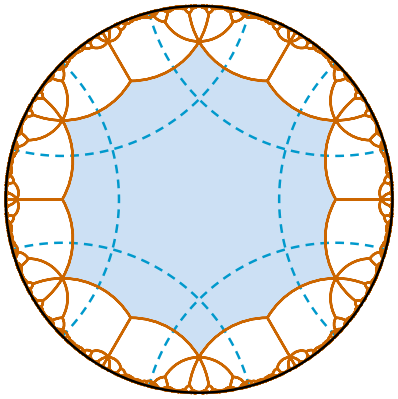
Variational Methods for Discrete Surface Parameterization. Applications and Implementation.

vorgelegt von
Diplom-Technomathematiker Stefan Sechelmann
geboren in Berlin

von der Fakultät II - Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation



Promotionsausschuss:

Vorsitzende:	Prof. Dr. Gitta Kutyniok
Gutachter:	Prof. Dr. Alexander I. Bobenko
Gutachter:	Prof. Dr. Max Wardetzky

Tag der wissenschaftlichen Aussprache: 25. April 2016

Berlin 2016

D 83

Contents

Introduction

The notion of discrete conformal equivalence of triangle meshes was introduced by Luo [45] and elaborated in detail by Springborn *et al.* [69] and Bobenko *et al.* [13]. Two combinatorially equivalent euclidean triangle meshes are conformally equivalent if there exist scale factors associated to vertices such that corresponding edge lengths are equal up to multiplication with adjacent scale factors. The problem of finding a discrete conformal map from a triangulated surface to the plane is formulated in terms of a variational principle using scale factors as variables.

Whereas the theoretical foundations of discrete conformal mappings via discrete conformal equivalence were arranged in previous work, this thesis focuses on the experimental side of the theory. In the spirit of discrete differential geometry we look at theorems and constructions from the geometry of Riemann surfaces and find analogous discrete constructions with similar properties. We investigate the generalization to cyclic polyhedral surfaces, which was only touched upon in previous work. We also give details on the spherical version of the theory that was put aside previously.

This thesis is divided into three parts.

Part I covers the discrete uniformization of Riemann surfaces via conformal equivalence of cyclic polyhedral surfaces. We generalize the previously established variational principles accordingly and present a wealth of constructions and examples. Taken from smooth differential geometry and translated to the discrete setting we observe properties known from classic theorems. This culminates in a theorem about hyperelliptic Riemann surfaces. A Riemann surface is hyperelliptic if and only if the axes of the uniformization group elements meet in a common point for a suitable basis of the group, see Figure 1 right. The tools created in the development of the methods used in this part lay the foundations for the applications presented in Part II.

In Part II we present three applications of discrete conformal mappings in the context of architectural geometry. In Chapter 2 we calculate regular patterns on architectural facade geometries with a period. We investigate how different boundary conditions affect the local behavior of the map. The resulting map is used to create new meshes with regular faces such as hexagons. We show how these meshes can be optimized towards a facade panel layout that can be fabricated in an efficient manner, see Figure 2 left.

In Chapter 3 we exploit the fact that isothermic surfaces possess a conformal curvature line parametrization to create meshes with planar quadrilaterals and touching incircles. This problem is formulated as a boundary value problem of discrete conformal maps. As a result we obtain circle pattern representations of surfaces relevant in the architectural context, see Figure 2 right.

In Chapter 4 we use discrete conformal maps as initialization for discrete Tschebyshev meshes,

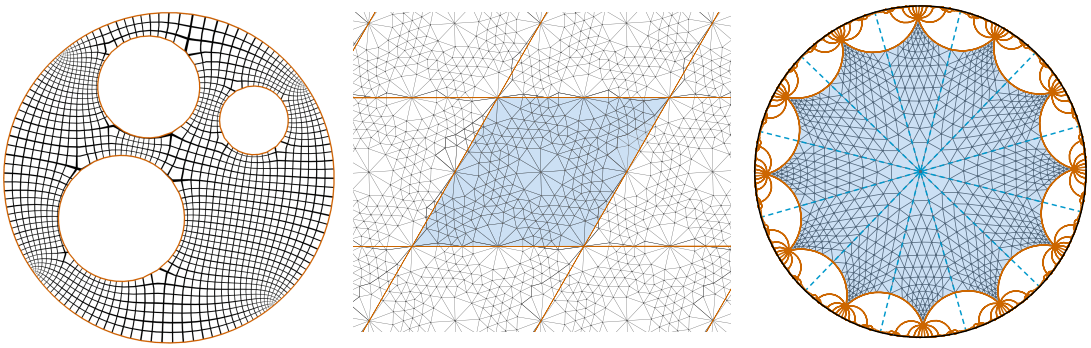


Figure 1: Discrete uniformizations of Riemann surfaces. Left: Region with four boundary components is mapped conformally to a domain bounded by circles, see Section 1.5.1. Middle: Uniformization of a discrete version of an elliptic curve. The image is a flat torus, see Section 1.7.2. Right: Uniformization of a hyperelliptic algebraic curve with a regular fundamental domain, see Section 1.8.3.
Diskrete Uniformisierungen von Riemannschen Flächen. Links: Ein Gebiet mit vier Randkomponenten wird auf einen von Kreisen begrenzten Bereich abgebildet, siehe Abschnitt 1.5.1. Mitte: Uniformisierung einer diskret elliptischen Kurve. Das Bild ist ein flacher Torus, siehe Abschnitt 1.7.2. Rechts: Uniformisierung einer hyperelliptischen algebraischen Kurve mit einem regulären Fundamentalpolygon, siehe Abschnitt 1.8.3.

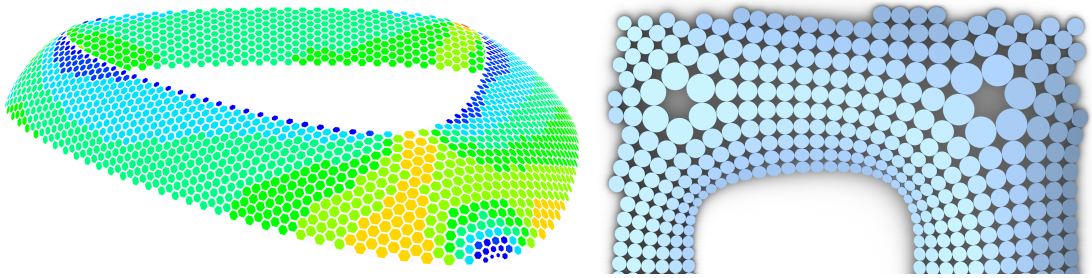


Figure 2: Two applications of discrete conformal mappings in architectural geometry. Left: A panel layout on a facade structure with size-quantized regular hexagons, see Chapter 2. Right: A circle pattern on a roof structure. We calculate discrete s-isothermic parameterizations for surfaces that are close to isothermic. The resulting meshes exhibit planar faces. The mesh layout follows the directions of principle curvature, see Chapter 3.
Zwei Anwendungen von diskret konformen Abbildungen in der Architekturgeometrie. Links: Ein Paneellayout auf einer architektonischen Fassadenoberfläche. Die Paneele sind größenquantisierte reguläre Sechsecke, siehe Kapitel 2. Rechts: Kreismuster auf einer Dachoberfläche. In Kapitel 3 berechnen wir diskrete s-isotherme Parametrisierungen für Flächen, die nahe an Isothermflächen sind. Die Ergebnisnetze haben ebene Facetten, die Kanten folgen den Hauptkrümmungsrichtungen der Fläche.

i.e., meshes with edges of equal length. Starting from a discrete conformal map, we perform non-linear optimization on quadrilateral meshes. In doing so we control the curvature and intersection angles of parameter curves in the resulting gridshell construction.

The third part of this work introduces the reader to the software framework built for calculation with discrete surfaces and in particular with discrete conformal mappings, see Chapter 8 about ConformalLab, and discrete surface optimization, Chapter 9 about VaryLab. Additionally the author has implemented the software packages HalfEdge and HalfEdgeTools, see Chapter 7, designed as a general tool for the calculation with discrete surfaces. Together with the user interface library JRWorkspace, Chapter 6, it constitutes a flexible framework for the creation of research applications in the context of discrete differential geometry.

The digital data accompanying this work includes most of the examples presented in Chapter 1 and the main files for the architectural applications. Additionally we include the current state of the source code of the programming framework presented in Part III. The organization of the data is presented in the appendix.

Deutsche Übersetzung

Diskret konforme Äquivalenz von Dreiecksnetzen wurde von Luo eingeführt [45] und von Springborn *et al.* [69] sowie Bobenko *et al.* [13] ausführlich behandelt. Zwei euklidische Triangulierungen mit gleicher Konnektivität sind diskret konform äquivalent, wenn es Faktoren pro Ecke gibt, so dass entsprechende Kanten bis auf Multiplikation mit den Faktoren die gleiche Länge haben. Die konforme Abbildung von triangulierten Flächen auf ebene Gebiete ist mit Hilfe eines Variationsprinzips auf den Skalierungsfaktoren formuliert.

Wir konzentrieren uns in dieser Abhandlung auf die experimentellen Aspekte der Theorie und stützen uns dabei auf die theoretischen Grundlagen aus den genannten Arbeiten. Ganz im Geiste der diskreten Differentialgeometrie untersuchen wir Aussagen und Konstruktionen aus der Differentialgeometrie von glatten Flächen und versuchen diese in den diskreten Strukturen wieder zu finden.

Diese Abhandlung ist in drei Teile aufgeteilt, wovon der erste die konformen Abbildungen behandelt, der andere Anwendungen der Selbigen in der Architektur enthält und der dritte die Implementation der verwendeten Methoden in einer Software beschreibt.

Im ersten Teil beschreiben wir die diskrete Uniformisierung von Riemannschen Flächen mittels diskret konformer Äquivalenz von zyklisch polyedrischen Flächen. Wir verallgemeinern dafür die bekannte Theorie der diskret konformen Dreiecksnetze auf zyklisch polyedrische Flächen. Dieser Teil der Theorie wurde in vorangegangenen Arbeiten nur kurz berührt. Weiter beleuchten wir den sphärischen Teil der Theorie, welcher bisher nicht behandelt worden ist. Dieser Teil enthält einen Satz über hyperelliptische Riemannsche Flächen: Eine Riemannsche Fläche ist hyperelliptisch genau dann, wenn es eine Basis der Uniformisierungsgruppe gibt, deren hyperbolische Achsen sich in einem gemeinsamen Punkt schneiden, siehe auch Abbildung 1 rechts. Die Methoden aus diesem Abschnitt bilden die Grundlage für die Anwendungen in Teil II.

Der zweite Teil enthält Anwendungen von konformen Abbildungen in der Architekturgeometrie. In Kapitel 2 werden reguläre Muster auf architektonischen Fassaden berechnet. Dafür verwenden wir periodische diskret konforme Abbildungen. Wir untersuchen, wie Randbedingungen das lokale Verhalten der Abbildungen beeinflussen. Am Ende verwenden wir diese Ab-

bildungen, um die Geometrie von regulären Paneelen, zum Beispiel Sechsecken, zu bestimmen. Wir beschreiben, wie diese Netze weiter optimiert werden um Paneele effizient produzieren zu können, siehe auch Abbildung 2 links.

In Kapitel 3 nutzen wir die Tatsache aus, dass Isothermflächen nach Krümmungslinien parametrisierbar sind. Wir berechnen s-isotherme Netze auf architektonischen Flächen, das sind Netze, deren ebene Facetten "sich an den Kanten berührende" Kreise besitzen. Dieses Problem ist als Randwertproblem von diskret konformen Abbildungen formuliert.

Kapitel 4 gibt eine Einführung in das Konzept der architektonischen Gitterschalen. Geometrisch sind das Netze mit gleich langen Kanten. Ausgehend von diskret konformen Abbildungen berechnen wir diese Strukturen mittels nicht-linearer Optimierung von Viereckснetzen.

Der dritte Teil führt den Leser in das Software-Framework ein, welches für Berechnungen an diskreten Flächen und speziell konformen Abbildungen entworfen wurde, siehe hierfür Kapitel 8 über ConformalLab. Der Teil, der sich mit nicht-linearer Optimierung von diskreten Flächen beschäftigt, ist im Kapitel 9 über VaryLab beschrieben.

Der Autor hat weiter die Bibliotheken HalfEdge and HalfEdgeTools entworfen, siehe Kapitel 7. Diese sind als allgemeine Werkzeuge für Berechnungen an diskreten Fächen konzipiert. Zusammen mit der Oberflächenbibliothek JRWorkspace, Kapitel 6, bilden sie ein flexibles Framework für den Entwurf komplexer Forschungsanwendungen im Bereich der diskreten Differentialgeometrie.

Diese Arbeit enthält eine CD mit den digitalen Daten der meisten Beispiele aus Kapitel 1 und den architektonischen Anwendungen aus Teil II. Zusätzlich ist der Quellcode der, in Teil III beschriebenen, Software in seiner aktuellen Fassung, sowie einer kompilierten, lauffähigen Version enthalten. Die Struktur dieser Daten ist im Anhang beschrieben.

Part I

Variational methods for discrete surface parameterization

Chapter 1

Discrete uniformization of Riemann surfaces

1.1 Introduction

In this chapter we investigate various applications of discrete conformal maps. It is a joint effort of Alexander Bobenko, Boris Springborn, and the author. Most of its content is also available in the article [14]. We build upon the work of Bobenko, Pinkall, and Springborn [13] and Springborn, Schröder, and Pinkall [69]. We present examples for uniformizations of Riemann surfaces of genus 0, 1, and $g > 1$ based on this theory, see Figure 1.1.

The notion of discrete conformal equivalence for polyhedral surfaces is based on a simple definition: Two polyhedral surfaces are discretely conformally equivalent if the edge lengths are related by scale factors assigned to the vertices. It leads to a surprisingly rich theory [45, 13, 28, 29].

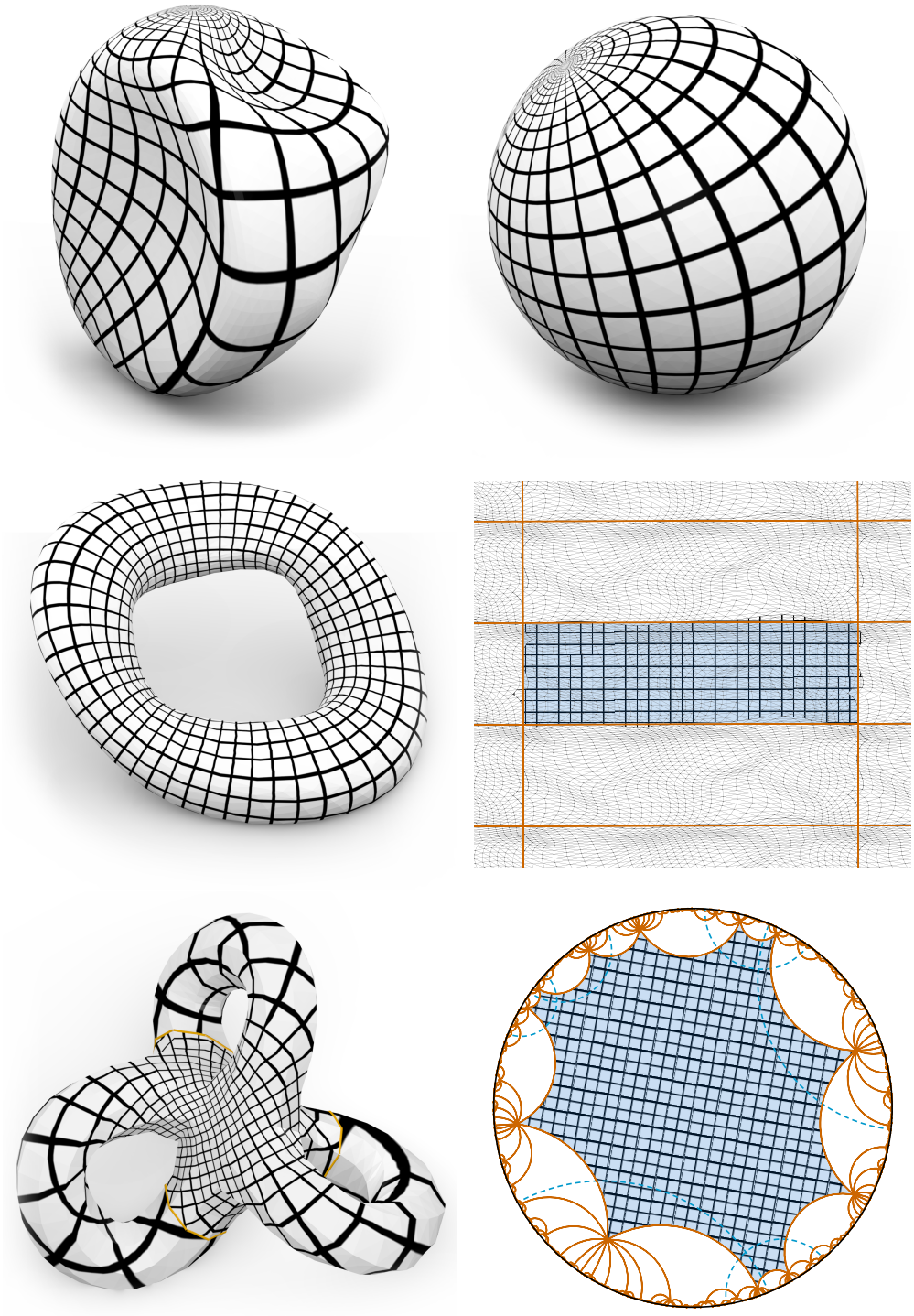
We extend the notion of discrete conformal equivalence from triangulated surfaces to polyhedral surfaces with faces that are inscribed in circles. The basic definitions and their immediate consequences are discussed in Section 1.2.

In Section 1.3, we generalize a variational principle for discretely conformally equivalent triangulations [13] to the polyhedral setting. This variational principle is the main tool for all our numerical calculations. It is also the basis for our uniqueness proof for discrete conformal mapping problems (Theorem 1.3.9).

Section 1.4 is concerned with the special case of quadrilateral meshes. We discuss the emergence of orthogonal circle patterns, a peculiar necessary condition for the existence of solutions for boundary angle problems, and we extend the method of constructing discrete Riemann maps from triangulations to quadrangulations.

In Section 1.5, we briefly discuss discrete conformal maps from multiply connected domains to circle domains, and special cases in which we can map to slit domains.

Section 1.6 deals with conformal mappings onto the sphere. We generalize the method for triangulations to quadrangulations, and we explain how the spherical version of the variational principle can in some cases be used for numerical calculations although the corresponding functional is not convex.



data/introduction/genus{0/1/3}_data.xml

Figure 1.1: Uniformization of compact Riemann surfaces. The uniformization of spheres is treated in Section 1.6. Tori are covered in Section 1.7, and Section 1.8 is concerned with surfaces of higher genus.

Section 1.7 is concerned with the uniformization of tori, i.e., the representation of Riemann surfaces as a quotient space of the complex plane modulo a period lattice. We consider Riemann surfaces represented as immersed surfaces in $\mathbb{R}^3$, and as elliptic curves. We conduct numerical experiments to test the conjectured convergence of discrete conformal maps. We consider the difference between the true modulus of an elliptic curve (which can be calculated using hypergeometric functions) and the modulus determined by discrete uniformization, and we estimate the asymptotic dependence of this error on the number of vertices.

In Section 1.8, we consider the Fuchsian uniformization of Riemann surfaces represented in different forms. We consider immersed surfaces in $\mathbb{R}^3$ (and S^3), hyperelliptic curves, and Riemann surfaces represented as a quotient of $\hat{\mathbb{C}}$ modulo a classical Schottky group. That is, we convert from Schottky uniformization to Fuchsian uniformization. The Section ends with two extended examples demonstrating, among other things, a remarkable geometric characterization of hyperelliptic surfaces due to Schmutz Schaller.

This text is accompanied by a compact disk which contains the data for all of the examples presented in this part of the work. Where applicable, we give the path to the data on the disk directly under the corresponding figure. We describe the usage of the software and the XML data format that was used to create all the results in Chapter 8.

1.2 Discrete conformal equivalence of cyclic polyhedral surfaces

1.2.1 Cyclic polyhedral surfaces

A *euclidean polyhedral surface* is a surface obtained from gluing euclidean polygons along their edges. (A *surface* is a connected two-dimensional manifold, possibly with boundary.) In other words, a euclidean polyhedral surface is a surface equipped with, first, an intrinsic metric that is flat except at isolated points where it has cone-like singularities, and, second, the structure of a CW complex with geodesic edges. The set of vertices contains all cone-like singularities. If the surface has a boundary, the boundary is polygonal and the set of vertices contains all corners of the boundary.

Hyperbolic polyhedral surfaces and *spherical polyhedral surfaces* are defined analogously. They are glued from polygons in the hyperbolic and elliptic planes, respectively. Their metric is locally hyperbolic or spherical, except at cone-like singularities.

We will only be concerned with polyhedral surfaces whose faces are all cyclic, i.e., inscribed in circles. We call them *cyclic polyhedral surfaces*. More precisely, we require the polygons to be cyclic before they are glued together. It is not required that the circumcircles persist after gluing; they may be disturbed by cone-like singularities. A polygon in the hyperbolic plane is considered cyclic if it is inscribed in a curve of constant curvature. This may be a circle (the locus of points at constant distance from its center), a horocycle, or a curve at constant distance from a geodesic.

A *triangulated surface*, or *triangulation* for short, is a polyhedral surface all of whose faces are triangles. All triangulations are cyclic.

1.2.2 Notation

We will denote the sets of vertices, edges, and faces of a CW complex Σ by V_Σ, E_Σ, and F_Σ, and we will often omit the subscript when there is no danger of confusion. For notational convenience, we require all CW complexes to be *strongly regular*. This means that we require that faces are not glued to themselves along edges or at vertices, that two faces are not glued together along more than one edge or one vertex, and that edges have distinct end-points and two edges have at most one endpoint in common. This allows us to label edges and faces by their vertices. We will write $ij \in E$ for the edge with vertices $i, j \in V$ and $ijkl \in F$ for the face with vertices $i, j, k, l \in V$. We will always list the vertices of a face in the correct cyclic order, so that for example the face $ijkl$ has edges ij, jk, kl, and li. The only reason for restricting our discussion to strongly regular CW complexes is to be able to use this simple notation. Everything we discuss applies also to general CW complexes.

