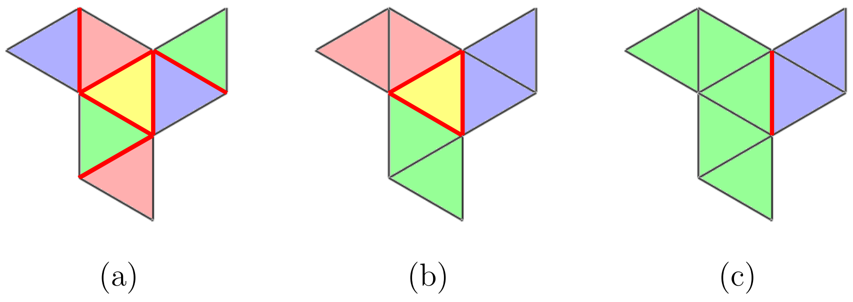


Figure 5.3: Separating edges (a), breaking edges (b) and a minimal set of breaking edges (c).

In fact, if we take the set of all the separating edges in a given component and slice along them (figure 5.3a), the component will be partitioned into a set of sub-components, where each sub-component is either aggregated or singular. It turns out that such a partition is too fine for our purpose.

A second choice for slicing is the set of all the breaking edges (figure 5.3b). This would give a set of sub-components that could cover all three basic types. However, this partition still produce unnecessary small pieces. In figure 5.3, for example, the singular sub-component in the middle can be actually merged into the linear sub-component on the right without changing the basic type of the later.

What we need in the algorithm is a *minimal set* of sub-components of basic types (figure 5.3c), meaning that any union of two or more sub-components in this set is not of any basic type. In another word, we require each sub-component to be *maximal* without changing its own basic type. Therefore, instead of using all the breaking edges in the given component, we only choose a subset of them so that the resulting sub-components are all maximal. Such a set of breaking edges is called a *minimal set of breaking edges*. Figure 5.4 shows several examples of cutting a component along a minimal set of breaking edges.

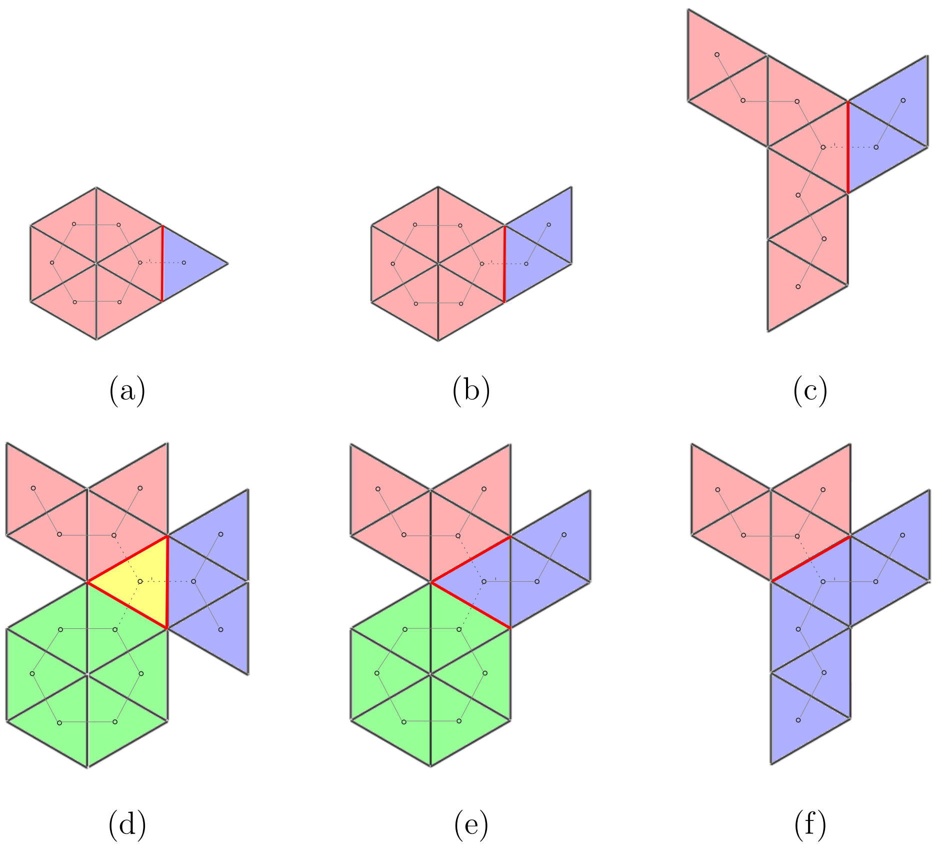


Figure 5.4: A component is partitioned into a minimal set of sub-components along a minimal set of breaking edges (marked in red). (a) to (f) shows six different components and their partitions.

Boundary Assignment Completion

In the second stage partition, every input component is divided into a set of sub-components. In a sub-component C, some edges on its boundary are actually also on the boundary of the input component, these edges are called *original boundary edges*. The other edges on the boundary of C are newly introduced by the second stage partition, they are called *new boundary edges*. In fact, every new boundary edge in sub-component C corresponds to a breaking edge in the input component.

After the partition, in any resulting sub-component C, every original boundary edge has a non-zero assignment (i.e. determined cutting pattern) inherited from the input sub-mesh, while every new boundary edge is initialized with assignment 0 and need to be solved here.

By definition, the second partition only happens along a minimal set of breaking edges. Based on the connectivity in the neighborhood, breaking edges in this set can be classified into several types.

Definition 5.2.4 (Classification of Breaking Edges). *Given a breaking edge e shared by two neighboring sub-component C_1 and C_2, based on the type of C_1 and C_2 and the position of e in C_1 and C_2, e can be classified into one of the following types:*

- *A2A: if both C_1 and C_2 are aggregated sub-components;*
- *A2Lm (or Lm2A): if C_1 is aggregated and C_2 is linear, and e belongs to a triangle in the middle of C_2;*
- *A2Le (or Le2A): if C_1 is aggregated and C_2 is linear, and e belongs to a triangle at one end of C_2;*
- *A2S (or S2A): if C_1 is aggregated and C_2 is singular;*
- *Lm2Lm: if both C_1 and C_2 are linear, and e belongs to a middle triangle in C_1 and another middle triangle in C_2;*
- *Lm2Le (or Le2Lm): if both C_1 and C_2 are linear, and e belongs to a middle triangle in C_1 and an end triangle in C_2;*
- *Lm2S (or S2Lm): if C_1 is linear and C_2 is singular, and e belongs to a middle triangle in C_1;*

Note that these are not all the combinations of two sub-components with basic types. However, other combinations, including Le2Le, Le2S and S2S, are impossible to sandwich a breaking edge, since they will violate the requirement that every sub-component should be maximal. For example, in a Le2Le combination, two linear sub-components would connect to each other at the ends; however, in such a situation these two sub-components could be merged into a larger linear sub-component. Therefore Le2Le is not a valid combination. The combination of Le2S and S2S can be excluded similarly.

Given a minimal set of breaking edges along which the given component has been partitioned, the breaking edges in this set are solved one after another using the following procedure.

Procedure 5.2.1 (Solving A Breaking Edge)**.** *Given a breaking edge e between two sub-components C_1 and C_2, do the following:*

- *If e is A2A, A2Lm or Lm2Lm: Suppose e is an A2A edge between aggregated sub-component C_1 and aggregated sub-component C_2. Assign e with $+1$ on C_1 side and -1 on C_2 side. According to definition 5.2.1, such an assignment is safe for both C_1 and C_2. In addition, the invariant of consistency (see definition 5.2.2) is preserved on e.*

  For A2Lm and Lm2Lm cases, it is processed and validated in the same way.

- *If e is A2Le or Lm2Le: Suppose e is an A2Le edge between aggregated sub-component C_1 and linear sub-component C_2. We will solve e on C_2 side first, and then on C_1 side.*

  On C_2 side, e is a boundary edge in an end triangle $f_2 \in C_2$. Let e' be the other boundary edge in triangle f_2; then within f_2, we can solve e based on whether or not e' is already solved.

  – *If e' is already solved in f_2, suppose its assignment is $a'_2 \in \{+1, -1\}$,* *then assign e with* $a_2 = -a'_2$ *in* f_2 *(thus preserving the invariant of completeness for* f_2*).*

  - *If* e' *is not solved in* f_2 *yet, then assign e with an arbitrary value* $a_2 \in \{+1, -1\}$ *(so that* e' *can be solved later with assignment* $a'_2 = -a_2$ *and thus preserving the invariant of completeness for* f_2*).*

  In either case, the two boundary edges e and e' *in end triangle* f_2 *would have opposite assignments, and it is therefore a safe boundary assignment for linear sub-component* C_2 *at the* f_2 *end.*

  Once e is solved on the C_2 *side with assignment* $a_2 \in \{+1, -1\}$*, we assign e on the* C_1 *side with* $a_1 = -a_2$ *(thus preserving the invariant of consistency on e). Note that* C_1 *is an aggregated sub-component and there is no actual constraint on its boundary assignment, therefore the above assignment on* C_1 *side is also safe.*

  For the Lm2Le case, it is processed and validated in the same way.

- *If e is A2S or Lm2S: Suppose e is an A2S edge between aggregated sub-component* C_1 *and singular sub-component* C_2*. We will solve e on* C_2 *side first, and then on* C_1 *side.*

  On C_2 *side, there is only one triangle* $f_2 \in C_2$*, and e is one of the three edges in* f_2*. Let* e' *and* e'' *denote the other two boundary edges in* f_2*; then within* f_2*, we can solve e based on whether or not* e' *and* e'' *are solved.*

  - *If at least one of these edges (say,* e'*) is already solved in* f_2*, suppose its assignment is* $a'_2 \in \{+1, -1\}$*, then assign e with* $a_2 = -a'_2$ *in* f_2 *(thus preserving the invariant of completeness for* f_2*).*

  - *If neither e' nor e'' is solved in f_2, then assign e with an arbitrary value $a_2 \in \{+1, -1\}$ (so that e' and e'' can be solved later with assignment ${a_2}' = -a_2$ and ${a_2}'' = -a_2$, and thus preserving the invariant of completeness for f_2).*

In either case, at least two boundary edges e and e' in the singular triangle f_2 would have opposite assignments, and it is therefore a safe boundary assignment for singular sub-component C_2.

Once e is solved on the C_2 side with assignment $a_2 \in \{+1, -1\}$, we assign e on the C_1 side with $a_1 = -a_2$ (thus preserving the invariant of consistency on e). Note that C_1 is an aggregated sub-component and there is no actual constraint on its boundary assignment, therefore the above assignment on C_1 side is also safe.

For the Lm2S case, it is processed and validated in the same way.

After this process, the algorithm produces a minimal set of sub-components of basic type. The boundary edges of each sub-component are completely solved and guaranteed to be a safe boundary assignment. Now we can go ahead to solve each sub-component based on its type.

5.2.5 Solving Singular Sub-components

For a singular sub-component (figure 5.5), there is only one triangle in it. Note that all the boundary edges are already solved in the previous step, and they constitute a safe boundary assignment. Therefore the whole sub-component is actually already solved.

Furthermore, it can be verified that the assignment for a singular sub-component C preserves all the related invariants in definition 5.2.2.

- For the only triangle $f \in C$, according to the discussion for the previous step in section 5.2.4, the assignment for three edges contains both $+1$ and -1, thus the invariant of completeness is preserved.

- For any edge $e \in C$:
  - If e is a boundary edge in the original mesh D, its cutting pattern is initialized upon input and is never changed;
  - Otherwise, e is an internal edge in the original mesh D and is exposed to the boundary of sub-component C during the previous step, where its assignment is already proved to preserve the invariant of consistency.

5.2.6 Solving Linear Sub-components

For a linear sub-component with a safe boundary assignment (figure 5.6), it can be solved completely.

In a linear sub-component, all the triangles are connected sequentially from one end to another end. Suppose a linear sub-component C has $n+1$ triangles ($n > 0$) ordered as

$$f_0,\ f_1,\ \cdots\ ,\ f_n$$

where f_0 and f_n are end triangles (i.e. dangling triangles) that each has two boundary edges, while the others are middle triangles that each has only one boundary edge. Let e_i be the internal edge shared by two consecutive triangles f_i and f_{i+1} ($1 \leqslant i \leqslant n$). Then the set of e_i are all the edges that need to be solved, and they can be ordered as a sequence

$$e_1,\ \cdots\ ,\ e_n$$

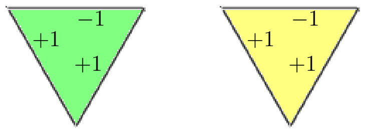


Figure 5.5: A singular sub-component is actually solved in the previous iteration already.

Let $a_i^{(i-1)}$ and $a_i^{(i)}$ be the cutting pattern assigned to e_i in f_{i-1} and f_i respectively. Then e_i can be solved as follows:

$$a_i^{(i-1)} = +1$$

$$a_i^{(i)} = -1$$

Since all the boundary edges are already solved in the previous step, now all the edges in this linear sub-component are solved and therefore all the triangles here are solved.

Furthermore, we can verify that such an assignment for this sub-component preserves all the related invariants in definition 5.2.2.

- For any triangle $f \in C$:
  - If f is an end triangle, say $f = f_0$ or $f = f_n$, it has two boundary edges and one internal edge. Since C is guaranteed to have a safe boundary assignment, the two boundary edges of f have opposite cutting patterns. Thus no matter what value is assigned to the internal edge, the invariant of completeness is already preserved for f.
  - If f is a middle triangle, say $f = f_i$ $(0 < i < n)$, it has one boundary edge e and two internal edges e_{i-1} and e_i. According to the above, the assignments on these two internal edges are $a_{i-1}^{(i)} = -1$ and $a_i^{(i)} = +1$, which are opposite to each other. Therefore no matter what value is assigned to the boundary edge e, the invariant of completeness is preserved for f.
- For any edge $e \in C$:
  - If e is an internal edge in C, say $e = e_i$ $(0 \leqslant i < n)$, it is assigned with opposite cutting patterns in the adjacent triangles, $a_i^{(i)} = -a_i^{(i-1)}$, thus preserving the invariant of consistency.
  - If e is a boundary edge in the original mesh D, its cutting pattern is initialized upon input.

– Otherwise, e is an internal edge in the original mesh D and is exposed to the boundary of sub-component C during the previous step, where its assignment is already proved to preserve the invariant of consistency.

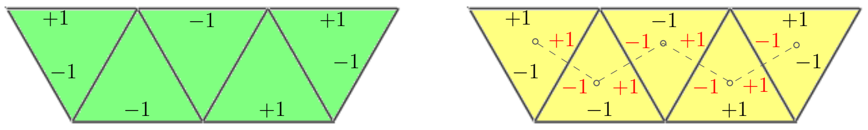


Figure 5.6: A linear sub-component can be completed solved in one iteration.

5.2.7 Solving Aggregated Sub-components

For an aggregated sub-component with a safe boundary assignment (figure 5.7), we will solve the most outside layer of triangles and leaving the rest part to be solved in later iterations.

Given an aggregated sub-component C, consider its dual graph C^*. Any triangular face $f_i \in C$ corresponds to a vertex $v_i^* \in C^*$, edge $e_j \in C$ to edge $e_j^* \in C^*$, vertex $v_k \in C$ to face $f_k^* \in C^*$. Note that C is a simply connected disk and every face $f_i \in C$ has at most one boundary edge. Accordingly, C^* is also a simply connected disk, its boundary is a loop connecting n vertices with n edges sequentially:

$$v_1^* - e_1^* - v_2^* - e_2^* - \cdots - v_n^* - e_n^* - v_1^*$$

where e_i^* is an edge connecting consecutive vertices v_i^* and v_{i+1}^*. This boundary loop in C^* corresponds to a looping sequence of n triangular faces in C that are connected via n edges:

$$f_1 - e_1 - f_2 - e_2 - \cdots - f_n - e_n - f_1$$

where e_i is the common edge between adjacent faces f_i and f_{i+1}. We call this looping sequence of triangles the *frontier* of C, denoted as C^F, and call the rest part the *interior* of C, denoted as C^I. The frontier C^F consists of the most outside layer of triangles in C, and the interior C^I consists of the rest.

For any face in the frontier, it either has only one edge (and therefore two vertices) exposed on the boundary, or has only one vertex (and no edge) exposed on the boundary.

In this step, we will solve all the triangles in frontier C^F and the newly exposed edges on the boundary of the interior C^I. We first solve the later and then the former.

For the interior C^I, some edges that were previously internal edges in C will be exposed to the boundary of C^I after the frontier C^F is solved and removed. Actually these edges are the border between C^F and C^I, shared by both sides. We call them boundary edges of C^I, and solve them one by one in the following way. For any edge e on the boundary of C^I, it belongs to a triangle in C^I; Denoted this triangle as f, and let e' and e'' be the other two edges in f. We solve e in f as follows.

- If e is the only edge of f that is on the boundary of C^I, then assign e with an arbitrary value $a \in \{+1, -1\}$.
- If besides e there is another edge (say e') on the boundary of C^I:
  - If e' is not solved yet, then assign e with $a = +1$;
  - If e' is already solved in f with assignment $a' \in \{+1, -1\}$, then assign e with $a = -a'$.
- If all the other two edges (e' and e'') are also on the boundary of C^I:
  - If neither e' nor e'' is solved in f, then assign e with an arbitrary value $a \in \{+1, -1\}$.
  - If at least one of them (say, e') is already solved in f with assignment $a' \in \{+1, -1\}$, then assign e with $a = -a'$;

By repeating this process for every boundary edge of C^I, the whole boundary of C^I is solved. And in all the above cases, for any $f \in C^I$ that has at least two edges (say e and e') on the boundary of C^I, these edges are assigned with opposite signs. Therefore the assignment on the boundary of C^I is safe, and C^I is ready to be solved in later iterations.

After the boundary of interior C^I is solved, we proceed to solve the frontier C^F. In C^F, those edges on the boundary C has already been solved in previous steps, thus they will be skipped here. For all the other edges in C^F, they can be classified and solved accordingly. For any one of such edges e:

- If e is a common edge shared by two consecutive frontier faces f_i and f_{i+1}, i.e. $e = e_i$, then solve e in these two faces with assignment $a_i^{(i)} = +1$ and $a_i^{(i+1)} = -1$ respectively.

- Otherwise, e is a common edge shared by a frontier face $f_i \in C^F$ and an interior face $f_i' \in C^I$. Thus e must already be solved in the C^I side with assignment, say, $a' \in \{+1, -1\}$. Then solve e in the C^F side with assignment $a = -a'$.

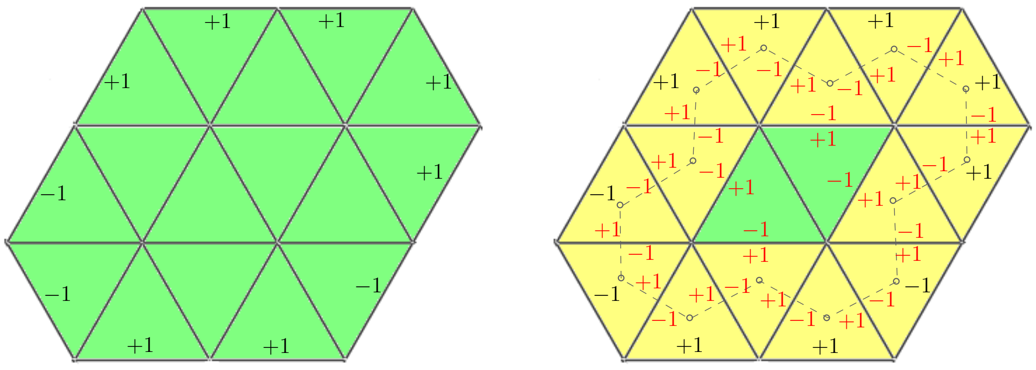


Figure 5.7: An aggregated sub-component is only solved in the frontier (in yellow), leaving the interior (in green) to later iterations.

5.2.8 Existence and Convergence

As shown in the explanation of each step above, all the invariants from definition 5.2.2 are preserved throughout algorithm 5.2.1. As a result, we have the following conclusion on the convergence of this algorithm.

Theorem 5.2.5 (Convergence of Algorithm 5.2.1)**.** *Given a triangular mesh D of a simply connected topological disk with a restricted boundary condition, algorithm 5.2.1 always terminates; and when it terminates, the output is a solution to the cutting pattern problem.*

Furthermore, according to theorem 5.2.5 we have the following conclusion on the existence of solutions.

Theorem 5.2.6 (Existence of Solutions for Restricted Boundary Conditions)**.** *The cutting pattern problem always has solutions under restricted boundary conditions.*

5.3 An Algorithm for General Boundary Conditions

At the beginning of this chapter, we convert the discrete unit tangent bundle problem to the cutting pattern problem (definition 5.1.2). In the previous section 5.2, we provide an algorithm for the cutting pattern problem under restricted boundary conditions. In this part, we generalize that algorithm to general boundary conditions (definition 5.1.3).

5.3.1 Algorithm Pipeline

Given a triangular mesh D with a general boundary condition, if there is any dangling or singular triangle, it is allowed to have a uniform cutting pattern ($+1$ or -1) on two boundary edges. In another word, the boundary assignment for D may not be safe (definition 5.2.1). In a special case where the boundary assignment is safe, the problem degenerates to restricted boundary conditions and can be solved with algorithm 5.2.1 directly. Otherwise, extra efforts are needed to address the dangerous parts and try to resolve them. The whole algorithm pipeline is defined as follows.

Algorithm 5.3.1 (Cutting Pattern Algorithm - II)**.**

- *Input: A simply connected topological disk represented as a triangular mesh D with a general boundary condition (definition 5.1.3).*

- *Output: A set of valid cutting patterns for the whole mesh D or an unsolvable sub-mesh.*

- *Procedures:*

  [1] Initialize all the inner edge with assignment 0*, and all the triangles as unsolved.*

  [2] Check the boundary assignment. If it is safe, go to step 4; otherwise, go to the next step.

  [3] Partition mesh D into a minimal set of sub-components of basic types, try to solve the breaking edges between sub-components and try to solve every sub-component. If any breaking edge or any sub-component is unsolvable, report it and exit; otherwise, put all the unsolved triangles into a sub-mesh and go to step 4.

  [4] Run step 2 in algorithm 5.2.1 to solve the unsolved sub-mesh from the previous step.

In this generalized algorithm, step 3 is the part where all the sub-components with dangerous boundary assignments are addressed, and where the algorithm might be terminated due to unsolvable sub-components. It is the major difference of this algorithm compared to the basic algorithm 5.2.1 for restricted boundary conditions. Therefore, we spend most of the rest of this section to elaborate step 3.

5.3.2 Classifying Boundary Assignments

In step 3 of algorithm 5.3.1, the input mesh D is partitioned into a minimal set of sub-components along a minimal set of breaking edges. This is carried out in the same way as in the basic algorithm for the restricted boundary conditions (section 5.2.4), and the resulting sub-components here also have three types: aggregated, linear and singular (definition 5.2.3).

