Project on Surface Parameterization

Xuan Li[1,*]

[1]Department of Computer Science, Stony Brook University
[*]SBU ID: 111676019

ABSTRACT

In this project, I implement three parametrization algorithms: Euclidean orbifold Tutte embedding [Aigerman and Lipman, 2015], hyperbolic orbifold Tutte embedding [Aigerman and Lipman, 2016], and boundary first flattening [Sawhney and Crane, 2017]. The Euclidean orbifold Tutte embedding uses orbifold structure to generalize the algorithm in [Gortler et al., 2006] to sphere-type meshes. Hyperbolic orbifold Tutte embedding generalize the Euclidean orbifold Tutte embedding in the sense of geometry. But this generalization in non-linear. Besides, the Euclidean orbifold Tutte is also claimed to be conformal, so this algorithm provide a new linear method to compute conformal maps. Boundary first flattening is another linear conformal map solver. But this algorithm is based on differential geometry.

1 Introduction

Parameterization is very important to many graphics applications. In this project, I explored two kinds of parameterization algorithms. The former is what in my **proposal**. The latter is what I did extra for the **bonus points**.

The first one is orbifold Tutte embedding [Aigerman and Lipman, 2015] [Aigerman and Lipman, 2016] . The behaviors of orbifolds is different in different background geometry, and I explored two kinds of geometry: Euclidean geometry and hyperbolic geometry. We will see later that algorithms for these two different settings are very different. One is linear and the other is non-linear. [Aigerman et al., 2017] is about spherical orbfolds. But in the level of implementation, there is no difference with hyperbolic orbifolds, except the distance function and isometries.

The second one is boundary first flattening [Sawhney and Crane, 2017]. This algorithm is in the mindset of differential geometry and conformal geometry. And this algorithm is linear.

This report is organized as follows: First I talk about the tools I used to implement my project. Then I separate the rest of report into two parts to introduce two kinds of algorithm separately.

The whole project and documents are uploaded onto GitHub: `https://github.com/xuan-li/GraphicsProject`

2 Project Overview

2.1 Toolbox

I implemented my whole project on Windows, using Visual Studio Community 2017. And I used some open-source libraries: Eigen [Guennebaud et al., 2010], Libigl [Jacobson et al., 2016], LBFGS++ [Qiu, 2016], and OpenMesh [Botsch et al., 2002]. The needed libraries are compiled into static libraries file (*.lib), and included in the projects.

2.1.1 OpenMesh

OpenMesh [Botsch et al., 2002] is the core data sturcture of my project. This library implements so-called halfedge data structure for polygonal mesh.

Each mesh $M = (V, E, F, H)$ can be represented by four components: V stands for vertex set, E stands for edge set, H stands for halfedge setsand F stands for face set. Each edge is corresponded with two opposite halfedges. The linking relations are shown in Fig.1.

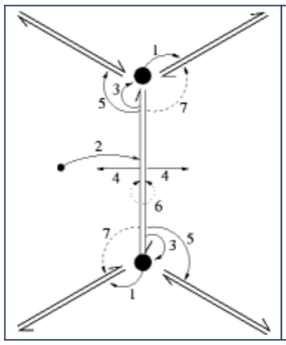
2.1.2 Eigen

Eigen [Guennebaud et al., 2010] is a standard library to handle matrices and their operations in C++. It's a head-only library, so it's very easy to deploy.

I will use this library to handle linear part in my project.

2.1.3 LBFGS++

LBFGS is a first-order optimization algorithm with line search. LBFGS++ [Qiu, 2016] is an implementation for this algorithm in C++. This is a project I found on GitHub, and it is very new.



- Each **vertex** references one outgoing halfedge, i.e. a halfedge that starts at this vertex (1).
- Each **face** references one of the halfedges bounding it (2).
- Each **halfedge** provides a handle to
  - the vertex it points to (3),
  - the face it belongs to (4)
  - the next halfedge inside the face (ordered counter-clockwise) (5),
  - the opposite halfedge (6),
  - (optionally: the previous halfedge in the face (7)).