1.2.3 Discrete metrics

The *discrete metric* of a euclidean (or hyperbolic or spherical) cyclic polyhedral surface Σ is the function $\ell : E_\Sigma \to \mathbb{R}_{>0}$ that assigns to each edge $ij \in E_\Sigma$ its length ℓ_{ij}. It satisfies the polygon inequalities (one side is shorter than the sum of the others):

$$\left.\begin{array}{r} -\ell_{i_1 i_2} + \ell_{i_2 i_3} + \ldots + \ell_{i_{n-1} i_n} > 0 \\ \ell_{i_1 i_2} - \ell_{i_2 i_3} + \ldots + \ell_{i_{n-1} i_n} > 0 \\ \vdots \\ \ell_{i_1 i_2} + \ell_{i_2 i_3} + \ldots - \ell_{i_{n-1} i_n} > 0 \end{array}\right\} \quad \text{for all } i_1 i_2 \ldots i_n \in F_\Sigma \tag{1.1}$$

In the case of spherical polyhedral surfaces, we also require that

$$\ell_{i_1 i_2} + \ell_{i_2 i_3} + \ldots + \ell_{i_{n-1} i_n} < 2\pi. \tag{1.2}$$

The polygon inequalities (1.1) are necessary and sufficient for the existence of a unique cyclic euclidean polygon and a unique cyclic hyperbolic polygon with the given edge lengths. Together with inequality (1.2) they are necessary and sufficient for the existence of a unique cyclic spherical polygon. For a new proof of these elementary geometric facts, see [38]. Thus, a discrete metric determines the geometry of a cyclic polyhedral surface:

Proposition and Definition 1.2.1. *If Σ is a surface with the structure of a CW complex and a function $\ell : E_\Sigma \to \mathbb{R}_{>0}$ satisfies the polygon inequalities* (1.1)*, then there is a unique euclidean cyclic polyhedral surface and also a unique hyperbolic cyclic polyhedral surface with CW complex Σ and discrete metric ℓ. If ℓ also satisfies the inequalities* (1.2)*, then there is a unique spherical cyclic polyhedral surface with CW complex Σ and discrete metric ℓ.*

We will denote the euclidean, hyperbolic, and spherical polyhedral surface with CW complex Σ and discrete metric ℓ by $(\Sigma, \ell)_{euc}$, $(\Sigma, \ell)_{hyp}$, and $(\Sigma, \ell)_{sph}$, respectively.

1.2.4 Discrete conformal equivalence

We extend the definition of discrete conformal equivalence from triangulations [45, 13] to cyclic polyhedral surfaces in a straightforward way (Definition 1.2.2). While some aspects of the theory

carry over to the more general setting (e.g., Möbius invariance, Proposition 1.2.5), others do not, like the characterization of discretely conformally equivalent triangulations in terms of length cross-ratios (Section 1.2.5). We will discuss similar characterizations for polyhedral surfaces with 2-colorable vertices and the particular case of quadrilateral faces in Sections 1.2.7 and 1.2.8.

We define discrete conformal equivalence only for polyhedral surfaces that are combinatorially equivalent (see Remark 1.2.4). Thus, we may assume that the surfaces share the same CW complex Σ equipped with different metrics ℓ, $\tilde{\ell}$.

Definition 1.2.2. Discrete conformal equivalence *is an equivalence relation on the set of cyclic polyhedral surfaces defined as follows:*

- *Two* euclidean *cyclic polyhedral surfaces* $(\Sigma, \ell)_{euc}$ *and* $(\Sigma, \tilde{\ell})_{euc}$ *are* discretely conformally equivalent *if there exists a function* $u : V_\Sigma \to \mathbb{R}$ *such that*

$$\tilde{\ell}_{ij} = e^{\frac{1}{2}(u_i+u_j)}\ell_{ij}. \tag{1.3}$$

- *Two* hyperbolic *cyclic polyhedral surfaces* $(\Sigma, \ell)_{hyp}$ *and* $(\Sigma, \tilde{\ell})_{hyp}$ *are* discretely conformally equivalent *if there exists a function* $u : V_\Sigma \to \mathbb{R}$ *such that*

$$\sinh\Big(\frac{\tilde{\ell}_{ij}}{2}\Big) = e^{\frac{1}{2}(u_i+u_j)}\,\sinh\Big(\frac{\ell_{ij}}{2}\Big). \tag{1.4}$$

- *Two* spherical *cyclic polyhedral surfaces* $(\Sigma, \ell)_{sph}$ *and* $(\Sigma, \tilde{\ell})_{sph}$ *are* discretely conformally equivalent *if there exists a function* $u : V_\Sigma \to \mathbb{R}$ *such that*

$$\sin\Big(\frac{\tilde{\ell}_{ij}}{2}\Big) = e^{\frac{1}{2}(u_i+u_j)}\,\sin\Big(\frac{\ell_{ij}}{2}\Big). \tag{1.5}$$

We will also consider mixed versions:

- *A euclidean cyclic polyhedral surface* $(\Sigma, \ell)_{euc}$ *and a hyperbolic cyclic polyhedral surface* $(\Sigma, \tilde{\ell})_{hyp}$ *are discretely conformally equivalent if*

$$\sinh\Big(\frac{\tilde{\ell}_{ij}}{2}\Big) = e^{\frac{1}{2}(u_i+u_j)}\ell_{ij}. \tag{1.6}$$

- *A euclidean cyclic polyhedral surface* $(\Sigma, \ell)_{euc}$ *and a spherical cyclic polyhedral surface* $(\Sigma, \tilde{\ell})_{sph}$ *are discretely conformally equivalent if*

$$\sin\Big(\frac{\tilde{\ell}_{ij}}{2}\Big) = e^{\frac{1}{2}(u_i+u_j)}\ell_{ij}. \tag{1.7}$$

- *A hyperbolic cyclic polyhedral surface* $(\Sigma, \ell)_{hyp}$ *and a spherical cyclic polyhedral surface* $(\Sigma, \tilde{\ell})_{sph}$ *are discretely conformally equivalent if*

$$\sin\Big(\frac{\tilde{\ell}_{ij}}{2}\Big) = e^{\frac{1}{2}(u_i+u_j)}\,\sinh\Big(\frac{\ell_{ij}}{2}\Big). \tag{1.8}$$

Remark 1.2.3. *Note that relation* (1.5) *for spherical edge lengths is equivalent to relation* (1.3) *for the euclidean lengths of the chords in the ambient* $\mathbb{R}^3$ *of the sphere (see Figure 1.2, left). Likewise, relation* (1.4) *for hyperbolic edge lengths is equivalent to* (1.3) *for the euclidean lengths of the chords in the ambient* $\mathbb{R}^{2,1}$ *of the hyperboloid model of the hyperbolic plane (see Figure 1.2, right).*

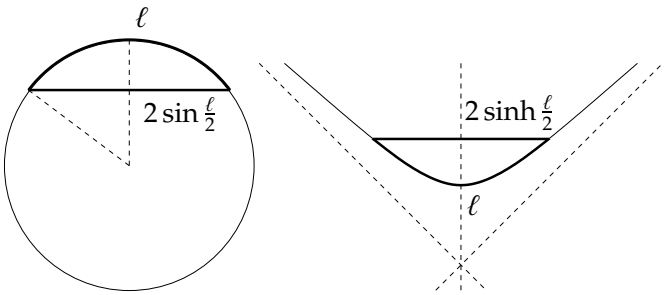


Figure 1.2: Spherical and hyperbolic chords.

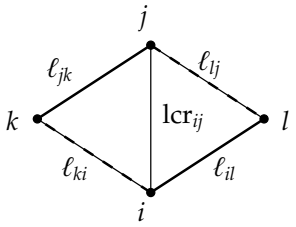


Figure 1.3: Length cross-ratio.

Remark 1.2.4. *For triangulations, the definition of discrete conformal equivalence has been extended to meshes that are not combinatorially equivalent [13, Definition 5.1.4] [28, 29]. It is not clear whether or how the following definitions for cyclic polyhedral surfaces can be extended to combinatorially inequivalent CW complexes.*

The discrete conformal class of a cyclic polyhedral surface embedded in n-dimensional euclidean space is invariant under Möbius transformations of the ambient space:

Proposition 1.2.5 (Möbius invariance)**.** *Suppose P and $\tilde{P}$ are two combinatorially equivalent euclidean cyclic polyhedral surfaces embedded in $\mathbb{R}^n$ (with straight edges and faces), and suppose there is a Möbius transformation of $\mathbb{R}^n \cup \{\infty\}$ that maps the vertices of P to the corresponding vertices of $\tilde{P}$. Then P and $\tilde{P}$ are discretely conformally equivalent.*

Note that only vertices are related by the Möbius transformation, not edges and faces, which remain straight. The simple proof for the case of triangulations [13] carries over without change.

1.2.5 Triangulations: Characterization by length cross-ratios

For euclidean triangulations, there is an alternative characterization of conformal equivalence in terms of length cross-ratios [13]. We review the basic facts in this section.

For two adjacent triangles $ijk \in F$ and $jil \in F$ (see Figure 1.3), the *length cross-ratio* of the common interior edge $ij \in E$ is defined as

$$\mathrm{lcr}_{ij} = \frac{\ell_{il}\ell_{jk}}{\ell_{lj}\ell_{ki}}. \tag{1.9}$$

(If the two triangles are embedded in the complex plane, this is just the modulus of the complex cross-ratio of the four vertices.) This definition of length cross-ratios implicitly assumes that an orientation has been chosen on the surface. For non-orientable surfaces, the length cross-ratio is well-defined on the oriented double cover.

The product of length cross-ratios around an interior vertex $i \in V$ is 1, because all lengths cancel:

$$\prod_{ij \ni i} \mathrm{lcr}_{ij} = 1. \tag{1.10}$$

Proposition 1.2.6. *Two euclidean triangulations $(\Sigma, \ell)_{euc}$ and $(\Sigma, \tilde{\ell})_{euc}$ are discretely conformally equivalent if and only if for each interior edge $ij \in E_\Sigma^{int}$, the induced length cross-ratios agree.*

Remark 1.2.7. *Analogous statements hold for spherical and hyperbolic triangulations. Equation* (1.9) *has to be modified by replacing ℓ with* $\sin\frac{\ell}{2}$ *or* $\sinh\frac{\ell}{2}$*, respectively (compare Remark 1.2.3).*

1.2.6 Triangulations: Reconstructing lengths from length cross-ratios

To deal with Riemann surfaces that are given in terms of Schottky data (Section 1.8.2) we will need to reconstruct a function $\ell : E_\Sigma \to \mathbb{R}_{>0}$ satisfying (1.9) from given length cross-ratios. (It is not required that the function ℓ satisfies the triangle inequalities.) To this end, we define auxiliary quantities c^i_{jk} attached to the angles of the triangulation. The value at vertex i of the triangle $ijk \in F$ is defined as

$$c^i_{jk} = \frac{\ell_{jk}}{\ell_{ij}\ell_{ki}}. \tag{1.11}$$

Then (1.9) is equivalent to

$$\mathrm{lcr}_{ij} = \frac{c^i_{jk}}{c^i_{lj}}. \tag{1.12}$$

Now, given a function $\mathrm{lcr} : E^{int} \to \mathbb{R}_{>0}$ defined on the set of interior edges E^{int} and satisfying the product condition (1.10) around interior vertices, one can find parameters c^i_{jk} satisfying (1.11) by choosing one value at each vertex and then successively multiplying length cross-ratios. The corresponding function ℓ is then determined by

$$\ell_{ij} = \frac{1}{\sqrt{c^i_{jk}c^j_{ki}}} = \frac{1}{\sqrt{c^i_{lj}c^j_{il}}}. \tag{1.13}$$

1.2.7 Bipartite graphs: Characterization by length multi-ratios

A different characterization of discrete conformal equivalence in terms of length multi-ratios holds if the 1-skeleton of the polyhedral surface is bipartite, i.e., if the vertices can be colored with two colors so that no two neighboring vertices share the same color.

Proposition 1.2.8. *(i) If two combinatorially equivalent euclidean cyclic polyhedral surfaces $(\Sigma, \ell)_{euc}$ and $(\Sigma, \tilde\ell)_{euc}$ with discrete metrics ℓ and $\tilde\ell$ are discretely conformally equivalent, then the* length multi-ratios *for even cycles*

$$i_1i_2, i_2i_3, \dots, i_{2n}i_1$$

are equal:

$$\frac{\ell_{i_1i_2}\ell_{i_3i_4}\cdots\ell_{i_{2n-1}i_{2n}}}{\ell_{i_2i_3}\ell_{i_4i_5}\cdots\ell_{i_{2n}i_1}} = \frac{\tilde\ell_{i_1i_2}\tilde\ell_{i_3i_4}\cdots\tilde\ell_{i_{2n-1}i_{2n}}}{\tilde\ell_{i_2i_3}\tilde\ell_{i_4i_5}\cdots\tilde\ell_{i_{2n}i_1}}\,. \tag{1.14}$$

(ii) If the 1-skeleton of Σ is bipartite, i.e., if all cycles are even, then this condition is also sufficient: If the length multi-ratios are equal for all cycles, then the polyhedral surfaces are discretely conformally equivalent.

Proof. (i) This is obvious, because all scale factors e^u cancel. (ii) It is easy to see that equations (1.3) can be solved for the scale factors $e^{u/2}$ if the length multi-ratios are equal. Note that the scale factors are not uniquely determined: They can be multiplied by λ and $1/\lambda$ on the two vertex color classes, respectively. To find a particular solution, one can fix the value of $e^{u/2}$ at one vertex,

and find the other values by alternatingly dividing and multiplying by $\tilde{\ell}/\ell$ along paths. The equality of length multi-ratios implies that the obtained values do not depend on the path. □

Remark 1.2.9. *If a polyhedral surface is simply connected, then its* 1*-skeleton is bipartite if and only if all faces are even polygons. If a polyhedral surface is not simply connected, then having even faces is only a necessary condition for being bipartite.*

A polyhedral surface with bipartite 1-skeleton has even faces. If a polyhedral surface has even faces and is simply connected, then its 1-skeleton is bipartite, and the face boundaries generate all cycles. Thus, Proposition 1.2.8 implies the following corollary.

Corollary 1.2.10. *Two simply connected combinatorially equivalent euclidean cyclic polyhedral surfaces with even faces and with discrete metrics ℓ and $\tilde{\ell}$ are discretely conformally equivalent if and only if the multi-ratio condition* (1.14) *holds for every face boundary cycle.*

Remark 1.2.11. *Analogous statements hold for spherical and hyperbolic cyclic polyhedral surfaces. In the multi-ratio condition, one has to replace non-euclidean lengths ℓ with* $\sin\frac{\ell}{2}$ *or* $\sinh\frac{\ell}{2}$*, respectively (compare Remark 1.2.3).*

1.2.8 Quadrangulations: Cross-ratio system on quad-graphs

The case of cyclic quadrilateral faces is somewhat special (and we will return to it in Section 1.4), because equal length cross-ratio implies equal complex cross-ratio:

Proposition 1.2.12. *If two euclidean polyhedral surfaces with cyclic quadrilateral faces are discretely conformally equivalent, then corresponding faces $ijkl \in F$ have the same complex cross-ratio (when embedded in the complex plane):*

$$\frac{(z_i - z_j)(z_k - z_l)}{(z_j - z_k)(z_l - z_i)} = \frac{(\tilde{z}_i - \tilde{z}_j)(\tilde{z}_k - \tilde{z}_l)}{(\tilde{z}_j - \tilde{z}_k)(\tilde{z}_l - \tilde{z}_i)}$$

Proof. This follows immediately from Proposition 1.2.8: The length multi-ratio of a quadrilateral is the modulus of the complex cross-ratio. If the (embedded) quadrilaterals are cyclic, then their complex cross-ratios are real and negative, so their arguments are also equal. □

For planar polyhedral surfaces, i.e., for quadrangulations in the complex plane, Proposition 1.2.12 connects discrete conformality with the cross-ratio system on quad-graphs. A *quad-graph* in the most general sense is simply an abstract CW cell decomposition of a surface with quadrilateral faces. Often, more conditions are added to the definition as needed. Here, we will require that the surface is oriented and that the vertices are bicolored black and white. For simplicity, we will also assume that the CW complex is strongly regular (see Section 1.2.2). The *cross-ratio system* on a quad-graph Σ imposes equations (1.15) on variables z_i that are attached to the vertices $i \in V_\Sigma$. There is one equation per face $ijkl \in F_\Sigma$:

$$\frac{(z_i - z_j)(z_k - z_l)}{(z_j - z_k)(z_l - z_i)} = Q_{ijkl}, \tag{1.15}$$

where we assume that i is a black vertex and the boundary vertices $ijkl$ are listed in the positive cyclic order. (Here we need the orientation). On the right hand side of the equation, $Q : F_\Sigma \to \mathbb{C} \setminus \{0, 1\}$ is a given function. In particular, it is required that the values z_i, z_j, z_k, z_l on a face are distinct.

By Proposition 1.2.12, two discretely conformally equivalent planar quadrangulations correspond to two solutions of the cross-ratio system on the same quad-graph with the same cross-ratios Q. The following proposition says that in the simply connected case, one can find complex factors w on the vertices whose absolute values $|w| = e^{u/2}$ govern the length change of edges according to (1.3), and whose arguments govern the rotation of edges. Note that (1.3) is obtained from (1.16) by taking absolute values.

Proposition 1.2.13. *Let Σ be a simply connected quad-graph. Two functions $z, \tilde{z} : V_\Sigma \to \mathbb{C}$ are solutions of the cross-ratio system on Σ with the same cross-ratios Q if and only if there is a function $w : V_\Sigma \to \mathbb{C}$ such that for all edges $ij \in E_\Sigma$*

$$\tilde{z}_j - \tilde{z}_i = w_i w_j (z_j - z_i). \tag{1.16}$$

Proof. As in the proof of Proposition 1.2.8, it is easy to see that the system of equations (1.16) is solvable for w if and only if the complex multi-ratios for even cycles are equal. Because Σ is simply connected, this is the case if and only if the complex cross-ratios of corresponding faces are equal. □

Remark 1.2.14. *The cross-ratio system on quad-graphs* (1.15) *is an integrable system (in the sense of 3D consistency [15, 16]) if the cross-ratios Q "factor", i.e., if there exists a function on the set of edges, $a : E_\Sigma \to \mathbb{C}$, that satisfies the following conditions for each quadrilateral $ijkl \in F$:*

(i) It takes the same value on opposite edges,

$$a_{ij} = a_{kl}, \quad a_{jk} = a_{li}. \tag{1.17}$$

(ii)

$$Q_{ijkl} = \frac{a_{ij}}{a_{jk}}\,. \tag{1.18}$$

In Adler, Bobenko & Suris' classification of integrable equations on quad-graphs [2], the integrable cross-ratio system is called $(\mathrm{Q1})_{\delta=0}$. It is also known as the discrete Schwarzian Korteweg–de Vries (dSKdV) equation, especially when it is considered on the regular square lattice [51] with constant cross-ratios.

If the cross-ratios Q have unit modulus, the cross-ratio system on quad-graphs is connected with circle patterns with prescribed intersection angles [15, 16].

Remark 1.2.15. *The system of equations* (1.16) *is also connected with an integrable system on quad-graphs. Let $b_{ij} = z_j - z_i$, so b is a function on the oriented edges with $b_{ij} = -b_{ji}$. Let us also assume that the quad-graph Σ is simply connected. Then the system* (1.16) *defines a function $z : V \to \mathbb{C}$ (uniquely up to an additive constant) if and only if the complex scale factors $w : V_\Sigma \to \mathbb{C}$ satisfy, for each face $ijkl \in F$ the closure condition*

$$b_{ij} w_i w_j + b_{jk} w_j w_k + b_{kl} w_k w_l + b_{li} w_l w_i = 0. \tag{1.19}$$

This system for w is integrable if, for each face $ijkl \in F$,

$$b_{ij} + b_{kl} = 0 \quad \textit{and} \quad b_{jk} + b_{li} = 0.$$

In this case, (1.19) *is known as discrete modified Korteweg–de Vries (dmKdV) equation [51], or as Hirota equation [15, 16].*

1.3 Variational principles for discrete conformal maps

1.3.1 Discrete conformal mapping problems

We will consider the following discrete conformal mapping problems. (The notation $(\Sigma, \ell)_g$ was introduced in Definition 1.2.1.)

Problem 1.3.1 (prescribed angle sums)**.** ***Given***

- *A euclidean, spherical, or hyperbolic cyclic polyhedral surface* $(\Sigma, \ell)_g$*, where* $g \in \{euc, hyp, sph\}$*,*
- *a desired total angle* $\Theta_i > 0$ *for each vertex* $i \in V_\Sigma$*,*
- *a choice of geometry* $\tilde{g} \in \{euc, hyp, sph\}$*,*

find *a discretely conformally equivalent cyclic polyhedral surface* $(\Sigma, \tilde{\ell})_{\tilde{g}}$ *of geometry* $\tilde{g}$ *that has the desired total angles* Θ *around vertices.*

For interior vertices, Θ prescribes a desired cone angle. For boundary vertices, Θ prescribes a desired interior angle of the polygonal boundary. If $\Theta_i = 2\pi$ for all interior vertices i, then Problem 1.3.1 asks for a flat metric in the discrete conformal class, with prescribed boundary angles if the surface has a boundary.

More generally, we will consider the following problem, where the logarithmic scale factors u (see Definition 1.2.2) are fixed at some vertices and desired angle sums Θ are prescribed at the other vertices. The problems to find discrete Riemann maps (Section 1.4.2) and maps onto the sphere (Section 1.6.1) can be reduced to this mapping problem with some fixed scale factors.

Problem 1.3.2 (prescribed scale factors and angle sums)**.** ***Given***

- *A euclidean, spherical, or hyperbolic cyclic polyhedral surface* $(\Sigma, \ell)_g$*, where* $g \in \{euc, hyp, sph\}$*,*
- *a partition* $V_\Sigma = V_0 \dot\cup V_1$
- *a prescribed angle* $\Theta_i > 0$ *for each vertex* $i \in V_1$*,*
- *a prescribed logarithmic scale factor* $u_i \in \mathbb{R}$ *for each vertex* $i \in V_0$*,*
- *a choice of geometry* $\tilde{g} \in \{euc, hyp, sph\}$*,*

find *a discretely conformally equivalent cyclic polyhedral surface* $(\Sigma, \tilde{\ell})_{\tilde{g}}$ *of geometry* $\tilde{g}$ *that has the desired total angles* Θ *around vertices in* V_1 *and the fixed scale factors* u *at vertices in* V_0*.*

Note that for $V_0 = \emptyset$, $V_1 = V$, Problem 1.3.2 reduces to Problem 1.3.1.

1.3.2 Analytic formulation of the mapping problems

We rephrase the mapping Problem 1.3.2 analytically as Problem 1.3.4. The sides of a cyclic polygon determine its angles, but practical explicit equations for the angles as functions of the sides exist only for triangles, e.g., (1.21). For this reason it makes sense to triangulate the polyhedral surface. For the angles in a triangulation, we use the notation shown in Figure 1.4. In triangle ijk, we denote the angle at vertex i by α^i_{jk}. We denote by β^i_{ij} the angle between the circumcircle and the edge jk. The angles α and β are related by

$$\alpha^i_{jk} + \beta^j_{ki} + \beta^k_{ij} = \pi,$$

so betas determine alphas and vice versa:

$$2\beta^i_{jk} = \pi + \alpha^i_{jk} - \alpha^j_{ki} - \alpha^k_{ij}, \quad \ldots \tag{1.20}$$

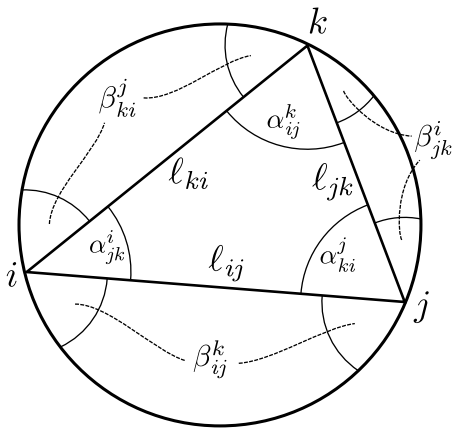


Figure 1.4: Notation of lengths and angles in a triangle $ijk \in F$.