Right after the partition, each sub-component will have an initial boundary assignment, where every boundary edge that is also on the boundary of the original mesh D has an assignment $+1$ or -1 that is prescribed in the input, while every other boundary edge is newly introduced in the partition and has an initial assignment 0. The new boundary edges with initial assignment 0 will be solved later and get an assignment of $+1$ or -1. Different to the restricted case, where the boundary assignments for sub-components are guaranteed to be save ones, the boundary assignments here could be of multiple types.

Definition 5.3.1 (Types of Boundary Assignments)**.** *For a sub-component C in step 3 of algorithm 5.3.1, its boundary assignments can be classified as follows*

[1] Safe boundary assignment: if and only if any dangling or singular triangle f (if there is one) has two boundary edges with opposite non-zero assignments ($+1$ and -1). Specifically, a safe boundary assignment could only be one of the following cases:

(a) C is aggregated and thus has no dangling or singular triangle; or

(b) C is linear, and in any end triangle (dangling triangle) f_i ($i = 1, 2$), two boundary edges $e_{i,1}$ and $e_{i,2}$ have opposite assignments $a_{i,1} = -a_{i_2} \neq 0$; or

(c) C is singular, and in the only triangle (singular triangle), two of the boundary edges (say e_1 and e_2) have opposite assignments $a_1 = -a_2 \neq 0$.

[2] Moderately dangerous boundary assignment: if and only if it is not safe and there are at least two boundary edges in C assigned with opposite

values ($+1$ and -1). Specifically, a moderately dangerous boundary assignment could only be one of the following cases:

(a) C is linear, and one of the end triangles f_1 has a non-zero assignment $a \neq 0$ on one of its boundary edges, and the other end triangle f_2 has an opposite assignment $-a$ on one of its boundary edges; or

(b) C is linear, and one of the end triangles f_1 has a non-zero assignment $a \neq 0$ on one of its boundary edges, and one of the middle triangles f_3 has an opposite assignment $-a$ on its only boundary edge; or

(c) C is linear, and one of the middle triangles f_3 has a non-zero assignment $a \neq 0$ on its only boundary edge, and another middle triangle f_4 has an opposite assignment $-a$ on its only boundary edge.

[3] Extremely dangerous boundary assignment: if and only if it is not safe and all the non-zero assignments (if there is any) on boundary edges are equal to a single value $a \in \{+1, -1\}$. This non-zero value a is called the forbidding value *of this assignment. Specifically, an extremely dangerous boundary assignment with forbidding value $a \in \{+1, -1\}$ could only be one of the following cases:*

(a) C is singular, and either there is no non-zero assignment on the boundary, or all the non-zero assignments are equal to a; or

(b) C is linear, and either there is no non-zero assignment on the boundary, or all the non-zero assignments are equal to a.

In this definition, the first one, safe boundary assignment, is actually the same as that for the restricted case (definition 5.2.1). Besides that, the last two are only for the general case here. They are also called *dangerous* boundary assignments. A dangerous boundary assignment could be either moderately dangerous or extremely dangerous, but not both.

It can be verified that a boundary assignment for a given sub-component C is dangerous if and only if it is not safe; that is, if and only if there is at least one dangling or singular triangle $f \in C$ that has a uniform non-zero assignment ($+1$ or -1) on all boundary edges. And such a triangle is called a *dangerous* triangle.

A sub-component with a safe boundary assignment is called a *safe sub-component*. Similarly, a sub-component with a dangerous boundary assignment is called a *dangerous sub-component*; if the boundary assignment is moderately dangerous or extremely dangerous, the sub-component is called a *moderately dangerous sub-component* or *extremely dangerous sub-component* respectively.

5.3.3 Completing Boundary Assignments

For any sub-component C resulting from the above partition, the new boundary edges of C are initialized with assignment 0 (i.e. open assignment) and need to be assigned with valid cutting patterns ± 1. This is equivalent to solving all the breaking edges along which the partition is carried out.

By completing cutting patterns for new boundary edges in a sub-component, a boundary assignment of one type can be transformed into another one of possibly different type. However, such transformations cannot happen between arbitrary pairs of types. As an interesting observation, a transformation can only keep or decrease the dangerousness of a boundary assignment, but not increase it. In particular,

- An extremely dangerous one can be transformed to another extremely dangerous one, a moderately dangerous one, or even a safe one;
- A moderately dangerous one can be transformed to another moderately dangerous one or a safe one, but not to any extremely dangerous one;
- A safe can only be transformed to another safe one, but not to any dangerous one.

Inspired by the above observations, we propose a *dangerousness relieving* process to solve this boundary assignment completion problem. That is, we process sub-components from the most dangerous ones to less dangerous ones, and for every sub-component, we solve its new boundary edges with best efforts, and thus transform itself and its adjacent sub-components to less dangerous ones. If any unsolvable edge is detected, the process should terminate and report it; otherwise, it will end when the boundary assignments for all sub-components are completely solved.

Procedure 5.3.1 defined in below implements such a dangerousness relieving process. For any sub-component C to be processed, some common notations are used in that procedure:

- $\{e_i^b \mid 1 \leqslant i \leqslant m\}$: the set of new boundary edges of C; or equivalently, the set of breaking edges that separate C from adjacent sub-components;
- C_i': an adjacent sub-component that shares a common edge e_i^b with C;
- a_i, a_i' ($\in \{+1, -1\}$): the assignment for breaking edge e_i^b in sub-component C and C_i' respectively;
- f_i, f_i': the triangle in C and C_i' that contains breaking edge e_i^b.

Procedure 5.3.1 (Boundary Assignment Completion)**.** *In step 3 of algorithm 5.3.1, given a minimal set of sub-components resulting from the partition of the input mesh D, do the following to complete their boundary assignments.*

[1] Repeat the following until all the extremely dangerous sub-components (if there is any) have been transformed into moderately dangerous or safe ones:

(a) Address all the extremely dangerous sub-components, put them in a set $\mathfrak{C}$;

(b) For every extremely dangerous sub-component $C \in \mathfrak{C}$, do the following:

- *If there is an adjacent sub-component (say C_i') that is safe, then assign e_i^b with $a_i = -b$ (thus C_i becomes moderately dangerous or safe) and $a_i' = b$ (thus C_i' remains safe).*
- *Otherwise, if there is an adjacent sub-component (say C_i') that is moderately dangerous, then assign e_i^b with $a_i = -b$ (thus C_i becomes moderately dangerous or safe) and $a_i' = b$ (thus C_i' remains moderately dangerous or becomes safe).*
- *Otherwise, if there is an adjacent sub-component (say C_i') that is extremely dangerous with opposite forbidding value $-b$, then assign e_i^b with $a_i = -b$ (thus C_i becomes moderately dangerous or safe) and $a_i' = b$ (thus C_i' becomes moderately dangerous or safe).*
- *Otherwise, all of the adjacent sub-components must be extremely dangerous and have the same forbidding value b; Then skip this sub-component C so that at a later point some adjacent sub-component might be transformed into less dangerous ones and thus C will have a chance.*

(c) *Address all the extremely dangerous sub-components again and put them in a set $\tilde{\mathfrak{C}}$.*

- *If $\tilde{\mathfrak{C}}$ is empty, stop step 1 of this procedure and go to step 2 of this procedure.*
- *Otherwise, if $\tilde{\mathfrak{C}}$ contains the same extremely dangerous sub-components as $\mathfrak{C}$ does, report $\tilde{\mathfrak{C}}$ as unsolvable and exit the procedure.*
- *Otherwise, $\mathfrak{B}$ is non-empty, then go back to step 1a of this procedure and start another iteration.*

[2] For every sub-component C that is moderately dangerous, do the following:

   (a) For every new boundary edge e_i^b in C that has assignment 0 (i.e. still open):

   - *If the corresponding adjacent sub-component C_i' is moderately dangerous, then assign e_i^b with arbitrary value $a_i \in \{+1, -1\}$ in C_i (thus C_i remains moderately dangerous or becomes safe) and $a_i' = -a_i$ in C_i' (thus C_i' remains moderately dangerous or becomes safe).*
   - *Otherwise, the corresponding adjacent sub-component C_i' must be safe, then assign e_i^b with arbitrary value $a_i \in \{+1, -1\}$ in C_i (thus C_i remains moderately dangerous or becomes safe) and $a_i' = -a_i$ in C_i' (thus C_i' remains safe).*

[3] Exit the procedure.

As shown in the above procedure, there are two points where it could terminate.

The first is in step 1c of the procedure, where a set of unsolvable sub-components is detected. Such a set of unsolvable sub-components is an obstruction that keeps the algorithm from generating a valid solution for the whole input mesh. We define it formally as follows.

Definition 5.3.2 (Cutting Pattern Obstruction)**.** *Let D be a triangular mesh of a simply connected topological disk, where every boundary edge is assigned with a cutting pattern $+1$ or -1. A cutting pattern obstruction in D is a maximal edge-connected component $C \subset D$ that is the union of a set of sub-components $\mathbb{S} = \{S_i \mid 1 \leqslant i \leqslant n\}$ $(n \geqslant 1)$, such that*

- *Every pair of sub-components S_i and S_j are triangle-disjoint to each other.*

- *Every $S_i \in \mathbb{S}$ is a maximal sub-component of basic types (i.e. singular, linear or aggregated). Or equivalently, the set $\mathbb{S}$ is a minimal set of basic type sub-components.*

- *All of the sub-components in $\mathbb{S}$ are extremely dangerous (definition 5.3.1) and have the same forbidding value $a \in \{+1, -1\}$.*

If there is any obstruction as defined above, procedure 5.3.1 will detect them and exit immediately.

The second point that the procedure could terminate is in step 3 at the end of the procedure. If there is no cutting pattern obstruction in the input mesh D, procedure 5.3.1 will finally terminate and output completed boundary assignments for all the sub-components.

5.3.4 Solving Sub-Components

Once the given mesh D is partitioned into sub-components and each sub-component gets a completely solved boundary assignment, we are ready to solve each sub-component based on its own type and the type of its boundary assignment.

Note that after running procedure 5.3.1 for boundary assignment completion, all the extremely dangerous sub-components are transformed to less dangerous ones, and only moderately dangerous sub-components and safe sub-components are left. Therefore we only need to solve the later two.

For safe sub-components, the problem is the same as that for the restricted boundary condition case. Safe sub-components that are singular, linear or aggregated can be directly processed by procedures from section 5.2.5, 5.2.6 and 5.2.7 respectively.

For moderately dangerous sub-components, by definition they must be linear. Recall that a linear sub-component C consists of a sequence of n ($n \geqslant 2$) triangles,

$$f_0,\ f_1,\ \cdots,\ f_n$$

where f_0 and f_n are end triangles (with valence 1) that each has two boundary edges $e^B_{i,1}$ and $e^B_{i,2}$ $(i = 0, n)$, and any other f_i is a middle triangle (with valence 2) that has only one boundary edge e^B_i $(1 \leqslant i \leqslant n-1)$. These boundary edges can be ordered as

$$(e^B_{0,1},\ e^B_{0,2}),\ e^B_1,\ e^B_2,\ \cdots,\ e^B_{n-1},\ (e^B_{n,1},\ e^B_{n,2})$$

Meanwhile, any pair of consecutive triangles f_{i-1} and f_i $(1 \leqslant i \leqslant n)$ share a common internal edge e^I_i, and all such internal edges can be ordered as a sequence

$$e^I_1,\ \cdots,\ e^I_n$$

Recalling the case for restricted boundary conditions in section 5.2.6 where a linear sub-component C always has a safe boundary assignment, the procedure there is equivalent to constructing a path P, such that P passes through all the triangles sequentially from f_0 all the way to f_n and penetrates the sequence of internal edges from e^I_1 through e^I_n; and whenever an internal edge e^I_i is penetrated, it is assigned with a pair of opposite non-zero values, $a\left(e^I_i, f_{i-1}\right) == 1$ and $a\left(e^I_i, f_i\right) = -1$.

In case of general boundary conditions, on the other hand, C could have a moderately dangerous boundary assignment. For this case, we use a similar procedure that also constructs a penetrating path through all the triangles and internal edges, but with more sophisticated rules to solve the internal edges.

According to definition 5.3.1, a moderately dangerous sub-component C must have at least two boundary edges $e^B_{i_1}$ and $e^B_{i_2}$ (assuming $i_1 < i_2$) in two different triangles f_{i_1} and f_{i_2} such that they have opposite assignments; that is, for some non-zero value $b \in \{+1, -1\}$, we have

$$a\left(e^B_{i_1}, f_{i_1}\right) = b$$

$$a\left(e^B_{i_2}, f_{i_2}\right) = -b$$

Taking f_{i_1} and f_{i_2} as two intermediate stops, the path we construct is a union of three segments

$$P = P_{01} \cup P_{12} \cup P_{2n}$$

where

$$P_{01} = [f_0, \cdots, f_{i_1}]$$

$$P_{12} = [f_{i_1}, \cdots, f_{i_2}]$$

$$P_{2n} = [f_{i_2}, \cdots, f_n]$$

In general, these path segments can be processed individually as follows.

- For P_{01}: In path segment P_{01}, the first triangle f_0 has two boundary edges $e^B_{0,1}$ and $e^B_{0,2}$, which already have assignments $a_{0,1}$ and $a_{0,2} \in \{+1, -1\}$. Then inner edge e^I_1 can be assigned in f_0 with $a\left(e^I_1, f_0\right) = -a_{0,1}$ (thus achieving completeness for f_0) and assigned in f_1 with $a\left(e^I_1, f_1\right) = a_{0,1}$ (thus achieving consistency for e^I_1).

  The last triangle f_{i_1} in path segment P_{01} has a boundary edge $e^B_{i_1}$ with a non-zero assignment b; consider the previous triangle f_{i_1-1}, it must also have a boundary edge $e^B_{i_1-1}$ with a non-zero assignment b_1. Then any inner edge e^I_i $(1 < i \leqslant i_1)$ can be assigned in f_{i-1} with $a\left(e^I_i, f_{i-1}\right) = -b_1$ (thus achieving completeness for f_{i-1}) and assigned in f_i with $a\left(e^I_i, f_i\right) = b_1$ (thus achieving consistency for e^I_i).

- For P_{12}: In path segment P_{12}, the first triangle f_{i_1} has a boundary edge $e^B_{i_1}$ with a non-zero assignment b, and the last triangle f_{i_2} has a boundary edge $e^B_{i_2}$ with a non-zero assignment $-b$. Then any inner edge e^I_i $(i_1 < i \leqslant i_2)$ can be assigned in f_{i-1} with $a\left(e^I_i, f_{i-1}\right) = b_2$ (thus achieving completeness for f_{i-1}) and assigned in f_i with $a\left(e^I_i, f_i\right) = -b_2$ (thus achieving consistency for e^I_i).

- For P_{2n} In path segment P_{2n}, the last triangle f_n has two boundary edges $e^B_{n,1}$ and $e^B_{n,2}$, which already have assignments $a_{n,1}$ and $a_{n,2} \in \{+1, -1\}$. Then inner edge e^I_n can be assigned in f_n with $a\left(e^I_n, f_n\right) = -a_{n,1}$ (thus achieving completeness for f_n) and assigned in f_{n-1} with $a\left(e^I_n, f_{n-1}\right) = a_{n,1}$ (thus achieving consistency for e^I_n).

  The first triangle f_{i_2} in path segment P_{2n} has a boundary edge $e^B_{i_2}$ with a non-zero assignment $-b$; consider the next triangle f_{i_2+1}, it must also have a boundary edge $e^B_{i_2+1}$ with a non-zero assignment b_2. Then any inner edge e_i^I ($i_2 < i < n$) can be assigned in f_{i-1} with $a\left(e_i^I, f_{i-1}\right) = b_2$ (thus achieving completeness for f_{i-1}) and assigned in f_i with $a\left(e_i^I, f_i\right) = -b_2$ (thus achieving consistency for e_i^I).

From the above discussion, one can see that for a moderately dangerous sub-component C, the completeness of cutting patterns is achieved on every triangle f_i, and the consistency of cutting patterns is achieved on every edge e_i^I that needs to be solved. Therefore, such a C is completely solved with a set of valid cutting patterns.

In summery, the overall procedure for solving a sub-component with completed boundary assignment in step 3 of algorithm 5.3.1 can be outlined as the follows.

Procedure 5.3.2 (Solving A Sub-component). *In step 3 of algorithm 5.3.1, given a sub-component C with a completed boundary assignment, do the following to solve C.*

- *If the boundary assignment for C is safe, then*
  - *If C is singular, run procedure in section 5.2.5 to solve it completely;*
  - *Otherwise, if C is linear, run procedure in section 5.2.6 to solve it completely;*
  - *Otherwise, C must be aggregated, run procedure in section 5.2.7 to solve its frontier and leave its interior to further iterations in step 4 of algorithm 5.3.1.*
- *Otherwise, the boundary assignment for C must be moderately dangerous and thus C must be linear. Define the following notations for C:*
  - *$F = \{f_i \mid 0 \leqslant i \leqslant n\}$: the set of triangles in C ordered sequentially;*
  - *$E^I = \{e_i^I \mid 1 \leqslant i \leqslant n\}$: the corresponding set of inner edges to be solved in C;*

- *$E^B = \{e^B_{0,1},\ e^B_{0,2}\} \cup \{e^B_i \mid 1 \leqslant i \leqslant n-1\} \cup \{e^B_{n,1},\ e^B_{n,2}\}$: the corresponding set of boundary edges in C;*
- *$a_{i,1}$ and $a_{i,2} \in \{+1, -1\}$ $(i = 0, n)$: the assignments for boundary edges $e^B_{i,1}$ and $e^B_{i,2}$ in end triangle f_i;*
- *$e^B_{i_1}$ and $e^B_{i_2} \in E^B$: two boundary edges with opposite assignments b and $-b$ $(b \in \{+1, -1\})$;*
- *f_{i_1} and $f_{i_2} \in F(0 \leqslant i_1 \leqslant i_2 \leqslant n)$: triangles in C that enclose $e^B_{i_1}$ and $e^B_{i_2}$ respectively;*
- *b_1 and $b_2 \in \{+1, -1\}$: the assignments for $e^B_{i_1-1} \in f_{i_1-1}$ and $e^B_{i_2+1} \in f_{i_2+1}$ respectively.*

Then solve the inner edges $\{e^I_1,\ e^I_2,\ \cdots,\ e^I_n\}$ as follows:

[1] For e^I_1, solve it in f_0 and f_1 with

$$a\left(e^I_1, f_0\right) = -a_{0,1}$$

$$a\left(e^I_1, f_1\right) = a_{0,1}$$

[2] For every e^I_i $(1 < i \leqslant i_1)$, solve it in f_{i-1} and f_i with

$$a\left(e^I_i, f_{i-1}\right) = -b_1$$

$$a\left(e^I_i, f_i\right) = b_1$$

[3] For every e^I_i $(i_1 < i \leqslant i_2)$, solve it in f_{i-1} and f_i with

$$a\left(e^I_i, f_{i-1}\right) = -b$$

$$a\left(e^I_i, f_i\right) = b$$

[4] For every e_i^I $(i_2 < i < n)$, solve it in f_{i-1} and f_i with

$$a\left(e_i^I, f_{i-1}\right) = b_2$$

$$a\left(e_i^I, f_i\right) = -b_2$$

[5] For e_n^I, solve it in f_{n-1} and f_n with

$$a\left(e_n^I, f_{n-1}\right) = a_{n,1}$$

$$a\left(e_n^I, f_n\right) = -a_{n,1}$$

- *Exit the procedure.*

5.3.5 Existence and Convergence

In the above discussions, we already address and validate all the major issues regarding the convergence of algorithm for general boundary conditions. In particular:

- For step 3 of algorithm 5.3.1, we show the follows.
  - In section 5.3.2, we show that the input mesh is partitioned into a minimal set of sub-components of basic types (singular, linear or aggregated) and with various types of initial boundary assignments (safe, moderately dangerous or extremely dangerous).
  - In section 5.3.3, we show that by procedure 5.3.1, if there is any cutting pattern obstruction (definition 5.3.2) in the input mesh D, the whole algorithm will terminate and output the obstruction. Otherwise, all the initial boundary assignments (consisting of $+1$ and -1 and 0) are completed (only consisting of $+1$ and -1), and all the extremely dangerous ones are downgraded to less dangerous one. Therefore after this process, all the boundary assignments are completed and become either moderately dangerous or safe.

  - In section 5.3.4, we show that by procedure 5.3.2, all the moderately dangerous sub-components are completely solved with valid cutting patterns. For safe sub-components, they are either completely solved or partially solved using procedures from the algorithm for restricted boundary conditions, where we already show that these procedures generate valid cutting patterns for newly solved part and the remaining unsolved triangles constitute a sub-mesh that has a safe boundary assignment. Therefore after this process, the original mesh is reduced to an unsolved sub-mesh with a restricted boundary condition.

- For step 4 of algorithm 5.3.1, we run algorithm 5.2.1 on the unsolved sub-mesh with a restricted boundary condition. According to theorem 5.2.5, this process will terminate and will generate a set of valid cutting patterns for the given sub-mesh.

Based on all these discussions, we have the following conclusion on the convergence of algorithm 5.3.1.

Theorem 5.3.3 (Convergence of Algorithm 5.3.1)**.** *Given a triangular mesh D of a simply connected topological disk with a general boundary condition, algorithm 5.4.1 always terminates; and when it terminates,*

- *If D has any cutting pattern obstruction (definition 5.3.2), the algorithm outputs such an obstruction (which is a connected component of D).*

- *Otherwise, the algorithm outputs a set of valid cutting patterns for the whole mesh D.*

Furthermore, according to theorem 5.3.3 we have the following conclusion on the existence of solutions.

Theorem 5.3.4 (Existence of Solutions for General Boundary Conditions)**.** *The cutting pattern problem has solutions under general boundary conditions if there is no cutting pattern obstruction in the input mesh.*

5.4 An Algorithm for Free Boundary Conditions

In the previous two sections, we propose algorithms for the cutting pattern problem under restricted boundary conditions (section 5.2) and general boundary conditions (5.3) respectively. Both of them can be considered as fixed boundary conditions. In this part, we discuss the cutting pattern problem under free boundary conditions (definition 5.1.3).

5.4.1 Algorithm Pipeline

Under a free boundary condition, given a triangular mesh D, the boundary edges will not be prescribed with any assignment. Instead, these values can be determined by the algorithm. Actually, at the beginning of the algorithm, the boundary edges can be automatically initialized with any safe boundary assignment; this way the problem with a free boundary condition is turned into one with a restricted boundary condition, and then it can be solved using algorithm 5.2.1. The new algorithm pipeline is outlined in below.

Algorithm 5.4.1 (Cutting Pattern Algorithm - III)**.**

- *Input: A simply connected topological disk represented as a triangular mesh D with a free boundary condition (definition 5.1.3).*
- *Output: A set of valid cutting patterns for the whole mesh D.*
- *Procedures:*

  [1] Initialize the boundary of D with a safe boundary assignment.

  [2] Run algorithm 5.2.1.