Figure 1. Halfedge data structure.

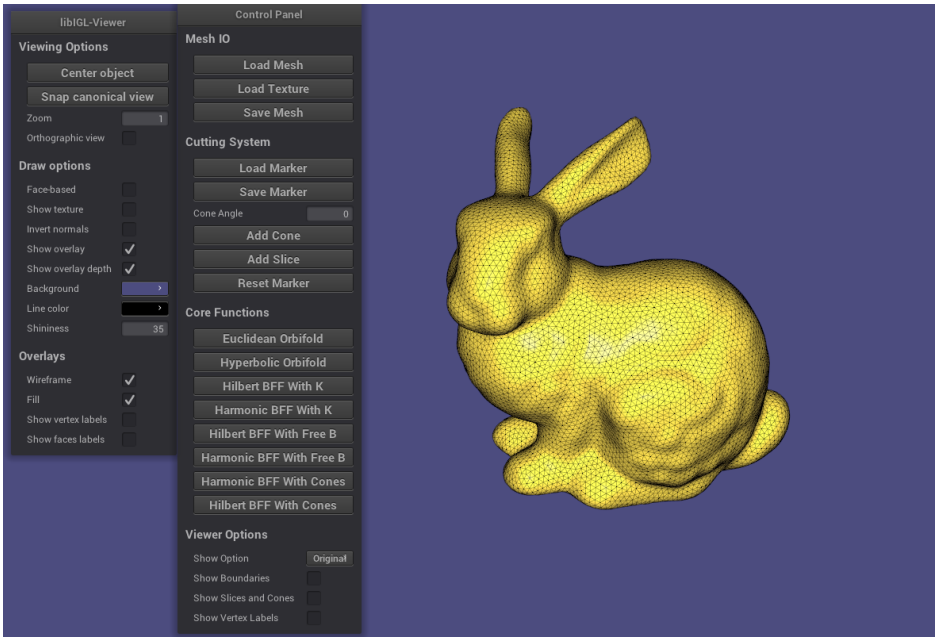


Figure 2. The GUI of my project

I use this algorithm to optimize the energy for hyperbolic orbifold.

2.1.4 Libigl

Libigl [Jacobson et al., 2016] is a simple C++ geometry processing library.

I use the Viewer class [1] of this library to implement my GUI. The GUI part of this library is based on Nanogui[2] and OpenGL3. The GUI of my project is shown in Fig.2

On the other hand, this library provides lots of functions used in geometry processing. I also use its quadratic programming algorithm [3]. It is used in the optimization problem for closed loop in boundary first flatten algorithm.

2.2 Project Structure

I put all codes in one solution of VS2017. The solution have five VS projects: Mesh, OrbifoldEmbedding, BoundaryFirstFlattening, Viewer and Utilities. They are linked together by reference between static libraries.

Mesh In this VS project, I extend the default definitions of OpenMesh. This mesh structure are used in all of other subprojects.

OrbifoldEmbedding This VS project implements the algorithms in [Aigerman and Lipman, 2015] [Aigerman and Lipman, 2016]. Both Euclidean orbifold Tutte embedding and hyperbolic orbifold Tutte embedding are included in this project.

[1] http://libigl.github.io/libigl/tutorial/tutorial.html#viewermenu
[2] https://github.com/wjakob/nanogui
[3] http://libigl.github.io/libigl/tutorial/tutorial.html#quadraticprogramming

BoundaryFirstFlattening This VS project implements the algorithms in [Sawhney and Crane, 2017].

Viewer This VS project implements the GUI part of my project. Meshes, embedding results, and texture mappings are showned here. Besides, I extend the Viewer class of Libigl so that I can select vertices on a mesh interactively.

Utilities This VS project implements some auxiliary functions needed in my project. The core function for my project is MeshSlicer, used to cut the mesh into a disk. This operation is needed for both orbifold and BFF algorithms. But most of this code is from my previous research projects. I modify it so that I can interactively choose slices. Most of other functions here are used to show my results.