For euclidean triangles,

$$\alpha^i_{jk} + \alpha^j_{ki} + \alpha^k_{ij} = \pi, \qquad \beta^i_{jk} = \alpha^i_{jk}.$$

The half-angle equation can be used to express the angles as functions of lengths:

$$\tan\left(\frac{\alpha^i_{jk}}{2}\right) = \begin{cases} \left(\dfrac{(-\ell_{ij}+\ell_{jk}+\ell_{ki})(\ell_{ij}+\ell_{jk}-\ell_{ki})}{(\ell_{ij}-\ell_{jk}+\ell_{ki})(\ell_{ij}+\ell_{jk}+\ell_{ki})}\right)^{\frac{1}{2}} & (euc) \\ \left(\dfrac{\sinh\left((\ell_{ij}-\ell_{jk}+\ell_{ki})/2\right)\sinh\left((\ell_{ij}+\ell_{jk}-\ell_{ki})/2\right)}{\sinh\left((-\ell_{ij}+\ell_{jk}+\ell_{ki})/2\right)\sinh\left((\ell_{ij}+\ell_{jk}+\ell_{ki})/2\right)}\right)^{\frac{1}{2}} & (hyp) \\ \left(\dfrac{\sin\left((\ell_{ij}-\ell_{jk}+\ell_{ki})/2\right)\sin\left((\ell_{ij}+\ell_{jk}-\ell_{ki})/2\right)}{\sin\left((-\ell_{ij}+\ell_{jk}+\ell_{ki})/2\right)\sin\left((\ell_{ij}+\ell_{jk}+\ell_{ki})/2\right)}\right)^{\frac{1}{2}} & (sph) \end{cases} \tag{1.21}$$

Lemma 1.3.3 (analytic formulation of Problem 1.3.2)**.** *Let*

- *the polyhedral surface* $(\Sigma, \ell)_g$,
- *the partition* $V_0 \dot\cup V_1$,
- Θ_i *for* $i \in V_1$,
- u_i *for* $i \in V_0$,
- *the geometry* $\tilde g \in \{euc, hyp, sph\}$

be given as in Problem 1.3.2. Let Δ *be an abstract triangulation obtained by adding non-crossing diagonals to non-triangular faces of* Σ*. (So* $V_\Sigma = V_\Delta$*,* $E_\Sigma \subseteq E_\Delta$*, and the set of added diagonals is* $E_\Delta \setminus E_\Sigma$*.) For* $ij \in E_\Sigma$*, define* λ_{ij} *by*

$$\lambda_{ij} = \begin{cases} 2\log \ell_{ij} & \text{if} \quad g = euc \\ 2\log \sinh \frac{\ell_{ij}}{2} & \text{if} \quad g = hyp \\ 2\log \sin \frac{\ell_{ij}}{2} & \text{if} \quad g = sph \end{cases} \tag{1.22}$$

Then solving Problem 1.3.2 is equivalent to solving Problem 1.3.4 with $E_0 = E_\Sigma$ *and* $E_1 = E_\Delta \setminus E_\Sigma$*.*

Problem 1.3.4. *Given*

- *an abstract triangulation* Δ,
- *a partition* $V_\Delta = V_0 \dot\cup V_1$,
- $u_i \in \mathbb{R}$ *for* $i \in V_0$
- $\Theta_i \in \mathbb{R}_{>0}$ *for* $i \in V_1$,
- *a partition* $E_\Delta = E_0 \dot\cup E_1$,
- λ_{ij} *for* $ij \in E_0$,
- $\tilde g \in \{euc, hyp, sph\}$,

find $u_i \in \mathbb{R}$ *for* $i \in V_1$ *and* λ_{ij} *for* $ij \in E_1$ *such that*

$$\tilde\ell : E_\Delta \to \mathbb{R}_{>0}$$

defined by

$$\tilde\lambda_{ij} = u_i + u_j + \lambda_{ij}, \tag{1.23}$$

and

$$\tilde\ell_{ij} = \begin{cases} e^{\frac{1}{2}\tilde\lambda_{ij}} & \text{if} \quad \tilde g = euc \\ 2\operatorname{arsinh} e^{\frac{1}{2}\tilde\lambda_{ij}} & \text{if} \quad \tilde g = hyp \\ 2\arcsin e^{\frac{1}{2}\tilde\lambda_{ij}} & \text{if} \quad \tilde g = sph \end{cases} \tag{1.24}$$

satisfies for all $ijk \in F_\Delta$ *the triangle inequalities*

$$\tilde\ell_{ij} < \tilde\ell_{jk} + \tilde\ell_{ki}, \qquad \tilde\ell_{jk} < \tilde\ell_{ki} + \tilde\ell_{ij}, \qquad \tilde\ell_{ki} < \tilde\ell_{ij} + \tilde\ell_{jk}, \tag{1.25}$$

and for $\tilde g = sph$ *also*

$$\tilde\ell_{ij} + \tilde\ell_{jk} + \tilde\ell_{ki} < 2\pi, \tag{1.26}$$

and such that

$$\sum_{jk:ijk\in F_\Delta} \tilde\alpha^i_{jk} = \Theta_i \quad \text{for all} \quad i \in V_1, \tag{1.27}$$

$$\tilde\beta^k_{ij} + \tilde\beta^l_{ji} = \pi \quad \text{for all} \quad ij \in E_1, \tag{1.28}$$

where $\tilde\alpha$ *and* $\tilde\beta$ *are defined by* (1.21) *and* (1.20) *(with* α, β, ℓ *replaced by* $\tilde\alpha$, $\tilde\beta$, $\tilde\ell$*). Note that for* $\tilde g = sph$ *it is also required that* $\tilde\lambda < 0$ *for* $\tilde\ell$ *to be well-defined.*

Proof of Lemma 1.3.3. Note that (1.27) says that the angle sums at vertices in V_1 have the prescribed values, and (1.28) says that neighboring triangles of $(\Delta, \tilde\ell)_{\tilde g}$ belonging to the same face of Σ share the same circumcircle. So deleting the edges in $E_\Delta \setminus E_\Sigma$, one obtains a cyclic polyhedral surface $(\Sigma, \tilde\ell|_{E_\Sigma})_{\tilde g}$. □

1.3.3 Variational principles

Definition 1.3.5. *For an abstract triangulation* Δ *and a function* $\Theta \in \mathbb{R}^{V_\Delta}_{>0}$*, define the three functions*

$$\begin{aligned} E^{euc}_{\Delta,\Theta}, E^{hyp}_{\Delta,\Theta}, E^{sph}_{\Delta,\Theta} : \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta} &\longrightarrow \mathbb{R}, \\ (\lambda, u) &\longmapsto E^{\tilde g}_{\Delta,\Theta}(\lambda, u) \end{aligned}$$

by

$$E^{\tilde{g}}_{\Delta,\Theta}(\lambda, u) = \sum_{ijk \in F_\Delta} \left(f^{\tilde{g}}(\tilde{\lambda}_{ij}, \tilde{\lambda}_{jk}, \tilde{\lambda}_{ki}) - \frac{\pi}{2}(\tilde{\lambda}_{jk} + \tilde{\lambda}_{ki} + \tilde{\lambda}_{ij}) \right) + \sum_{i \in V_\Delta} \Theta_i u_i, \tag{1.29}$$

where $\tilde{g} \in \{euc, hyp, sph\}$, $\tilde{\lambda}$ is defined as function of λ and u by (1.23)*, and the functions f^{euc}, f^{hyp}, f^{sph} are defined in Section 1.3.4.*

We will often omit the subscripts and write simply $E^{euc}, E^{hyp}, E^{sph}$ when this is unlikely to cause confusion.

Definition 1.3.6. *We define the* feasible regions *of the functions $E^{\tilde{g}}_{\Delta,\Theta}$ as the following open subsets of their domains:*

- *The feasible region of E^{euc} and E^{hyp} is the set of all $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ such that $\tilde{\ell} \in \mathbb{R}^E_{>0}$ defined by* (1.23) *and* (1.24) *satisfies the triangle inequalities* (1.25)
- *The feasible region of E^{sph} is the set of all $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ such that $\tilde{\lambda}$ defined by* (1.23) *is negative, and $\tilde{\ell}$, which is then well-defined by* (1.24)*, satisfies the triangle inequalities* (1.25) *and the inequalities* (1.26)*.*

Theorem 1.3.7 (Variational principles)**.** *Every solution $(\Sigma, \tilde{\ell})_{\tilde{g}}$ of Problem 1.3.2 corresponds via* (1.23) *and* (1.24) *to a critical point $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ of the function $E^{\tilde{g}}_{\Delta,\Theta}$ under the constraints that λ_{ij} and u_i are fixed for $ij \in E_0$ and $i \in V_0$, respectively. (The triangulation Δ, and $E_0 = E_\Sigma$ and $E_1 = E_\Delta \setminus E_\Sigma$ are as in Lemma 1.3.3, and the given function Θ is extended from V_1 to V by arbitrary values on V_0.)*

Conversely, if $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ is a critical point of the function $E^{\tilde{g}}_{\Delta,\Theta}$ under the same constraints, and if (λ, u) is contained in the feasible region of $E^{\tilde{g}}_{\Delta,\Theta}$, then $(\Sigma, \tilde{\ell})_{\tilde{g}}$ defined by (1.23) *and* (1.24) *is a solution of Problem 1.3.2.*

Proof. This follows from the analytic formulation of Problem 1.3.2 (see Section 1.3.2) and Proposition 1.3.8. □

Proposition 1.3.8 (First derivative of $E^{\tilde{g}}$)**.** *The partial derivatives of $E^{\tilde{g}}$ are*

$$\frac{\partial E^{\tilde{g}}}{\partial u_i}(\lambda, u) = \Theta_i - \sum_{ijk \ni i} \tilde{\alpha}^i_{jk} \tag{1.30}$$

$$\frac{\partial E^{\tilde{g}}}{\partial \lambda_{ij}}(\lambda, u) = \tilde{\beta}^k_{ij} + \tilde{\beta}^l_{ij} - \pi. \tag{1.31}$$

Here $\tilde{\alpha}$, $\tilde{\beta}$ are defined by (1.21) *and* (1.20) *(with α, β, ℓ replaced by $\tilde{\alpha}$, $\tilde{\beta}$, $\tilde{\ell}$) if (λ, u) is contained in the feasible region of $E^{\tilde{g}}$. For (λ, u) not contained in the feasible region, the definition of $\tilde{\alpha}$, $\tilde{\beta}$ is extended like in Definition 1.3.12.*

Proof. Equations (1.30) and (1.31) follow from the definition of $E^{\tilde{g}}$ and Proposition 1.3.14 on the partial derivatives of f^g. □

Theorem 1.3.9 (Uniqueness for mapping problems)**.** *If Problem 1.3.2 with target geometry $\tilde{g} \in \{euc, hyp\}$ has a solution, then the solution is unique — except if $\tilde{g} = euc$ and $V_0 = \emptyset$ (the case of Problem 1.3.1). In this case, the solution is unique up to scale.*

The critical point $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ that corresponds, via (1.23) *and* (1.24)*, to a solution $(\Sigma, \tilde{\ell})_{\tilde{g}}$ of Problem 1.3.2 with $\tilde{g} \in \{euc, hyp\}$ is a minimizer of $E^{\tilde{g}}_{\Delta,\Theta}$ under the constraints described in Theorem 1.3.7.*

The minimizer is unique except in the following cases. If $\tilde{g} = euc$ and $V_0 = \emptyset$, then $E^{\tilde{\delta}}_{\Delta,\Theta}$ is constant along all lines in the "scaling direction" $(0, 1_{V_\Delta}) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$. If the 1-skeleton of Σ is bipartite and $V_0 = \emptyset$, then $E^{\tilde{\delta}}_{\Delta,\Theta}$ is constant in the direction that is ± 1 on the two color classes of V_Δ, respectively, and takes appropriate values on $E_\Delta \setminus E_\Sigma$ so that $\tilde{\lambda}_{ij}$ defined by (1.23) *remains constant for all $ij \in E_\Delta$. (In both exceptional cases, one can obtain a unique minimizer by adding the constraint of fixing u_i for some $i \in V_\Delta$.)*

Proof. The theorem follows from Theorem 1.3.7 and the following observations.

(1) If the point $(\lambda, u) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ corresponds to a solution of Problem 1.3.2, it is contained in the feasible region of $E^{\tilde{\delta}}_{\Delta,\Theta}$.
(2) By (1.29) and Proposition 1.3.16, the functions E^{euc} and E^{hyp} are convex.
(3) For (λ, u) in the feasible region, the second derivative $D^2E^{hyp}(\lambda, u)$ is a positive definite quadratic form of $d\tilde{\lambda}$, i.e., $D^2E^{hyp}(\lambda, u)(\dot{\lambda}, \dot{u}) \geq 0$ for all $(\dot{\lambda}, \dot{u}) \in \mathbb{R}^{E_\Delta} \times \mathbb{R}^{V_\Delta}$ and $D^2E^{hyp}(\lambda, u)(\dot{\lambda}, \dot{u}) = 0$ if and only if

$$\dot{\lambda}_{ij} + \dot{u}_i + \dot{u}_j = 0 \quad \text{for all } ij \in E_\Delta.$$

(4) Similarly, for (λ, u) in the feasible region, the second derivative $D^2E^{euc}(\lambda, u)$ is a positive semidefinite quadratic form with $D^2E^{euc}(\lambda, u)(\dot{\lambda}, \dot{u}) = 0$ if and only if

$$\dot{\lambda}_{ij} + \dot{u}_i + \dot{u}_j = c \quad \text{for all } ij \in E_\Delta, \text{ for some } c \in \mathbb{R}.$$

□

In the following proposition, we collect explicit formulas for the second derivatives of the functions $E^{\tilde{\delta}}$. They are useful for the numerical minimization of E^{euc} and E^{hyp}, and even for finding critical points of E^{sph}, as explained in Section 1.6.2.

Proposition 1.3.10 (Second derivative of $E^{\tilde{\delta}}$)**.** *The second derivatives of E^{euc}, E^{hyp}, and E^{sph} are the quadratic forms*

$$D^2E^{\tilde{\delta}}(\lambda, u) = \frac{1}{2} \sum_{ijk \in F_\Delta} \left(q^k_{ij}(\lambda, u) + q^i_{jk}(\lambda, u) + q^j_{ki}(\lambda, u) \right),$$

where $q^k_{ij}(\lambda, u) = 0$ if $\tilde{\ell}_{ij}, \tilde{\ell}_{jk}, \tilde{\ell}_{ki}$ defined by (1.23), (1.24) *violate the triangle inequalities* (1.25)*, or, in the case of $\tilde{g} = sph$, inequality* (1.26)*. Otherwise, the quadratic forms $q^k_{ij}(\lambda, u)$ are defined by*

$$q^k_{ij} = \begin{cases} \cot \tilde{\alpha}^k_{ij} \, (d\lambda_{ki} - d\lambda_{jk} + du_i - du_j)^2 & (euc) \\ \cot \tilde{\beta}^k_{ij} \left((d\lambda_{ik} - d\lambda_{kj} + du_i - du_j)^2 + \tanh^2 \left(\frac{\tilde{\ell}_{ij}}{2} \right) (d\lambda_{ij} + du_i + du_j)^2 \right) & (hyp) \\ \cot \tilde{\beta}^k_{ij} \left((d\lambda_{ik} - d\lambda_{kj} + du_i - du_j)^2 - \tan^2 \left(\frac{\tilde{\ell}_{ij}}{2} \right) (d\lambda_{ij} + du_i + du_j)^2 \right) & (sph) \end{cases}$$

where $\tilde{\alpha}$, $\tilde{\beta}$ are defined by (1.21) *and* (1.20) *(with α, β, ℓ replaced by $\tilde{\alpha}$, $\tilde{\beta}$, $\tilde{\ell}$).*

Proposition 1.3.10 follows from (1.29) and Proposition 1.3.15 about the second derivatives of f^g.

1.3.4 The triangle functions

This section is concerned with three real valued functions f^{euc}, f^{hyp}, f^{sph} of three variables that are the main building blocks for the action functions E^{euc}, E^{hyp}, E^{sph} of the variational principles.

Since we consider single triangles in this section, not triangulations, we can use simpler notation. For $\{i, j, k\} = \{1, 2, 3\}$, let

$$\lambda_i = \lambda_{jk}, \quad \ell_i = \ell_{jk}, \quad \alpha_i = \alpha^i_{jk}, \quad \beta_i = \beta^i_{jk}.$$

The terminology introduced in the following definition makes Definition 1.3.12 easier to state.

Definition 1.3.11. *Let the* feasible region *of f^{euc} and f^{hyp} be the open subset of all $\lambda \in \mathbb{R}^3$ such that $\ell \in \mathbb{R}^3_{>0}$ determined by* (1.22) *satisfies the triangle inequalities, i.e.,*

$$\ell_k < \ell_i + \ell_j \tag{1.32}$$

for $\{i, j, k\} = \{1, 2, 3\}$.

Let the feasible region *of f^{sph} be the open subset of all $\lambda \in \mathbb{R}^3$ such that $\lambda < 0$, and such that $\ell \in \mathbb{R}^3_{>0}$, which is then well-defined by* (1.22)*, satisfies the triangle inequalities* (1.32) *and*

$$\ell_1 + \ell_2 + \ell_3 < 2\pi. \tag{1.33}$$

Definition 1.3.12. *We define the three functions*

$$f^{euc}, f^{hyp}, f^{sph} : \mathbb{R}^3 \to \mathbb{R}$$

by

$$\begin{aligned} f^g(\lambda_1, \lambda_2, \lambda_3) = \beta_1\lambda_1 + \beta_2\lambda_2 + \beta_3\lambda_3 + \text{Л}(\alpha_1) + \text{Л}(\alpha_2) + \text{Л}(\alpha_3) \\ + \text{Л}(\beta_1) + \text{Л}(\beta_2) + \text{Л}(\beta_3) + \text{Л}\Big(\tfrac{1}{2}(\pi - \alpha_1 - \alpha_2 - \alpha_3)\Big), \end{aligned} \tag{1.34}$$

where $g \in \{euc, hyp, sph\}$, $\text{Л}(x)$ denotes Milnor's Lobachevsky function [47]

$$\text{Л}(x) = -\int_0^x \log\left|2\sin(t)\right| dt, \tag{1.35}$$

and,

- *if λ is in the feasible region of f^g, then the angles α, β are defined as the angles (shown in Figure 1.4) in a euclidean, hyperbolic, or spherical triangle (depending on g) with sides ℓ_1, ℓ_2, ℓ_3 determined by* (1.22)*. That is, α and β are defined by* (1.21) *and* (1.20)*.*
- *Otherwise, if $g = sph$, and if either at least two λs are non-negative or $\lambda < 0$ and inequality* (1.33) *is violated, let*

$$\alpha_k = \alpha_i = \alpha_j = \pi, \quad \beta_k = \beta_i = \beta_j = 0.$$

- *Otherwise, if the triangle inequality* (1.32) *is violated, or if $g = sph$ and $\lambda_k \geq 0$, let*

$$\alpha_k = \beta_k = \pi, \quad \alpha_i = \alpha_j = \beta_i = \beta_j = 0.$$

Figure 1.5 shows a graph of Milnor's Lobachevsky function. It is continuous, π-periodic, odd, has zeros at the integer multiples of $\pi/2$, and is real analytic except at integer multiples of π, where the derivative tends to $+\infty$.

Remark 1.3.13. *In the euclidean case,* (1.34) *simplifies to*

$$f^{\text{euc}}(\lambda) = \alpha_i\lambda_i + \alpha_j\lambda_j + \alpha_k\lambda_k + 2\text{Л}(\alpha_i) + 2\text{Л}(\alpha_j) + 2\text{Л}(\alpha_k). \tag{1.36}$$

This follows immediately from $\alpha_1 + \alpha_2 + \alpha_3 = \pi$, $\alpha = \beta$, and $\text{Л}(0) = 0$.

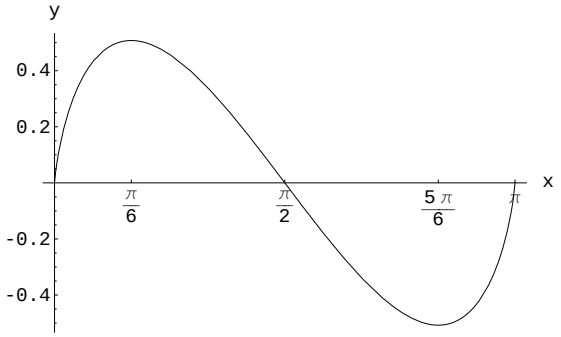


Figure 1.5: Graph of Milnor's Lobachevsky function, $y = \Pi(x)$.

Proposition 1.3.14 (first derivative)**.** *The functions f^g, $g \in \{euc, hyp, sph\}$, are continuously differentiable and*

$$\frac{\partial f^g}{\partial \lambda_i} = \beta_i. \tag{1.37}$$

Proof. Note that the angles α, β are continuous functions of λ on $\mathbb{R}^3$. Hence f^g defined by (1.34) is also continuous. We will show that f^g is continuously differentiable with derivative (1.37) on an open dense subset of the domain, namely, the union of (a) the feasible region and (b) the interior of its complement. Since f^g is continuous and df^g extends continuously to $\mathbb{R}^3$, the claim follows.

(a) First, suppose λ is contained in the feasible region of f^g. By symmetry, it suffices to consider the derivative with respect to λ_1. From (1.34) and (1.35) one obtains

$$\begin{aligned} \frac{\partial f^g}{\partial \lambda_1} \;=\; & \beta_1 + \sum_{i=1}^{3} \Big(\\ & \big(\lambda_i - \log(2\sin\beta_i)\big)\frac{\partial \beta_i}{\partial \lambda_1} + \\ & \Big(-\log(2\sin\alpha_i) + \tfrac{1}{2}\log\Big|2\sin\big(\tfrac{\pi-\alpha_1-\alpha_2-\alpha_3}{2}\big)\Big|\Big)\frac{\partial \alpha_i}{\partial \lambda_1} \\ & \Big). \end{aligned} \tag{1.38}$$

For hyperbolic and spherical triangles, one derives from the respective cosine rules

$$\begin{aligned} \sinh^2\frac{\ell_i}{2} &= \frac{\sin\beta_i \sin\frac{\pi-\alpha_1-\alpha_2-\alpha_3}{2}}{\sin\alpha_2\sin\alpha_3} \qquad \text{(hyperbolic)}, \\ \sin^2\frac{\ell_i}{2} &= \frac{\sin\beta_i \sin\frac{\alpha_1+\alpha_2+\alpha_3-\pi}{2}}{\sin\alpha_2\sin\alpha_3} \qquad \text{(spherical)}. \end{aligned}$$

In both cases, expand the fraction on the right hand side by four and take logarithms to find

$$\lambda_i = \log(2\sin\beta_i) + \log\Big|2\sin\tfrac{\pi-\alpha_1-\alpha_2-\alpha_3}{2}\Big| - \log(2\sin\alpha_j) - \log(2\sin\alpha_k).$$

Substitute this expression for λ_i in (1.38) and use $d\beta_i = \frac{1}{2}(d\alpha_i - d\alpha_j - d\alpha_k)$ to see that all terms on the right hand side of (1.38) cancel, except β_1.