In this algorithm, step 2 has been elaborated in section 5.2. The only step that needs to be explained is step 1.

5.4.2 Initializing The Boundary

Consider step 1 in the above algorithm pipeline. Given a triangular mesh D of a simply connected topological disk, all the boundary edges need to be initialized with a valid cutting pattern $+1$ or -1. This can be easily achieved by considering the triangles that contain boundary edges.

Procedure 5.4.1 (Initialize Boundary Assignments)**.** *Given an input mesh D, for every triangle f containing boundary edge(s), do the following:*

- *If there is only one boundary edge e in f, initialize e with assignment $a = +1$;*
- *If there are two boundary edges e_1 and e_2 in f, i.e. f is a dangling triangle, initialize them with assignments $a_1 = +1$ and $a_2 = -1$ respectively;*
- *If all three edges e_1, e_2 and e_3 are boundary edges, i.e. f is a singular triangle, assign them with $a_1 = a_2 = +1$ and $a_3 = -1$ respectively;*

By calling procedure 5.4.1, the input mesh D is assigned with an initial boundary assignment $\mathfrak{A}$. And it is easy to verify that $\mathfrak{A}$ satisfies all the requirements for a restricted boundary condition (definition 5.1.3). Therefore, the problem is converted to one with a restricted boundary condition, where $\mathfrak{A}$ serves as a prescribed boundary assignment.

5.4.3 Existence and Convergence

In algorithm 5.4.1, step 1 converts a free boundary condition to a restricted boundary condition, which is given as input to step 2 to be processed by algorithm 5.2.1. According to theorem 5.2.5, algorithm 5.2.1 always terminates for any input mesh D with a restricted boundary condition, and always generates a solution for the cutting pattern problem.

Therefore we have the following conclusion on the convergence of algorithm 5.4.1.

Theorem 5.4.1 (Convergence of Algorithm 5.4.1)**.** *Given a triangular mesh D of a simply connected topological disk with a free boundary condition, algorithm 5.4.1 always terminates; and when it terminates, the output is a solution to the cutting pattern problem.*

Furthermore, according to theorem 5.4.1 we have the following conclusion on the existence of solutions.

Theorem 5.4.2 (Existence of Solutions for Free Boundary Conditions)**.** *The cutting pattern problem always has solutions under free boundary conditions.*

Chapter 6

Discrete Unit Tangent Bundles for $g = 0$ Closed Surfaces

This chapter is devoted to discrete unit tangent bundles for topological 2-spheres, i.e. $g = 0$ closed and oriented surfaces. We start from the intuitions behind our construction (section 6.1), then present the local construction (section 6.2) and global construction (section 6.3) respectively, and give validations of our methods (section 6.4).

6.1 Intuitions

Unit tangent bundles for S^2 surfaces are non-trivial. In order to represent such a complicated 3-manifold, we utilize the idea of *local trivialization*. Specifically, we partition a given surface into a set of covering disks, build a unit tangent bundle (which is trivial) over each disk, then glue these trivial bundles into a non-trivial bundle for the original surface.

Without loss of generality, we use a sphere that has radius one and is centered at the origin of R^3 (figure 6.1); Any other $g = 0$ surface can be mapped to such a standard sphere. Given a standard sphere M, it can be sliced open along the equator on the XY plane into two semi-spheres, which can be further mapped to two unit disks D_1 and D_2 on the XY plane by stereographic projection. These two disks are called covering disks for the original surface M.

For each covering disk, its unit tangent bundle is simply a direct product of the disk with a circle, $D_i \times S^1$, which is a solid torus (figure 5.1). In fact, two such solid tori can cover a unit tangent bundle over M, overlapping each other only on their boundary surfaces, because the intersection of the two covering disks for M only happens on their boundary loops.

For the purpose of a global construction, we need to consider the *transition function* ϕ_{21} between two covering disks. It can be formulated as a complex function

$$\phi_{21}(z) = \frac{1}{z} \tag{6.1}$$

By changing to the polar coordinates $z = re^{i\theta}$ and restricting to the boundary of unit disk, we have:

$$\phi_{21}(z) = e^{-i\theta} \tag{6.2}$$

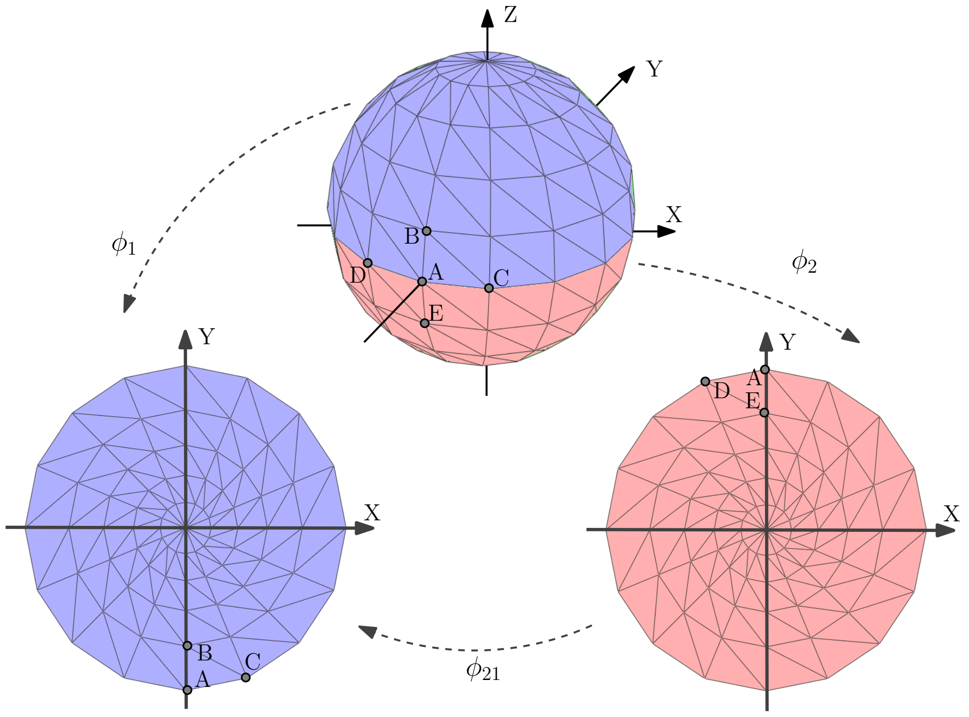


Figure 6.1: A 2-sphere can be parameterized by two unit disks. The map ϕ_1 and ϕ_2 are both orientation-preserving, verified by the oriented triangle ACB and ADE. The chart transition function is $\phi_{21} = 1/z$ in complex form.

By taking the differential

$$d\phi_{21}(z) = -\frac{1}{z^2} = -\frac{1}{r^2}e^{-2\theta i} \tag{6.3}$$

the Jacobian J_{21} on the boundary of unit disk can be written as:

$$J_{21} = \begin{pmatrix} \cos(\pi - 2\theta) & -\sin(\pi - 2\theta) \\ \sin(\pi - 2\theta) & \cos(\pi - 2\theta) \end{pmatrix} \tag{6.4}$$

The transition function (equation 6.2) and Jacobian (equation 6.4) give information about how to match the boundary surfaces of two local bundles. The following proposition states the requirements that our discrete construction should meet.

Proposition 6.1.1. *A discrete construction of unit tangent bundles over a S^2 surface M using two local bundles over covering disks D_i should satisfy the* *following requirements:*

[1] According to the transition function ϕ_{21}, a vertex v_j^2 in ∂D_2 with polar angle θ_j should be mapped to a vertex v_i^1 in ∂D_1 with polar angle $\theta_i = -\theta_j$, and the fiber over v_j^2 should be mapped to the fiber over v_i^1.

[2] According to the Jacobian J_{21}, when one starts from a vertex $v_j^2 \in \partial D_2$ and goes around ∂D_2 once (counterclockwise) to get back to v_j^2, the corresponding fiber should be rotated (or twisted) by 4π at the end.

6.2 Local Construction

Once two covering disks $\{D_k \,|\, 1 \leqslant k \leqslant 2\})$ are given, we can build local trivial bundles $\{T_k = D_k \times S^1 \,|\, 1 \leqslant k \leqslant 2\})$. Local construction refers to the stage where each local bundle T_k is tessellated with tetrahedra. Such a bundle can be constructed using algorithms from chapter 5, with extra constraints on the boundary triangulation. In particular, we need a special grid structure (section 6.2.1) on the boundary surface ∂T_k, which allows us to assign meaningful coordinates (section 6.2.2) to the vertices in ∂T_k. Then an appropriate gluable triangulation (section 6.2.3) is needed for ∂T_k so that the tessellation algorithms from chapter 5 can be adapted here (section 6.2.4) and generates discrete local bundles that meet the requirements from the global construction in a later stage.

6.2.1 Grid Structure

The boundary surface ∂T_k of a local bundle T_k is a genus one closed surface. Considering the global construction that glues two local bundles along their boundary, it turns out that vertices in boundary triangulation should have certain grid structures.

Generally speaking, a grid structure over such a torus surface could be a 2D tensor product of two discretized circles $\partial D_k \times S^1$, where ∂D_k is the boundary of covering disk D_k, S^1 is the fiber. Each circle should be discretized by a set of sampling points (i.e. vertices), and the number of vertices is called the *discretizing resolution*. Suppose the discretizing resolution of D_k and S^1 is M_b and M_f respectively, a grid structure Γ can be generally defined over the boundary of a local bundle T_k as a pair of integers

$$\Gamma(\partial T_k) = [M_b, M_f] \tag{6.5}$$

Furthermore, some other requirements arise from our specific construction of unit tangent bundles. First, according to proposition 6.1.1, when one goes around D_k once, he should go around the fiber S_1 twice. It implies that we need the discretization resolution along D_k two times of that along S^1, $M_b = 2M_f$. Second, according to equation 6.4, there should be a sample point in the midway of S^1. It implies that the resolution of S^1 should be an even number, $M_f = 2N$ for some positive integer N. As a consequence, the grid structure we actually use in our construction should have the following form

$$\Gamma(\partial T_k) = [4N, 2N] \tag{6.6}$$

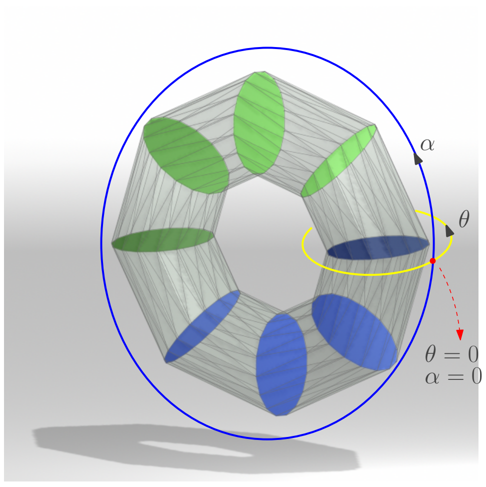
for some positive integer N and for $1 \leqslant k \leqslant 2$.

6.2.2 Vertex Coordinates

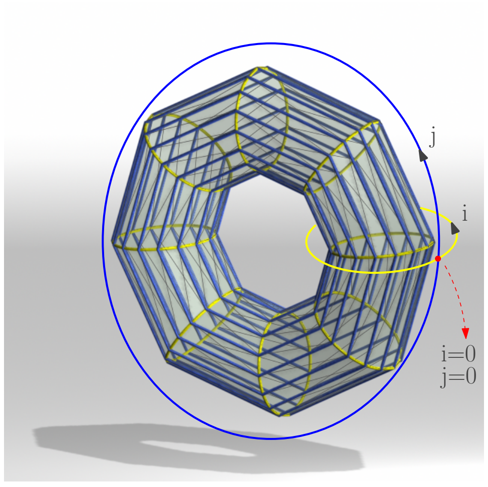
A grid structure as in section 6.2.1 allows us to assign simple yet meaningful coordinates to vertices of ∂T. Here we introduce two different coordinate systems.

The first one is called *angle coordinate system* (figure 6.2a). Note that each vertex is the intersection of two circles, a copy of ∂D_k and a fiber S^1. Along a copy of ∂D_k the vertices can be parameterized by polar angle θ, while along a fiber S^1 the vertices can be parameterized by polar angle α. Therefore each vertex v can be represented as a pair of real values θ and α. Plus, to distinguish vertices from two local bundles, we put an extra integer k $(1 \leqslant k \leqslant 2)$ in the coordinate, which results in a three-tuple coordinate for each vertex

$$(k,\, \theta,\, \alpha)$$



(a)



(b)

Figure 6.2: In angle coordinates (a), θ and α should be normalized to $[0..2\pi)$. In grid coordinates (b), i and j should be modulated by m and n respectively, if there are m fiber circles (in blue) and n base circles (in yellow).

Note that the valid values for both θ and α are between 0 and 2π; any value beyond this range should be normalized to this range by module 2π.

The second system is called *grid coordinate system* (figure 6.2b). Similarly, along each copy of ∂D_k, the vertices can be numbered by $\{0, 1, ..., M_b - 1\}$ in the increasing order of polar angle θ. Along each fiber S^1, the vertices can be numbered by $\{0, 1, ..., M_f - 1\}$ in the increasing order of polar angle α. Plus an extra integer k $(1 \leqslant k \leqslant 2)$ to indicate which local bundle the vertex belongs to, we have another three-tuple coordinate for each vertex

$$(k,\ i,\ j)$$

Note that the valid values for i and j is $[0..M_b - 1]$ and $[0..M_f - 1]$ respectively; any value beyond the range should be modulated by M_b or M_f accordingly.

Under certain re-parameterization that gives uniform angle difference between adjacent vertices along D_k and S^1 respectively, these two kinds of coordinates can be transformed to each other:

$$(k,\, i,\, j) = \left(k,\, \left[\frac{\theta}{2\pi} \cdot m\right],\, \left[\frac{\alpha}{2\pi} \cdot n\right]\right) \tag{6.7}$$

$$(k, \theta, \alpha) = \left(k,\, \frac{2\pi}{m} \cdot i,\, \frac{2\pi}{n} \cdot j\right) \tag{6.8}$$

By fixing the grid size to $4N \times 2N$, the transformation between angle coordinates and grid coordinates can be re-written as:

$$(k,\, i,\, j) = \left(k,\, \left[\frac{\theta}{\pi} \cdot 2N\right],\, \left[\frac{\alpha}{\pi} \cdot N\right]\right) \tag{6.9}$$

$$(k,\, \theta,\, \alpha) = \left(k,\, \frac{\pi}{2N} \cdot i,\, \frac{\pi}{N} \cdot j\right) \tag{6.10}$$

6.2.3 Gluable Triangulation

Having a grid structure 6.6 is the first step to discretize ∂T_k. Here we will define a special triangulation for ∂T_k using the grid structure. Such a triangulation allows further gluing of two local bundles and therefore is called a *gluable triangulation.*

Definition 6.2.1 (Gluable Triangulation (for $UT(S^2)$))**.** *Given a trivial local bundle $T_k = D_k \times S^1$, where D_k $(1 \leqslant k \leqslant 2)$ is a covering disk of a S^2 surface, a triangulation of its boundary $\partial T_k = \partial D_k \times S^1$ is called a gluable triangulation $\Delta(\partial T_k)$ (figure 6.4) if :*

[1] The vertices of the triangulation form a grid structure on ∂T_k with the following form:

$$\Gamma(\partial T_k) = [4N, 2N]$$

for some positive integers N;

[2] The triangles are defined by all possible combinations of vertices (repre-

sented with grid coordinates) in the following forms:

$$[(k,i,j),(k,i+1,j),(k,i+1,j+1)]$$

$$[(k,i,j),(k,i+1,j+1),(k,i,j+1)]$$

Figure 6.4 illustrates an gluable triangulations on both ∂T_k, where ∂T_k is sliced open along D_k and a fiber S^1 and flattened into a 2D domain for visualization purposes. A 3D view of gluable triangulations is shown in figure 6.3.

It is easy to notice that the edges in a gluable triangulation can be classified into three types:

- Fiber edge: (for $0 \leqslant i \leqslant 4N$, $0 \leqslant j < 2N$)

$$[(k,i,j),(k,i,j+1)]$$

- Base edge: (for $0 \leqslant i < 4N$, $0 \leqslant j < 2N$)

$$[(k,i,j),(k,i+1,j)]$$

- Diagonal edge: (for $0 \leqslant i < 4N$, $0 \leqslant j < 2N$)

$$[(k,i,j),(k,i+1,j+1)]$$

Furthermore, the edges of the same type will connect into loops, and all the edges on ∂T_k are grouped into a set of cycles in one of the following categories:

- Fiber cycles $\{\gamma_i^F(k) \,|\, 0 \leqslant i \leqslant 4N\}$:

$$\gamma_i^F(k) = [(k,i,0),(k,i,1),\cdots,(k,i,2N-1),(k,i,0)]$$

- Base cycles $\{\gamma_j^B(k) \,|\, 0 \leqslant j < 2N\}$:

$$\gamma_j^B(k) = [(k,0,j), (k,1,j), \cdots, (k,4N-1,j), (k,0,j)]$$

- Diagonal cycles $\{\gamma_j^D(k) \,|\, 0 \leqslant j \leqslant 2N\}$:

$$\gamma_j^D(k) = [(k,0,j), (k,1,j+1), \cdots, (k,4N-1,j+4N-1), (k,0,j)]$$

Figure 6.4 illustrates different types of cycles on ∂T_1 and ∂T_2; the fiber cycles, base cycles, and diagonal cycles are drawn in black, red and blue respectively.

On $\partial T_k = \partial D_k \times S^1$, every cycle of the above can be projected onto either ∂D_k or S^1, or both. Each fiber cycle $\gamma_i^F(k)$ can be 1-to-1 mapped to a fiber S^1. Each base cycle $\gamma_j^B(k)$ can be 1-to-1 mapped to ∂D_k. Each diagonal cycle $\gamma_j^D(c)$ covers both directions; it can be 2-to-1 mapped onto a fiber S^1 and 1-to-1 mapped to ∂D_k.

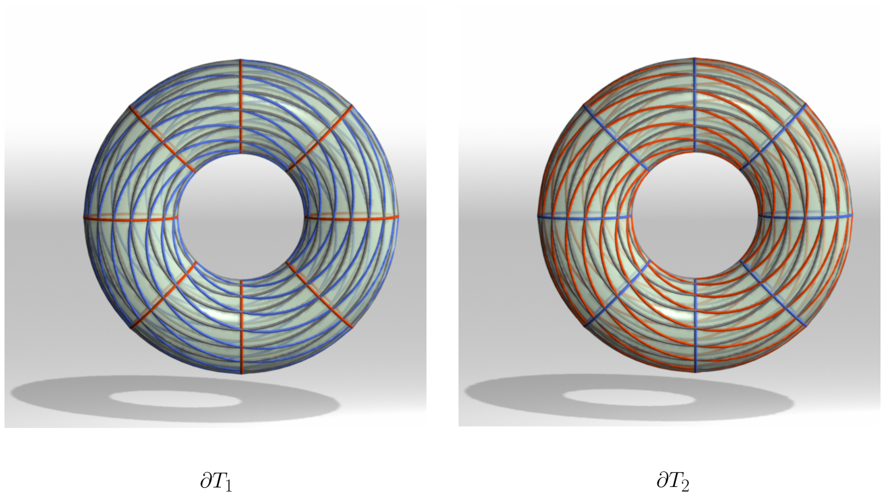


Figure 6.3: Gluable triangulations on the boundary of two local bundles for a $g = 0$ surface.

6.2.4 Local Bundle

Once a gluable triangulation is defined on ∂T_k, we are ready to triangulate the whole local bundle T_k. Note that the local bundle is a direct product both in itself $T_k = D_k \times S^1$ and on the boundary $\partial T_k = \partial D_k \times S^1$, and the gluable triangulation of ∂T_k also has a grid structure which is a tensor product. It meets all the requirements in definition 5.1.1 and can therefore be triangulated using algorithms in chapter 5.

By definition 6.2.1, each fiber has a discretizing resolution of $2N$. Thus in a tetrahedral mesh for local bundle $T_k = D_k \times S^1$, the base D_k should have $2N$ pre-images (i.e. base copies) $\{(D_k)_j | 0 \leqslant j < 2N\}$, where each $(D_k)_j$ is a triangular mesh consisting of all the vertices that have local coordinate j along fiber S^1. The whole tetrahedral mesh T_k is then partitioned by these base copies into $2N$ bundle segments $\{(D_k)_j \times I_j | 0 \leqslant j < 2N\}$, where I_j is a one-dimensional interval corresponding to an edge $(j, j+1)$ along a fiber.

In order to extend the boundary triangulation into the whole volume T_k, we can actually do this for each bundle segment $(D_k)_j \times I_j$ individually, where the problem is converted to the cutting pattern problem (definition 5.1.2) for base copy $(D_k)_j$. To do the conversion, we need to represent the boundary triangulation on each $\partial(D_k)_j \times I_j$ in the language of cutting patterns (section 5.1.2).

Definition 6.2.2 (Boundary Cutting Pattern for $UT(S^2)$)**.** *Given a gluable triangulation for $\partial T_k = \partial D_k \times S^1$, where D_k is a covering disk of a S^2 surface, T_k is a trivial local bundle $T_k = D_k \times S^1$, let $\partial(D_k)_j$ denote the j'th copy of ∂D_k in ∂T_k. The boundary cutting pattern is a function f that assigns every edge in each $\partial(D_k)_j$ with integer $+1$.*

Using the above boundary cutting patterns as boundary constraints, we can compute cutting patterns for each base copy $(D_k)_j$ using algorithms from chapter 5, which will give us a tetrahedral tessellation for each bundle segment $(D_k)_j \times I_j$ and thus a tetrahedral mesh for the whole local bundle T_k.

6.3 Global Construction

The global construction considers how to glue two local bundles (constructed in section 6.3) into a global one, and the later will be a discrete representation for the unit tangent bundle $UT(S^2)$ for the original surface S^2. As explained in section 6.1, such a gluing only involves the boundary surface of local bundles T_1 and T_2. In order to glue two local bundles together, we need to define a gluing map between ∂T_1 and ∂T_2.