3 Orbfiold Tutte Embeding

In this section, I will get into details of my project on orbifold Tutte embedding. I explore two geometry background: Euclidean and hyperbolic. This section is organized as follows: I will first introduce the theoretical background. Then I will give details on my implementation.

3.1 Theoretical Background

3.1.1 Tutte Embedding

Tutte embedding is very famous in the field of parameterization, because it can generate bijective parameterization. Tutte embedding is based on the following theorem [Tutte, 1963]:

Theorem 3.1 *Let $G = \langle V, E, F \rangle$ be a 3-connected planar graph with boundary vertices $B \subset V$ defining a unique unbounded exterior face f_e . Suppose ∂f_e is embedded in the plane as a (not necessarily strictly) convex planar polygon, and each interior vertex is positioned in the plane as a strictly convex combination of its neighbors, then the straight-line drawing of G with these vertex positions is an embedding. In addition, this embedding has strictly convex interior faces.*

The embedding problem is the following full-rank linear system:

$$\begin{aligned} \sum_{v_j \in N(v_i)} w_{ij}(z_j - z_i) = 0, \quad & v_i \in V - B \\ z_i = z_i^0, \quad & v_i \in B \end{aligned} \tag{1}$$

where $w > 0$.

[Gortler et al., 2006] generalized this algorithm to high genus surface, but spherical surface wasn't touched.

System 1 is equivalent to the following optimization problem:

$$\begin{aligned} \min_{\Phi} \quad & E(z) = \frac{1}{2} \sum_{(i,j) \in E} w_{ij} d(z_i, z_j)^2 \\ s.t \quad & z_i = z_i^0, \quad v_i \in B \end{aligned} \tag{2}$$

We will use this equivalence in hyperbolic background

3.1.2 Orbifolds

The orbifold is a generalization of the manifold. Formal definition for general orbifold is very abstract. We only focus on what we will use: 2-dimensional sphere-type orbifolds, that is, they have the topology of sphere. In the following context, we assume all orbifolds are of this kind.

In this project, we simply treat an orbifold as a seamlessly tiled plane by a basic tile. The transformations between copies are orientation-preserved isometries in their corresponding geometric backgrounds. Fix point of some isometry is called cones or singularities. And if two points is differed by an transformation between two copies, they are equivalent. The quotient plane by these equivalent relations are called orbifolds.

We know cones are rotation center of some isometries. So an orbifold is determined by cones and their rotation angles. Examples are shown in Fig.3. We use Poincare disk as hyperbolic geometry model.

3.2 Algorithm

The purpose of [Aigerman and Lipman, 2015] [Aigerman and Lipman, 2016] is generalize Tutte embedding onto sphere-type surfaces.

The algorithm is as follows:

1) Choose some cone set C, and find a path through all of them, cut the mesh M into a disk type mesh $\bar{M}$.