For euclidean triangles, (1.38) simplifies to

$$\frac{\partial f^g}{\partial \lambda_1} = \beta_1 + \sum_{i=1}^{3} \big(\lambda_i - 2\log(2\sin\alpha_i)\big)\frac{\partial \alpha_i}{\partial \lambda_1},$$

where

$$\lambda_i - 2\log(2\sin\alpha_i) = 2\log\frac{\ell_i}{2\sin\alpha_i} = 2\log R$$

does not depend on i. (R denotes the circumradius.) Equation (1.37) follows because the angle sum is constant.

(b) Now suppose λ is contained in the interior of the complement of the feasible region of f^g. Since $\beta_1, \beta_2, \beta_3$ are constant on each connected component of the complement of the feasible region, and since

$$f^g(\lambda_1, \lambda_2, \lambda_3) = \beta_1\lambda_1 + \beta_2\lambda_2 + \beta_3\lambda_3,$$

outside the feasible region, equation 1.37 holds also in this case. This completes the proof. □

Proposition 1.3.15 (second derivative)**.** *For $g \in \{euc, hyp, sph\}$ the function f^g is twice continuously differentiable on its feasible set and the second derivative is*

$$D^2 f^{euc} = \frac{1}{2}\sum_{i=1}^{3} \cot\alpha_i\,(d\lambda_j - d\lambda_k)^2, \tag{1.39}$$

$$D^2 f^{hyp} = \frac{1}{2}\sum_{i=1}^{3} \cot\beta_i \Big((d\lambda_j - d\lambda_k)^2 + \tanh^2\Big(\tfrac{\ell_i}{2}\Big)\,d\lambda_i^2\Big), \tag{1.40}$$

$$D^2 f^{sph} = \frac{1}{2}\sum_{i=1}^{3} \cot\beta_i \Big((d\lambda_j - d\lambda_k)^2 - \tan^2\Big(\tfrac{\ell_i}{2}\Big)\,d\lambda_i^2\Big). \tag{1.41}$$

On each component of the complement of its feasible set, the function f_g is linear so the second derivative vanishes.

A proof of (1.39) is contained in the work of Bobenko *et al.* [13, Proposition 4.2.3], see Remark 1.3.17 below. Equations (1.40) and (1.41) can be derived by lengthy calculations.

Proposition 1.3.16. *(i) The function f^{euc} is convex. On its feasible set, the second derivative $D^2 f^{euc}$ is positive semidefinite with one-dimensional kernel spanned by the "scaling direction"* $(1, 1, 1)$.

(ii) The function f^{hyp} is convex. On its feasible set, the second derivative $D^2 f^{hyp}$ is positive definite, so the function is locally strictly convex.

Part (i) is proved by Bobenko *et al.* [13, Propositions 4.2.4, 4.2.5, note the following remark] directly from (1.39). We do not know a similarly straightforward proof of part (ii). The proof by Bobenko *et al.* (Section 6.2) is based on a connection with 3-dimensional hyperbolic geometry: f^{hyp} is the Legendre dual of the volume of an ideal hyperbolic prism considered as a function of the dihedral angles. This volume function is strictly concave, as shown by Leibon [43]. His argument uses the decomposition of an ideal prism into three ideal tetrahedra.

Remark 1.3.17. *The functions f and $\hat{V}_h$ defined by Bobenko et al. [13, Equations (4-3), (6-4)] are related to the functions f^{euc} and f^{hyp} by*

$$f^{euc}(\lambda_1, \lambda_2, \lambda_3) = 2f(\tfrac{\lambda_1}{2}, \tfrac{\lambda_2}{2}, \tfrac{\lambda_3}{2}), \tag{1.42}$$

$$f^{hyp}(\lambda_1, \lambda_2, \lambda_3) = 2\hat{V}_h(\lambda_1, \lambda_2, \lambda_3, 0, 0, 0). \tag{1.43}$$

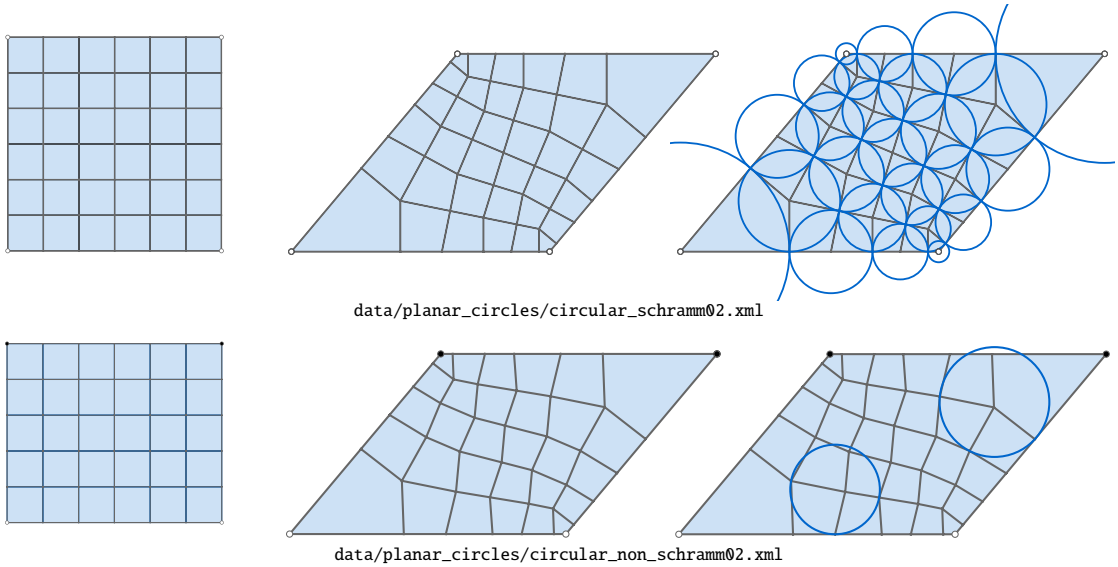


Figure 1.6: Mapping a rectangle to a parallelogram. Note the orthogonal circle pattern in the top row and the wiggly vertical lines in the bottom row.

1.4 Conformal maps of cyclic quadrangulations

Having introduced the mapping problems and variational principles, we return to conformal maps of cyclic quadrangulations. Some basic facts were already discussed in Section 1.2.8. Here, in Section 1.4.1, we consider a simple experiment that demonstrates the somewhat unexpected appearance of orthogonal circle patterns, and also a necessary condition for the boundary angles. In Section 1.4.2, we discuss a discrete version of the Riemann mapping problem for quadrangulations.

1.4.1 Emerging circle patterns and a necessary condition

Consider the two discrete conformal maps shown in the two rows of Figure 1.6. The domains (shown left) are a square and a rectangle, subdivided into 6×6 and 6×5 squares, respectively. We solve the mapping Problem 1.3.1 by minimizing E^{euc} as explained in Section 1.3.3, prescribing boundary angles to obtain maps to parallelograms: $\Theta = 50°$ and $130°$ for the corner vertices, $\Theta = 180°$ for the other boundary vertices, and $\Theta = 360°$ for interior vertices. The resulting quadrangulations are shown in the middle.

On first sight, the 6×6 example shown in the top row behaves rather like one would expect from a conformal map. The horizontal and vertical "coordinate lines" of the domain are mapped to polygonal curves that look more or less like they could be discretizations of reasonable smooth curves. In the 6×5 example shown in the bottom row, the images of the vertical lines zigzag noticeably.

A closer look at the 6×6 example reveals a remarkable phenomenon. Let us bicolor the vertices black and white so that neighboring vertices have different colors, with the corners colored white. Then, in the image quadrangulation, the edges incident with a black vertex meet at right angles, and the edges incident with a white vertex have the same length. One can therefore

draw a circle around each white vertex through the neighboring black vertices as shown in Figure 1.6 (top right). At the black vertices, these circles touch and intersect orthogonally. Such circle patterns were studied by Schramm [61] as discrete analogs of conformal maps.

Given such a circle pattern with orthogonally intersecting circles, the quadrangulation formed by drawing edges between circle centers and intersection points consists of quadrilaterals that are right-angled kites. Such kites have complex cross-ratio -1. Hence, the quadrangulation coming from an orthogonal circle pattern is discretely conformally equivalent (in our sense) to a combinatorially equivalent quadrangulation consisting of squares.

The conformal map shown in the top row of Figure 1.6 "finds" the orthogonal circle pattern because that circle pattern exists and the conformal map is unique (by Theorem 1.3.9). For the 6×5 example shown in the bottom row, a corresponding orthogonal circle pattern does not exist. No matter which coloring is chosen, there are two black vertices at which the total angle changes (from 90° to 50° and 130°, respectively). The neighbors of a vertex do not lie on a circle. Figure 1.6 (bottom right) shows two circles drawn through three out of four neighbors.

If we map an $m \times n$ square grid to a parallelogram like in Figure 1.6, an orthogonal circle pattern will appear if m an n are even. No such pattern will appear if one of the numbers is even and the other is odd. What happens if both m and n are odd? In this case, the conformal map does not exist. The corners with increasing angle and the corners with decreasing angle would have different colors. This violates the necessary condition expressed in the following theorem.

Theorem 1.4.1 (Necessary condition for the existence of a conformal map)**.** *Let Σ be an abstract quadrangulation of the closed disk, and let*

$$z, \tilde{z} : V_\Sigma \to \mathbb{C}$$

determine two discretely conformally equivalent immersions of Σ into the complex plane. Denote their angle sums at boundary vertices $v \in V_\Sigma$ by Θ_v and $\tilde{\Theta}_v$, respectively. Since the 1*-skeleton of Σ is bipartite, we may assume the vertices are colored black and white. Let V_b^∂ and V_w^∂ denote the sets of black and white boundary vertices of Σ. Then*

$$\sum_{v \in V_b^\partial} (\tilde{\Theta}_v - \Theta_v) \equiv 0 \pmod{2\pi}, \tag{1.44}$$

$$\sum_{v \in V_w^\partial} (\tilde{\Theta}_v - \Theta_v) \equiv 0 \pmod{2\pi}. \tag{1.45}$$

(Since $\sum_{V_b^\partial \cup V_w^\partial} (\tilde{\Theta}_v - \Theta_v) = 0$, equations (1.45) *and* (1.44) *are equivalent.)*

Proof. Since z and $\tilde{z}$ are two solutions of the cross-ratio system on Σ with the same cross-ratios (see Section 1.2.8), there exists by Proposition 1.2.13 a function $w : V_\Sigma \to \mathbb{C}$ such that (1.16) holds for all edges $ij \in E_\Sigma$. Now suppose $v_0, \dots, v_{2n-1} \in V_\Sigma$ are the boundary vertices in cyclic order (with indices taken modulo $2n$). Then

$$e^{i(\tilde{\Theta}_{v_k} - \Theta_{v_k})} = \frac{(\tilde{z}_{v_{k+1}} - \tilde{z}_{v_k})(z_{v_{k-1}} - z_{v_k})}{(\tilde{z}_{v_{k-1}} - \tilde{z}_{v_k})(z_{v_{k+1}} - z_{v_k})} = \frac{w_{v_{k+1}}}{w_{v_{k-1}}},$$

so

$$\prod_{k=0}^{n-1} e^{i(\tilde{\Theta}_{v_{2k}} - \Theta_{v_{2k}})} = \prod_{k=0}^{n-1} e^{i(\tilde{\Theta}_{v_{2k+1}} - \Theta_{v_{2k+1}})} = 1.$$

Equations (1.44) and (1.45) follow. □

1.4.2 Riemann maps with cyclic quadrilaterals

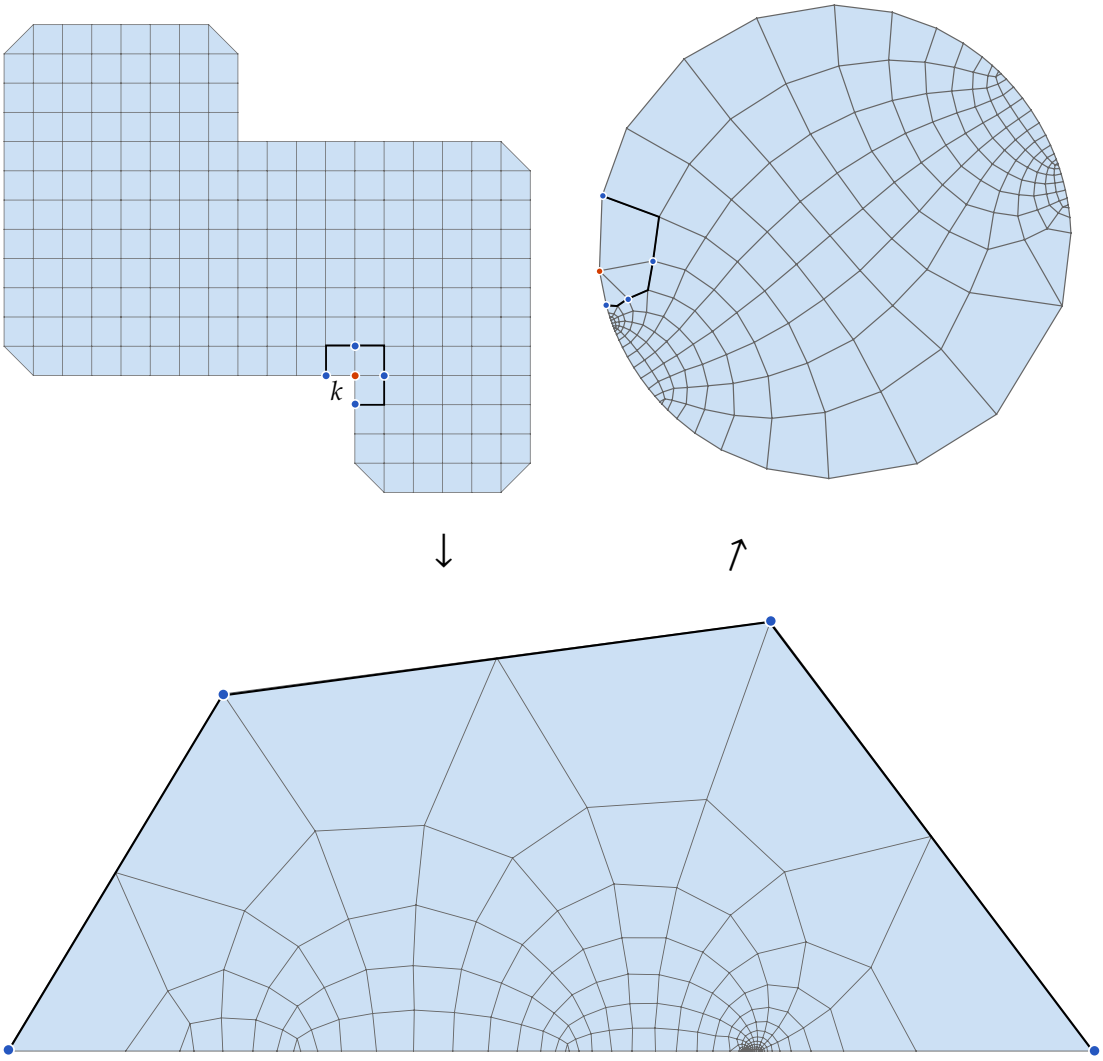
Consider the following discrete version of the Riemann mapping problem: Map a cyclic polyhedral surface that is topologically a closed disk discretely conformally to a planar polygonal region with boundary vertices on a circle. An example is shown in Figure 1.7, top row. This type of problem can often be reduced to Problem 1.3.2. Then, by the variational principle, if a solution exists, it can be found by minimizing a convex function. For triangulations, the reduction of the discrete Riemann mapping problem to Problem 1.3.2 is explained by Bobenko *et al.* [13, Section 3.3]. Here, we consider the case of quadrangulations. (The arguments can be extended to even polygons with more than four sides. We restrict our attention to quadrilaterals because the combinatorial restrictions discussed in the following paragraph become even more involved yet not more interesting for surfaces with hexagons, octagons, etc.)

The basic idea is the same as for triangulations: First, map the polyhedral surface to the half plane with one boundary vertex at infinity. Then apply a Möbius transformation. This leads to a combinatorial restriction: No face may have more than one edge on the boundary. (The face would degenerate when the boundary is mapped to a straight line.) For triangulations, this means that no triangle may be connected to the surface by only one edge. If this condition is violated, cutting off such "ears" often leads to an admissible triangulation. For quadrangulations, this fix does not work in typical situations. Instead, if a quadrilateral contains two consecutive edges on the boundary, cut off a triangle. The resulting polyhedral surface will consist mostly of quadrilaterals with some triangles on the boundary, as in the example shown in Figure 1.7.

Suppose $(\Sigma, \ell)_{euc}$ is a euclidean cyclic polyhedral surface that is homeomorphic to the closed disk and consists mostly of quadrilaterals. (For the following construction we really only need a boundary vertex that is incident with quadrilateral faces.) To map it to a polygonal region inscribed in a circle, proceed as follows (see Figure 1.7):

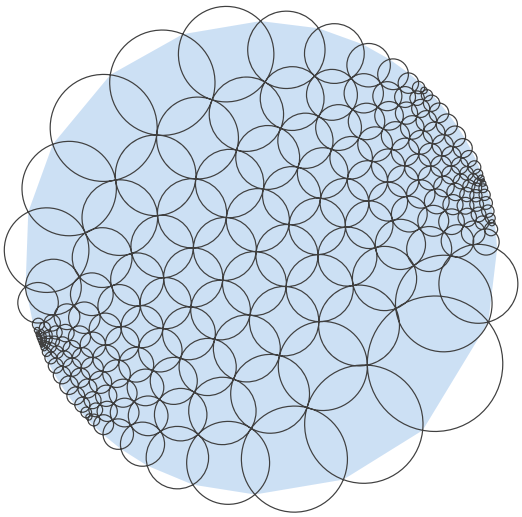
(1) Choose a vertex k on the boundary of Σ such that all incident faces are quadrilaterals.
(2) Apply a discrete conformal change of metric (1.3) such that all edges incident with k have the same length. One may choose $u = 0$ for all vertices except the neighbors of k. It does not matter if polygon inequalities are violated after this step.
(3) Let $(\Sigma', \ell')_{euc}$ be the cyclic polyhedral complex obtained by removing vertex k and all incident quadrilaterals.
(4) Solve Problem 1.3.2 for $(\Sigma', \ell')_{euc}$ with prescribed total angles $\Theta_i = 2\pi$ for interior vertices of Σ', $\Theta_i = \pi$ for boundary vertices of Σ' that were not neighbors of k in Σ, and fixed logarithmic scale factors $u_i = 0$ for those that were neighbors of k. The result is a planar polyhedral surface as shown in Figure 1.7, bottom. The boundary consists of one straight line segment containing all boundary edges of Σ' that were also boundary edges of Σ, and two or more straight line segments, each consisting of two edges that were incident with a removed quadrilateral.
(5) Apply a Möbius transformation (e.g., $z \mapsto 1/z$) to the vertices that maps the boundary vertices of Σ to a circle and the other vertices to the inside of this circle. Reinsert k at the image point of ∞ under this Möbius transformation. Each face $ijmk \in \Sigma$ incident with k is cyclic because the three vertices i, j, and m are contained in a line before transformation.
(6) Optionally apply a 2-dimensional version of the Möbius normalization described in Section 1.6.3.

Proposition 1.4.2. *The result of this procedure is a planar cyclic polyhedral surface that is discretely conformally equivalent to* $(\Sigma, \ell)_{euc}$ *and has its boundary polygon inscribed in a circle.*



data/planar_circles/riemann_shape.xml

Figure 1.7: Riemann mapping with cyclic quadrilaterals. Corners with valence two in the original domain are modeled as triangles, upper left. We first map to the upper half-plane. Quadrilaterals adjacent to the point at infinity are removed. Points opposite to the point at infinity have a prescribed boundary angle of π. The four vertices in between have a fixed $u = 0$. Quadrilaterals adjacent to the point at infinity are cyclic after the Möbius transformation to the disk, bottom.



data/planar_circles/riemann_shape.xml

Figure 1.8: Here we show the face circumcircles of the solution to the Riemann mapping problem of Figure 1.7. It looks conspicuously like an orthogonal circle pattern. But the face circumcircles intersect only approximately but not exactly at right angles.

Proof. That the boundary polygon is inscribed in a circle is obvious from the construction. Using the Möbius invariance of discrete conformal equivalence (Proposition 1.2.5), it is not difficult to see that the surfaces without quadrilaterals incident with k are discretely conformally equivalent. To show that the whole surfaces are equivalent, it suffices to show that corresponding quadrilaterals incident with k have the same complex cross-ratio.

After step (2), the length cross-ratio of a quadrilateral incident with k is equal to the simple length ratio of the two edges that are not incident with k.

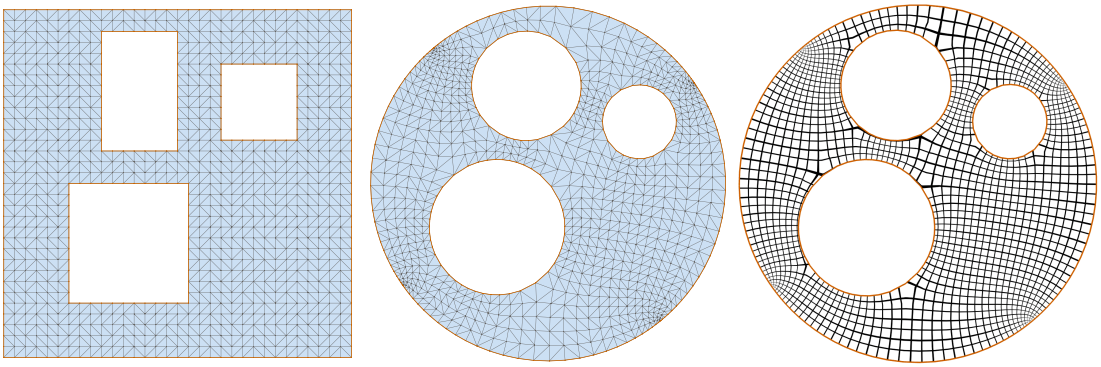
After step (4), the length cross-ratio of these edges is unchanged due to the fixed logarithmic scale factors $u = 0$ on the neighbors of k. Also, these edges are now collinear because of the prescribed angle $\Theta = \pi$ between them.

After applying the Möbius transformation in step (5), the image of the point at infinity and the other three vertices of our quadrilateral incident with k form again a cyclic quadrilateral with the same complex cross-ratio as in the beginning. □

1.5 Multiply connected domains

1.5.1 Circle domains

Koebe's generalization of the Riemann mapping theorem says that multiply connected domains are conformally equivalent to domains bounded by circles, and the uniformizing map to such a circle domain is unique up to Möbius transformations. A method to construct discrete Riemann maps is described by Bobenko *et al.* [13, Section 3.3] for triangulations and for mostly quadrilateral meshes in the previous Section 1.4.2. Having generalized the notion of discrete conformal equivalence from triangulations to cyclic polyhedral surfaces, it is straightforward to adapt this



data/circle_domain_euclidean/three_holes.xml

Figure 1.9: Discrete conformal map of a multiply-connected domain (left) to a circle domain (middle). The images of vertical and horizontal "parameter lines" are shown on the right.

method to construct discrete maps to circle domains:

(1) Fill holes by gluing faces to all but one boundary component, so that the resulting surface is homeomorphic to a disk.
(2) Construct the discrete Riemann map.
(3) Remove the faces that were added in step (1).

Figure 1.9 shows an example.