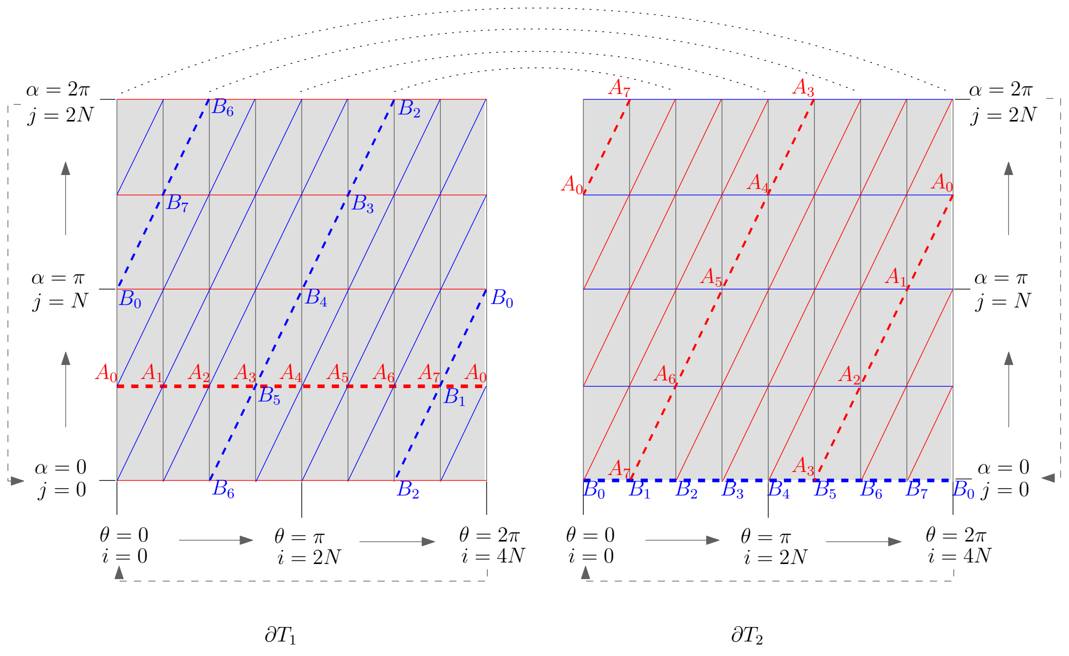

$\alpha = 2\pi$, $j = 2N$
$\alpha = \pi$, $j = N$
$\alpha = 0$, $j = 0$
$\theta = 0$, $i = 0$
$\theta = \pi$, $i = 2N$
$\theta = 2\pi$, $i = 4N$
B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7
A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7
∂T_1
∂T_2

Figure 6.4: *Gluable triangulations* (Definition 6.2.1) and a *gluing map* (Definition 6.3.1) in between. It maps each cycle in ∂T_1 to a unique cycle in ∂T_2 that has the same color, namely, grey (vertical) to grey (vertical), red (horizontal) to red (diagonal), blue (diagonal) to blue (horizontal). In particular, the high-lighted red cycle $A_0A_1 \cdots A_7A_0$ from both sides are identified under the map; same for the blue cycle $B_0B_1 \cdots B_7B_0$. Note that this map is orientation-reversing, it maps pairs of oriented triangles from both sides against the normal of each other.

6.3.1 Gluing Map

To design a gluing map between ∂T_1 and ∂T_2, one needs to consider how a gluing map should behave. The behavior of a gluing map should be consistent with the way we partition a given surface in section 6.1, and is therefore determined by the transition function between the covering disks and the jacobian of the transition function.

According the transition function in equation 6.2, ∂D_2 is identified with ∂D_1 by reversing the orientation. That means $\partial T_2 = \partial D_2 \times S^1$ should be identified with $\partial T_2 = \partial D_1 \times S^1$ with orientation reversed. Furthermore, according to the jacobian in equation 6.4, fibers in ∂T_2 need to be twisted before getting identified with fibers in ∂T_1, and the speed of fiber twisting is twice the speed of traveling along ∂D_2, plus a shift of π in angle coordinates, or equivalently, N in grid coordinates.

Based the above observations, we define a gluing map from ∂T_2 to ∂T_1 as follows.

Definition 6.3.1 (Gluing Map for $UT(S^2)$)**.** *Given an S^2 surface M, its two covering disks $\{D_k \,|\, k = 1, 2\}$ and two local bundles $\{T_k = D_k \times S^1\}$ whose boundaries $\{\partial T_k = \partial D_k \times S^1\}$ have gluable triangulations $\{\Delta(\partial T_k)\}$ with grid structure $[4N, 2N]$, a map*

$$\Psi : \partial T_2 \to \partial T_1$$

is called a gluing map if it acts on the vertices of the gluable triangulations as follows:

$$\Psi\left((2, i, j)\right) = (1, -i, j - i + N)$$

Using this map, we can glue two local bundles together along their boundary surfaces nicely. As validated in the next section, such a construction will generate a valid tetrahedral mesh that has no boundary, and this mesh is a faithful discrete representation of the unit tangent bundle over the original surface.

6.4 Validations

In previous sections, we introduce gluable triangulations on the boundary surface of local bundles, and define a gluing map between them. In this section we will show that these constructions will produce desired discrete representations for unit tangent bundles of S^2 surfaces.

6.4.1 Bijection

First of all we will show that a gluing map induces a bijection of the gluable triangulations on ∂T_1 and ∂T_2. To achieve this, we use three lemmas to show the bijection of vertices (lemma 6.4.1), edges (lemma 6.4.2) and triangles (lemma 6.4.3) respectively.

Lemma 6.4.1. *A gluing map Ψ induces a bijection between vertices in ∂T_1 and vertices in ∂T_2.*

Proof. By definition of $\Psi \partial T_2 \to \partial T_1$, any vertex in ∂T_2 has a unique image in ∂T_1 under map Ψ. So it suffices to show that every vertex in ∂T_1 has a unique pre-image in ∂T_2.

For any vertex $(1, i', j') \in \partial T_1$ ($0 \leqslant i' \leqslant 4N$, $0 \leqslant j' \leqslant 2N$), suppose it has a pre-image, denoted as $(2, i, j) \in \partial T_2$. By definition we have

$$(1, i', j') = (1, -i, j - i + N)$$

Solving this for i and j, we have

$$i = -i'$$

$$j = j' - i' - N$$

With a modulation of $4N$ for i' and $2N$ for j', it corresponds to a unique vertex in ∂T_1. Therefore Ψ is a bijection between vertices on ∂T_2 and vertices on ∂T_1. And the inverse map can be defined as:

$$\Psi^{-1}\left((1, i', j')\right) = (2, -i', j' - i' + N)$$

$\square$

Lemma 6.4.2. *A gluing map Ψ induces a bijection between edges in ∂T_1 and edges in ∂T_2.*

Proof. It suffices to show that under $\Psi \partial T_2 \to \partial T_1$, any edge in ∂T_2 has a unique image in ∂T_1, and any edge in ∂T_1 has a unique pre-image in ∂T_2.

We first show that any edge in ∂T_2 has a unique image in ∂T_1 under map Ψ.

- For any fiber edge $\left[(2, i, j), (2, i, j+1)\right]$ ($0 \leqslant i \leqslant 4N$, $0 \leqslant j < 2N$), the end vertices are mapped to

  $$\Psi\left((2, i, j)\right) = (1, -i, j - i + N)$$

  $$\Psi\left((2, i, j+1)\right) = (1, -i, j - i + N + 1)$$

  With a substitution

  $$i' = -i \text{ (modulated by } 4N)$$

  $$j' = j - i + N \text{ (modulated by } 2N)$$

  it corresponds to a valid and unique fiber edge in ∂T_1

  $$[(1, i', j'),\ (1, i', j' + 1)]$$

- For any base edge $\left[(2, i, j), (2, i+1, j)\right]$ ($0 \leqslant i < N_k$, $0 \leqslant j < N$), the end vertices are mapped to

$$\Psi\left((2,i,j)\right) = (1,-i,j-i+N)$$

$$\Psi\left((2,i+1,j)\right) = (1,-i-1,j-i-1+N)$$

With a substitution

$$i' = -i-1 \text{ (modulated by } 4N)$$

$$j' = j-i-1+N \text{ (modulated by } 2N)$$

it corresponds to a valid and unique diagonal edge in $1 \times S^1$

$$[(1,i'+1,j'+1),\ (1,i',j')]$$

- For any diagonal edge $[(2,i,j), (2,i+1,j+1)]$ $(0 \leqslant i < N_k, 0 \leqslant j < N)$, the end vertices are mapped to

$$\Psi\left((2,i,j)\right) = (1,-i,j-i+N)$$

$$\Psi\left((2,i+1,j+1)\right) = (1,-i-1,j-i+N)$$

With a substitution

$$i' = -i-1 \text{ (modulated by } 4N)$$

$$j' = j-i+N \text{ (modulated by } 2N)$$

it corresponds to a valid and unique base edge in $1 \times S^1$

$$[(1, i'+1, j'), \; (1, i', j')]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any edge in ∂T_1 is mapped to a valid and unique edge in ∂T_2.

In summary, Ψ induces a bijection between edges in ∂T_1 and edges in ∂T_2. □

Lemma 6.4.3. *A gluing map Ψ induces a bijection between triangles in ∂T_1 and triangles in ∂T_2.*

Proof. It suffices to show that under $\Psi \partial T_2 \to \partial T_1$, any triangle in ∂T_2 has a unique image in ∂T_1, and any triangle in ∂T_1 has a unique pre-image in ∂T_2.

We first show that any triangle in ∂T_2 has a unique image in ∂T_1 under map Ψ, with orientation reversed.

- For any triangle

  $$[(2, i, j), (2, i+1, j), (2, i+1, j+1)]$$

  ($0 \leqslant i < 4N$, $0 \leqslant j < 2N$), the end vertices are mapped to

  $$\Psi\left((2, i, j)\right) = (1, -i, j-i+N)$$

  $$\Psi\left((2, i+1, j)\right) = (1, -i-1, j-i-1+N)$$

  $$\Psi\left((2, i+1, j+1)\right) = (1, -i-1, j-i+N)$$

  With a substitution

  $$i' = -i-1 \text{ (modulated by } 4N)$$

$$j' = j - i - 1 + N \text{ (modulated by } 2N\text{)}$$

it corresponds to a unique but orientation-reversed triangle in ∂T_1

$$[(1, i' + 1, j' + 1), (1, i', j'), (1, i', j' + 1)]$$

- For any triangle

$$[(2, i, j), (2, i + 1, j + 1), (2, i, j + 1)]$$

$(0 \leqslant i < 4N,\ 0 \leqslant j < 2N)$, the end vertices are mapped to

$$\Psi((2, i, j)) = (1, -i, j - i + N)$$

$$\Psi((2, i + 1, j + 1)) = (1, -i - 1, j - i + N)$$

$$\Psi((2, i, j + 1)) = (1, -i, j + 1 - i + N)$$

With a substitution

$$i' = -i - 1 \text{ (modulated by } 4N\text{)}$$

$$j' = j - i + N \text{ (modulated by } 2N\text{)}$$

it corresponds to a unique but orientation-reversed triangle in ∂T_1

$$[(1, i' + 1, j'), (1, i', j'), (1, i' + 1, j' + 1)]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any triangle in ∂T_1 is mapped to a unique but reversed triangle in ∂T_2.

In summary, Ψ induces a bijection between triangles in ∂T_1 and triangles in ∂T_2 with orientation reversed. $\square$

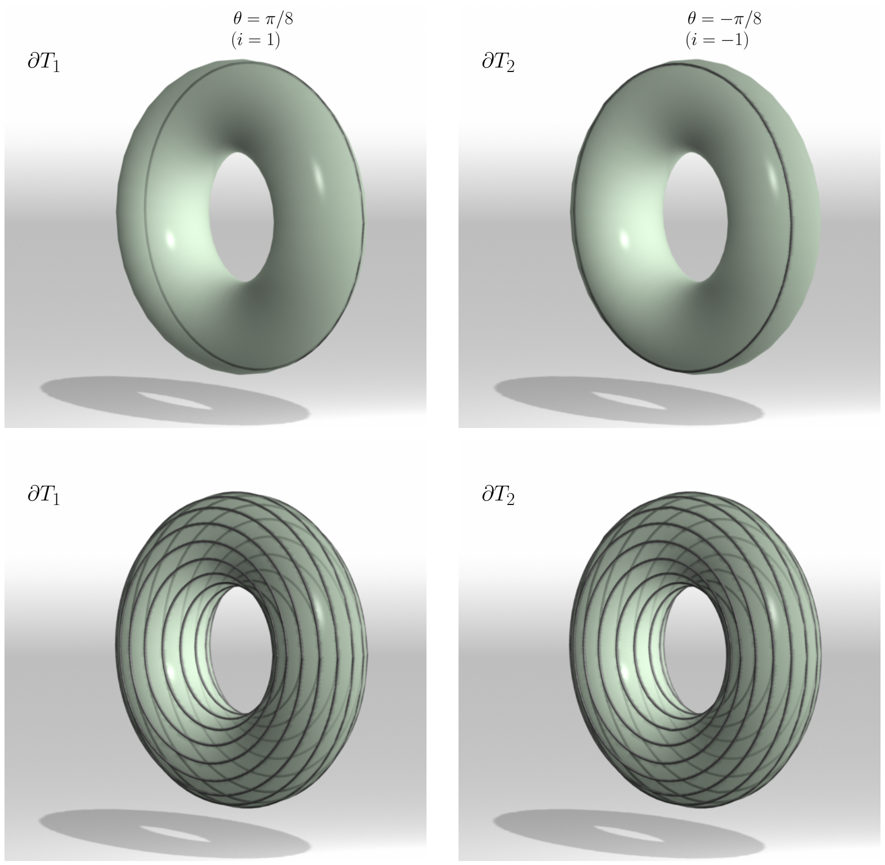


Figure 6.5: Fiber cycles on ∂T_1 are mapped to fiber cycles on ∂T_2.

According to lemma 6.4.1, 6.4.2 and 6.4.3, we claim that a gluing map induces a bijection of gluable triangulations for ∂T_1 and ∂T_2.

Theorem 6.4.1. *A gluing map Ψ induces a bijection between the gluable triangulation for ∂T_1 and the gluable triangulation for ∂T_2.*

In addition, from lemma 6.4.2 and its proof, we can get a corollary regarding the mapping among cycles of different types.

Corollary 6.4.1. *A gluing map Ψ induces a bijection between cycles on ∂T_1 and cycles on ∂T_2. In particular:*

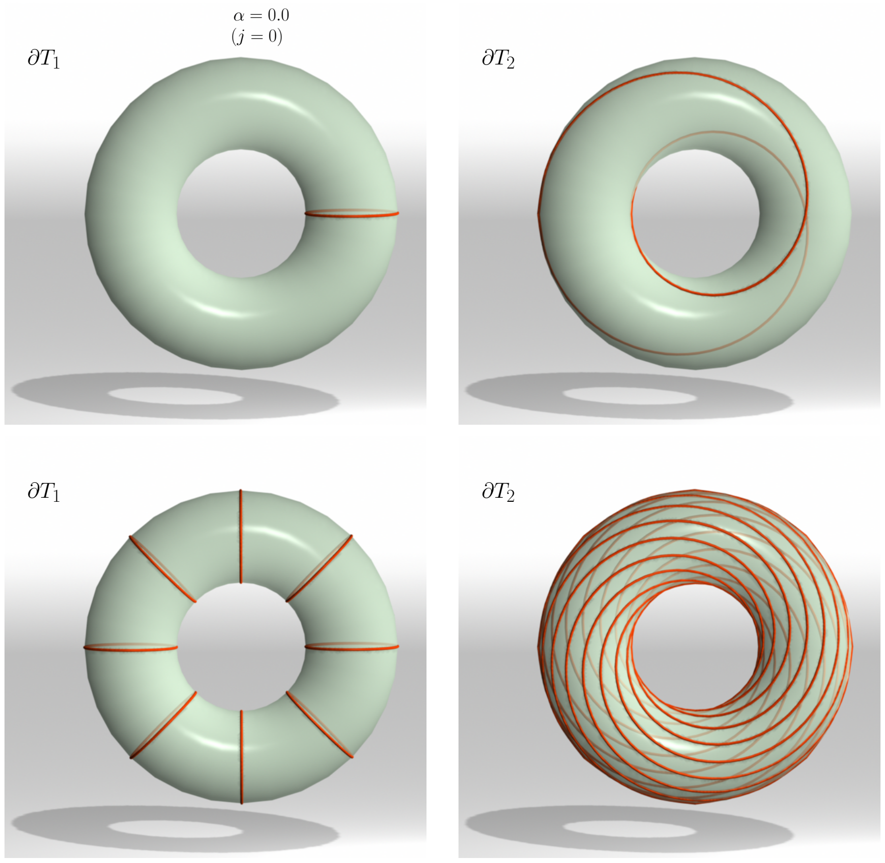


Figure 6.6: Base cycles on ∂T_1 are mapped to diagonal cycles on ∂T_2.

- *Any fiber cycle on ∂T_1 is mapped to a unique fiber cycle on ∂T_2, and vice versa;*
- *Any base cycle on ∂T_1 is mapped to a unique diagonal cycle on ∂T_2, and vice versa;*
- *Any diagonal cycle on ∂T_1 is mapped to a unique base cycle on ∂T_2, and vice versa;*

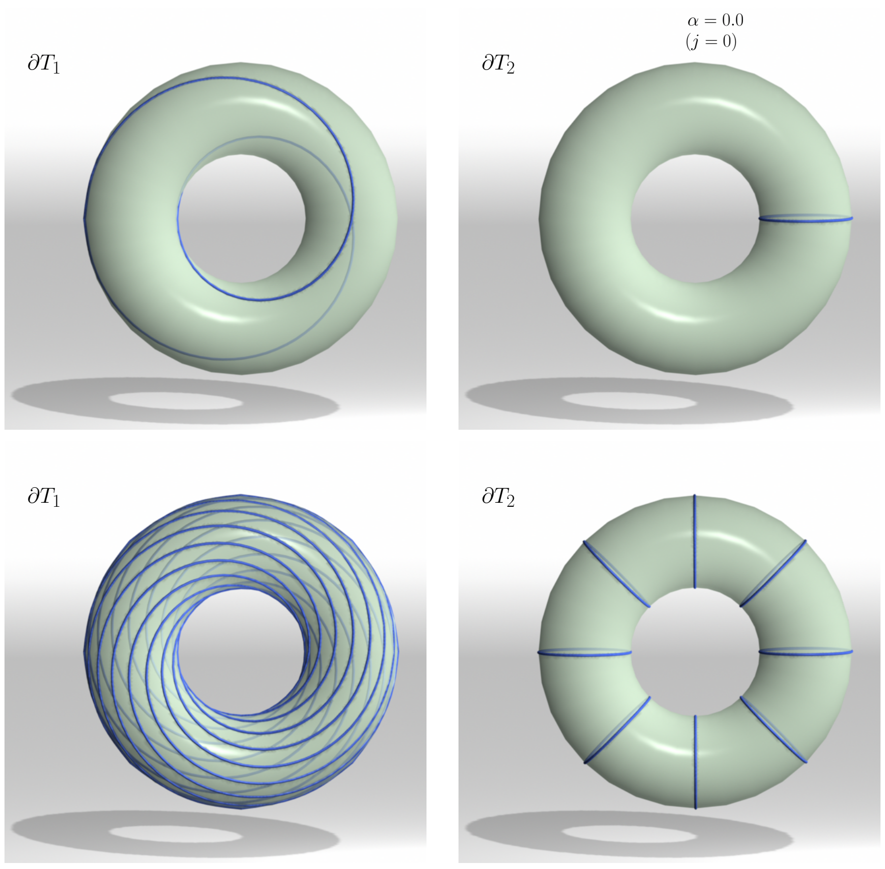


Figure 6.7: Diagonal cycles on ∂T_1 are mapped to base cycles on ∂T_2.

6.4.2 Continuity

In the previous section we show that a gluing map Ψ induces bijections of vertices, edges and triangles between ∂T_1 and ∂T_2. Actually Ψ has an even nicer property, it is continuous on the triangulations over each band. Before getting to a theorem about this, we need some definitions.

Definition 6.4.2 (One-Ring Neighborhoods for Vertices). *Given a triangulation Δ and a vertex $v \in \Delta$, the one-ring neighborhood of v is a sequence of triangles*

$$\Theta(v) = [f_1, f_2, \cdots, f_t]$$

such that

- *$v \in f_i$ for $1 \leqslant i \leqslant t$;*
- *There is a common edge shared by f_i and f_{i+1} for $1 \leqslant i < t$, and by f_t and f_1 if v is not on the boundary;*
- *All the f_i's are ordered counter-clockwisely.*

We call t the face-valence of v.

Definition 6.4.3 (Continuous Map for Triangulations). *Given two triangulated domain Δ_1 and Δ_2, a map $\Phi : \Delta_1 \to \Delta_2$, and a vertex $v \in \Delta_1$; denote the one-ring neighborhood of v as*

$$\Theta(v) = [f_1, f_2, \cdots, f_t]$$

denote the image vertex as $v' = \Phi(v)$ and its one-ring neighborhood as

$$\Theta(v')) = [f'_1, f'_2, \cdots, f'_t]$$

where

$$f_i' = \Phi(f_i)$$

The given map Φ *is continuous if and only if for every* $v \in \Delta_1$*,*

$$\Phi(f_i) = f_i' \ (1 \leqslant i \leqslant t)$$

Φ *is reversely continuous if and only if for every* $v \in \Delta_1$

$$\Phi(f_i) = (f_{t-i}')^{-1} \ (1 \leqslant i \leqslant t)$$

where $(f_i')^{-1}$ *denotes* f_i' *with orientation reversed.*

In another word, a map between two triangulations are continuous or reversely continuous if and only if it keeps one-ring neighborhoods invariant for all the vertices with orientation preserved or reversed. And as proved in the following theorem, a gluing map in definition 6.3.1 is a continuous map with orientation reversed.

Theorem 6.4.4. *Gluing map* Ψ *and its inverse* Ψ^{-1} *are both reversely continuous.*

Proof. It suffices to show that both Ψ and Ψ^{-1} keeps one-ring neighborhoods invariant with orientation reversed.

For any vertex $v = (2, i, j)$ in ∂T_2, its one-ring neighborhood has six triangles

$$\Theta(v) = [f_1, f_2, f_3, f_4, f_5, f_6]$$

where

$$f_1 = [(2, i, j), (2, i, j-1), (2, i+1, j)]$$

$$f_2 = [(2, i, j), (2, i+1, j), (2, i+1, j+1)]$$

$$f_3 = [(2,i,j),(2,i+1,j+1),(2,i,j+1)]$$

$$f_4 = [(2,i,j),(2,i,j+1),(2,i-1,j)]$$

$$f_5 = [(2,i,j),(2,i-1,j),(2,i-1,j-1)]$$

$$f_6 = [(2,i,j),(2,i-1,j-1),(2,i,j-1)]$$

Under Ψ they are mapped to

$$\Psi(f_1) = [(2,i',j'),(2,i',j'-1),(2,i'-1,j'-1)]$$

$$\Psi(f_2) = [(2,i',j'),(2,i'-1,j'-1),(2,i'-1,j')]$$

$$\Psi(f_3) = [(2,i',j'),(2,i'-1,j'),(2,i',j'+1)]$$

$$\Psi(f_4) = [(2,i',j'),(2,i',j'+1),(2,i'+1,j'+1)]$$

$$\Psi(f_5) = [(2,i',j'),(2,i'+1,j'+1),(2,i'+1,j')]$$

$$\Psi(f_6) = [(2,i',j'),(2,i'+1,j'),(2,i',j'-1)]$$

where

$$i' = -i\,, \quad j' = j - i + N$$

For the image vertex $v' = \Psi(v) = (1,i',j')$, it also has a one-ring neighborhood of six triangles

$$\Theta(v') = [f'_1, f'_2, f'_3, f'_4, f'_5, f'_6]$$

where

$$f'_1 = [(2,i',j'),(2,i',j'-1),(2,i'+1,j')]$$

$$f'_2 = [(2,i',j'),(2,i'+1,j'),(2,i'+1,j'+1)]$$

$$f'_3 = [(2,i',j'),(2,i'+1,j'+1),(2,i',j'+1)]$$

$$f'_4 = [(2,i',j'),(2,i',j'+1),(2,i'-1,j')]$$

$$f'_5 = [(2,i',j'),(2,i'-1,j'),(2,i'-1,j'-1)]$$

$$f'_6 = [(2,i',j'),(2,i'-1,j'-1),(2,i',j'-1)]$$

Obviously

$$\Psi(f_1) = (f'_6)^{-1}$$

$$\Psi(f_2) = (f'_5)^{-1}$$

$$\Psi(f_3) = (f'_4)^{-1}$$

$$\Psi(f_4) = (f'_3)^{-1}$$

$$\Psi(f_5) = (f'_2)^{-1}$$

$$\Psi(f_6) = (f'_1)^{-1}$$

which means that Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$. Therefore Ψ is reversely continuous from ∂T_2 to ∂T_1.