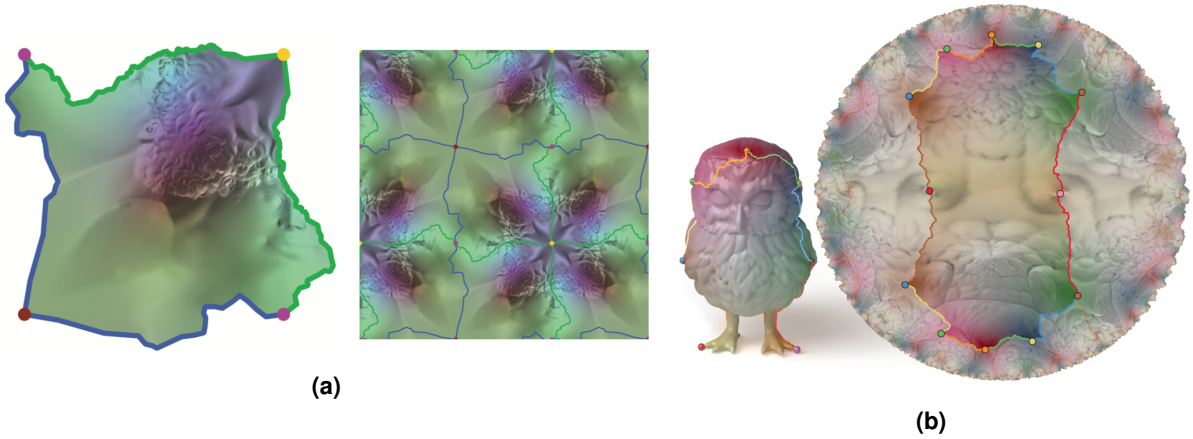


Figure 3. (a) A Euclidean orbfold. (b) A hyperbolic orbifold.

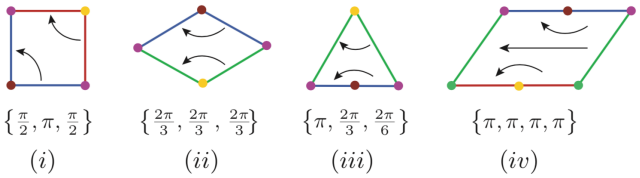


Figure 4. Four kinds of sphere-type Euclidean orbifolds

2) Embed cones of $\bar{M}$ to some basic tile.

3) Solve the orbifold system.

In step 1, some of cones in the original mesh will be splitted into two copies. They are both cones in cutted mesh.

In step 2, the positions of cones should satisfy the requirement of some orbifold. There are only four Euclidean orbifolds, shown in Fig.4. And for hyperbolic background, there are infinitely many. We simply set all cone angles be π, the method to compute the positions of cones are in the appendix of [Aigerman and Lipman, 2016]

In step 3, we solve Tutte embedding system with orbifold constraints.

For Euclidean orbifolds, we solve the following linear system.

$$\begin{aligned} \sum_{v_j\in N(v_i)} w_{ij}(z_j-z_i)=0 \quad & v_i\in \bar{V}-\bar{B} \\ \textstyle\sum_{v_j\in N(v_i)} w_{ij}(z_j-z_i)+\sum_{v_j\in N(v_{i'})} w_{i'j}R_{i'i}(z_j-z_{i'})=0 \quad & (v_i,v_{i'}) \text{ boundary pair} \\ R_{i'i}z_{i'}-z_i=t_{ii'} & \\ z_i=z_i^0 \quad & v_i\in\bar{C} \end{aligned} \tag{3}$$

Here $R_{i'i}$, $t_{i'i}$ is the rotation part and translation part of the isometry from i' to i.

For hyperbolic orbifolds, we solve the following non-linear system:

$$\begin{aligned} & \min_{\Phi} E(z), \\ s.t \quad & z_i=m_{i'i}(z_{i'}), \quad v_i\in\bar{B}-\bar{C} \\ & z_i=z_i^0, \qquad\qquad v_i\in\bar{C} \end{aligned} \tag{4}$$

The gradient is given by:

$$\nabla_{z_i}E=\sum j\in N_i w_{ij}\nabla_{z_i}d^2(z_i,z_j)+\sum j\in N_{i'}w_{i'j}\nabla_{z_i}d^2(z_i,m_{i'i}(z_j)) \tag{5}$$