1.5.2 Special slit domains

Any multiply connected domain can be mapped to the complex plain with parallel slits [49]. In principle, it is possible to construct discrete conformal maps that map holes to slits by solving Problem 1.3.1. On each boundary component that should be mapped to a slit, set the desired total angle $\Theta = 2\pi$ for the two vertices that should be mapped to the endpoints of the slit, and set $\Theta = \pi$ for all other vertices on that boundary component. However, this will not work in general. While the resulting surface will be flat, the developing map to the plane will in general have translational monodromy for a cycle around the hole. The surface will only close up in the plane if the vertices that should be mapped to the endpoints of the slit are chosen exactly right. (This will in general require modifying the original mesh.)

Sometimes, the symmetry of the problem determines the right positions of the end-vertices, so that discrete conformal maps to slit surfaces can be computed. The first two rows of Figure 1.10 show examples. The bottom row visualizes a discrete conformal map where circular holes are mapped to slits. Here, we use the following trick: We start with the slit surface and map it to a surface with circular holes as described in Section 1.5.1.

1.6 Uniformization of spheres

This section is concerned with discrete conformal maps of polyhedral surfaces of genus 0 onto the round sphere. For triangulations, this is described by Bobenko *et al.* [13, Section 3.2]. In Section 1.6.1, we adapt this method to quadrangulations. This is similar to the discrete Riemann

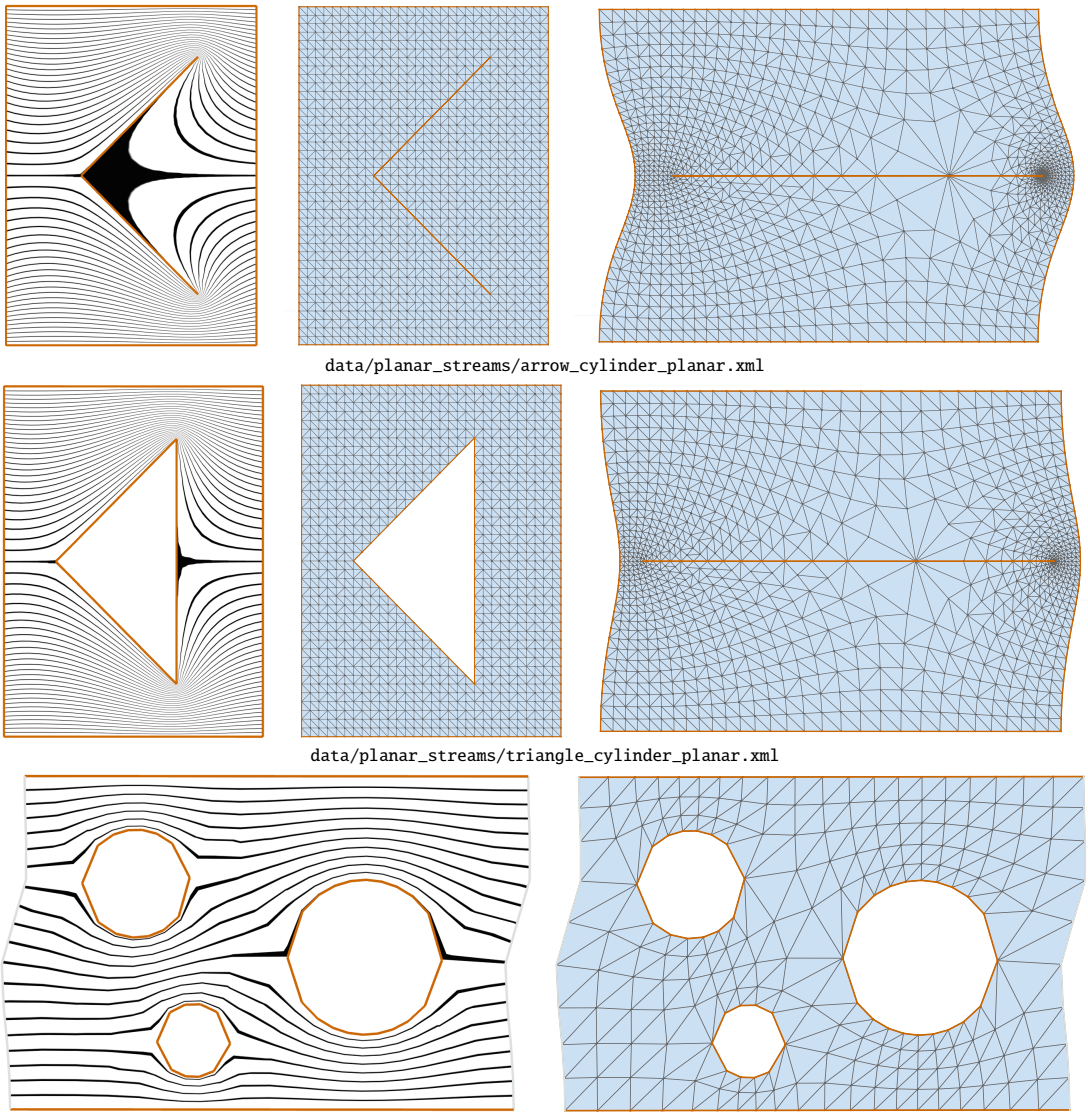
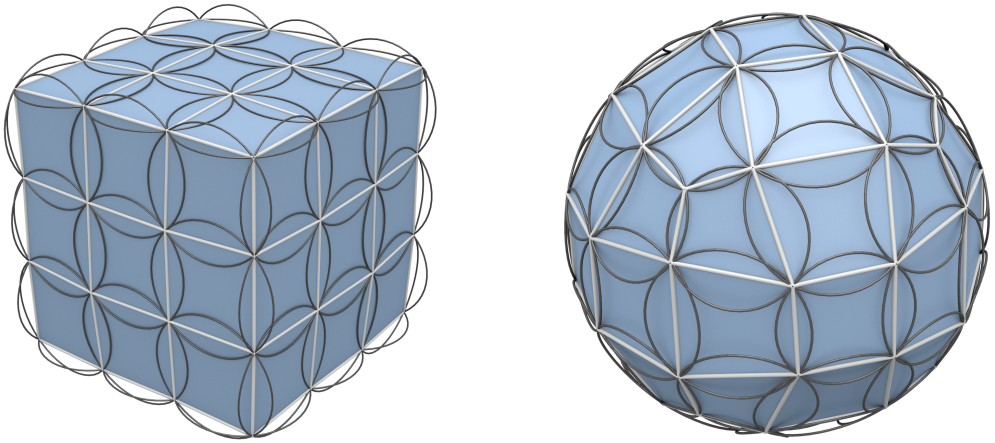


Figure 1.10: Mapping surfaces with holes to slit surfaces. In all images, the left and right parts of the boundary are identified by a horizontal translation. Preimages of horizontal lines visualize the flow of an incompressible inviscid fluid around the hole in a channel with periodic boundary conditions. Top row: A cylinder with a triangular hole is mapped to a cylinder with a slit. One vertex of the triangle and the midpoint of the opposite side are mapped to the endpoints of the slit. Middle row: An arrow shaped slit is mapped to a straight slit. The two vertices at the arrow's tip, on either side of the slit, are mapped to the endpoints of the straight slit. Bottom row: Three circular boundary components are mapped to horizontal slits. (The slit surface is not shown.)



data/spherical_cyclic/spherical_circular.xml

Figure 1.11: Discrete conformal map from the cube to the sphere, calculated with the method described in Section 1.6.1. We apply Möbius normalization (Section 1.6.3) to the polyhedral surface with vertices on the sphere to achieve rotational symmetry.

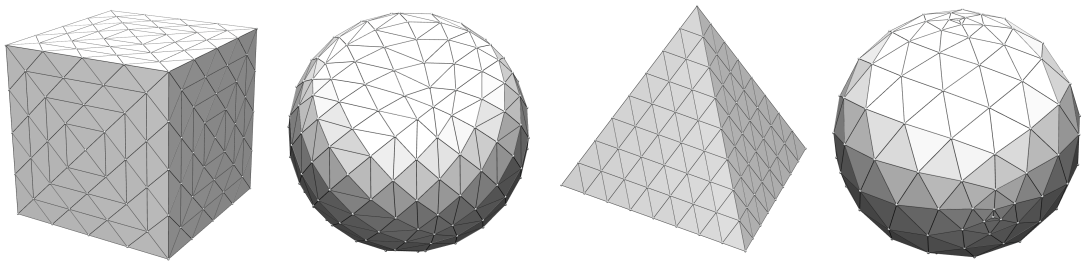
mapping with quadrilaterals described in Section 1.4.2. Effectively, this method reduces the problem to minimizing the convex euclidean functional E^{euc}. The spherical version of the variational principle (Theorem 1.3.7) involves the non-convex function E^{sph}. It is not as practical for calculations, because one has to find a saddle point instead of a minimum. Nevertheless, the spherical functional can often be used to calculate maps to the sphere. This is explained in Section 1.6.2.

1.6.1 Uniformizing quadrangulations of the sphere

Suppose $(\Sigma, \ell)_{euc}$ is a cyclic polyhedral surface with quadrilateral faces that is homeomorphic to the sphere.

(1) Choose a vertex $k \in V_\Sigma$.
(2) Apply a discrete conformal change of metric (1.3) such that all edges incident with k have the same length. One may choose $u = 0$ for all vertices except the neighbors of k. It does not matter if polygon inequalities are violated after this step.
(3) Let $(\Sigma', \ell')_{euc}$ be the complex obtained by removing vertex k and all incident quadrilaterals.
(4) Solve Problem 1.3.2 for $(\Sigma', \ell')_{euc}$ with prescribed total angles $\Theta_i = 2\pi$ for interior vertices of Σ', $\Theta_i = \pi$ for boundary vertices of Σ' that were not neighbors of k in Σ, and fixed scale factors $u_i = 0$ for vertices that were neighbors of k in Σ. The result is a planar polyhedral surface with cyclic quadrilaterals. Consecutive boundary edges that belonged to a face incident with vertex k in Σ are contained in a straight line.
(5) Map the vertices to the unit sphere by stereographic projection and reinsert the vertex k at the image point of ∞.
(6) Optionally apply Möbius normalization, see Section 1.6.3.

Proposition 1.6.1. *The result is a cyclic polyhedral surface with vertices on the unit sphere that is discretely conformally equivalent to* $(\Sigma, \ell)_{euc}$.



data/spherical/cube_data.xml data/spherical/tetrahedron_data.xml

Figure 1.12: Mapping conformally to the sphere using the spherical functional. The spherical surfaces are Möbius-normalized to achieve rotational symmetry.

This can be seen in the same way as the corresponding statement about discrete Riemann maps with quadrilaterals (Proposition 1.4.2). Figure 1.11 shows a discrete conformal map calculated by this method.

1.6.2 Using the spherical functional

It is possible to use the spherical functional E^{sph} to calculate maps to the sphere even though it is not convex. For simplicity, we consider only triangulations, so all λ variables are fixed and we may consider E^{sph} as function of the logarithmic scale factors u only (see Section 1.3.3). A numerical method has to find a saddle point of $E^{sph}(u)$.

Note that the scaling direction $1_{V_\Delta} \in \mathbb{R}^{V_\Delta}$ is a negative direction of the Hessian at a critical point: Suppose $(\Delta, \ell)_{sph}$ is a spherical triangulation with the desired angle sum Θ_i at each vertex i. Then $0 \in \mathbb{R}^{V_\Delta}$ is a critical point of $E^{sph}_{\Delta,\Theta}(u)$. If we enlarge all edge lengths by a common factor $e^h > 1$, then all angles become larger, so every component (1.30) of the gradient of E^{sph} becomes negative. Following the negative gradient would result in even longer lengths.

The following minimax method works in many cases. Define the function $\tilde{E}$ by *maximizing* the functional E^{sph} in the scaling direction,

$$\tilde{E}(u) = \max_{h \in \mathbb{R}} \left\{ E^{sph}(u + h1_{V_\Delta}) \right\} \tag{1.46}$$

Minimize functional $\tilde{E}$ in a hyperplane of $\mathbb{R}^{V_\Delta}$ transverse to the direction 1_{V_Δ}.

Figure 1.1 (top) and Figure 1.12 show examples of discrete conformal maps to polyhedral surfaces inscribed in a sphere that were calculated using this method.

1.6.3 Möbius normalization

The notion of discrete conformal equivalence of euclidean polyhedral surfaces $(\Sigma, \ell)_{euc}$ in $\mathbb{R}^3$ is Möbius-invariant (Proposition 1.2.5). If all vertices $v \in V_\Sigma$ are contained in the unit sphere $S^2 \subset \mathbb{R}^3$, then there is a Möbius transformation T of S^2 such that the center of mass of the transformed vertices is the origin [70],

$$\sum_{v \in V_\Sigma} T(v) = 0.$$

The Möbius transformation T is uniquely determined up to post-composition with a rotation around the origin.

The following method can be used to calculate such a Möbius transformation: Find the unique minimizer of the function δ defined below. Then choose for T a Möbius transformation that maps S^2 to itself and the minimizer to the origin. Here, we only provide explicit formulas for the function δ and its first two derivatives. For a more detailed account, we refer the reader to Springborn [70]. The function δ is defined on the open unit ball in $\mathbb{R}^3$ by

$$\delta(x) = \sum_{v \in V} \log\left(\frac{-\langle x, v\rangle}{\sqrt{-\langle x, x\rangle}}\right), \tag{1.47}$$

where

$$\langle x, y\rangle = x_1 y_1 + x_2 y_2 + x_3 y_3 - 1. \tag{1.48}$$

The gradient and Hessian matrix of δ are

$$\operatorname{grad} \delta(x) = \sum_{v \in V} \left(\frac{v}{\langle x, v\rangle} - \frac{x}{\langle x, x\rangle}\right), \tag{1.49}$$

$$\operatorname{Hess} \delta(x) = \sum_{v \in V} \left(2\frac{x^T x}{\langle x, x\rangle^2} - \frac{v^T v}{\langle x, v\rangle^2} - \operatorname{diag}\left(\frac{1}{\langle x, x\rangle}\right)\right). \tag{1.50}$$

1.7 Uniformization of tori

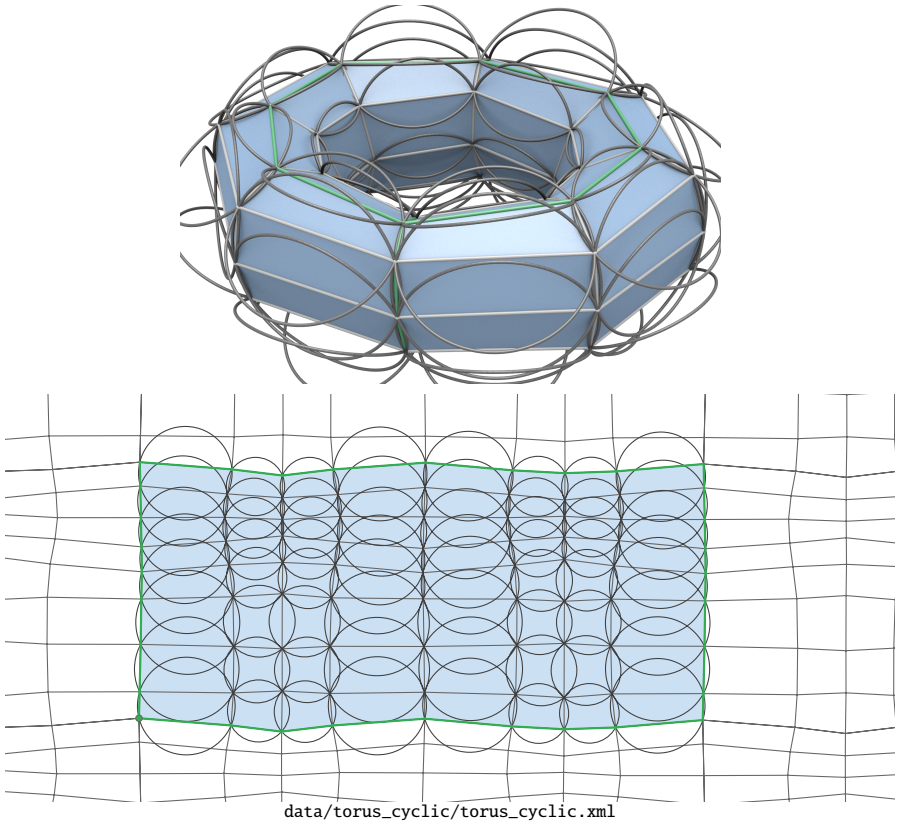
Every Riemann surface R of genus one is conformally equivalent to a flat torus, i.e., to a quotient space $\mathbb{C}/\Gamma$, where $\Gamma = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ is some two-dimensional lattice in $\mathbb{C}$. The biholomorphic map from R to $\mathbb{C}/\Gamma$, or from the universal cover of R to $\mathbb{C}$, is called a uniformizing map. For a polyhedral surface of genus one, constructing a discrete uniformizing map amounts to solving Problem 1.3.1 with prescribed total angle $\Theta = 2\pi$ at all vertices. This provides us with a method to calculate approximate uniformizing maps for Riemann surfaces of genus one given in various forms. We consider examples of tori immersed in $\mathbb{R}^3$ in Section 1.7.1 and elliptic curves in Section 1.7.2. (We will also consider tori in the form of Schottky uniformization in Section 1.8.2, as a toy example after treating the higher genus case.)

The belief that discrete conformal maps approximate conformal maps is not based on a proven theorem but on experimental evidence like the Wente torus example of Section 1.7.1 and the numerical experiments of Section 1.7.4.

1.7.1 Immersed tori

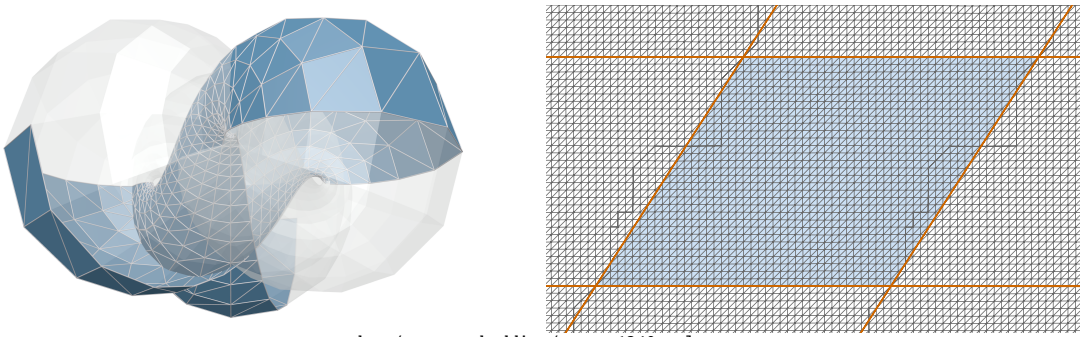
First we consider a simple example with quadrilateral faces. Figure 1.13 (left) shows a coarse discretization of a torus whose faces are inscribed in circles. On the right, the figure shows the uniformization obtained by solving Problem 1.3.1 with prescribed total angle $\Theta = 2\pi$ at all vertices.

To test the numerical accuracy of our discrete uniformizing maps, we consider the famous torus of constant mean curvature discovered by Wente [74]. Explicit doubly-periodic conformal immersion formulas (i.e., formulas for the inverse of a uniformizing map) are known in terms of elliptic functions [1, 73, 10].



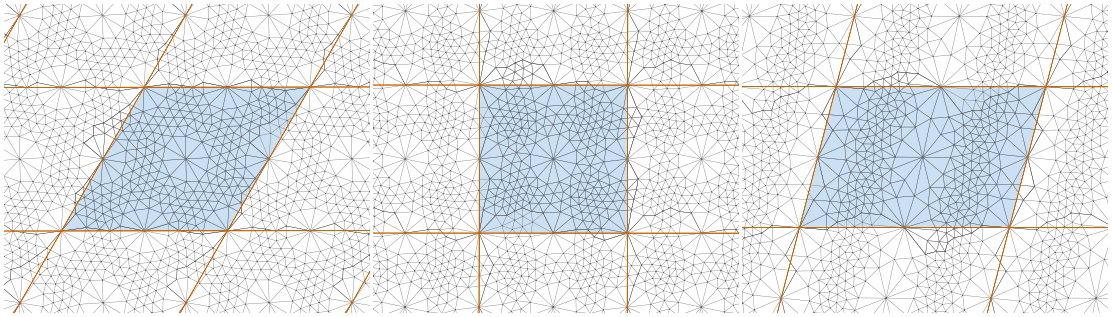
data/torus_cyclic/torus_cyclic.xml

Figure 1.13: Uniformization of an immersed torus with cyclic quadrilateral faces.



data/wente_embedding/wente_1240.xml

Figure 1.14: Uniformization of the Wente torus. Discretely sampled conformal immersion (left), domain of uniformization (right). Orange curves indicate the lattice of the flat torus in the domain. The faces of the polyhedral surface are approximately conformal squares. In the discrete uniformization their images are approximately squares, as it should be.



data/elliptic_curves/{tetrahedron|square|generic}.xml

Figure 1.15: Discrete uniformization of elliptic curves. Left: If the branch points in S^2 are the vertices of a regular tetrahedron, period lattice is very close to a hexagonal lattice. Middle: If the branch points form a square on the equator, the period lattice is very close to a square lattice. Right: an example with branch points in unsymmetric position.

Figure 1.14 (left) shows a triangulated model of the Wente torus constructed by sampling an explicit immersion formula [10] on a nearly square lattice containing the period lattice Γ. On the right, the figure shows the discrete uniformization, which reproduces the regular lattice Γ to high accuracy. The modulus $\tau = \omega_2/\omega_1$ of the Wente torus has been determined numerically [30] as $\tau = 0.41300\ldots + i\,0.91073\ldots$. The modulus of the discrete uniformization of the discretized surface shown in the figure is $\tilde{\tau} = 0.41341\ldots + i\,0.91061\ldots$.

1.7.2 Elliptic curves

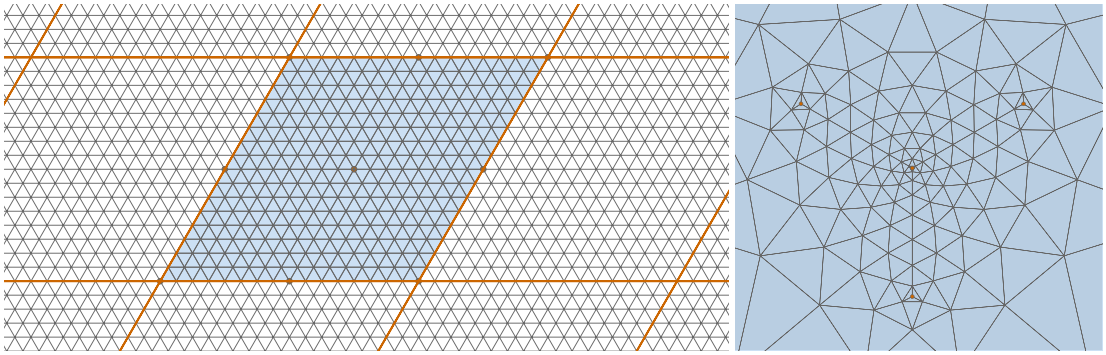
An algebraic curve of the form

$$\mu^2 = a\prod_{j=1}^{k}(\lambda - \lambda_j), \tag{1.51}$$

where the $\lambda_j \in \mathbb{C}$ are distinct and $k = 3$ (an elliptic curve) or $k = 4$ (with the singularity at infinity resolved), represents a Riemann surface of genus one as branched double cover of the λ-sphere $\mathbb{C}\mathrm{P}^1$, which we identify conformally with the unit sphere $S^2 \subset \mathbb{R}^3$. The branch points are $\lambda_1, \lambda_2, \lambda_3, \infty$ if $k = 3$ and $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ if $k = 4$. Every Riemann surface of genus one can be represented in this way.