By a similar analysis on Ψ^{-1} from ∂T_1 to ∂T_2, it can be shown that Ψ^{-1} is also reversely continuous. □

6.4.3 Fiber Twisting

Corollary 6.4.1 describes the behavior of gluing map Ψ on different types of cycles. Here we give further investigations on its behavior on fiber cycles, which will give rise to insights about how the global bundle is twisted under gluing map Ψ.

On each fiber cycle, there is a unique vertex with zero polar angle (i.e. $\alpha = 0$). We call this vertex the *zero point* of this fiber cycle. According to corollary 6.4.1, every fiber cycle on ∂T_2 is mapped to a unique fiber cycle on ∂T_1. In our construction, it turns out that their zero points are not identified. Instead, a fiber cycle from ∂T_2 is twisted by certain amount to meet the image cycle on

∂T_1. Actually how their zero points are mapped tells how a band is twisted. The following theorem states that such twisting satisfies the requirements from proposition 6.1.1.

Theorem 6.4.5. *A gluing map Ψ induces a 4π twist of fiber cycles across ∂T_2 and ∂T_1.*

Proof. Consider the sequence of fiber cycles on ∂T_2:

$$\left[\gamma_0^F(2),\ \gamma_1^F(2),\ \cdots\ \gamma_{4N-1}^F(2),\ \gamma_{4N}^F(2)\right]$$

where the last one is actually the same fiber as the first one under modulation of $4N$. According to corollary 6.4.1 they are mapped to a sequence of fiber cycles on ∂T_1:

$$\left[\gamma_{4N}^F(1),\ \gamma_{4N-1}^F(1),\ \cdots\ \gamma_1^F(1),\ \gamma_0^F(1)\right]$$

Note that the zero points of fiber cycles on ∂T_1 are:

$$[(1,4N,0),\ (1,4N-1,0),\ \cdots,\ (1,1,0),\ (1,0,0)]$$

Now consider the corresponding pre-image vertices on ∂T_2. Under gluing map

$$\Psi((2,i,j)) = (1,-i,j-i+N)$$

and considering the modulation of $4N$ and $2N$ for i and j respectively, the zero points of fiber cycles on ∂T_1 are mapped to:

$$[(2,0,0),\ (2,1,1,\ \cdots,\ (2,4N-1,4N-1),\ (2,4N,4N)]$$

That is, using grid coordinates, for $i = 0,\ 1,\ \cdots,\ 4N-1, 4N$, fiber $\gamma_i^F(2)$ needs to be rotated (in terms of j) by $0,\ 1,\ \cdots,\ 4N-1, 4N$ to meet the zero point on the image fiber $\Psi\left(\gamma_i^F(2)\right)$. Translating to the angle coordinates (equation 6.10), an increment of $4N$ in terms of i is equivalent to a rotation of 2π, while an increment of $4N$ in terms of j is equivalent to a rotation of 4π. That means traversing ∂D_2 once will incur a fiber twisting of 4π. By a similar analysis on the inverse map Ψ^{-1}, one will reach a symmetric result that traversing ∂D_1 once will also incur a fiber twisting of 4π. □

6.4.4 Final Remarks

In this chapter, we present a local-to-global framework to build unit tangent bundles for a S^2 surface. We first partition the surface into two covering disks D_1 and D_2 that intersecting only in their boundaries. Then build two trivial local bundles T_1 and T_2 with gluable triangulations on their boundary, and gluing T_1 and T_2 along their boundary by a gluing map Ψ. We also prove some claims that characterize this construction, they can be summarized in two conclusions as below.

First, gluing map Ψ glues local bundles T_1 and T_2 into a valid tetrahedral mesh UT that has no boundary. With a gluable triangulation (definition 6.2.1) defined on ∂T_1 and ∂T_2, the gluing map Ψ (definition 6.3.1) between them is both bijective (theorem 6.4.1) and reversely continuous (theorem 6.4.4). That means the two local bundles are glued to each other along the boundary seamlessly and the result is a valid tetrahedral mesh without any boundary.

Second, the tetrahedral mesh UT faithfully represents a unit tangent bundle over a given S^2 surface. Under gluing map Ψ the fibers along ∂D_2 are twisted (theorem 6.4.5) and then identified (corollary 6.4.1) with fibers along ∂D_1. The twisting conforms to the transition function (equation 6.2) between two covering disks and to the jacobian (equation 6.4) of the transition function, and the total twisting in the global bundle UT amounts to 4π (theorem 6.4.5). Thus the final tetrahedral mesh UT is a valid representation of unit tangent bundle over the original surface.

In summary, we claim that our construction will generate valid discrete representations of unit tangent bundles over S^2 surfaces.

Theorem 6.4.6. *Given a S^2 surface M, the local-to-global construction generates a valid tetrahedral mesh that faithfully represents the unit tangent bundle $UT(M)$.*

Chapter 7

Discrete Unit Tangent Bundles for $g > 0$ Closed Surfaces

This chapter is devoted to discrete unit tangent bundles for $g > 0$ closed and orientable surfaces, which are surfaces with g handles T_g^2. It covers $g = 1$ surfaces whose unit tangent bundle is trivial, and $g > 1$ surfaces whose unit tangent bundle is non-trivial, they share the same philosophy of construction. We start from the intuitions behind our construction (section 7.1), then present the local construction (section 7.2) and global construction (section 7.3) respectively, and finally give validations of our methods (section 7.4).

7.1 Intuitions

For discrete unit tangent bundles of $g > 0$ surfaces, we follow a *local-to-global* philosophy that is similar to that of the $g = 0$ case (section 6). Namely, we first construct some local bundles that are trivial, then glue them into a global bundle that could be non-trivial. The motivation to keep following such a philosophy is multi-folded.

- According to our construction of discrete unit tangent bundles for disks D^2 (section 5.1), the local bundle of $T^2(g)$ have regular inner structures; by gluing a set of local bundles into a global bundle of $T^2(g)$, most part of the regularity in local bundles will be preserved in the global bundle as well.

- Based on the algorithms in section 5, each local bundle over D^2 is guaranteed to have a nice tessellation with tetrahedra, which can also be transferred into the global bundle of $T^2(g)$ with careful design of boundary triangulations.

- The unit tangent bundle of $T^2(g)$ is a closed 3-manifold that cannot be embedded in R^3 and is therefore hard to visualize. A discrete local bundle over D^2, however, can be easily embedded in R^3 and therefore provides an indirect way to visualize the internal mesh structure of the global bundle of $T^2(g)$.

Despite the similarity to the $g = 0$ case, the construction in this chapter has prominent difference in several aspects, such as the way to partition the surface to disks, and the way to enforce twisting into a non-trivial global bundle.

7.1.1 Construct Covering Disks

The first step to build local bundles for a $T^2(g)$ surface is to find a set of topological disks covering the whole surface. In fact, any closed and oriented surface with one or more handles can be naturally covered by one disk. We will explain this in below with some basic knowledge of surface topology.

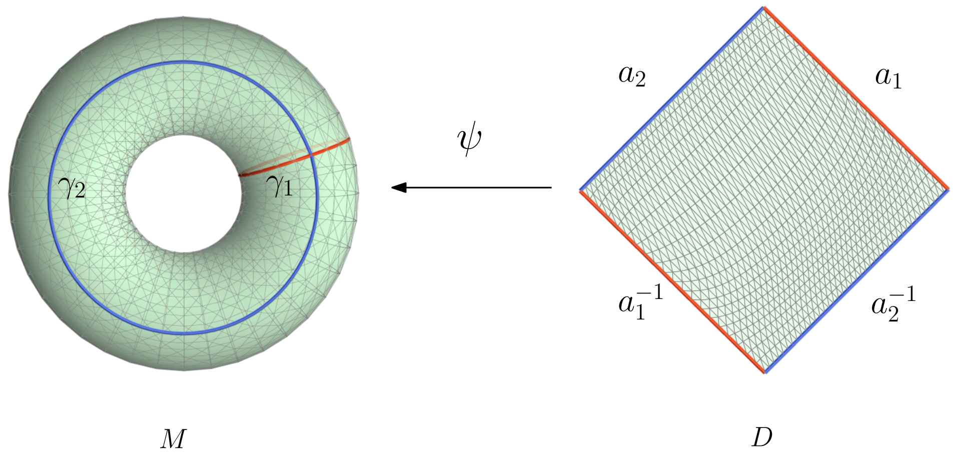


Figure 7.1: A $g = 1$ surface M can be covered by a single disk D under covering map ψ. The boundary of the disk ∂D consists of 4 segments that are identified in pairs under map ψ.

On a closed and oriented surface M of genus g, we can always find $2g$ loops, $\{\gamma_1, \gamma_2, \cdots, \gamma_{2g}\}$, such that none of them can be deformed to one another smoothly without leaving the surface. For example, figure 7.1 shows 2 such loops on a genus 1 surface, figure 7.2 shows 4 such loops on a genus 2 surface. These loops represent the generators of the first *homology group* of the given surface. When the given surface is not knotted, these loops can also be named *handle loops* and *tunnel loops* respectively (as in [16]). In this work, we call them *cutting loops*.

Cutting loop a_i and b_i intersect with each other at somewhere on the surface. Without loss of generality, we can always assume that all the loops

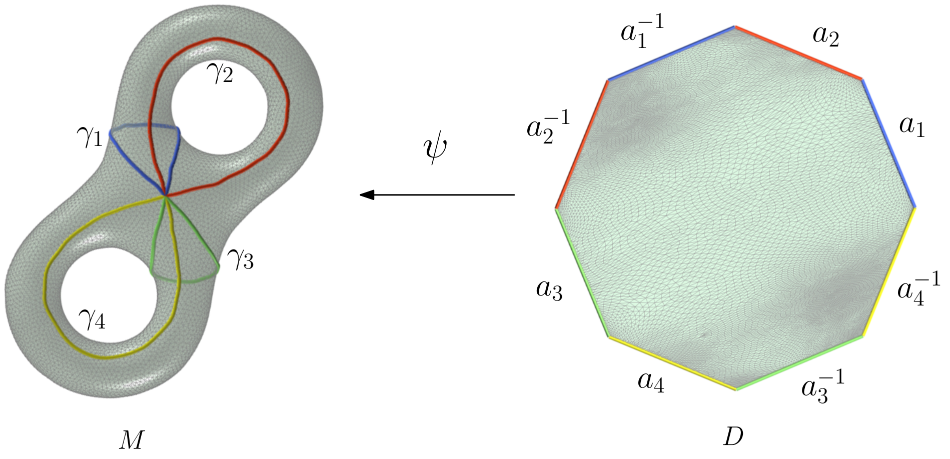


Figure 7.2: A $g = 2$ surface M can be covered by a single disk D under covering map ψ. The boundary of the disk ∂D consists of 8 segments that are identified in pairs under map ψ.

intersect at a single common point; in case two loops having no intersection or more than one intersections, they can always be perturbed on the surface so that they meet one another at only one common point. This common point is called the *base point*. A set of cutting loops sharing a single base point are called *canonical cutting loops*.

By slicing a surface M open along a set of canonical cutting loops, we can get a topological disk D (see figure 7.1 and 7.2). Each cutting loop γ_k on M is split into two boundary curves a_k and a_k^{-1} in D, and the base point p of M is split into $4g$ copies in D. Namely, D can be viewed as a $4g$-sided polygonal region, whose boundary ∂D consists of $4g$ segments

$$\partial D = c_1 c_2 \cdots c_{4g} \tag{7.1}$$

where each c_i $(1 \leqslant i \leqslant 4g)$ represents one of the boundary segments $\{a_j, a_j^{-1} \,|\, 1 \leqslant j \leqslant 2g\}$ in D and therefore corresponds to a cutting loop γ_k $(1 \leqslant k \leqslant 2g)$in M. In particular, by appropriate choice of base point and cutting loops, ∂D can be expressed in a canonical form

$$\partial D = a_1 a_2 a_1^{-1} a_2^{-1} \cdots a_{2g-1} a_{2g} a_{2g-1}^{-1} b_{2g}^{-1} \tag{7.2}$$

In algebraic topology, such a polygonal disk can be used as a representation of the so called *fundamental domain*, and can serve as one period in the *universal covering space*, which is a simply connected surface that covers M by infinitely many periods. In computational topology, this disk is called a *(canonical) polygonal schema*. In this work, we name it a *covering disk*, which covers the original surface M through a *covering map*

$$\psi : D \to M$$

which is a surjective map such that (for $1 \leqslant j \leqslant 2g$ and $1 \leqslant i \leqslant 4g$)

$$\psi(a_j) = \psi(a_j^{-1}) = \gamma_j$$

$$\psi(p_i) = p$$

where $p_i \in D$ is a common point between two consecutive segments

$$p_i = c_i \cap c_{i+1}$$

Also, we can define an *identifying map*

$$\varphi : \partial D \to \partial D$$

which is a bijective self-map that identifies boundary segments in D in pairs ($1 \leqslant j \leqslant 2g$)

$$\varphi(a_j) = a_j^{-1}, \varphi(a_j^{-1}) = a_j$$

In our construction of discrete unit tangent bundles, we use such a covering disk D to build a local bundle.

7.1.2 Enforce Twisting

A unit tangent bundle for a compact surface is in general non-trivial and may have certain amount of twisting. As a well known result in algebraic topology, the total twisting in a unit tangent bundle should be

$$\text{total twisting} = 2\pi\chi = 2\pi(2-2g)$$

where χ is the Euler characteristic of the base surface, which equals $2-2g$ if the base surface has genus g. It implies that, for example, for genus 0, 1 and 2 closed surfaces the total twisting in their unit tangent bundles should be 4π, 0 and -4π respectively.

In our discrete construction of unit tangent bundles, a key challenge is how to enforce the right amount of twisting into the final tetrahedral mesh. For genus 0 closed surface (i.e. S^2), as we have seen in chapter 6, the total twisting is enforced along a cutting loop that partitions the sphere into two covering disks D_1 and D_2. Recall that two local bundles T_1 and T_2 are built over those two disks, and they are glued into a global bundle along their boundary. Then the twisting is directly reflected by the gluing map between the boundaries ∂T_1 and ∂T_2 of local bundles.

For $g > 0$ surfaces, we hope to repeat the process for $g = 0$ surfaces by enforcing the twisting along cutting loops. Unlike the sphere case, now we have $2g$ cutting loops on a genus g surface, while having $2\pi(2-2g)$ twisting to be enforced. In this work, we distribute the twisting among $2g-2$ cutting loops with -2π per loop, while leaving the remaining 2 cutting loops free of twisting. As a special case, for $g = 1$ surfaces, there are no twisting in the unit tangent bundle, and the only two cutting loops are both free of twisting.

7.2 Local Construction

In the stage of local construction, we are given a covering disk D for a genus g ($g > 0$) surface M, and need to build a tetrahedral mesh for the trivial local bundle $T = D \times S^1$. First of all, we design a special grid structure (section 7.2.1) on the boundary surface ∂T of the local bundle, which allows us to assign meaningful coordinates (section 7.2.2) to the vertices in ∂T. Then we design an appropriate gluable triangulation (section 7.2.3) on ∂T so that a tetrahedral mesh can be built (section 7.2.4) from this boundary triangulation using algorithms from chapter 5. Note that the construction in this stage meets the requirements for a unit tangent bundle of M and is consistent with the global construction in a later stage.

7.2.1 Grid Structure

A local bundle T over a covering disk D is a solid torus, its boundary ∂T is a genus one closed surface. In our construction, ∂T needs to be triangulated appropriately, and it turns out that the vertices in this triangulation should have certain grid structure.

Note that ∂T is a direct product $\partial T = \partial D \times S^1$, the grid structure could naturally be a tensor product of discretized ∂D and discretized S^1, both of which are piece-wise linear approximations for a one-dimensional loop. Further, since ∂D consists a set of segments c_k, the discretization of ∂D is a union of the discretization of each segment c_k. As a tensor product, the grid structure can therefore be described by the *discretizing resolution*, i.e. amount of edges, along fiber S^1 and that along each c_k.

Formally, for the boundary surface ∂T of a local bundle T for a genus g ($g > 0$) surface, we can define a general grid structure $\Gamma(\partial T)$ as a $(1+4g)$-tuple of integers

$$\Gamma(\partial T) = \left[M_f, M_{c_1}, M_{c_2}, \cdots, M_{c_{4g}}\right] \tag{7.3}$$

where M_f specifies the number of edges along each fiber S^1, M_{c_k} represents the discretizing resolution of the k'th segment in ∂D ($1 \leqslant k \leqslant 4g$). Note that each c_k is a curve segment bounded at both ends, the number of vertices along c_k is therefore $M_{c_k}+1$, and every pair of consecutive segments share a common vertex.

The last $4g$ integers can not be assigned arbitrarily, they are subject to certain restrictions.

First, as explained in section 7.1.1, boundary segment a_k should be identified with a_k^{-1}. Consequently for $1 \leqslant k \leqslant 2g$, we have

$$M_{a_k} = M_{a_k^{-1}}$$

Thus the representation for a grid structure in 7.3 can be simplified to

$1 + 2g$ integers:

$$\Gamma(\partial T) = [M_f, M_1, M_2, \cdots, M_{2g}] \tag{7.4}$$

where M_k represents the discretizing resolution for both a_k and a_k^{-1}.

Secondly, our discrete construction of $UT\left(T^2(g)\right)$ imposes a further requirement. As elaborated in section 7.1.2, among all the $2g$ cutting loops on the original surface M, only two of them are free of twisting, while any other loop will bear a twisting of -2π. Without loss of generality, assume γ_1 and γ_2 are the two twisting-free cutting loops. Then as further explained in later discussions (section 7.2.3, 7.3.1), we require that for $3 \leqslant k \leqslant 2g$:

$$M_k = M_f$$

Therefore the grid structure that we actually use in later constructions can be defined by only three integers:

$$\Gamma(\partial T) = [N, N_1, N_2] \tag{7.5}$$

where

$$N = M_f = M_k \, (3 \leqslant k \leqslant 2g)$$

$$N_1 = M_1$$

$$N_2 = M_2$$

Namely, N denotes the discretizing resolution of the fiber S^1, as well as that of the $4g - 4$ twisting-bearing boundary segments; N_1 and N_2 denote the discretizing resolution of those two pairs of twisting-free boundary segments. These three integers are independent to one another and can be chosen individually.

7.2.2 Vertex Coordinates

In any grid structure defined generally in 7.3, every vertex in it can be assigned with a coordinate, either locally or globally.

One local coordinate system can be built along each fiber, where every vertex can be indexed by an integer j $(0 \leqslant j < M_f)$. Note that since each fiber is a closed loop, any addition and substraction for this coordinate is modulated by M_f, so that the value is always in the range of $[0, M_f - 1]$. For example, $j = M_f$ is equivalent to $j = 0$, $j = -1 - M_f$ is equivalent to $j = M_f - 1$.

Another local coordinate system can be built along each boundary segment $c_k \subset \partial D$, where every vertex can be indexed by an integer i $(0 \leqslant i \leqslant M_{c_k})$. Recall that each c_k is a curve segment rather than a loop, and there are M_{c_k} edges along c_k; therefore, there should be $M_{c_k} + 1$ distinguished vertices along c_k, and the end vertex $i = 0$ and $i = M_{c_k}$ should be identified with some end vertex of adjacent boundary segment c_{k-1} and c_{k+1}.

If we enlarge the scope to the whole grid structure $\Gamma(\partial T)$, every vertex can be assigned with a global coordinate

$$[c_k, i, j]$$

where c_k could be any boundary segment among $\{a_l, a_l^{-1} \,|\, 1 \leqslant l \leqslant 2g\}$. This coordinate represents the j'th vertex on a fiber over the i'th vertex of the boundary segment $c_k \subset \partial D$. In another word, if the vertex is projected to boundary segment $c_k \subset \partial D$, it has local coordinate i; projected to a fiber, it has local coordinate j.

7.2.3 Gluable Triangulation

Once a grid structure is defined on ∂T, we can build a triangulation over the vertices in the grid structure. Here we define a special triangulation $\Delta(\partial T)$ such that the local bundle T can be turned into a global bundle later. We call it a *gluable triangulation.*

Definition 7.2.1 (Gluable Triangulation for $UT\left(T^2(g)\right)$)**.** *Given a trivial local bundle $T = D \times S^1$, where D is the covering disk of a $T^2(g)$ surface ($g > 0$), a triangulation of its boundary $\partial T = \partial D \times S^1$ is called a gluable triangulation $\Delta(\partial T)$ if :*

[1] The vertices of the triangulation form a grid structure on ∂T with the following form:

$$\Gamma(\partial T) = [N, N_1, N_2]$$

for some positive integers N, N_1 and N_2. Namely,

- *Every fiber S^1 consists of N edges;*
- *Every boundary segment $c \in \{a_1, a_1^{-1}\}$ consists of N_1 edges;*
- *Every boundary segment $c \in \{a_2, a_2^{-1}\}$ consists of N_2 edges;*
- *Every boundary segment $c \in \{a_k, a_k^{-1} \,|\, 3 \leqslant k \leqslant 2g\}$ consists of N edges.*

[2] The triangles in each band $c \times S^1 \subset \partial D$ are defined by all possible combinations of vertices in the following forms:

- *For $c \in \{a_1, a_2\}$:*

$$[(c, i, j), (c, i+1, j), (c, i+1, j+1)]$$

$$[(c, i, j), (c, i+1, j+1), (c, i, j+1)]$$

- *For $c \in \{a_1^{-1}, a_2^{-1}\}$:*

$$[(c, i, j+1), (c, i, j), (c, i+1, j)]$$

$$[(c, i, j+1), (c, i+1, j), (c, i+1, j+1)]$$

- *For $c \in \{a_k, a_k^{-1} \,|\, 3 \leqslant k \leqslant 2g\}$:*

$$[(c, i, j+1), (c, i, j), (c, i+1, j)]$$

$$[(c,i,j+1),(c,i+1,j),(c,i+1,j+1)]$$

In the part of defining triangles, the first two cases are for untwisted bands, as shown in figure 7.5, while the third case is for twisted bands, as shown in figure 7.6. In both of these figures, a band is cut off along a base path and flattened into a 2D domain for visualization purposes. Moreover, a 3D view of gluable triangulations is shown in figure 7.3 and 7.4 for $g = 1$ and $g = 2$ base surfaces respectively.