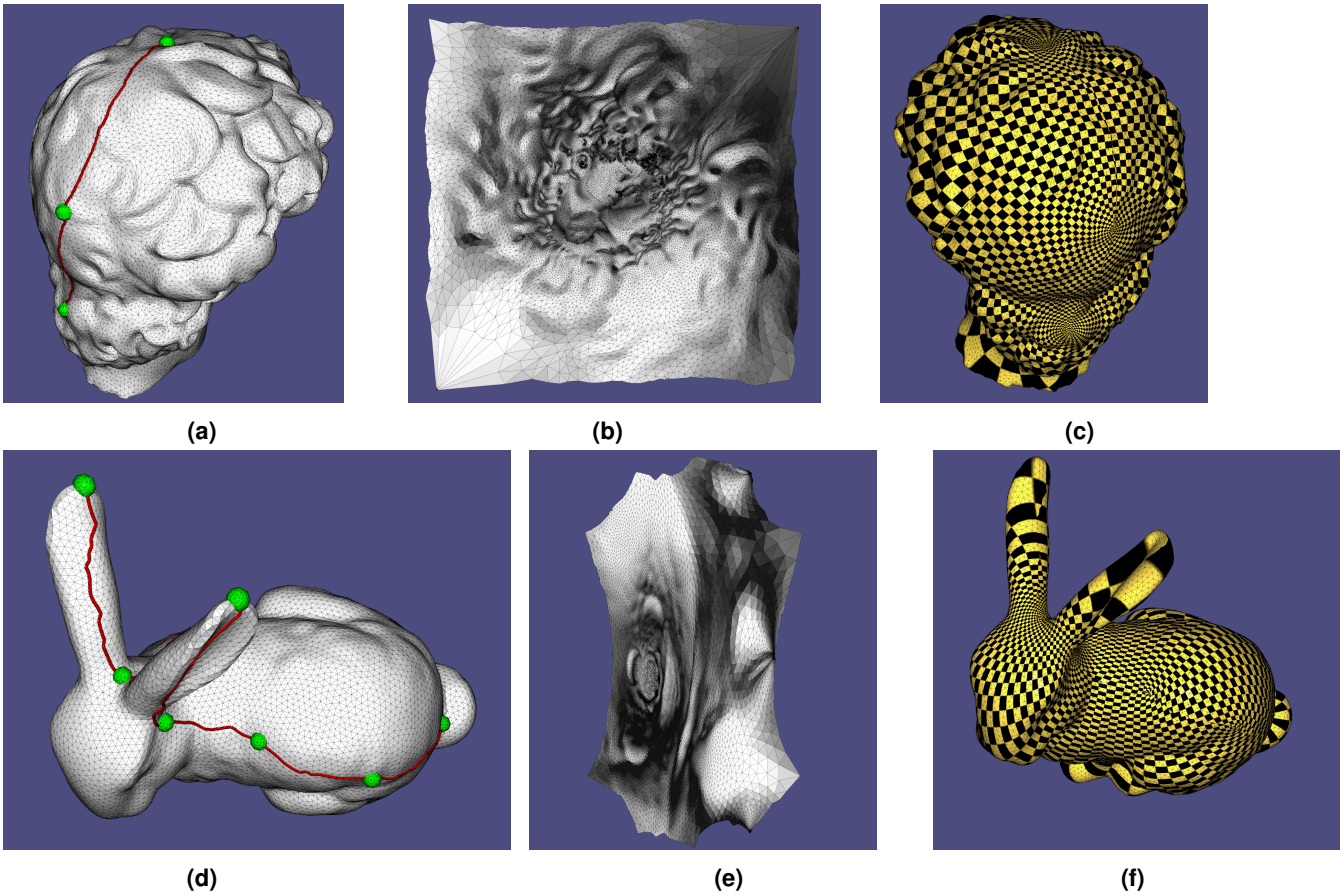


Figure 5. (a)(d) Models and their cones and slices; (b)(e) Orbifold Tutte embeddings; (c)(f) Texture mappings.

3.3 Implementation Details and Experiments

3.3.1 Vertex Selecting System.

I implement a vertex selecting system, so user can interactively choose cones and determine cone angles to get different orbifolds. The idea of this algorithm is as follows:

1) Find all visible vertices from the camera.

2) Compute all projected coordinates of all visible vertices.

3) Get position of mouse, find the nearest projected coordinates and then select the corresponding vertex.

3.3.2 Covering Space.

To say the structure of orbifold, I implement the tiling algorithm. This is an incremental algorithm which extends the big boundary of tiled area one copy by one copy. The pipeline is as follows:

1) Given a basic tile and segments, initiate ordered segment of tiled area sequence by initial segments.