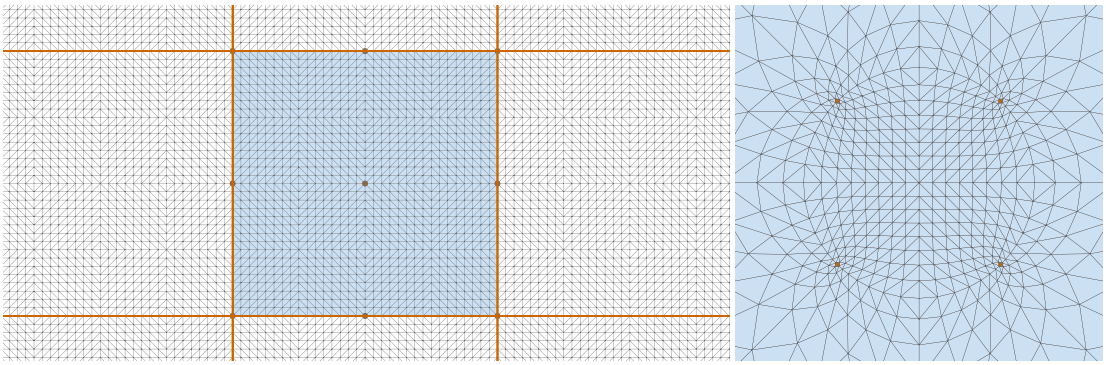
We construct a discrete model for a double cover of S^2 branched at four points $\lambda_1, \ldots, \lambda_4$ in the following way. Choose n other points $p_1, \ldots, p_n \in S^2$ and let P be the boundary of the convex hull of the points $\{\lambda_1, \ldots, \lambda_4, p_1, \ldots, p_n\}$. Then P is a convex polyhedron with $n + 4$ vertices and with faces inscribed in circles. (Generically, the faces will be triangles. In Section 1.7.3 we explain the method we used to obtain "good" triangles.) Find two disjoint simple edge paths γ_1, γ_2 joining the branch points λ_j in pairs. Take a second copy $\hat{P}$ of the polyhedron P. Cut and glue P and $\hat{P}$ along the paths γ_1, γ_2 to obtain a polyhedral surface of genus 1. Uniformize it by solving Problem 1.3.1. One obtains a discrete conformal map to a flat torus, whose inverse can be seen as a discrete elliptic function. Figure 1.15 shows examples. We will treat hyperelliptic curves in a similar fashion in Section 1.8.3.

Remark 1.7.1. *Instead of constructing a doubly-covered convex euclidean polyhedron with vertices on the unit sphere as described above, one could also construct a spherical triangulation of the doubly-covered*



data/elliptic_tetrahedron_uniform/tetrahedron_uniform_branched.xml

Figure 1.16: Mapping the hexagonal torus $\mathbb{C}/(\mathbb{Z} + \tau\mathbb{Z})$, $\tau = \frac{1}{2} + i\frac{\sqrt{3}}{2}$ (left) to a double cover of the sphere (right). Because the regular triangulation of the torus on the left is symmetric with respect to the elliptic involution, its image projects to a triangulation of the sphere seen on the right.



data/elliptic_square_uniform/square_branched.xml

Figure 1.17: Mapping the square torus $\mathbb{C}/(\mathbb{Z} + i\mathbb{Z})$ (left) to a double cover of the sphere (right). Again, the triangulation on the left is symmetric with respect to the elliptic involution, so the image on the right projects to a triangulation of the sphere.

sphere that is invariant under the elliptic involution (exchanging sheets). These two approaches are in fact equivalent due to Remark 1.2.3.

Mapping a flat torus to an elliptic curve. We can also go the opposite way, mapping a flat torus to a double cover of S^2. Start with a triangulated flat torus. The triangulation should be symmetric with respect to the elliptic involution, i.e., symmetric with respect to a half turn around one vertex (which is then also a half turn around three other vertices). The quotient space of the triangulated torus modulo the elliptic involution is then a triangulated sphere. Map it to the sphere by the procedure explained by Bobenko *et al.* [13, Section 3.2], see also Section 1.6.1 of the present work.

Figures 1.16 and 1.17 show examples where we started with a hexagonal and a square torus respectively.

1.7.3 Choosing points on the sphere

The uniformization procedure for elliptic curves described in Section 1.7.2 requires choosing points on the sphere in addition to the four given branch points. For numerical reasons, these points should be chosen so that taking the convex hull leads to triangles that are close to equilateral. We obtained good triangulations by minimizing the following energy for n points in $\mathbb{R}^3$ while fixing the subset of branch points:

$$\mathcal{E} = n^2 \sum_{v \in V} (\langle v, v \rangle - 1)^2 + \sum_{\substack{v,w \in V \\ w \neq v}} \frac{1}{\langle w - v, w - v \rangle}, \tag{1.52}$$

$$\frac{\partial \mathcal{E}}{\partial v} = 4n^2 \langle v, v \rangle v + 4 \sum_{\substack{w \in V \\ w \neq v}} \frac{w - v}{\langle w - v, w - v \rangle^2} \tag{1.53}$$

where $\langle ., . \rangle$ denotes the standard euclidean scalar product of $\mathbb{R}^3$. We do not enforce the constraint that the points should lie in the unit sphere S^2. Instead, we simply project back to S^2 after the optimization.

As initial guess we choose points uniformly distributed in S^2. To achieve this we choose points with normally distributed coordinates and project them to S^2 [48].

1.7.4 Numerical experiments

Given the branch points of an elliptic curve, the modulus τ can be calculated in terms of hypergeometric functions. In this section, we compare the theoretical value of τ with the value $\hat{\tau}$ that we obtain by the discrete uniformization method explained in Section 1.7.2.

We consider elliptic curves in Weierstrass normal form

$$\begin{aligned} \mu^2 &= 4(z - \lambda_1)(z - \lambda_2)(z - \lambda_3) \\ &= 4z^3 - g_2 z - g_3, \end{aligned} \tag{1.54}$$

so the branch points $\lambda_1, \lambda_2, \lambda_3, \infty$ satisfy $\lambda_1 + \lambda_2 + \lambda_3 = 0$, and

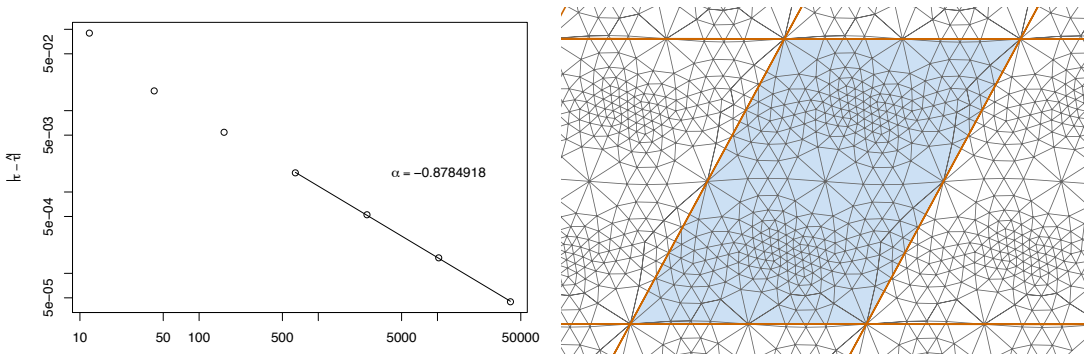
$$g_2 = -4(\lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1), \qquad g_3 = 4\lambda_1\lambda_2\lambda_3. \tag{1.55}$$

We calculate the modulus τ with Mathematica using the built-in function `WeierstrassHalfPeriods[{g2, g3}]`. We normalize τ and the value $\hat{\tau}$ obtained by discrete uniformization so that they lie in the standard fundamental domain of the modular group, $|\tau| > 1$ and $|\mathrm{Re}(\tau)| < \frac{1}{2}$, and we consider the error $|\tau - \hat{\tau}|$. (We stay away from the boundary of the fundamental domain.)

Subdivided icosahedron.

In this experiment we start with the twelve vertices of a regular icosahedron and choose the branch points $\lambda_1, \ldots, \lambda_4$ among them. The remaining points act the role of $p_1, \ldots, p_n$. To study the dependence of $|\tau - \hat{\tau}|$ on the number of points we repeatedly subdivide all triangles into four similar triangles and project the new vertices to S^2. The number of vertices grows exponentially while the triangles remain close to equilateral. Figure 1.18 shows the result of this experiment. It suggests the error behaves like

$$|\tau - \hat{\tau}| = O(n^\alpha), \quad \alpha \approx -0.88. \tag{1.56}$$



data/convergence/subdivision/icosubdivision01.dat

data/convergence/subdivision/data/icosubdivision01.xml

Figure 1.18: Left: Error for zero to six subdivision steps. The log-log plot shows the error $|\tau - \hat{\tau}|$ against the number of vertices of the subdivided icosahedron (i.e., in one sheet of the doubly-covered sphere). To estimate the asymptotic behavior of the error, we determine the slope $\alpha \approx -0.88$ of a line through the last four points by linear regression. Right: Result of the discrete uniformization after two subdivision steps.

Dependence on mesh quality.

In the second experiment we choose the additional points randomly to analyze how the quality of the triangulation affects the approximation error. We use the following quantities to measure the quality of the triangulation based on length cross-ratios for edges:

$$\begin{aligned} Q_{\mathrm{lcr}}(e) &:= \frac{1}{2}\left(\mathrm{lcr}(e) + \frac{1}{\mathrm{lcr}(e)}\right) - 1, \\ Q_{\mathrm{lmr}}(f) &:= \frac{1}{2}\left(\mathrm{lmr}(f) + \frac{1}{\mathrm{lmr}(f)}\right) - 1, \end{aligned}$$

where lcr denotes the length cross-ratio (1.9) of an edge, and lmr denotes the length multi-ratio defined for faces by $\mathrm{lmr}(f) = \prod_{e \in f} \mathrm{lcr}(e)$. If $Q_{\mathrm{lcr}} = 0$ for all edges, then the mesh is discretely conformally equivalent to a mesh consisting of equilateral triangles. So less is better for these quality measures. To get enough "good" triangulations in our samples, we improve random meshes with the procedure described in Section 1.7.3.

Figure 1.19 (left) shows a plot of 2600 triangulations ranging from $n = 20$ to $n = 1500$ vertices. No clear convergence rate is discernible. The situation improves when only samples with a certain minimal mesh quality are considered. For the plot in Figure 1.19 (right) we selected only triangulations with $\max_e\{Q_{\mathrm{lmr}}(e)\} < 0.3$. (The results are similar when using the quality measures $\max_e\{Q_{\mathrm{lcr}}(e)\} < x$ or $\mathrm{mean}_e\{Q_{\mathrm{lcr}}(e)\} < x$.)

The results from these two experiments suggest that the error depends on the number of vertices asymptotically like n^α, where the exponent $\alpha < 0$ depends on the mesh.

1.7.5 Putting a square pattern on a spherical mesh

We can use a variant of the discrete uniformization of elliptic curves (Section 1.7.2) to put a square pattern on a surface that is homeomorphic to a sphere. Figure 1.20 shows an example.

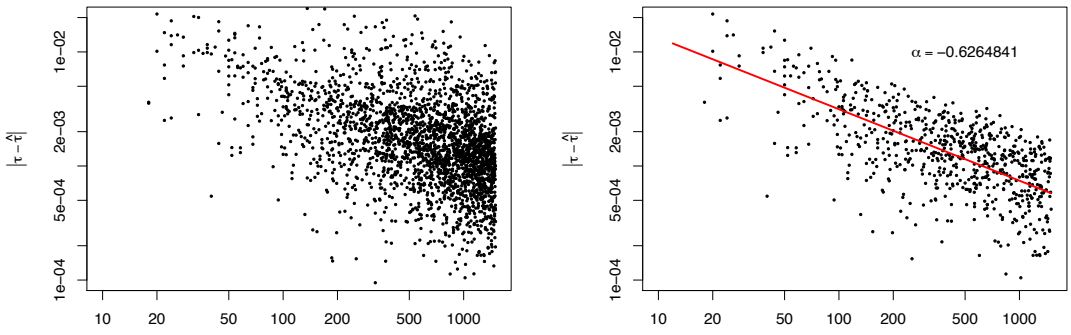


Figure 1.19: Left: log-log plot of the error $|\tau - \hat{\tau}|$ against the number of vertices for a sample of optimized random triangulations with no quality constraint. Right: Only triangulations with $\max_e\{Q_{\mathrm{lmr}}(e)\} < 0.3$ are considered. The regression line with slope $\alpha \approx -0.63$ is shown in red.

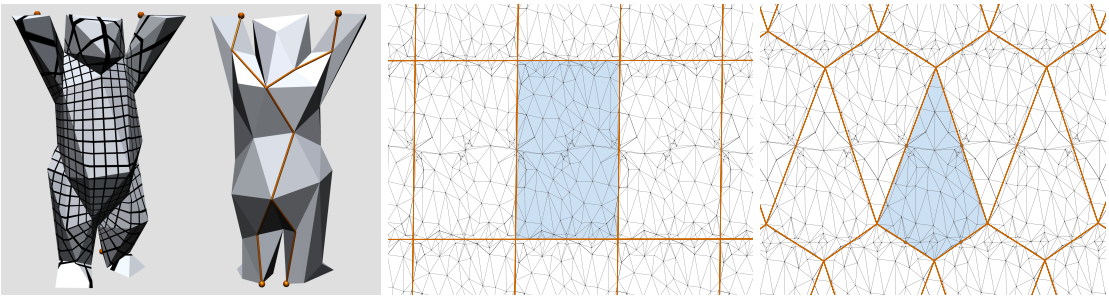


Figure 1.20: The discrete "Berlin Buddy Bear", a mascot of the SFB/Transregio 109 "Discretization in Geometry and Dynamics". The square pattern is put on a bear model as described in Section 1.7.5. Four ramification vertices (marked in red) are chosen at the paws. The uniformization of the branched double cover is shown in the middle. Each fundamental domain covers the bear twice. Fundamental domains of the group generated by rotations around the branch points are shown on the right. Each covers the bear once.

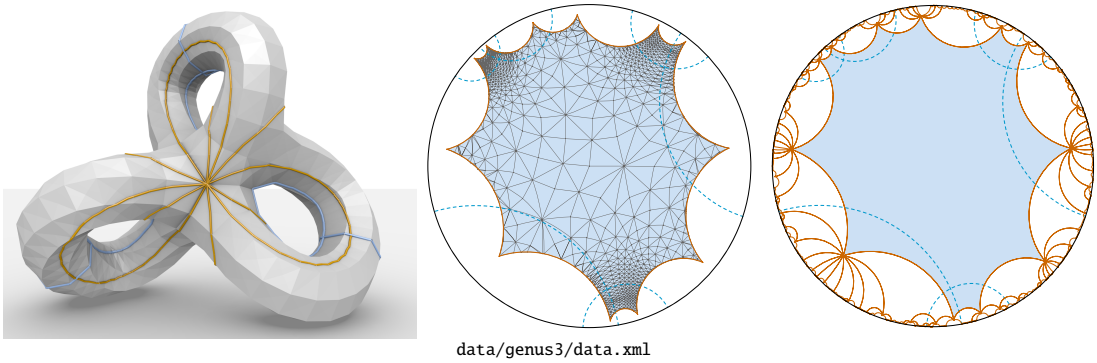


Figure 1.21: Discrete uniformization of an embedded triangulated surface of genus 3. A fundamental polygon with "canonical" edge pairing is shown on the right together with the image mesh. The edges of the polygon (brown) and the axes of the edge-pairing translations (blue) are pulled back to the embedded surface shown on the left.

Pick four vertices of the mesh as ramification points and create a two-sheeted branched cover of the mesh by gluing two copies along paths connecting the selected vertices. The resulting surface is a torus. It can be uniformized using the euclidean functional. The uniformizing group is generated by two translations. This group is a subgroup of the group generated by rotations around the branch vertices. Hence we can achieve the same result as follows. Instead of doubling the surface, prescribe total angles $\Theta = \pi$ at the ramification vertices and $\Theta = 2\pi$ at all other vertices. The result is a flat surface with four cone-like singularities of cone-angle π. The monodromy of the developing map is generated by half-turns. Avoiding the double cover is more efficient because one only has to minimize a function of (approximately) half the number of variables.

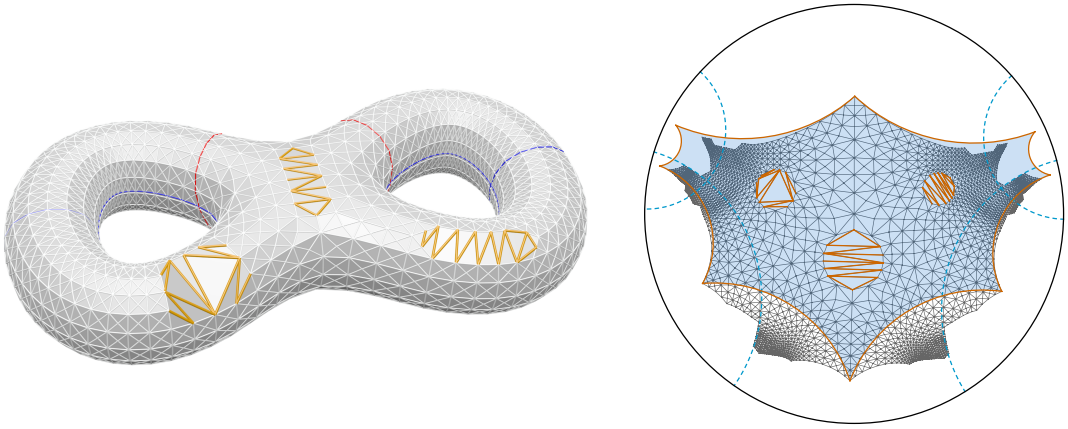
1.8 Uniformization of surfaces of higher genus

As in the case of tori (Section 1.7), we can find uniformizing maps for cyclic polyhedral surfaces of genus $g \geq 2$ by solving the hyperbolic version ($\tilde{g} = hyp$) of Problem 1.3.1 with prescribed total angle $\Theta = 2\pi$ at all vertices. (We will only consider triangulations in the following.) This allows us to calculate approximate uniformizations for Riemann surfaces of genus $g \geq 2$ given in various forms, by approximating them with polyhedral surfaces.

In Section 1.8.1, we briefly discuss how to construct fundamental polygons and group generators.

Not much needs to be said about the uniformization of immersed surfaces. Examples are shown in Figures 1.1 (bottom) and 1.21. In Section 1.5.1 we discussed mappings from multiply connected domains to circle domains. Analogously, one can construct uniformizations of polyhedral surfaces of genus $g \geq 2$ with holes over the hyperbolic plane with circular holes. An example is shown in Figure 1.22. More precisely, the holes are bounded by hyperbolic polygons with vertices on a circle.

We explain how to calculate the Fuchsian uniformization for Riemann surfaces given in the form of a Schottky uniformization in Section 1.8.2. We discuss the uniformization of hyperelliptic curves in Section 1.8.3 and a geometric characterization of hyperelliptic Riemann surfaces in Section 1.8.4.



data/circle_domain_hyperbolic/brezel_{one,two,three}_holes.xml

Figure 1.22: Uniformization of a genus-2 surface with three boundary components over the hyperbolic plane with three circular holes. The three holes are filled with polygons, which are then triangulated during the calculation, see Sections 1.5.1 and 1.3.

1.8.1 Fundamental polygons and group generators

Basic facts and notation. Every compact Riemann R of genus $g \geq 2$ can be represented as the quotient of the hyperbolic plane H^2 modulo the action of a discrete group G of hyperbolic translations,

$$R = H^2/G. \tag{1.57}$$

Presentations of the group G play an important role. We will denote generators by capital letters and their inverses by primes,

$$A' := A^{-1} \in G. \tag{1.58}$$

The uniformization group G can be presented with a finite set of generators

$$A, B, C, D, \ldots \in \mathrm{Isom}(H^2)$$

subject to a single relation $r = 1$,

$$G = \langle A, B, C, D, \ldots \mid r = 1 \rangle, \tag{1.59}$$

where r is a product in which all generators and their inverses appear exactly once. Such presentations are closely related with fundamental polygons: Every fundamental polygon in which all vertices are identified leads to such a presentation.

A fundamental domain of G is an open connected subset D of the hyperbolic plane such that the G-orbit of the closure $\bar{D}$ covers H^2, and $gD \cap D = \emptyset$ for all $g \in G \setminus \{1\}$. A fundamental polygon of G is a fundamental domain with polygonal boundary, i.e., the boundary consists of geodesic segments, the *edges* of the fundamental polygon, which are identified in pairs by the action of the group G. For each edge a, there is exactly one partner edge a' such that there exists a translation $A \in G$ mapping a to a'. These edge-gluing translations form a generating set of G. If all vertices of the fundamental polygon are identified (i.e., they belong to the same G-orbit), then the fundamental polygon has $4g$ edges. In this case there is only one relation for these generators. The relation can be determined from the edge labels, which we always list in

counterclockwise order. For example, if the edges of an octagon are labeled "canonically",

$$aba'b'cdc'd', \tag{1.60}$$

then the relation for the corresponding edge pairing translations is

$$DC'D'CBA'B'A, \tag{1.61}$$

and if opposite edges are identified,

$$abcda'b'c'd', \tag{1.62}$$

the relation is

$$DC'BA'D'CB'A = 1. \tag{1.63}$$

Computational aspects.

Let (Σ, ℓ) be a closed (euclidean, spherical, or hyperbolic) triangulated surface of genus $g \geq 2$. We solve Problem 1.3.1 to obtain a combinatorially equivalent hyperbolic triangulated surface $(\Sigma, \tilde{\ell})_{hyp}$ with angle sum $\Theta = 2\pi$ at every vertex. We lay out the triangles in the hyperbolic plane one-by-one, following a breadth-first search of the the 1-skeleton of the dual cell complex of Σ. (Alternatively, one could use a shortest spanning tree of the 1-skeleton of the dual complex [23].) The result is a fundamental polygon with many vertices. An example is shown in Figure 1.23 (a).

We simplify this fundamental polygon by connecting vertices that are identified with more than one partner by geodesic arcs, as shown in Figure 1.23 (b). The resulting polygon has in general more than one vertex class.