By definition, the edges in a gluable triangulation can be classified as

- Fiber edge: (for $0 \leqslant i \leqslant M_c$, $0 \leqslant j < M_f$)

$$[(c,i,j)\,,(c,i,j+1)]$$

- Base edge: (for $0 \leqslant i < M_c$, $0 \leqslant j < M_f$)

$$[(c,i,j)\,,(c,i+1,j)]$$

- Diagonal edge: (for $0 \leqslant i < M_c$, $0 \leqslant j < M_f$)

$$[(c,i,j)\,,(c,i+1,j+1)]$$

- Anti-diagonal edge: (for $0 \leqslant i < M_c$, $0 \leqslant j < M_f$)

$$[(c,i,j)\,,(c,i+1,j-1)]$$

Moreover, the edges of the same type will connect into paths (or loops). And all the edges within each band $c \times S^1 \subset \partial T$ can be grouped into paths in one of the following types:

- Fiber paths $\{\gamma_i^F(c)\,|\,0 \leqslant i \leqslant M_c\}$:

$$\gamma_i^F(c) = [(c,i,0)\,,(c,i,1)\,,\cdots,(c,i,M_f-1)\,,(c,i,0)]$$

- Base paths $\{\gamma_j^B(c) \,|\, 0 \leqslant j < M_f\}$:

$$\gamma_j^B(c) = [(c, 0, j), (c, 1, j), \cdots, (c, M_c, j)]$$

- Diagonal paths $\{\gamma_j^D(c) \,|\, 0 \leqslant j \leqslant M = \min\{M_f, M_c\}\}$:

$$\gamma_j^D(c) = [(c, 0, j), (c, 1, j+1), \cdots, (c, M, j+M)]$$

- Anti-diagonal paths $\{\gamma_j^A(c) \,|\, 0 \leqslant j \leqslant M = \min\{M_f, M_c\}\}$:

$$\gamma_j^A(c) = [(c, 0, j), (c, 1, j-1), \cdots, (c, M, j-M)]$$

where c denotes a boundary segment in $\{a_k, a_k^{-1} \,|\, 1 \leqslant k \leqslant 2g\}$, M_c denotes the discretizing resolution of c, M_f denotes the discretizing resolution of a fiber, and (c, i, j) is the global coordinate of a vertex, where the value of j is always normalized to $[0, M_f - 1]$. For example, figure 7.5 illustrates different types of paths on a pair of untwisted bands; the fiber paths, base paths, and diagonal (or anti-diagonal) paths on $a_k \times S^1$ (or $a_k^{-1} \times S^1$)are drawn in black, red and blue respectively. Similarly, figure 7.6 shows these paths on a pair of twisted bands.

On $\partial T = \partial D \times S^1$, every path can be projected onto either ∂D or S^1, or both. Each fiber path $\gamma_i^F(c)$ can be projected onto a fiber S^1; actually it is a loop by itself that represents a fiber. Each base path $\gamma_j^B(c)$ can be projected to boundary segment c, and the later will be mapped to a cutting loop on the original surface. Each diagonal path $\gamma_j^D(c)$ or anti-diagonal path $\gamma_j^A(c)$ actually covers both directions; projected onto fiber S^1, it is mapped to a fiber path $\gamma_0^F(c)$; projected onto ∂D, it is mapped to a base path $\gamma_0^B(c)$.

7.2.4 Local Bundle

Once a gluable triangulation is defined on the boundary ∂T, we are ready to triangulate the whole local bundle T. Note that the local bundle is a direct product both in itself $T = D \times S^1$ and on the boundary $\partial T = \partial D \times S^1$, and

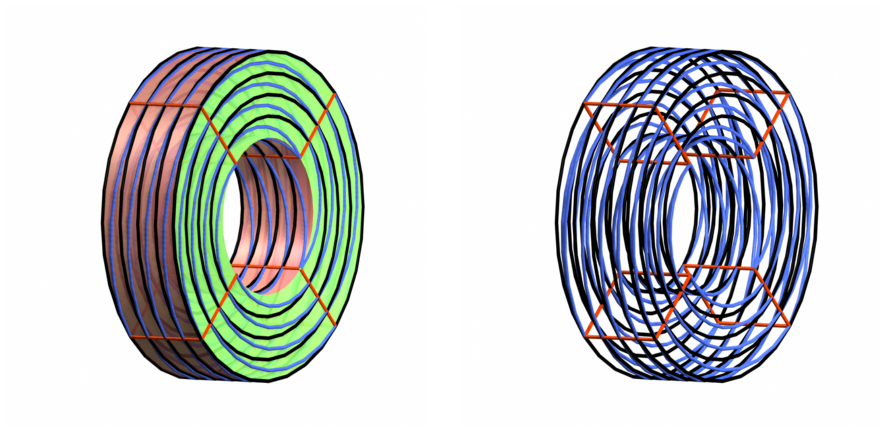


Figure 7.3: Gluable triangulation on the boundary of the local bundle for a $g = 1$ surface.

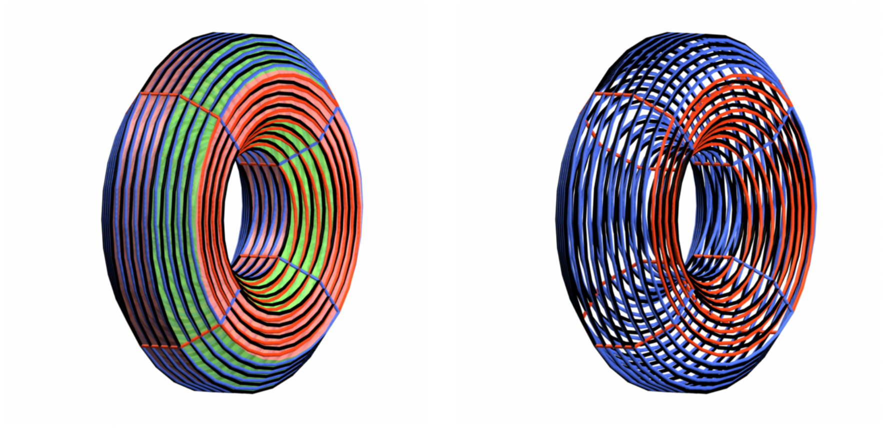


Figure 7.4: Gluable triangulation on the boundary of the local bundle for a $g = 2$ surface.

the gluable triangulation of ∂T also has a grid structure which is a tensor product. It meets all the requirements in definition 5.1.1 and can therefore be triangulated using algorithms in chapter 5.

By definition 7.2.1, each fiber has a discretizing resolution of N. Thus in a tetrahedral mesh for local bundle $T = D \times S^1$, the base D should have N pre-images (i.e. base copies) $\{D_j | 0 \leqslant j < N\}$, where each D_j is a triangular mesh consisting of all the vertices that have local coordinate j along fiber S^1. The whole tetrahedral mesh T is then partitioned by these base copies into N bundle segments $\{D_j \times I_j | 0 \leqslant j < N\}$, where I_j is a one-dimensional interval corresponding to an edge $(j, j+1)$ along a fiber.

In order to extend the boundary triangulation into the whole volume T, we can actually do this for each bundle segment $D_j \times I_j$ individually, where the problem is converted to the cutting pattern problem (definition 5.1.2) for base copy D_j. To do the conversion, we need to represent the boundary triangulation on each $\partial D_j \times I_j$ in the language of cutting patterns (section 5.1.2).

Definition 7.2.2 (Boundary Cutting Pattern for $UT\left(T^2(g)\right)$)**.** *Given a gluable triangulation for $\partial T = \partial D \times S^1$, where D is the covering disk of a $T^2(g)$ surface ($g > 0$), T is a trivial local bundle $T = D \times S^1$, let ∂D_j denote the j'th copy of ∂D in ∂T, which consists of $4g$ boundary segments $\{(a_j)_i \,|\, 1 \leqslant i \leqslant 4g\}$. The boundary cutting pattern is a function f that assigns every edge in each ∂D_j with an integer $+1$ or -1 in the following way:*

- *For every edge e along $c \in \{(a_j)_1, (a_j)_2\}$:*

$$f(e) = +1$$

- *For every edge e along $c \in \{(a_j)_1^{-1}, (a_j)_2^{-1}\}$:*

$$f(e) = -1$$

- *For every edge e along $c \in \{(a_j)_k, (a_j)_k^{-1} \,|\, 3 \leqslant k \leqslant 2g\}$:*

$$f(e) = -1$$

Using the above boundary cutting patterns as boundary constraints, we can compute cutting patterns for each base copy D_j using the algorithms defined in chapter 5, which will give us a tetrahedral tessellation for each bundle segment $D_j \times I_j$ and thus a tetrahedral mesh for the whole local bundle T.

7.3 Global Construction

In the local construction, we compute a tetrahedral mesh for local bundle $T = D \times S^1$. In this section we will do the global construction that turns the local bundle into a global bundle, and the later will be a discrete representation for the unit tangent bundle $UT\left(T^2(g)\right)$ for a genus g ($g > 0$) surface.

Here we are in a different situation to the $g = 0$ case (chapter 6). For a S^2 surface, there are two local bundles T_1 and T_2, and we glue them together along their boundaries. For a $T^2(g)$ surface, however, we only have one local bundle T. In order to construct a global bundle, this local bundle need to be glued to itself along its own boundary. In another word, we need a gluing map Ψ over ∂T.

In the following discussion, we divide all those $4g$ bands on ∂T into two groups

$$\partial T = (\partial T)^+ \cup (\partial T)^-$$

with each group containing $2g$ bands

$$(\partial T)^+ = \bigcup_{k=1}^{2g} (a_k \times S^1)$$

$$(\partial T)^{-} = \bigcup_{k=1}^{2g}(a_k^{-1} \times S^1)$$

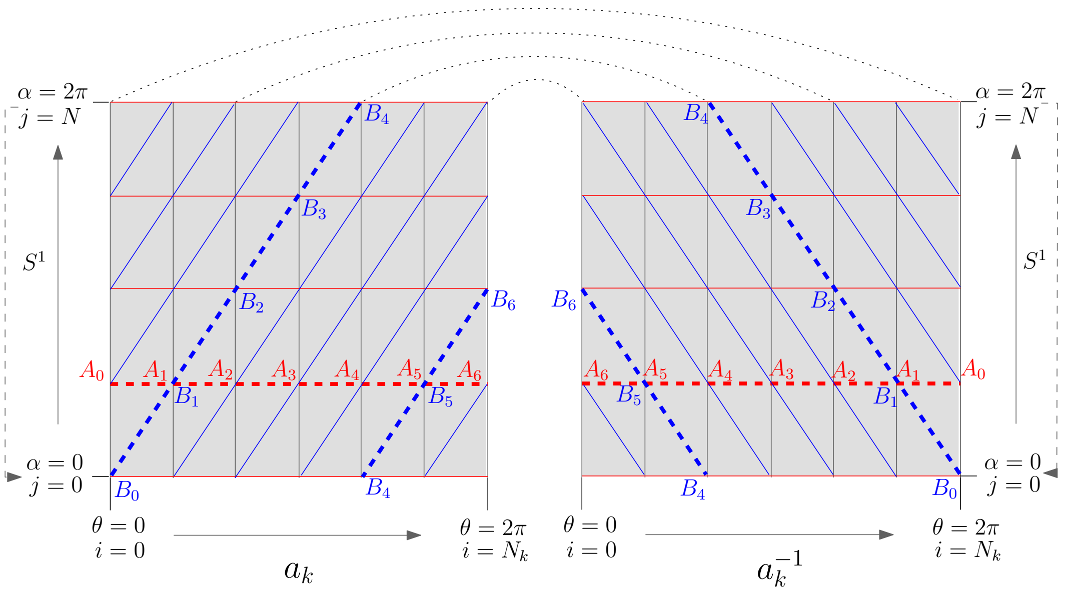


Figure 7.5: Gluable triangulation and gluing map for $UT(T^2(g))$ on a pair of untwisted boundary bands $a_k \times S^1$ and $a_k^{-1} \times S^1$ $(1 \leqslant k \leqslant 2)$. A path in grey, red or blue on the left side is mapped to a unique path in the same color on the right side. Two of such paths, $A_0A_1 \cdots A_6$ (in red) and $B_0B_1 \cdots B_6$ (in blue), are highlighted as examples.

7.3.1 Gluing Map

A gluing map over ∂T is partially determined by the identifying map φ over ∂D (section 7.1.1). Recall that φ identifies boundary segments a_i and a_i^{-1} with their orientation reversed. Consequently, the gluing map should identify boundary bands $a_i \times S^1$ and $a_i^{-1} \times S^1$ in an orientation-reversing manner. Namely, it is a map between $(\partial T)^{+}$ and $(\partial T)^{-}$.

But different to φ, the gluing map over ∂T may involve twisting. As stated in section 7.1.2, the total twisting should be $2\pi(2 - 2g)$. In this work, we design a gluing map Ψ in a special way so that the twisting is uniformly distributed to $2g - 2$ pairs of boundary segments around D.

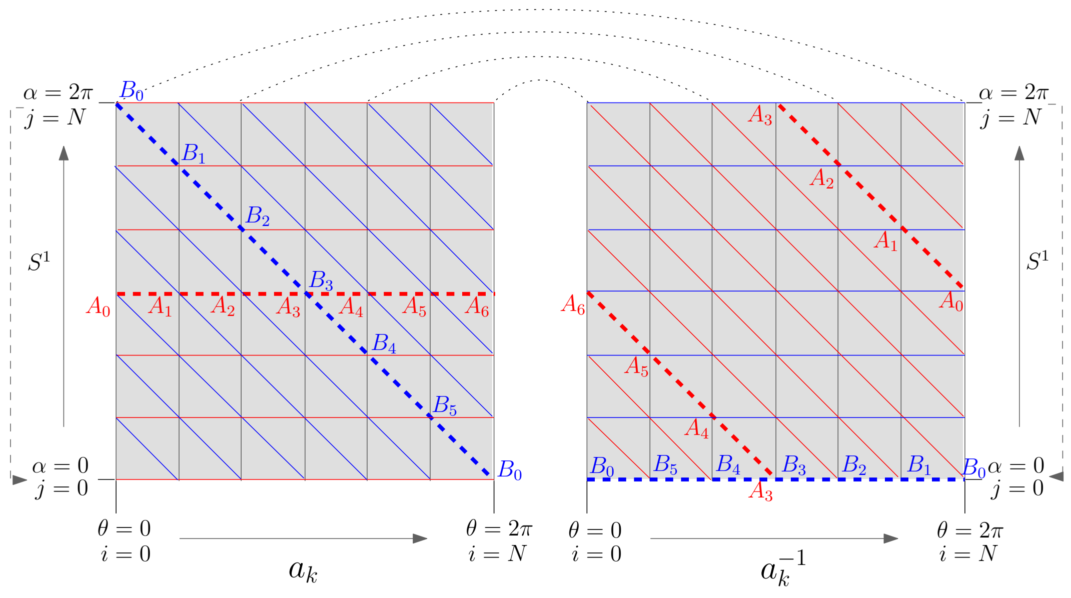


Figure 7.6: Gluable triangulation and gluing map for $UT\,(T^2(g))$ on a pair of twisted boundary bands $a_k \times S^1$ and $a_k^{-1} \times S^1$ ($3 \leqslant k \leqslant 2g$). A path in grey, red or blue on the left side is mapped to a unique path in the same color on the right side. Two of such paths, $A_0A_1\cdots A_6$ (in red) and $B_0B_1\cdots B_6$ (in blue), are highlighted as examples.

Definition 7.3.1 (Gluing Map for $UT\,(T^2(g))$)**.** *Given a local bundle $T = D \times S^1$ and a gluable triangulation with grid structure $[N, N_1, N_2]$ for $\partial T = \partial D \times S^1$, where D is a covering disk of a $T^2(g)$ surface ($g > 0$), a map*

$$\Psi : (\partial T)^+ \to (\partial T)^-$$

is called a gluing map if it acts on the vertices of the gluable triangulation as follows:

[1] For $1 \leqslant k \leqslant 2$, $0 \leqslant i \leqslant N_k$, $0 \leqslant j < N$:

$$\Psi\left((a_k, i, j)\right) = \left(a_k^{-1}, N_k - i, j\right),$$

[2] For $3 \leqslant k \leqslant 2g$, $0 \leqslant i \leqslant N$, $0 \leqslant j < N$:

$$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N - i, j + i)\,,$$

Under such a map, we can glue a local bundle to itself along its own boundary. As verified in the next section, such a construction will glue a local bundle nicely into a closed tetrahedral mesh, which is a faithful discrete representation of the unit tangent bundle over a given $T^2(g)$ surface.

7.4 Validations

In local construction, we define a gluable triangulation Δ on the boundary of a local bundle T; in global construction, we define a gluing map Ψ over ∂T. In this section, we will give several claims and proofs to validate that the above constructions will generate a desired discrete unit tangent bundle.

7.4.1 Bijection

In this part we will show that a gluing map induces a bijection of the gluable triangulation on bands in $(\partial T)^+$ and those in $(\partial T)^-$. We use three lemmas to show the bijection of vertices (lemma 7.4.1), edges (lemma 7.4.2) and triangles (lemma 7.4.3) respectively.

Lemma 7.4.1. *A gluing map Ψ induces a bijection between vertices in $a_k \times S^1$ and vertices in $a_k^{-1} \times S^1$ $(1 \leqslant k \leqslant 2g)$.*

Proof. It can be verified for untwisted bands $(1 \leqslant k \leqslant 2)$ and twisted bands $(3 \leqslant k \leqslant 2g)$ separately.

[1] For $1 \leqslant k \leqslant 2$, for any vertex $(a_k^{-1}, i', j') \in a_k^{-1} \times S^1$, suppose it has a pre-image, denoted as (a_k, i, j); By definition,

$$(a_k^{-1}, i', j') = (a_k^{-1}, N_k - i, j)$$

Solving this for i and j, we have

$$i = N_k - i'$$

$$j = j'$$

which gives a unique vertex in $a_k \times S^1$. And therefore the inverse map of $\Psi : a_k^{-1} \times S^1 \to a_k \times S^1$ can be defined as:

$$\Psi^{-1}\left((a_k^{-1}, i', j')\right) \to (a_k^{-1}, N_k - i', j')$$

[2] For $3 \leqslant k \leqslant 2g$, for any vertex $(a_k^{-1}, i', j') \in a_k^{-1} \times S^1$, suppose it has a pre-image, denoted as (a_k, i, j); By definition,

$$(a_k^{-1}, i', j') = (a_k^{-1}, N - i, j + i)$$

Solving this for i and j, we have

$$i = N - i'$$

$$j = j' - i = j' - N + i' = j' + i'$$

which gives a unique vertex in $a_k \times S^1$. And therefore the inverse map of $\Psi : a_k^{-1} \times S^1 \to a_k \times S^1$ can be defined as:

$$\Psi^{-1}\left((a_k^{-1}, i', j')\right) \to (a_k^{-1}, N_k - i', j' + i')$$

In both cases, for any vertex in $a_k^{-1} \times S^1$, there is a unique pre-image in $a_k \times S^1$, and the inverse map Ψ^{-1} can be represented as above. $\square$

Lemma 7.4.2. *A gluing map Ψ induces a bijection between edges in $a_k \times S^1$ and edges in $a_k^{-1} \times S^1$ ($1 \leqslant k \leqslant 2g$).*

Proof. It can be verified for untwisted bands ($1 \leqslant k \leqslant 2$) and twisted bands ($3 \leqslant k \leqslant 2g$) separately.

[1] $1 \leqslant k \leqslant 2$: We will show that any edge on untwisted band $a_k \times S^1$ is mapped to a unique edge on untwisted band $a_k^{-1} \times S^1$ under map Ψ.

- For any fiber edge $[(a_k, i, j), (a_k, i, j+1)]$ ($0 \leqslant i \leqslant N_k$, $0 \leqslant j < N$), the end vertices are mapped to

$$\Psi((a_k, i, j)) = (a_k^{-1}, N_k - i, j)$$

$$\Psi((a_k, i, j+1)) = (a_k^{-1}, N_k - i, j+1)$$

  With a substitution

$$i' = N_k - i \; (0 \leqslant i' \leqslant N_k)$$

$$i' = j \; (0 \leqslant j' < N)$$

  it corresponds to a valid and unique fiber edge on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i', j'),\, (a_k^{-1}, i', j'+1)\right]$$

- For any base edge $[(a_k, i, j), (a_k, i+1, j)]$ ($0 \leqslant i < N_k$, $0 \leqslant j < N$), the end vertices are mapped to

$$\Psi((a_k, i, j)) = (a_k^{-1}, N_k - i, j)$$

$$\Psi((a_k, i+1, j)) = (a_k^{-1}, N_k - i - 1, j)$$

With a substitution

$$i' = N_k - i - 1 \; (0 \leqslant i' < N_k)$$

$$j' = j \; (0 \leqslant j' < N)$$

it corresponds to a valid and unique base edge on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i'+1, j'),\, (a_k^{-1}, i', j')\right]$$

- For any diagonal edge $\left[(a_k, i, j), (a_k, i+1, j+1)\right]$ $(0 \leqslant i < N_k$, $0 \leqslant j < N)$, the end vertices are mapped to

  $$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N_k - i, j)$$

  $$\Psi\left((a_k, i+1, j+1)\right) = (a_k^{-1}, N_k - i - 1, j+1)$$

  With a substitution

  $$i' = N_k - i - 1 \; (0 \leqslant i' < N_k)$$

  $$j' = j + 1 \; (0 \leqslant j' < N)$$

  it corresponds to a valid and unique anti-diagonal edge on $a_k^{-1} \times S^1$

  $$\left[(a_k^{-1}, i'+1, j'-1),\, (a_k^{-1}, i', j')\right]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any edge on band $a_k^{-1} \times S^1$ is mapped to a valid and unique edge on band $a_k \times S^1$ under map Ψ^{-1}.

[2] $3 \leqslant k \leqslant 2g$: We will show that any edge on twisted band $a_k \times S^1$ is mapped to a unique edge on twisted band $a_k^{-1} \times S^1$ under map Ψ.