2) In each iteration, find the nearest segment s to the origin. Find its equivalent segment s' in the basic tile. Compute the isometry between these two segments. Transform the basic tile using this isometry. Erase overlapped segments. Continue until the area is big enough.

To quickly find the nearest segment, I maintain a min heap besides the segment sequence.

3.3.3 Results.

I use the first kind of orbifold to parameterize David model, and use a hyperbolic orbifold with 7 cones to parameterize Bunny model. The result are shown in Fig.5.

Note that Euclidean orbifold Tutte embeddings with 3 cones are actually conformal map. It can be seen in Fig.5c that the angle are well preserved.

4 Boundary First Flattening

In this section, I will get into details of my project on boundary first flattening. The algorithm is based on differential geometry. This section is organized as follows: I will first introduce the theoretical background. Then I will give details on my implementation.

4.1 Theoretical Background

4.1.1 Conformal Maps

In complex analysis, conformal maps are equivlent to holomorphic maps, that is, the differentials of these map satisfy Cauchy-Riemann equation:

$$\frac{\partial f}{\partial \bar{z}} = 0 \tag{6}$$

One important property is that all comformal maps are harmonic maps, that is,

$$\Delta f = 0 \tag{7}$$

But the reverse direction is not true.

4.1.2 Poisson Problems

Poisson problem is stated as follows:

Dirichlet-type condition:

$$\begin{aligned} \Delta a &= \phi \qquad on\ M \\ a &= g, \qquad on\ \partial M \end{aligned} \tag{8}$$

Or Riemann-type condition:

$$\begin{aligned} \Delta a &= \phi \qquad on\ M \\ \frac{\partial a}{\partial n} &= h, \qquad on\ \partial M \end{aligned} \tag{9}$$

We will use discrete Poisson equation on a mesn M:

$$\begin{bmatrix} A_{II} & A_{IB} \\ A_{BI} & A_{BB} \end{bmatrix} \begin{bmatrix} a_I \\ a_B \end{bmatrix} = \begin{bmatrix} \phi_I \\ \phi_B - h \end{bmatrix} \tag{10}$$

where M is the cotangent Laplacian matrix.

Known A, Dirichlet-type conditions and Riemann-type conditions can be converted to each other: Given h, we simply solve the Poisson equation and set

$$g = \Lambda_\phi^* h = a_B, \tag{11}$$

Given g, we can convert it to h by

$$h = \Lambda_\phi g = \phi_B - A_{IB}^T A_{II}^{-1} (\phi_I - A_{IB} g) - A_{BB} g \tag{12}$$

4.1.3 Cherrier Formula

Cherrier formula [Cherrier, 1984] is the core fact which induces the BFF algorithm.

For a conformal map $f : M \to \tilde{M}$, we have

$$\begin{aligned} \Delta u &= K - e^{2u}\tilde{K} \quad on \qquad M \\ \frac{\partial u}{\partial n} &= k - e^{u}\tilde{k} \quad on \qquad \partial M \end{aligned} \tag{13}$$

where u is conformal factor, K,$\tilde{K}$ are Gauss curvature of the source surface and the target surface, $k, \tilde{k}$ is geodesic curvature of the source surface and the target surface.

The discrete version is the root of the algorithm:

$$\begin{aligned} Au &= \Omega - \tilde{\Omega} \quad on\ Int\ M \\ h &= k - k \qquad on\ \partial M \end{aligned} \tag{14}$$

where $\Omega_i = 2\pi - \sum_{ijk \in F} \theta_i^{jk}$ defined on interior vertices, $k_i = \pi - \sum_{ijk \in F} \theta_i^{jk}$ defined on boundary vertices, which is the exterior angle at v_i. Note that Ω is defined to be zero at boundary vertices.