Now we perform the standard algorithm involving cut-and-glue operations (see, e.g., [33]) to obtain a fundamental polygon with one vertex class and so-called canonical edge identification

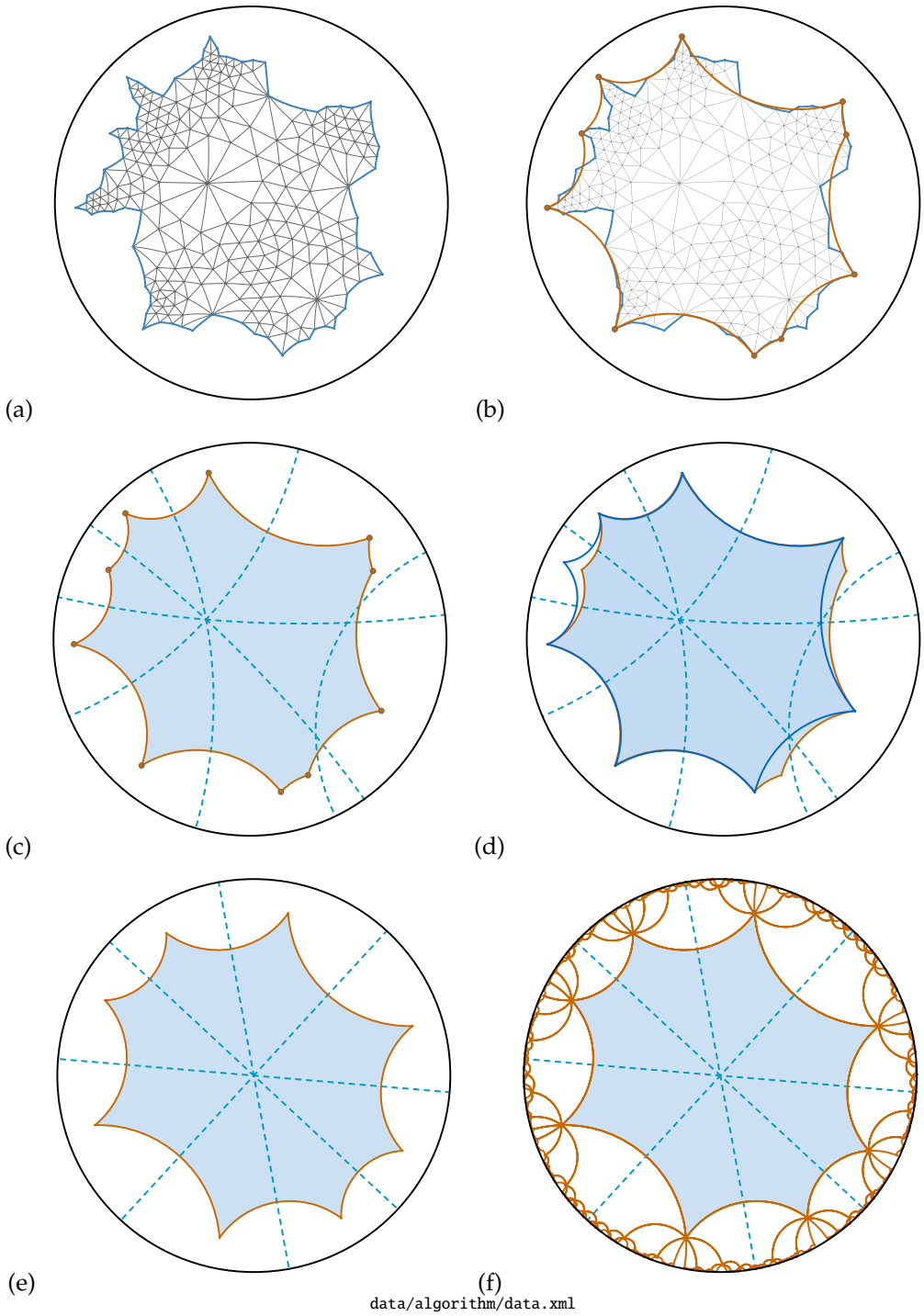
$$aba'b'cdc'd' \ldots . \tag{1.64}$$

During this process we maintain edge-identification transformations, which we represent as $SO^+(2,1)$ matrices.

Hyperbolic translations tend to accumulate numerical errors quite fast when building products. The situation could be ameliorated somewhat by using the $PSL(2,\mathbb{R})$-representation of hyperbolic isometries [25], but the fundamental problem remains. For this reason, it is desirable to perform the cut-and-glue algorithm in such a way that the number of matrix products required to maintain the gluing translations is small.

We start with a fundamental polygon with a single vertex class that is not in canonical form. Then choose sides a and b such that the order in the polygon is $a \cdots b \cdots a' \cdots b' \cdots$. We perform a cut-and-paste operation such that sides a and b are next to each other after transformation, see Figure 1.24. In order to minimize the number of products we choose a and b such the number of sides in between a, b, a', and b' is minimal. We proceed cutting as described before to bring b next to a' and analogously a' next to b'. We end up with a new polygon and the order $aba'b' \cdots$. Repeat this algorithm for the rest of the sides to arrive at canonical form. Again a proof can be found, e.g., in the book by Jost [33]. See Figure 1.25 for an example of this algorithm.

The number of products built during the transformation of a linked pair a, b depends on the number of polygon sides between a, b, a', and b'. Let C be the number of sides between a and b. Let D and E be the number of sides between b, a', and b' respectively. Then the total number of products in this transformation is $6C + 4D + 2E$. In step 1, connecting a and b, we can perform



data/algorithm/data.xml

Figure 1.23: Constructing a fundamental polygon with opposite edges identified. (a) Laying out hyperbolic triangles creates a fundamental polygon with many vertices. (b) Straighten the edges between vertices that are identified with more than one partner (shown in red). (c) Axes of the edge-pairing translations are shown in blue. (d,e) Two cut-and-paste operations lead to a fundamental polygon with one vertex class and opposite edges identified. The axes intersect in one point (see Section 1.8.4). We move this point to the origin. (f) Tiling the hyperbolic plane with fundamental polygons.

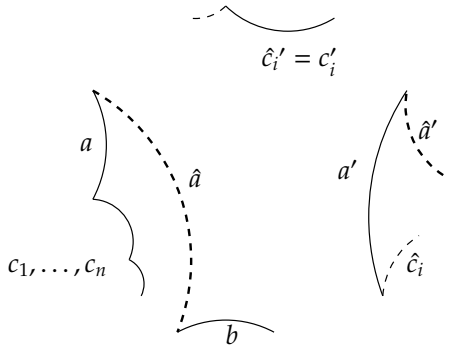


Figure 1.24: Cut-and-paste: Cut from the start of a to the start of side b. Glue the piece along a' using transformation A that moves side a to $'a$. Intermediate sides $c_1, \ldots, c_n$ between a and b are transformed accordingly and group elements transform as $\hat{C}_i = C_i A'$. Their inverses C_i' become $\hat{C}_i' = AC_i'$

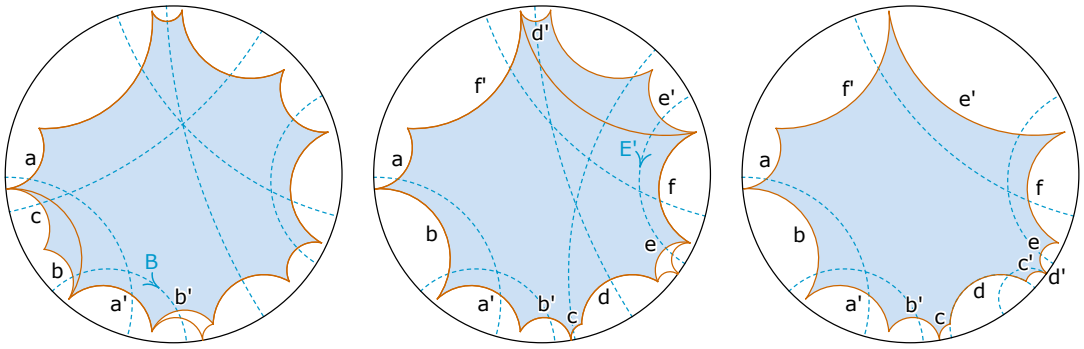


Figure 1.25: Motions to transform a given fundamental domain into canonical form. The shaded area is the current fundamental polygon. Step 1: Choose the linked pair a, b. Cut at c and b and identify via B to put side a next to side b. Choose linked pair e, f. Cut between e' and d' and identify via E' to put side e' next to f'. The remaining handle $cdc'd'$ is already in place and we arrive at canonical form $aba'b'cdc'd'efe'f'$.

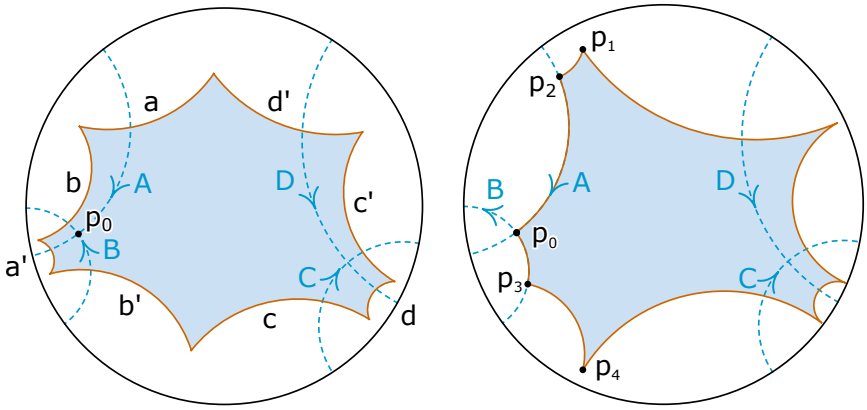


Figure 1.26: The algorithm of Linda Keen to construct strictly convex fundamental polygons. Start with any canonical fundamental polygon $aba'b'cdc'd'$ with a corresponding relation $DC'D'CBA'B'A = 1$ (left). We choose the intersection p_0 of the axes of transformations A and B as base point for the new domain. The new vertices of the fundamental domain are calculated as $p_1 = A'Bp_0$, $p_2 = A'p_0$, $p_3 = Bp_0$, and $p_4 = BA'p_0$. The other vertices are obtained similarly from p_4 by applying C and D.

the cut-and-past operation using transformation b and a cut between the end of a and the end of b in counter-clockwise order. This moves the intermediate sides between a and b out of the way for the next 2 steps. By this we have a total number of products $2C + 4D + 2E$. We use a greedy implementation that chooses the smallest number of products in each step to select a and b. Alternatively one could enumerate the transformation tree and choose the cheapest path of all transformation sequences.

The polygon with canonical edge identifications may not be convex. Following Linda Keen [36], we can transform this domain into a strictly convex fundamental polygon by choosing a different base vertex for the same group of transformations. Let $aba'b'cdc'd'\ldots$ be a fundamental polygon and let

$$G = \langle A, B, C, D, \ldots \in PSL(2, \mathbb{R}) \mid \ldots DC'D'CBA'B'A = 1 \rangle \tag{1.65}$$

be the corresponding presentation of the uniformization group, see Figure 1.26 (left). Then the axes of the generators A and B intersect in a point p_0. Choosing p_0 as the base point of a new fundamental polygon as shown in Figure 1.26 (right) renders it convex and uniquely determined for the given group and presentation.

Fundamental polygons with opposite sides identified.

When we consider the geometric characterization of hyperelliptic surfaces in Section 1.8.4, we want to transform fundamental polygons into fundamental polygons with opposite sides identified, i.e., polygons with edge labeling

$$abcd\cdots a'b'c'd'\cdots .$$

Any fundamental polygon can be transformed into a fundamental polygon with opposite edges identified by cut-and-glue operations: First transform the polygon to canonical form $aba'b'cdc'd'\ldots$ by the standard algorithm. Playing a sequence of steps that transforms a polygon with opposite edges identified to canonical form backwards, transforms the canonical polygon to a polygon with opposite edges identified.

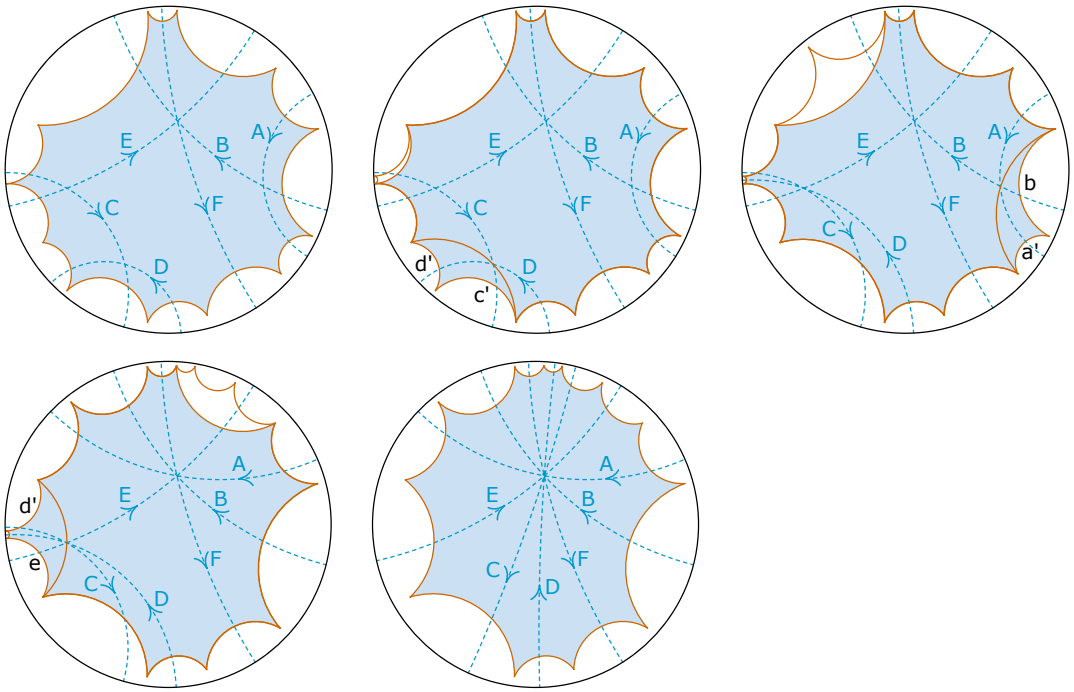


Figure 1.27: Algorithm example: Start with group relation $B'F'D'E'A'FEC'DCBA = 1$. Step 1: Cut from the start of d' to the end of c' and identify via C' to move C' behind D'. The resulting relation is clustered such that the elements of the first half form the inverses of the second half: $B'F'D'C'E'A'FEDCBA = 1$. We now sort elements $A, \ldots, B$ to match the order of $A', \ldots, F'$ or vice versa. Step 2: Cut at a' and b and identify via B to move B behind A'. We get relation $B'F'D'C'E'A'BFEDCA = 1$. Step 3: Cut between d' and e and identify via E to move E behind C. We get $B'F'D'C'E'A'BFDCEA = 1$ as the final relation and a fundamental polygon with oppositely identified sides.

This algorithm is not optimal with respect to the number of multiplications necessary to maintain the edge-gluing translations. Especially if the original polygon is already "close" to one with opposite sides identified, the detour via a canonical polygon is inefficient.

In all examples, we use a heuristic method based on the following idea: Find one of the longest sequences of different letters in the edge labeling (ignoring primes), and then try to move a different letter into this sequence by cutting and gluing.

For a given fundamental polygon and the ordering of its sides, we can reconstruct the relation of the group by moving around the base vertex on the surface. That means if we start with transformation A moving side a to a', the next element in the relation belongs to the side next to a' in counter-clockwise order. We proceed until we get back to element A. Note that we could have chosen clockwise orientation, but for drawing pictures we stick to counter-clockwise.

For an oppositely identified fundamental polygon (and for canonically identified polygons as well) the order of transformations in the relation has the same structure as the sides of the polygon (by reading the relation backwards and possibly relabeling of sides). For example the group relation corresponding to a polygon $ab'cd'a'bc'd$ is $D'C'B'A'DCBA = 1$.

For the order of group elements in the relation we give cut-and-paste sequences that move single

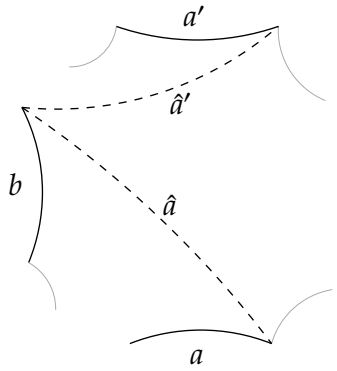


Figure 1.28: Move element A in front of B in a relation that moves counter-clockwise around the base vertex. Choose the upper cut (dashed lines) and paste via transformation A' moving side a' to a. Or cut at the lower one and paste via transformation A.

elements to different locations in the relation leaving all other elements in place. Using these transformations we normalize the polygon as follows.

(1) Find largest sequence of different (excluding inverse) generators in the relation.
(2) Move missing elements into the sequence such that it forms a basis for the group.
(3) Sort elements in the sequence to match the order of the corresponding inverse elements.

Let the order of sides in the polygon be $a' \cdots b \cdots a \cdots$. The corresponding cut-and-paste operation that moves the group element A behind element B in the relation is one of

- Cut from the end of a to the start of b and glue using A.
- Cut from the start of a' to the start of b and glue via A'.

Choose the one with the lowest number of products, see Figure 1.28. If the order of sides in the polygon is $a' \cdots a \cdots b \cdots$ we can however not perform the operation, the resulting domain would be disconnected. We therefor define the cost for such an operation as infinite. In step 2 we minimize over all possible transformations that move a selected element into the sequence. We do not show here that the cost is always a finite during the algorithm and call this algorithm heuristic due to this fact. Depending on the sort algorithm we use in step 3 we have the freedom to choose between the motion of the inverse elements or their partners. In our examples this gives enough freedom to create all oppositely identified fundamental domains. See Figure 1.27 for an example of this algorithm.

1.8.2 From Schottky to Fuchsian uniformization

In this section, we consider Riemann surfaces presented as quotient spaces of classical Schottky groups.

Definition 1.8.1. *Let $C_1, C_1', \ldots, C_g, C_g'$ be circles in $\hat{\mathbb{C}}$ that bound disjoint disks. A* classical Schottky group *is a Kleinian group generated by Möbius transformations $\sigma_1, \ldots, \sigma_g$, where σ_j maps the outside of C_j onto the inside of C_j'.*

Each generator σ_j has fixed points A_j, B_j inside C_j and C_j', respectively. The limiting set A of G is the union of orbits of the fixed points A_j, B_j. G acts freely and properly discontinuously on the domain of discontinuity $\Omega = \hat{\mathbb{C}} \setminus A$. The quotient space $R = \Omega/G$ is a Riemann surface

of genus g. The domain outside all of the circles is a fundamental domain of G. The identified pairs of circles form handles.

We discretize the Riemann surface $R = \Omega/G$ determined by a classical Schottky group G as follows. First, construct a triangulation of Ω whose vertex set and combinatorics are invariant under the action of G. (Ignore the fact that a Möbius transformation maps straight edges to circular arcs as in Proposition 1.2.5 on the Möbius invariance of conformal classes.) For example, the triangulation may be the Delaunay triangulation of a G-invariant point set. The following construction avoids Delaunay triangulations of infinite (but symmetric) point configurations:

If necessary, choose a Möbius normalization for which the fundamental domain is bounded in $\mathbb{C}$. For each pair of circles C_j, C_j' we construct polygons $p_{1j}, \ldots, p_{n_j j}$ inscribed in C_j and $p'_{1j'} \ldots, p'_{n_j j}$ inscribed in C' such that $\sigma_j(p_{kj}) = p'_{kj}$. For example, we may choose a regular n-gon inscribed in C_j and map the vertices by σ_j to C_j'. Triangulate the compact region bounded by these polygons, adding vertices in the interior as wanted. (For example, use a constrained Delaunay triangulation.) The images of this triangulation under the action of G (again, considering only combinatorics and vertex positions) form a G-invariant triangulation $\hat{\Delta}$ of the universal cover of R, hence a triangulation Δ of R. More precisely, the triangulations $\hat{\Delta}$ and Δ are only defined up to isotopy fixing the vertices. The edge-lengths $\hat{\ell}$ (distances of vertices) do not project from $\hat{\Delta}$ to Δ, but the length cross-ratios $\widehat{\text{lcr}}$ calculated from these edge lengths do, because they are Möbius-invariant. The projected length cross-ratios lcr determine a discrete conformal class for Δ (see Section 1.2.5).

To obtain a Fuchsian uniformization of R, construct edge lengths ℓ from the length cross-ratios lcr as described in Section 1.2.6. Then solve Problem 1.3.1 (or rather the corresponding analytic version, Problem 1.3.4) for $(\Delta, \ell)_{euc}$ with $\tilde{g} = hyp$ and desired angle sums $\Theta = 2\pi$ at all vertices.

Note that the lengths ℓ calculated from length cross-ratios lcr may not satisfy all triangle inequalities. This does not matter for the corresponding analytic Problem 1.3.4 (with $V = V_1$, $E = E_1$). If Problem 1.3.4 has a solution, it is in the discrete conformal class determined by the length cross-ratios lcr. Also, whether or not Problem 1.3.4 has a solution does not depend on the choice of edge lengths ℓ provided they lead to the same length cross-ratios.

Figure 1.29 shows examples of the Fuchsian uniformization of a genus two and three surface presented by its Schottky uniformization.

Tori given by Schottky data. For tori, the Schottky data consists of one generator

$$\frac{\sigma(z) - A}{\sigma(z) - B} = \mu \frac{z - A}{z - B} \tag{1.66}$$

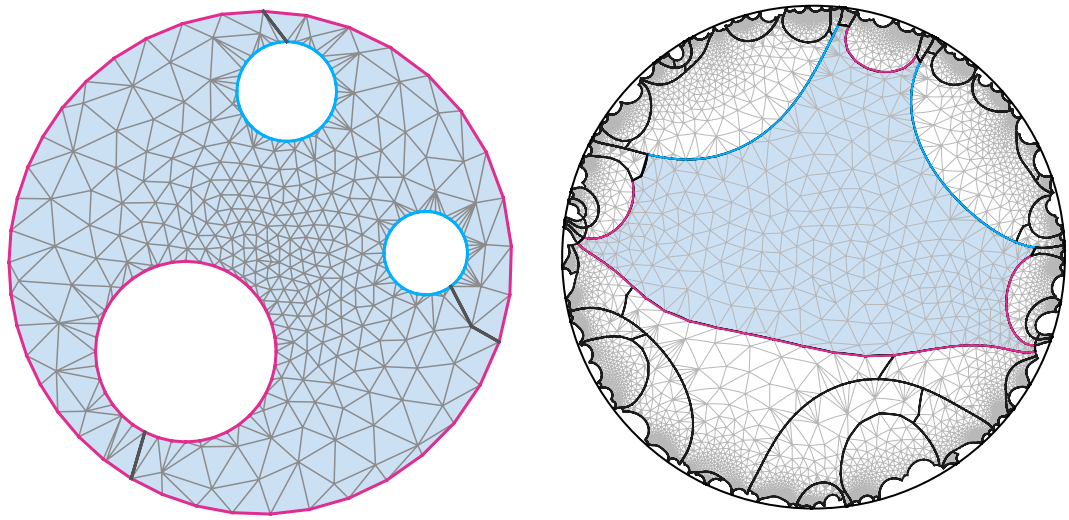
and one pair of circles. To find a uniformization $\mathbb{C}/\Gamma$ is elementary. It suffices to consider the case where $A = B = 0$ (and C, C' are concentric circles around 0 with radii i and μ. Figure 1.30 shows two examples where we apply the discrete method without adding extra points inside the fundamental domain of the Schottky group.