- For any fiber edge $[(a_k, i, j), (a_k, i, j+1)]$ $(0 \leqslant i \leqslant N,\ 0 \leqslant j < N)$, the end vertices are mapped to

  $$\Psi((a_k, i, j)) = (a_k^{-1}, N-i, j+i)$$

  $$\Psi((a_k, i, j+1)) = (a_k^{-1}, N-i, j+i+1)$$

  With a substitution

  $$i' = N - i\ (0 \leqslant i' \leqslant N)$$

  $$j' = j + i\ (0 \leqslant j' < N)$$

  it corresponds to a valid and unique fiber edge on $a_k^{-1} \times S^1$

  $$\left[(a_k^{-1}, i', j'),\ (a_k^{-1}, i', j'+1)\right]$$

- For any base edge $[(a_k, i, j), (a_k, i+1, j)]$ $(0 \leqslant i < N,\ 0 \leqslant j < N)$, the end vertices are mapped to

  $$\Psi((a_k, i, j)) = (a_k^{-1}, N-i, j+i)$$

  $$\Psi((a_k, i+1, j)) = (a_k^{-1}, N-i-1, j+i+1)$$

  With a substitution

  $$i' = N - i - 1\ (0 \leqslant i' < N)$$

$$j' = j + i \; (0 \leqslant j' < N)$$

it corresponds to a valid and unique anti-diagonal edge on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i' + 1, j' - 1),\, (a_k^{-1}, i', j')\right]$$

- For any anti-diagonal edge $[(a_k, i, j), (a_k, i+1, j-1)]$ $(0 \leqslant i < N,$ $0 \leqslant j < N)$, the end vertices are mapped to

$$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N - i, j + i)$$

$$\Psi\left((a_k, i+1, j-1)\right) = (a_k^{-1}, N - i - 1, j + i)$$

With a substitution

$$i' = N - i - 1 \; (0 \leqslant i' < N)$$

$$j' = j + i \; (0 \leqslant j' < N)$$

it corresponds to a valid and unique base edge on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i' + 1, j'),\, (a_k^{-1}, i', j')\right]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any edge on band $a_k^{-1} \times S^1$ is mapped to a valid and unique edge on band $a_k \times S^1$ under map Ψ^{-1}.

In summary, under Ψ any edge in $a_k \times S^1$ is mapped to a unique edge in $a_k^{-1} \times S^1$, and vice versa. Therefore Ψ induces a bijection of edges between each pair of bands. $\square$

Lemma 7.4.3. *A gluing map Ψ induces a bijection between triangles in $a_k \times S^1$ and triangles in $a_k^{-1} \times S^1$ ($1 \leqslant k \leqslant 2g$).*

Proof. It can be verified for untwisted bands ($1 \leqslant k \leqslant 2$) and twisted bands ($3 \leqslant k \leqslant 2g$) separately.

[1] $1 \leqslant k \leqslant 2$: We will show that under map Ψ any triangle on untwisted band $a_k \times S^1$ is mapped to a unique triangle on untwisted band $a_k^{-1} \times S^1$ in an orientation-reversing manner.

- For any triangle on $a_k \times S^1$

  $$[(a_k, i, j), (a_k, i+1, j), (a_k, i+1, j+1)]$$

  ($0 \leqslant i < N_k$, $0 \leqslant j < N$), the end vertices are mapped to

  $$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N_k - i, j)$$

  $$\Psi\left((a_k, i+1, j)\right) = (a_k^{-1}, N_k - i - 1, j)$$

  $$\Psi\left((a_k, i+1, j+1)\right) = (a_k^{-1}, N_k - i - 1, j+1)$$

  With a substitution

  $$i' = N_k - i - 1 \; (0 \leqslant i' < N_k)$$

  $$j' = j \; (0 \leqslant j' < N)$$

  it corresponds to a unique but reversed triangle on $a_k^{-1} \times S^1$

  $$\left[(a_k^{-1}, i'+1, j'), (a_k^{-1}, i', j'), (a_k^{-1}, i', j'+1)\right]$$

- For any triangle on $a_k \times S^1$

$$[(a_k, i, j), (a_k, i+1, j+1), (a_k, i, j+1)]$$

($0 \leqslant i < N_k$, $0 \leqslant j < N$), the end vertices are mapped to

$$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N_k - i, j)$$

$$\Psi\left((a_k, i+1, j+1)\right) = (a_k^{-1}, N_k - i - 1, j+1)$$

$$\Psi\left((a_k, i, j+1)\right) = (a_k^{-1}, N_k - i, j+1)$$

With a substitution

$$i' = N_k - i - 1 \ (0 \leqslant i' < N_k)$$

$$j' = j \ (0 \leqslant j' < N)$$

it corresponds to a unique but orientation-reversed triangle on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i'+1, j'), (a_k^{-1}, i', j'+1), (a_k^{-1}, i'+1, j'+1)\right]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any triangle on band $a_k^{-1} \times S^1$ is mapped to a unique but reversed triangle on band $a_k \times S^1$ under map Ψ^{-1}.

[2] $3 \leqslant k \leqslant 2g$: We will show that under map Ψ any triangle on twisted band $a_k \times S^1$ is mapped to a unique triangle on twisted band $a_k^{-1} \times S^1$ in an orientation-reversing manner.

- For any triangle on $a_k \times S^1$

$$[(a_k, i, j+1), (a_k, i, j), (a_k, i+1, j)]$$

$(0 \leqslant i < N,\ 0 \leqslant j < N)$, the end vertices are mapped to

$$\Psi\left((a_k, i, j+1)\right) = (a_k^{-1}, N-i, j+i+1)$$

$$\Psi\left((a_k, i, j)\right) = (a_k^{-1}, N-i, j+i)$$

$$\Psi\left((a_k, i+1, j)\right) = (a_k^{-1}, N-i-1, j+i+1)$$

With a substitution

$$i' = N - i - 1\ (0 \leqslant i' < N)$$

$$j' = j + i\ (0 \leqslant j' < N)$$

it corresponds to a unique but reversed triangle on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i'+1, j'+1), (a_k^{-1}, i'+1, j'), (a_k^{-1}, i', j'+1)\right]$$

- For any triangle on $a_k \times S^1$

$$[(a_k, i, j+1), (a_k, i+1, j), (a_k, i+1, j+1)]$$

$(0 \leqslant i < N,\ 0 \leqslant j < N)$, the end vertices are mapped to

$$\Psi\left((a_k, i, j+1)\right) = (a_k^{-1}, N-i, j+i+1)$$

$$\Psi\left((a_k, i+1, j)\right) = (a_k^{-1}, N-i-1, j+i+1)$$

$$\Psi\left((a_k, i+1, j+1)\right) = (a_k^{-1}, N-i-1, j+i+2)$$

With a substitution

$$i' = N - i - 1 \; (0 \leqslant i' < N)$$

$$j' = j + i + 1 \; (0 \leqslant j' < N)$$

it corresponds to a unique but reversed triangle on $a_k^{-1} \times S^1$

$$\left[(a_k^{-1}, i'+1, j'), (a_k^{-1}, i', j'), (a_k^{-1}, i', j'+1)\right]$$

By a similar analysis on the inverse map Ψ^{-1}, it can be shown that any triangle on band $a_k^{-1} \times S^1$ is mapped to a unique but reversed triangle on band $a_k \times S^1$ under map Ψ^{-1}.

In summary, under Ψ any triangle in $a_k \times S^1$ is mapped to a unique but reversed triangle in $a_k^{-1} \times S^1$, and vice versa. Therefore Ψ induces a bijection of triangles between each pair of bands. □

According to lemma 7.4.1, 7.4.2 and 7.4.3, we claim that a gluing map induces a bijection of gluable triangulations on each pair of bands.

Theorem 7.4.1. *A gluing map Ψ induces a bijection of the gluable triangulation on band $a_k \times S^1$ and $a_k^{-1} \times S^1$ $(1 \leqslant k \leqslant 2g)$.*

In addition, from lemma 7.4.2 and its proof, we can get a corollary regarding the mapping among paths of different types.

Corollary 7.4.1. *A gluing map Ψ induces a bijection between paths on $a_k \times S^1$ and paths on $a_k^{-1} \times S^1$ $(1 \leqslant k \leqslant 2g)$. In particular:*

- *For untwisted bands $\{a_k \times S^1, a_k^{-1} \times S^1 \,|\, 1 \leqslant k \leqslant 2\}$:*

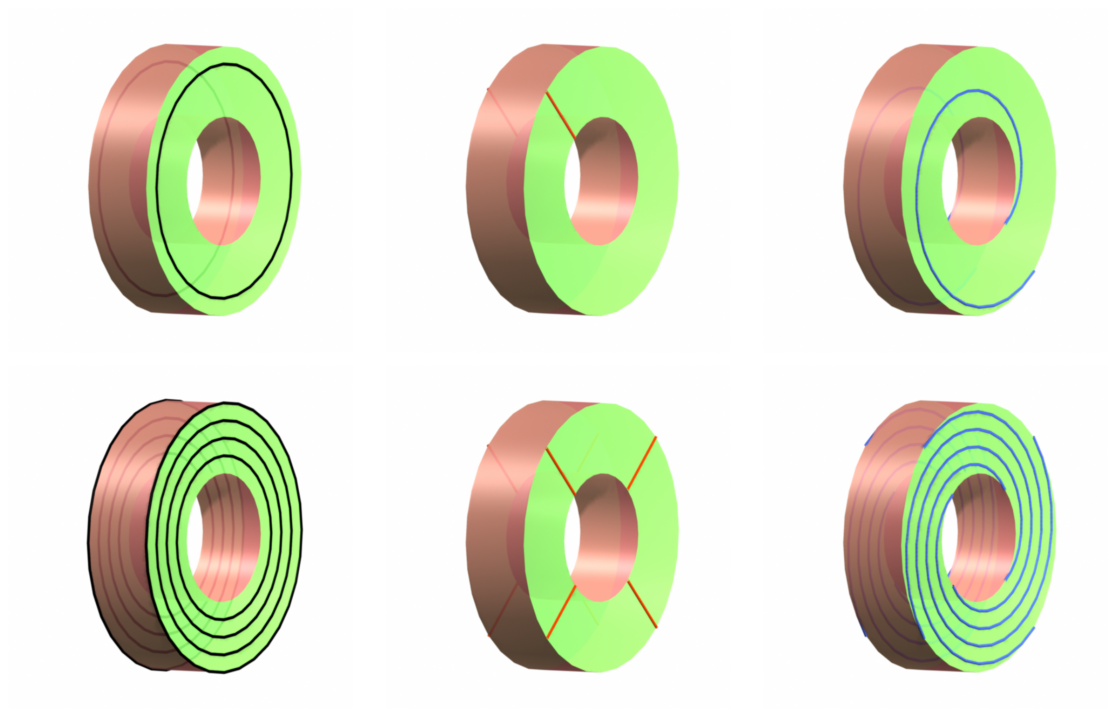
  - *Any fiber path on $a_k \times S^1$ is mapped to a unique fiber path on $a_k^{-1} \times S^1$, and vice versa;*
  - *Any base path on $a_k \times S^1$ is mapped to a unique base path on $a_k^{-1} \times S^1$, and vice versa;*
  - *Any diagonal path on $a_k \times S^1$ is mapped to a unique anti-diagonal path on $a_k^{-1} \times S^1$, and vice versa;*

- *For twisted bands $\{a_k \times S^1, a_k^{-1} \times S^1 \,|\, 3 \leqslant k \leqslant 2g\}$:*
  - *Any fiber path on $a_k \times S^1$ is mapped to a unique fiber path on $a_k^{-1} \times S^1$, and vice versa;*
  - *Any base path on $a_k \times S^1$ is mapped to a unique anti-diagonal path on $a_k^{-1} \times S^1$, and vice versa;*
  - *Any anti-diagonal path on $a_k \times S^1$ is mapped to a unique base path on $a_k^{-1} \times S^1$, and vice versa;*

7.4.2 Continuity

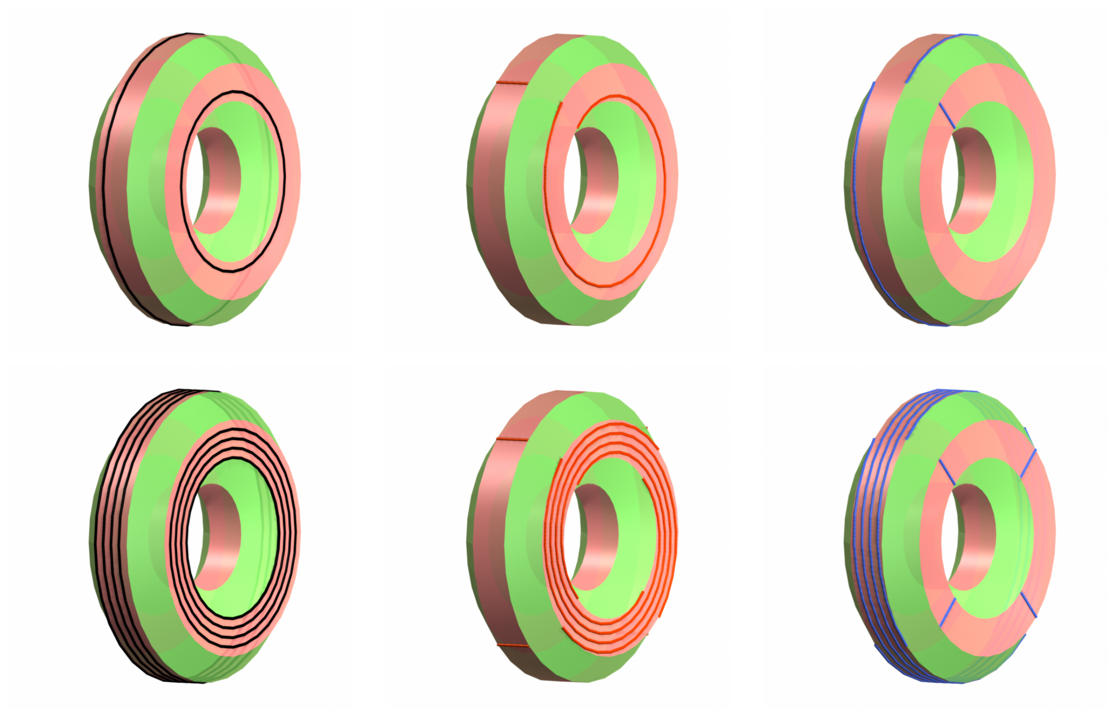
In lemma 7.4.1, 7.4.2 and 7.4.3, we show that a gluing map Ψ induces bijections of vertices, edges and triangles between each pair of bands. As we will show in this part, Ψ is continuous on the triangulations over each band. Again, we use the concepts of one-ring neighborhood for vertices (definition 6.4.2) and preservingly or reversely continuous map for triangulations (definition 6.4.3). The following theorem guarantees that a gluing map in definition 7.3.1 is a continuous map with orientation reversed.

Theorem 7.4.2. *Gluing map Ψ and its inverse Ψ^{-1} are both reversely continuous on each pair of bands $a_k \times S^1$ and $a_k^{-1} \times S^1$.*



(a) (b) (c)

Figure 7.7: Path map on untwisted bands for $UT\,(T^2(1))$. The gluing map over ∂T induces a path map between a pair of untwisted bands $a_2 \times S^1$ and $a_2^{-1} \times S^1$, mapping fiber paths to fiber paths (a), base paths to base paths (b), anti-diagonal paths to anti-diagonal paths (c) respectively.



(a) (b) (c)

Figure 7.8: Path map on untwisted bands for $UT\,(T^2(2))$. The gluing map over ∂T induces a path map between a pair of twisted bands $a_3 \times S^1$ and $a_3^{-1} \times S^1$, mapping fiber paths to fiber paths (a), base paths to anti-diagonal paths (b), anti-diagonal paths to base paths (c) respectively.

Proof. It suffices to show that for both untwisted bands and twisted bands, both Ψ and Ψ^{-1} keep one-ring neighborhoods invariant with orientation reversed.

[1] $1 \leqslant k \leqslant 2$: We will show that Ψ is reversely continuous from untwisted band $a_k \times S^1$ to untwisted band $a_k^{-1} \times S^1$.

- $i = 0$: vertex $v = (a_k, i, j)$ is on the boundary of $a_k \times S^1$. Its one-ring neighborhood has three triangles

$$\Theta(v) = [f_1, f_2, f_3]$$

where

$$f_1 = [(a_k, i, j), (a_k, i, j-1), (a_k, i+1, j)]$$

$$f_2 = [(a_k, i, j), (a_k, i+1, j), (a_k, i+1, j+1)]$$

$$f_3 = [(a_k, i, j), (a_k, i+1, j+1), (a_k, i, j+1)]$$

Under Ψ they are mapped to

$$\Psi(f_1) = [(a_k, i', j'), (a_k, i', j'-1), (a_k, i'-1, j')]$$

$$\Psi(f_2) = [(a_k, i', j'), (a_k, i'-1, j'), (a_k, i'-1, j'+1)]$$

$$\Psi(f_3) = [(a_k, i', j'), (a_k, i'-1, j'+1), (a_k, i', j'+1)]$$

where

$$i' = N_k - i\,, \quad j' = j$$

Note that the image vertex $v' = \Psi(v) = (a_k^{-1}, i', j')$ has a one-ring neighborhood of

$$\Theta(v') = [f'_1, f'_2, f'_3]$$

where

$$f'_1 = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i', j'+1), (a_k^{-1}, i'-1, j'+1)\big]$$

$$f'_2 = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i'-1, j'+1), (a_k^{-1}, i'-1, j')\big]$$

$$f'_3 = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i'-1, j'), (a_k^{-1}, i', j'-1)\big]$$

Obviously

$$\Psi(f_l) = (f'_{3-l})^{-1} \ (1 \leqslant l \leqslant 3)$$

Therefore Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$.

- $i = N_k$: vertex $v = (a_k, i, j)$ is on the boundary of $a_k \times S^1$. By a similar analysis to the $i = 0$ case, we have that Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$
- $0 < i < N_k$: vertex $v = (a_k, i, j)$ is an interior vertex of $a_k \times S^1$. Its one-ring neighborhood has six triangles

$$\Theta(v) = [f_1, f_2, f_3, f_4, f_5, f_6]$$

and so does the one-ring neighborhood of the image vertex $v' = \Psi(v) = (a_k^{-1}, i', j')$ ($i' = N_k - i$, $j' = j$)

$$\Theta(v') = [f'_1, f'_2, f'_3, f'_4, f'_5, f'_6]$$

Similar to the $i = 0$ case it can be verified that

$$\Psi(f_l) = (f'_{6-l})^{-1} \ (1 \leqslant l \leqslant 6)$$

Therefore Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$.

Using a similar deduction, it can be shown that Ψ^{-1} is also reversely continuous from untwisted band $a_k^{-1} \times S^1$ to untwisted band $a_k \times S^1$.

[2] $3 \leqslant k \leqslant 2g$: We will show that Ψ is reversely continuous from twisted band $a_k \times S^1$ to twisted band $a_k^{-1} \times S^1$.

- $i = 0$: vertex $v = (a_k, i, j)$ is on the boundary of $a_k \times S^1$. Its one-ring neighborhood has three triangles

$$\Theta(v) = [f_1, f_2, f_3]$$

where

$$f_1 = [(a_k, i, j), (a_k, i, j-1), (a_k, i+1, j-1)]$$

$$f_2 = [(a_k, i, j), (a_k, i+1, j-1), (a_k, i+1, j)]$$

$$f_3 = [(a_k, i, j), (a_k, i+1, j), (a_k, i, j+1)]$$

Under Ψ they are mapped to

$$\Psi(f_1) = [(a_k, i', j'), (a_k, i', j'-1), (a_k, i'-1, j')]$$

$$\Psi(f_2) = [(a_k, i', j'), (a_k, i'-1, j'), (a_k, i'-1, j'+1)]$$

$$\Psi(f_3) = [(a_k, i', j'), (a_k, i'-1, j'+1), (a_k, i', j'+1)]$$

where

$$i' = N - i\,, \quad j' = j + i$$

Note that the image vertex $v' = \Psi(v) = (a_k^{-1}, i', j')$ has a one-ring neighborhood of

$$\Theta(v') = [f_1', f_2', f_3']$$

where

$$f_1' = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i', j'+1), (a_k^{-1}, i'-1, j'+1)\big]$$

$$f_2' = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i'-1, j'+1), (a_k^{-1}, i'-1, j')\big]$$

$$f_3' = \big[(a_k^{-1}, i', j'), (a_k^{-1}, i'-1, j'), (a_k^{-1}, i', j'-1)\big]$$

Obviously

$$\Psi(f_l) = (f_{3-l}')^{-1} \ (1 \leqslant l \leqslant 3)$$

Therefore Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$.

- $i = N$: vertex $v = (a_k, i, j)$ is on the boundary of $a_k \times S^1$. By a similar analysis to the $i = 0$ case, we have that Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$
- $0 < i < N$: vertex $v = (a_k, i, j)$ is an interior vertex of $a_k \times S^1$. Its one-ring neighborhood has six triangles

$$\Theta(v) = [f_1, f_2, f_3, f_4, f_5, f_6]$$

and so does the one-ring neighborhood of the image vertex $v' =$

$\Psi(v) = (a_k^{-1}, i', j')$ $(i' = N_k - i,\ j' = j + i)$

$$\Theta(v') = [f'_1, f'_2, f'_3, f'_4, f'_5, f'_6]$$

Similar to the $i = 0$ case it can be verified that

$$\Psi(f_l) = (f'_{6-l})^{-1} \ (1 \leqslant l \leqslant 6)$$

Therefore Ψ maps $\Theta(v)$ to the reversed $\Theta(\Psi(v))$.

Using a similar deduction, it can be shown that Ψ^{-1} is also reversely continuous from twisted band $a_k^{-1} \times S^1$ to twisted band $a_k \times S^1$.

In summary, under the gluing map Ψ any triangulated band $a_k \times S^1$ is mapped to $a_k^{-1} \times S^1$ (and vice versa) continuously with orientation reversed. $\square$

Note that we only claim the continuity of Ψ within each band. Actually the gluing map is not continuous across the border of two consecutive bands. However, we will show later that all the border fibers will be nicely identified, and the continuity and bijection on each band is enough to guarantee that the result of the gluing is a valid tetrahedral mesh.

7.4.3 Fiber Twisting

In corollary 7.4.1 we show the behavior of a gluing map Ψ over different types of paths, namely fiber paths, base paths and (anti-)diagonal paths. With thorough investigation, we can get further information on the mapping of fiber paths, which will provide us with information about how the global bundle is twisted under gluing map Ψ.