5 Algorithm

The pipeline of the BFF algorithm is as follows:

1) If boundray conformal factor u is known, we compute bounadry target k by:

$$\tilde{k} = k - \Lambda_\Omega u.$$

If target boundary $\tilde{k}$ is known, we compute boundary conformal factor u by::

$$u = \Lambda_\Omega^*(k - \tilde{k}).$$

2) Define new boundary edge lengths as:

$$l_{ij}^* = e^{\frac{u_i+u_j}{2}} l_{ij}.$$

With $\tilde{k}$, which is exterior angles, we can integrate these two data into a closed loop.

3) Extend the loop into interior conformally.

5.1 Loop Integration

In step 2, usually, directly integrating $\tilde{k}$ over l_{ij}^* won't give a closed loop, we should change edge length a little. So we solve the following quadratic optimization problem:

$$\begin{aligned} &\min_{\tilde{l}} \ ||\tilde{l} - l^*||_2^2 \\ &s.t. \sum_{ij \in \partial M} \tilde{l}_i j \tilde{T}_{ij} \end{aligned} \tag{15}$$

Here, $\tilde{T}_{ij}$ is the tangent vector of edge ij. We use the following procedure to compute tangent vectors:

1) Assume boundary vertex list is $\{0, 1, 2, 3, ..., |l| - 1\}$. Compute cumulative angle as

$$\phi_k = \sum_{i=0}^{k-1} k_i.$$

2) Define tangent vectors as

$$\tilde{T}_{ij} = (\cos\phi_i, \sin\phi_j).$$

5.2 Conformal Interpolation

In step 3, we need to get interior coordinates from fixed boundary data.

We know that all conformal maps are harmonic maps. So if the boundary data is truely from a conformal map, we can get this conformal map using the solver for harmonic maps. So the first method is using harmonic mapping directly. **Harmonic system** is Eq.1 with cotangent weights. But since we've changed the boundary lengths, conformality may not well preserved.

Another method is to only use harmonic mapping on one component, and try to minimize conformal energy on the other component. This is done by Hilbert transform, which is defined by:

$$(Ha_B)_j = \frac{1}{2}(a_k - a_i) \tag{16}$$

where i, j, k are three consecutive vertices on the boundary.

Then we use Ha_B as boundary data to solve the harmonic system. The solution is the other component. The shortcoming for this method is that the boundary exterior angles can not be well preserved.

5.3 Implementation Details and Experiments

References

Aigerman et al., 2017. Aigerman, N., Kovalsky, S. Z., and Lipman, Y. (2017). Spherical orbifold tutte embeddings. *ACM Trans. Graph.*, 36(4):90:1–90:13.

Aigerman and Lipman, 2015. Aigerman, N. and Lipman, Y. (2015). Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190:1–190:12.

Aigerman and Lipman, 2016. Aigerman, N. and Lipman, Y. (2016). Hyperbolic orbifold tutte embeddings. *ACM Trans. Graph.*, 35(6):217:1–217:14.

Botsch et al., 2002. Botsch, M., Steinberg, S., Bischoff, S., and Kobbelt, L. (2002). Openmesh - a generic and efficient polygon mesh data structure. https://www.openmesh.org.

Cherrier, 1984. Cherrier, P. (1984). Problèmes de neumann non linéaires sur les variétés riemanniennes. *Journal of Functional Analysis*, 57(2):154 – 206.

Gortler et al., 2006. Gortler, S. J., Gotsman, C., and Thurston, D. (2006). Discrete one-forms on meshes and applications to 3d mesh parameterization. *Comput. Aided Geom. Des.*, 23(2):83–112.

Guennebaud et al., 2010. Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Jacobson et al., 2016. Jacobson, A., Panozzo, D., et al. (2016). libigl: A simple C++ geometry processing library. http://libigl.github.io/libigl/.

Qiu, 2016. Qiu, Y. (2016). Lbfgs++. https://github.com/yixuan/LBFGSpp.

Sawhney and Crane, 2017. Sawhney, R. and Crane, K. (2017). Boundary first flattening. https://arxiv.org/abs/1704.06873.

Tutte, 1963. Tutte, W. T. (1963). How to draw a graph. *Proceedings of the London Mathematical Society*, s3-13(1):743—-767.