On each fiber path, there is a unique vertex with zero local coordinate (i.e. $j = 0$); we call this vertex the *zero point* of this fiber path. According to corollary 7.4.1, every fiber path on $c_k \times S^1$ is mapped to a unique fiber path on $c_k^{-1} \times S^1$. In general there is no guarantee that their zero points are identified.

$$\begin{tikzcd}
 & \gamma^F_{N_1}(a_1^{-1}) \arrow[d, equal] & \gamma^F_{N_2}(a_2) \arrow[d, equal] & \gamma^F_{N_1}(a_1) \arrow[d, equal] & \gamma^F_{N_2}(a_2^{-1}) \arrow[d, equal] \\
\gamma^F_0(a_1) \arrow[ru] & \gamma^F_0(a_2^{-1}) \arrow[ru] & \gamma^F_0(a_1^{-1}) \arrow[ru] & \gamma^F_0(a_2) \arrow[ru] & \gamma^F_0(a_3) \arrow[ddllll, dashed] \\
 & \gamma^F_N(a_3^{-1}) \arrow[d, equal] & \gamma^F_N(a_4) \arrow[d, equal] & \gamma^F_N(a_3) \arrow[d, equal] & \gamma^F_N(a_4^{-1}) \arrow[d, equal] \\
\gamma^F_0(a_3) \arrow[ru] & \gamma^F_0(a_4^{-1}) \arrow[ru] & \gamma^F_0(a_3^{-1}) \arrow[ru] & \gamma^F_0(a_4) \arrow[ru] & \gamma^F_0(a_5) \arrow[dllll, dashed] \\
\cdots \arrow[dd, dashed] & & \cdots & & \\
 & \gamma^F_N(a_{2g-1}^{-1}) \arrow[d, equal] & \gamma^F_N(a_{2g}) \arrow[d, equal] & \gamma^F_N(a_{2g-1}) \arrow[d, equal] & \gamma^F_N(a_{2g}^{-1}) \arrow[d, equal] \\
\gamma^F_0(a_{2g-1}) \arrow[ru] & \gamma^F_0(a_{2g}^{-1}) \arrow[ru] & \gamma^F_0(a_{2g-1}^{-1}) \arrow[ru] & \gamma^F_0(a_{2g}) \arrow[ru] & \gamma^F_0(a_0) \arrow[uuuuullll, dashed, bend right]
\end{tikzcd}$$

Figure 7.9: All the border fiber paths on ∂T are identified. The red arrow means that the lower-left fiber path is mapped to the upper right fiber path by Ψ; The blue equal symbol means that the upper fiber path and the lower fiber path are actually the same fiber shared by two consecutive bands.

Actually how their zero points are mapped tells how a band is twisted. The following theorem quantifies the twisting induced by a gluing map Ψ.

Theorem 7.4.3. *A gluing map Ψ induces a -2π twist of fiber paths across each twisted band $a_k \times S^1$ ($1 \leqslant k \leqslant 2$), while zero twist across each untwisted band $a_k \times S^1$ ($3 \leqslant k \leqslant 2g$).*

Proof. Consider the sequence of fiber paths on band $a_k \times S^1$:

$$\left[\gamma^F_0(a_k),\ \gamma^F_1(a_k),\ \cdots\ \gamma^F_{N_{k-1}}(a_k),\ \gamma^F_{N_k}(a_k)\right]$$

According to corollary 7.4.1 they are mapped to a sequence of fiber paths on band $a_k^{-1} \times S^1$:

$$\left[\gamma^F_{N_k}(a_k^{-1}),\ \gamma^F_{N_{k-1}}(a_k^{-1}),\ \cdots\ \gamma^F_1(a_k^{-1}),\ \gamma^F_0(a_k^{-1})\right]$$

Note that the zero points of fiber paths on band $a_k^{-1} \times S^1$ are:

$$\left[(a_k^{-1}, N, 0),\ (a_k^{-1}, N-1, 0),\ \cdots,\ (a_k^{-1}, 1, 0),\ (a_k^{-1}, 0, 0)\right]$$

Now consider the corresponding pre-image vertices on band $a_k \times S^1$.

[1] For $1 \leqslant k \leqslant 2$, according to the gluing map

$$\Psi((a_k, i, j)) = (a_k^{-1}, N-i, j)$$

the pre-images on band $a_k \times S^1$ are:

$$[(a_k, 0, 0),\ (a_k, 1, 0),\ \cdots,\ (a_k, N-1, 0),\ (a_k, N, 0)]$$

That is, for $i = 0,\ 1,\ \cdots,\ N_k - 1, N_k$, the fiber $\gamma^F_i(a_k)$ will always have its own zero point mapped to the zero point on the image fiber $\Psi\left(\gamma^F_i(a_k)\right)$. It means that there is no twisting on such a band.

[2] For $3 \leqslant k \leqslant 2g$, according to the gluing map

$$\Psi((a_k, i, j)) = (a_k^{-1}, N-i, j)$$

the pre-images on band $a_k \times S^1$ are:

$$[(a_k, 0, 0),\ (a_k, 1, -1),\ \cdots,\ (a_k, N-1, -(N-1)),\ (a_k, N, -N)]$$

That is, for $i = 0,\ 1,\ \cdots,\ N-1, N$, fiber $\gamma^F_i(a_k)$ is rotated (in terms of j)

by $0,\ -1,\ \cdots,\ -(N-1), -N$ to meet the zero point on the image fiber $\Psi\left(\gamma_i^F(a_k)\right)$. Since every fiber path is actually a cycle with discretizing resolution of N, a shift of $-N$ in terms of j is equivalent to a rotation of -2π. It means that such a band has a twisting of -2π.

$\square$

From theorem 7.4.3, we see that for a genus g surface, the gluing map will induce a -2π twisting on each pair of twisted bands. Since there are $2g-2$ pairs twisted bands, the total twisting is $2\pi(2-2g)$. As a special case, for $g=1$ closed surface, there is no twisted bands and therefore there is no twisting induced by gluing map Ψ.

Furthermore, note that each band $c_k \times S^1$ is bounded by two fiber paths (cycles) $\gamma_0^F(c_k)$ and $\gamma_{N_k}^F(c_k)$, and every two consecutive bands share a common fiber path

$$\gamma_{N_k}^F(c_k) = \left(c_k \times S^1\right) \cup \left(c_{k+1} \times S^1\right) = \gamma_0^F(c_{k+1})$$

.

We name it a *border* fiber path, and without loss of generality we use $\gamma_0^F(c_k)$ to represent the border fiber path between $c_{k-1} \times S^1$ and $c_k \times S^1$. From theorem 7.4.3, we can have an easy corollary 7.4.2 about the mapping among border fiber paths.

Corollary 7.4.2. *A gluing map Ψ identifies all the border fiber paths:*

$$\Psi(\gamma_0^F(c_k)) = \Psi(\gamma_0^F(c_l)) \ \ (1 \leqslant k,\ l \leqslant 4g)\}$$

The mapping between any pair of border fiber paths is an identity map in terms of local coordinates along fibers:

$$\Psi((c_k, 0, j)) = (c_l, 0, j) \ \ (1 \leqslant k,\ l \leqslant 4g,\ 0 \leqslant j \leqslant N)$$

The identification of all the border fiber paths is illustrated in figure 7.9.

Actually they are all mapped to the fiber cycle over the base point in the global bundle.

7.4.4 Final Remarks

In the above discussions, we investigate local and global constructions and prove some important properties of the gluable triangulation and gluing map. Based on all those theorems and corollaries, we can draw two conclusions.

First, a gluing map Ψ turns a local bundle T into a valid tetrahedral mesh UT that has no boundary. With a gluable triangulation (definition 7.2.1) defined on the boundary of local bundle, a gluing map Ψ (definition 7.3.1) induces a map between each pair of boundary bands $a_k \times S^1$ and $a_k^{-1} \times S^1$ that is both bijective (theorem 7.4.1) and reversely continuous (theorem 7.4.2). Therefore, T is glued to itself along its boundary seamlessly, and the result is again a tetrahedral mesh without any boundary.

Second, the tetrahedral mesh UT is a valid discrete representation for a unit tangent bundle over a given $T^2(g)$ surface. Note that the local bundle is a trivial S^1 bundle over the covering disk. Under the gluing map Ψ, every fiber that is not on ∂T will remain in UT as a single fiber, and every fiber on ∂T will be identified with some other fibers on ∂T (corollary 7.4.1) and become a fiber in UT. In particular, all the fibers on the border of bands will be identified (theorem 7.4.2) to a single fiber in UT. Therefore, UT is a valid S^1 bundle over the original $T^2(g)$ surface. Since the total twisting induced by gluing map Ψ is $2\pi(2-2g)$ (theorem 7.4.3), this UT actually represents a unit tangent bundle over $T^2(g)$.

In summary, we claim that our construction will generate valid discrete representations of unit tangent bundles over $T^2(g)$ surfaces.

Theorem 7.4.4. *Given a $T^2(g)$ surface M $(g > 0)$, the local-to-global construction generates a valid tetrahedral mesh that represents the unit tangent bundle $UT(M)$.*

Chapter 8

Conclusion

In this dissertation we investigate two major problems in geometric design and computer graphics, namely discrete metric design for surfaces and 3-manifolds, and discrete unit tangent bundle design for surfaces.

For discrete metric design, we study three different types of surfaces or 3-manifolds, and develop computational methods for each of them.

In chapter 2, we present a discrete metric design method for genus-zero surfaces with multiple holes. Given a surface with multiple holes, we compute several flat metrics so that the surface can be mapped to four different flat domains, including cylindric, parallel, rectangle and circular slit domains (figure 1.1). They can be used to solve different problems, such as surface matching, brain mapping, quad-mesh generation and etc. The underlying computation is based on rigorous results on Riemann surfaces and complex analysis. It involves compute the basis of holomorphic one-forms on the given surface, and a special holomorphic one-form that can generate the slit domains. The algorithm is linear and the mapping is conformal.

In chapter 3, we present a volumetric parameterization method for handle bodies. This method utilizes the fact that such volumes are direct product of certain surfaces and a one-dimensional line segment. The algorithm starts from partitioning the boundary surface into bases and walls, and then extend a given flat metric on the bases into the volume by computing volumetric harmonic functions. In order to relieve distortion, polycube domains are utilized to model the boundary surface and to parameterize the whole volume.

In chapter 4, we present discrete metric design algorithms for hyperbolic 3-manifolds with boundaries. The final metric we compute induces constant curvature in the interior of the given 3-manifolds, and zero curvature on the boundary. We also provide algorithms to simplify a tetrahedral mesh and convert it to a truncated tetrahedral mesh, and algorithms to embed and visualize the given 3-manifold in 3-dimensional hyperbolic space.

For discrete unit tangent bundle designs, we first give a solution for topological disks, and then use that as part of the algorithms to solve closed surfaces.

In chapter 5, we present methods to generate discrete unit tangent bundles for topological disks (D^2). Due to the fact that such a bundle is trivial, we are able to generate a tetrahedral mesh that has a regular structure both on the boundary and in the interior. This problem is converted to an equivalent 2D problem, the cutting pattern problem. We provide algorithms to solve this 2D problem under three different boundary conditions, from the most restricted one to the most general and flexible one. These algorithms are proved to terminate with a solution under most circumstances. In case a solution does not exist, the unsolvable part in the input mesh can be addressed and reported.

In chapter 6, we present a discrete construction of unit tangent bundles for $g = 0$ closed and oriented surfaces (S^2). For such a non-trivial bundle, we utilize the idea of local trivialization. Namely, we partition a given surface into two covering disks, build trivial local bundles over each disk, and then glue those local bundles into a global bundle nicely along their boundary. In order to meet the topological requirements for a unit tangent bundle, we design a special triangulation on the boundary of local bundles, and a gluing map in between to glue two local bundles to a global bundle. We validate that such a construction generates a valid tetrahedral mesh that faithfully represent the unit tangent bundle.

In chapter 7, we present a discrete construction of unit tangent bundles for $g > 0$ closed and oriented surfaces ($T^2(g)$). Similar to the $g = 0$ case, we still follow a local-to-global philosophy. But here we use a single disk to cover a given surface, and build a single local bundle over the covering disk. Then the local bundle is glued to itself along its own boundary. Again we design gluable triangulations on the boundary of the local bundle, and a gluing map over it. We also validate that such a construction generates a valid tetrahedral mesh with the right amount of twisting that required by a unit tangent bundle. In particular, when $g = 1$ this construction will give a trivial bundle.

Based on this dissertation, there are several other topics that draw interests for further investigation. First, for discrete metric design, the current methods in this work target at conformal parameterizations, which can generate parameter domains with regular shape, but usually incur large distortion. One future direction is to compute metrics that takes area distortion into consideration and try to minimize the distortion. Second, for volumetric metric design, only two types of 3-manifolds are involved in the current work. It is challenging to extend the study to other types of 3-manifolds, or to find alternative methods for current types. Third, in this work we only focus on discrete constructions of unit tangent bundles. Any interesting extension would be other types of fiber bundles that bear different amount of twisting and have different properties and behaviors. Another future direction is to use them to discretize other types of geometric objects, such as covariant differentiation, connection and etc.

Bibliography

[1] Lars Ahlfors. *Complex Analysis.* McGraw-Hill Science / Engineering / Math, 3 edition, 1979.

[2] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. In *ACM Trans. Graph.*, pages 617–625, 2005.

[3] Mirela Ben-Chen, Craig Gotsman, and Guy Bunin. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum (Proc. Eurographics)*, 27(2), 2008.

[4] Alexander I. Bobenko and Boris A. Springborn. Variational principles for circle patterns and koebe's theorem. *Transactions of the American Mathematical Society*, 356:659–689, 2004.

[5] Philip L. Bowers and Monica K. Hurdal. Planar conformal mapping of piecewise flat surfaces. In *Visualization and Mathematics III*, pages 3–34, Berlin, 2003. Springer-Verlag.

[6] Patrick J. Callahan, Martin V. Hildebrand, and Jeffrey R. Weeks. A census of cusped hyperbolic 3-manifolds. *Math. Comput.*, 68(225):321–332, 1999.

[7] Guoning Chen, Konstantin Mischaikow, Robert S. Laramee, Pawel Pilarczyk, and Eugene Zhang. Vector field editing and periodic orbit extraction using morse decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):769–785, 2007.

[8] Guoning Chen, Konstantin Mischaikow, Robert S. Laramee, and Eugene Zhang. Efficient morse decompositions of vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):848–862, 2008.

[9] Siu-Wing Cheng, Tamal K. Dey, and Joshua A. Levine. A practical delaunay meshing algorithm for a large class of domains*. In *Proceedings of the 16th International Meshing Roundtable*, pages 477–494. 2008.

[10] Siu-Wing Cheng, Tamal K. Dey, and Edgar A. Ramos. Delaunay refinement for piecewise smooth complexes. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1096–1105, New Orleans, Louisiana, 2007. Society for Industrial and Applied Mathematics.

[11] Bennett Chow and Feng Luo. Combinatorial Ricci flows on surfaces. *Journal Differential Geometry*, 63(1):97–129, 2003.

[12] Chuck Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry: Theory and Applications*, 25:233–256, 2003.

[13] Colin de Verdiere Yves. Un principe variationnel pour les empilements de cercles. *Invent. Math.*, 104(3):655–669, 1991.

[14] Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. *Computer Graphics Forum*, 21(3):209–218, 2002.

[15] Tamal K. Dey. Delaunay mesh generation of three dimensional domains. In *Technical report*. Ohio State University, 2007.

[16] Tamal K. Dey, Kuiyu Li, and Jian Sun. On computing handle and tunnel loops. In *Proceedings of the 2007 International Conference on Cyberworlds*, pages 357–366. IEEE Computer Society, 2007.

[17] Tamal K. Dey, Kuiyu Li, Jian Sun, and David Cohen-Steiner. Computing geometry-aware handle and tunnel loops in 3d models. *ACM Trans. Graph.*, 27(3):1–9, 2008.

[18] Qiang Du and Desheng Wang. Tetrahedral mesh generation and optimization based on centroidal voronoi tessellations. *International Journal for Numerical Methods in Engineering*, 56(9):1355–1373, 2003.

[19] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge University Press, 1 edition, May 2001.

[20] Jeffrey Gordon Erickson and Kim Whittlesey. Greedy optimal homotopy and homology generators. In *Proc. 16th Symp. Discrete Algorithms*, pages 1038–1046. ACM and SIAM, 2005.

[21] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Trans. Graph.*, 26(3):56, 2007.

[22] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. *Advances in Multiresolution for Geometric Modelling*, pages 157–186, 2005.

[23] Roberto Frigerio, Bruno Martelli, and Carlo Petronio. Small hyperbolic 3-manifolds with geodesic boundary. *Experimental Mathematics*, 13:171–184, 2004.

[24] Michihiko Fujii. Hyperbolic 3-manifolds with totally geodesic boundary. *Osaka Journal of Mathematics*, 27:539–553, 1990.

[25] Steven J. Gortler, Craig Gotsman, and Dylan Thurston. Discrete one-forms on meshes and applications to 3D mesh parameterization. *Comput. Aided Geom. Des.*, 23(2):83–112, 2006.

[26] Xianfeng Gu, Ying He, and Hong Qin. Manifold splines. *Graphical Models*, 68(3):237–254, 2006.

[27] Xianfeng Gu and Shing-Tung Yau. Global conformal surface parameterization. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 127–137, Aachen, Germany, 2003. Eurographics Association.

[28] Xianfeng Gu and Shing-Tung Yau. *Computational Conformal Geometry.* Advanced Lectures in Mathematics. High Education Press and International Press, 2008.

[29] Richard S. Hamilton. Three manifolds with positive Ricci curvature. *Journal of Differential Geometry.*, 17:255–306, 1982.

[30] Richard S. Hamilton. The Ricci flow on surfaces. *Mathematics and general relativity (Santa Cruz, CA, 1986), Contemp. Math. Amer.Math.Soc. Providence, RI*, 71, 1988.

[31] James L. Helman and Lambertus Hesselink. Representation and display of vector field topology in fluid flow data sets. *Computer*, 22(8):27–36, 1989.

[32] Miao Jin, Junho Kim, and Xianfeng Gu. Discrete surface ricci flow: Theory and applications. *IMA Conference on the Mathematics of Surfaces*, pages 209—232, 2007.

[33] Miao Jin, Junho Kim, Feng Luo, and Xianfeng Gu. Discrete surface ricci flow. *IEEE Transaction on Visualization and Computer Graphics*, 2008.

[34] Y. Kallinderis, A. Khawaja, and H. Mcmorris. Hybrid Prismatic/Tetrahedral grid generation for complex geometries. *AIAA PAPER*, 34:93—0669, 1996.

[35] Liliya Kharevych, Boris Springborn, and Peter Schröder. Discrete conformal mappings via circle patterns. *ACM Transactions on Graphics*, 25(2):412–438, 2006.

[36] François Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. In *ACM Trans. Graph.*, 2007.

[37] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Transactions on Graphics*, 21(3):362–371, 2002.

[38] Hongyu Li, Wenbin Chen, and I-Fan Shen. Segmentation of discrete vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):289–300, 2006.

[39] Xin Li, Xiaohu Guo, Hongyu Wang, Ying He, Xianfeng Gu, and Hong Qin. Harmonic volumetric mapping for solid modeling applications. In *Proceeding of Symposium on Solid and Physical Modeling*, pages 109–120, 2007.

[40] Feng Luo. A combinatorial curvature flow for compact 3-manifolds with boundary. *Electron. Res. Announc. Amer. Math. Soc.*, 11:12–20, 2005.

[41] Karim Mahrous, Janine Bennett, Gerik Scheuermann, Bernd Hamann, and Kenneth I. Joy. Topological segmentation in Three-Dimensional vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):198–205, 2004.

[42] Tobias Martin, Elaine Cohen, and Mike Kirby. Volumetric parameterization and trivariate b-spline fitting using harmonic functions. In *Proceeding of Symposium on Solid and Physical Modeling*, 2008.

[43] R. Montenegro, J. M. Cascón, J. M. Escobar, E. Rodríguez, and G. Montero. An automatic strategy for adaptive tetrahedral mesh generation. *Applied Numerical Mathematics*, 59(9):2203–2217, September 2009.

[44] George D. Mostow. Quasi-conformal mappings in n-space and the rigidity of the hyperbolic space forms. *Publ.Math.IHES*, 34:53–104, 1968.

[45] Steve Owen. A survey of unstructured mesh generation technology, 1998.

[46] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 26(3):55, 2007.

[47] Grisha Perelman. The entropy formula for the Ricci flow and its geometric applications. Technical Report arXiv.org, November 11 2002.

[48] Grisha Perelman. Finite extinction time for the solutions to the Ricci flow on certain three-manifolds. Technical Report arXiv.org, July 17 2003.

[49] Grisha Perelman. Ricci flow with surgery on three-manifolds. Technical Report arXiv.org, March 10 2003.

[50] Fabiano Petronetto, Afonso Paiva, Marcos Lage, Geovan Tavares, Hlio Lopes, and Thomas Lewiner. Meshless Helmholtz-Hodge decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 99(2):338–349, 5555.

[51] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2:1, 1993.

[52] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1):1–11, 2009.

[53] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lvy. N-symmetry direction field design. *ACM Trans. Graph.*, 27(2):1–13, 2008.

[54] Burt Rodin and Dennis Sullivan. The convergence of circle packings to the riemann mapping. *Journal of Differential Geometry*, 26(2):349–360, 1987.

[55] Alla Sheffer, Emil Praun, and Kenneth Rose. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2), 2006.

[56] Jonathan Richard Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 86–95, Minneapolis, Minnesota, United States, 1998. ACM.

[57] Hang Si. Tetgen: A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. http://tetgen.berlios.de/.

[58] Boris Springborn, Peter Schröder, and Ulrich Pinkall. Conformal equivalence of triangle meshes. *ACM Transactions on Graphics*, 27(3):1–11, 2008.

[59] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube maps. In *ACM Transactions on Graphics*, pages 853–860, 2004.

[60] Holger Theisel. Designing 2D vector fields of arbitrary topology. *Computer Graphics Forum*, 21(3):595–604, 2002.

[61] William P. Thurston. *Geometry and Topology of Three-Manifolds*. lecture notes at Princeton university, 1980.

[62] William P. Thurston. The finite riemann mapping theorem. 1985.

[63] Yiying Tong, Pierre Alliez, David Cohen-Steiner, and Mathieu Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 201–210, Cagliari, Sardinia, Italy, 2006. Eurographics Association.

[64] Yiying Tong, Santiago Lombeyda, Anil N. Hirani, and Mathieu Desbrun. Discrete multiscale vector field decomposition. *ACM Trans. Graph.*, 22(3):445–452, 2003.

[65] Hongyu Wang, Ying He, Xin Li, Xianfeng Gu, and Hong Qin. Polycube splines. In *ACM Solid and Physical Modeling*, pages 241–251, 2007.

[66] Hongyu Wang, Miao Jin, Ying He, Xianfeng Gu, and Hong Qin. Usercontrollable polycube map for manifold spline construction. In *ACM Symposium On Solid And Physical Modeling*, pages 397–404, 2008.

[67] Yalin Wang, Xianfeng Gu, Paul M. Thompson, and Shing-Tung Yau. 3d harmonic mapping and tetrahedral meshing of brain imaging data. In *Proceeding of Medical Imaging Computing and Computer Assisted Intervention (MICCAI), St. Malo, France*, 2004.

[68] Xiaotian Yin, Junfei Dai, Shing-Tung Yau, and Xianfeng Gu. Slit map: Conformal parameterization for multiply connected surfaces. In *Geometric Modeling and Processing*, 2008.

[69] Xiaotian Yin, Miao Jin, Feng Luo, and Xianfeng Gu. Discrete curvature flows for surfaces and 3-manifolds. In *Emerging Trends in Visual Computing*, volume 5416 of *LNCS*, pages 38–74, 2009.

[70] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Trans. Graph.*, 25(4):1294–1326, 2006.