1.8.3 Hyperelliptic curves

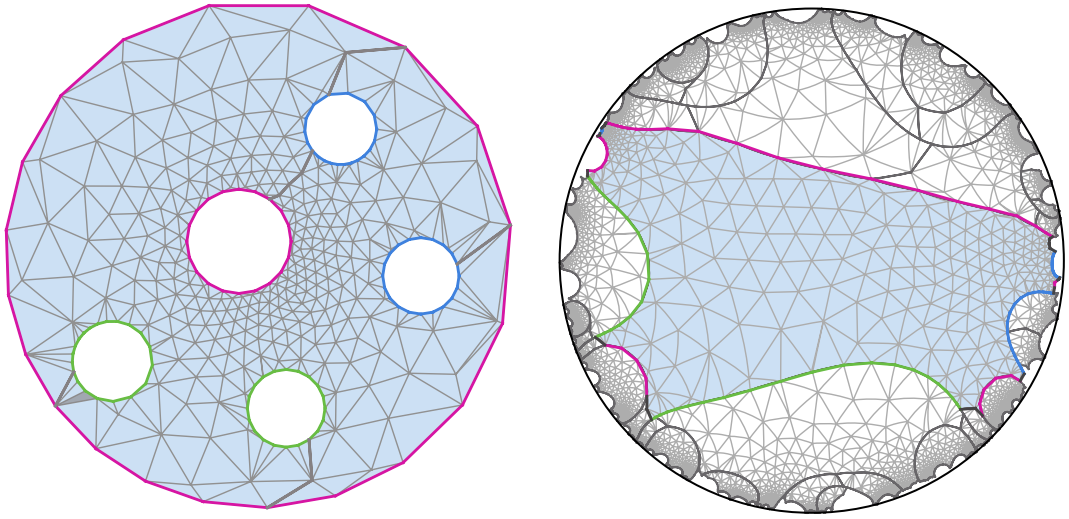
A hyperelliptic curve is a complex algebraic curve of the form

$$\mu^2 = p(\lambda), \tag{1.67}$$

where p is a polynomial of degree $d \geq 5$ with d distinct roots. For $d = 2g + 2$ or $d = 2g + 1$, the hyperelliptic curve becomes a compact Riemann surface of genus g after singularities at



data/schottky_g2/genus2_fine.xml



data/schottky_g3/schottky.xml

Figure 1.29: Discrete Riemann surface of genus 2 and 3 given by Schottky data (left) and the corresponding Fuchsian uniformizations (right). Circles with the same color are identified. The extra points of the triangulation are chosen so that the triangles are close to equilateral where possible. The shaded region in the right images corresponds to the fundamental domain of the Schottky group in the left image. Its boundary consists of curves corresponding to the circles and curves corresponding to lines connecting the circles (drawn in gray).

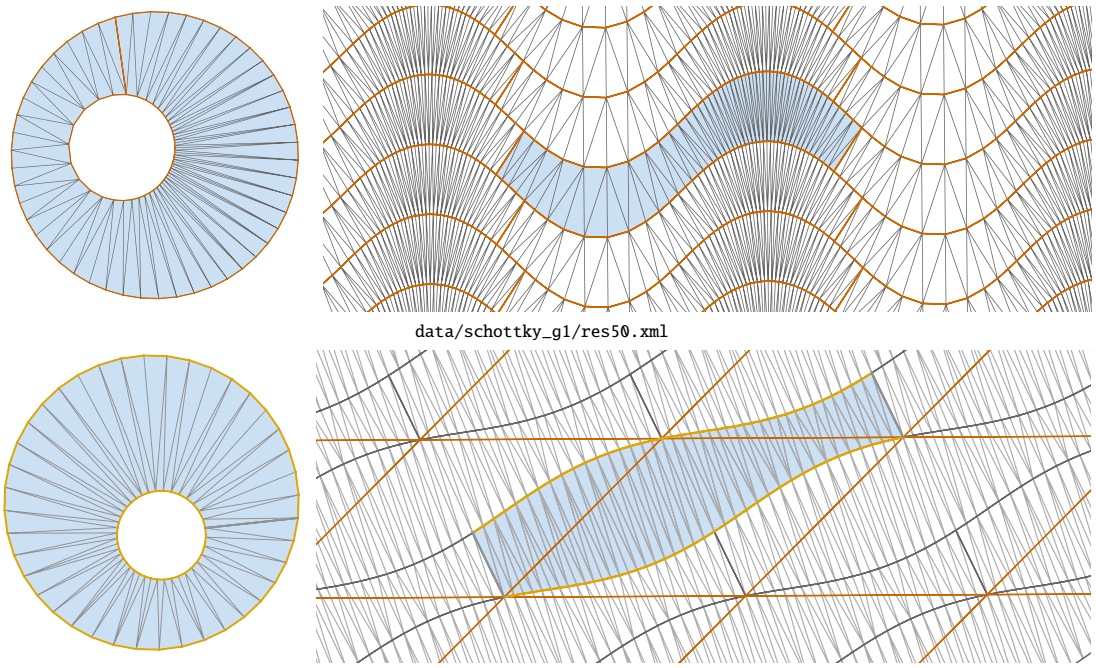
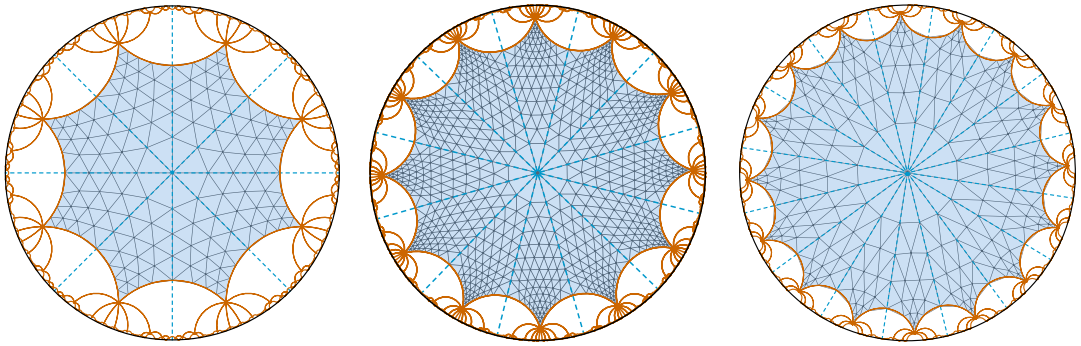


Figure 1.30: Left: Fundamental domains of Riemann surfaces of genus 1 given by Schottky data C, C', A, B, μ (see (1.66)). The triangulations use only points on the circles C, C'. We deliberately chose non-concentric circles, i.e., with centers $A \neq B$. Right: Representation of the same surfaces as $\mathbb{C}/\Gamma$ for a lattice Γ. Top: For real $\mu = 0.3$ we get a rectangular lattice. Bottom: $\mu = 0.08 + 0.01i$ yields a parallelogram.



data/hyperelliptic_{g2,g3,g4}/data.xml

Figure 1.31: Uniformizations of the hyperbolic curves (1.68) with genus 2, 3, and 4. The triangulation of the surfaces is a regular 1-to-4 subdivision of the convex hull of the branch points. Due to the symmetries of these curves, the fundamental domains are regular hyperbolic $4g$-gons. Since the triangulation is as symmetric as the curves, and because the solution of the discrete uniformization problem is unique, the fundamental domains of the polyhedral surfaces are also exactly regular hyperbolic $4g$-gons. Any error in the domains is therefore due to numerics and not due to the discretization.

infinity are resolved. For our purposes, a hyperelliptic curve is just a branched double cover of the λ-sphere with branch points $\lambda_1, \dots, \lambda_{2g+1}, \infty$ if $d = 2g + 1$ and branch points $\lambda_1, \dots, \lambda_{2g+2}$ if $d = 2g + 2$.

We construct a polyhedral approximation of a hyperelliptic curve in the same way as for elliptic curves (Section 1.7.2). We choose points $p_1, \dots, p_n$ in addition to the $2g + 2$ branch points and take the convex hull. We cut the resulting polyhedron open along edge paths joining pairs of branch points and glue a second copy along the cuts.

Figure 1.31 shows uniformizations of the curves

$$\mu^2 = \lambda \prod_{k=1}^{2g} \left(\lambda - e^{\frac{ik\pi}{g}}\right). \tag{1.68}$$

for $g = 2, 3, 4$ that were obtained this way. The curves are branched at the $2g^{\text{th}}$ roots of unity and at 0 and ∞.

Mapping a polyhedral surface to a hyperelliptic curve. We can also map a triangulated surface of genus g to a branched double cover of the sphere, provided it is symmetric with respect to a discrete conformal involution with $2g + 2$ fixed points, which are vertices. In the simplest case, the involution is an isometry. (Compare Section 1.7.2, where we map flat tori to elliptic curves.) Taking the quotient of the triangulation with respect to the involution, we get a triangulated sphere with a discrete conformal structure, which we map discretely conformally to the sphere. Figure 1.32 shows an example.

1.8.4 Geometric characterization of hyperelliptic surfaces

A Riemann surface R of genus $g \geq 2$ is called *hyperelliptic*, if one of the following equivalent conditions is true (and hence all are):

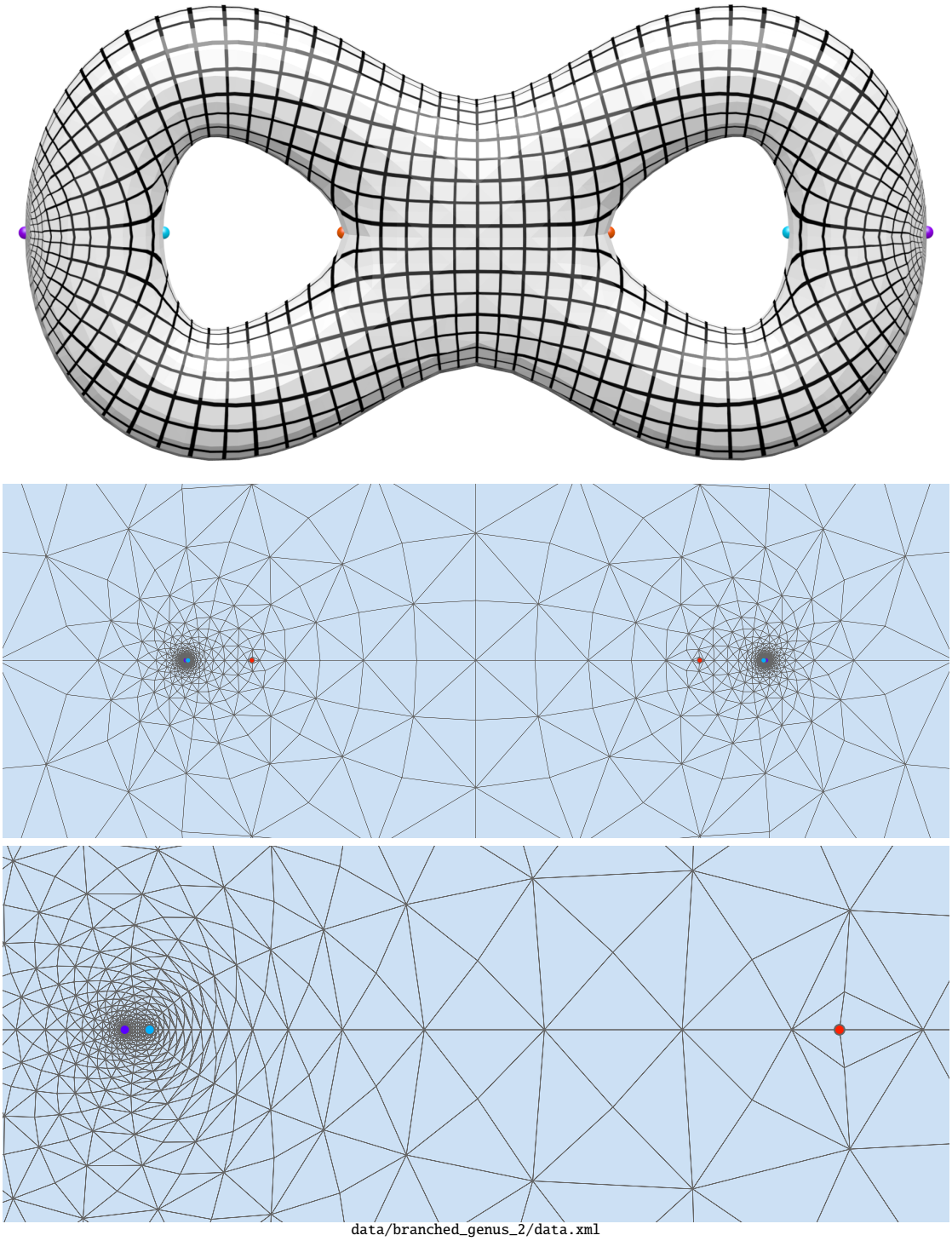


Figure 1.32: A triangulated genus-2 surface is mapped to a branched cover of $\hat{\mathbb{C}}$. The 180° rotation about the horizontal symmetry axis is a discrete conformal involution with 6 fixed points marked in red, blue, and purple. The texture is a square grid in the plane, pulled back to the doubly-covered sphere by Mercator projection, then pulled back to the surface. Middle and bottom: branched cover of $\hat{\mathbb{C}}$, and a closeup of three branch points (stereographic projection).

(i) R is conformally equivalent to some hyperelliptic curve.
(ii) R is conformally equivalent to a branched cover of the sphere with $2g + 2$ branch points.
(iii) There is a conformal involution $\tau : R \to R$ with exactly $2g + 2$ fixed points.

The involution τ is called the *hyperelliptic involution* of R. By the Riemann-Hurwitz formula, the quotient surface R/τ is a sphere.

All Riemann surfaces of genus two are hyperelliptic, but for every genus greater than two, there are Riemann surfaces that are not hyperelliptic. The following geometric characterization of hyperelliptic Riemann surfaces is due to Schmutz Schaller [59, 60].

Theorem 1.8.2. *Let R be a closed hyperbolic surface of genus g. Then the following statements are equivalent:*

(i) R is hyperelliptic.
(ii) R has a set of $2g - 2$ simple closed geodesics which all intersect in one point and which intersect in no other point.
(iii) R has a set of $2g$ simple closed geodesics which all intersect in one point and which intersect in no other point.
(iv) R has a fundamental polygon that is a $4g$-gon with opposite sides identified and equal opposite angles.

The fundamental polygon of condition (iv) is symmetric with respect to a 180° rotation around its center, which corresponds to the hyperelliptic involution on R. The $2g+2$ fixed points on R are the vertex of the polygon, its center, and the $2g$ edge midpoints. The axes of the $2g$ edge-gluing translations all go through the center. They project to $2g$ simple closed geodesics on R that all intersect in one point and intersect in no other point.

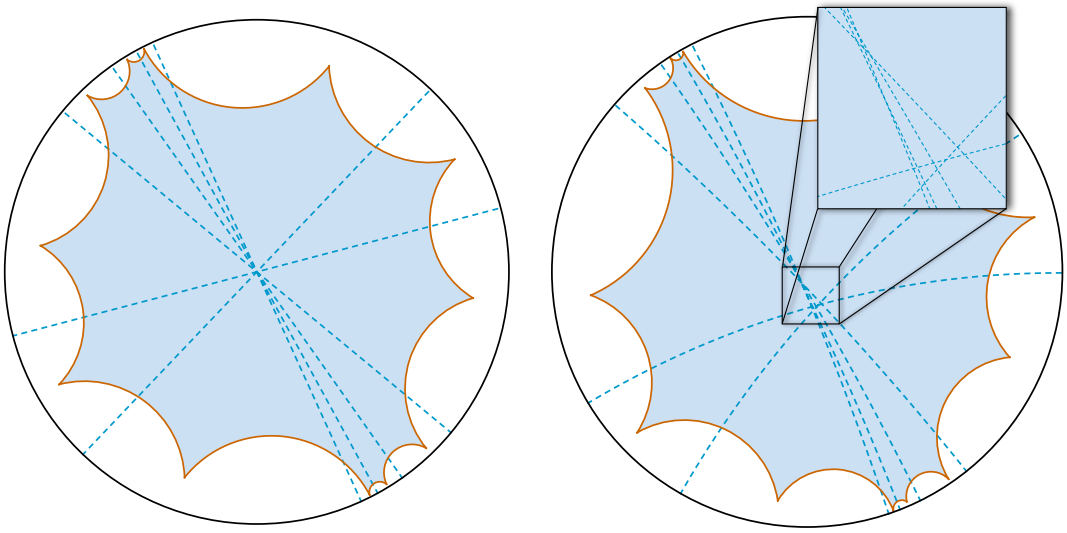
1.8.5 Example: Deforming a hyperelliptic surface

We uniformize a hyperelliptic surface obtaining a centrally symmetric fundamental polygon with opposite edges identified as predicted by Theorem 1.8.2. The axes of the generators meet in one point. Then we deform the surface slightly to a non-hyperelliptic surface to see how the fundamental polygon and the axes change. The result is shown in Figure 1.33.

For this example, we construct an elliptic-hyperelliptic triangulated surface with additional symmetry. A surface is called *elliptic-hyperelliptic* if it is conformally equivalent to a two-sheeted branched cover of the torus.

Take two regular tetrahedra (the faces of which are subdivided several times to obtain a finer mesh), cut them across pairs of opposite edges and glue them together to obtain a two-sheeted cover of a tetrahedron branched at the four vertices. Now choose two paths in one of the sheets that connect the centers of the tetrahedron's faces in pairs. Cut the surface along these paths, take another copy of this cut surface and glue corresponding cuts together to form an elliptic-hyperelliptic surface of genus three that is a four-fold cover of a regular tetrahedron. The surface possesses six anti-holomorphic involutions corresponding to the six reflectional symmetries of the tetrahedron, and three holomorphic involutions corresponding to the rotational symmetries of the tetrahedron of order two. Each of the holomorphic involutions has eight fixed points covering the midpoints of a pair of opposite edges. Thus, this elliptic-hyperelliptic surface is also hyperelliptic.

Figure 1.33 (left) shows a uniformization of the hyperelliptic elliptic-hyperelliptic surface. Destroying the symmetry by moving all points of the polyhedral surface in space by a small random



data/non-hyperelliptic/tetrahedron_g3_01_symmetric.xml

Figure 1.33: Hyperelliptic vs. non-hyperelliptic. Left: Uniformization of a hyperelliptic surface with a centrally symmetric fundamental polygon. The axes of the generators meet in a common point. Right: Uniformization of the deformed surface, which is not hyperelliptic. The axes do not meet in one point.

offset destroys the hyperellipticity of the surface, see Figure 1.33 (right).

Numerical Data. We list the numerical $SO^+(2,1)$ matrices of the generators of the group

$$\langle T_1, T_2, T_3, T_4, T_5, T_6 \mid T_6 T_5{}^{-1} T_4 T_3{}^{-1} T_2 T_1{}^{-1} T_6{}^{-1} T_5 T_4{}^{-1} T_3 T_2{}^{-1} T_1 = 1 \rangle \tag{1.69}$$

representing the hyperelliptic elliptic-hyperelliptic surface constructed in this section (see Figure 1.34). The matrices satisfy the relation with error $\approx 10^{-7}$.

$$T_1 = \begin{bmatrix} 2.05443154523212 & -4.021591426903446 & -4.403849064057392 \\ -4.021591427085276 & 16.338309707059754 & 16.796236533536394 \\ -4.403849064222335 & 16.796236533484112 & 17.392741252301292 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 7.906334736200989 & -6.57792280760043 & -10.236171033333449 \\ -6.5779228079025245 & 7.265127613618063 & 9.749417813849163 \\ -10.236171033527825 & 9.749417813638956 & 14.171462349831586 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 933.210063638192 & 509.0929753776527 & 1063.0407708335915 \\ 509.09297492442374 & 279.0228056502974 & 580.5414569092936 \\ 1063.0407706165242 & 580.5414573067374 & 1211.2328692884857 \end{bmatrix}$$

$$T_4 = \begin{bmatrix} 47.8208492808903 & 21.282776040302117 & -52.33345184418173 \\ 21.28277609643665 & 10.67424906068982 & -23.788571865092973 \\ -52.333451867010325 & -23.788571814871467 & 57.49509834158029 \end{bmatrix}$$

$$T_5 = \begin{bmatrix} 933.2100574645401 & 509.09297238055706 & -1063.040763978619 \\ 509.092972765322 & 279.02280467565924 & -580.5414545474814 \\ -1063.0407641628826 & -580.5414542100707 & 1211.2328621402066 \end{bmatrix}$$

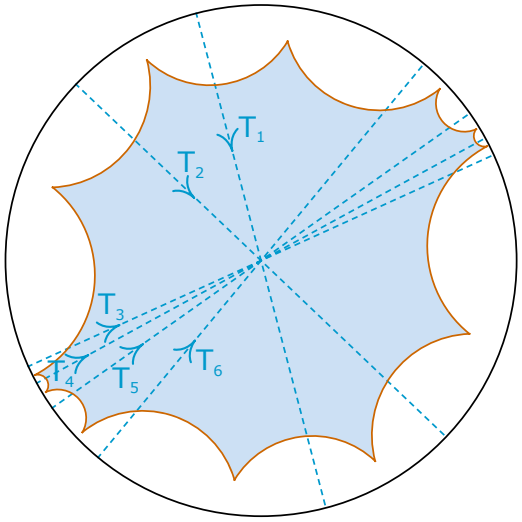


Figure 1.34: Generator labels

$$T_6 = \begin{bmatrix} 128.62265665383228 & 90.05086671584104 & -157.0093831621644 \\ 90.05086668827934 & 64.5401174556973 & -110.78621463208009 \\ -157.00938314635744 & -110.78621465448322 & 192.16277410952506 \end{bmatrix}$$

1.8.6 Example: Different forms of the same genus-2 surface

In this section we present Fuchsian uniformizations of the same Riemann surface represented in three different ways:

- as hyperelliptic curve $\mu^2 = \lambda^6 - 1$ (Figure 1.35),
- as Lawson's genus-2 minimal surface in S^3 [42] (Figure 1.36),
- and as a surface glued from six squares (Figure 1.37).

For each representation we choose corresponding fundamental polygons that allow the comparison of the uniformization:

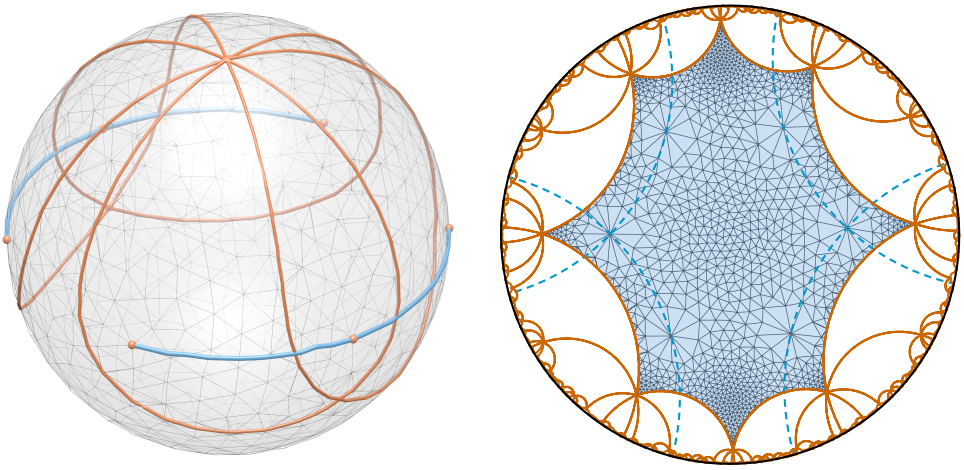
- an octagon with canonical edge pairing $aba'b'cdc'd'$,
- an octagon with opposite sides identified, $abcda'b'c'd'$,
- a 12-gon that is adapted to the six-squares surface.

All data presented in this section is available on the DGD Gallery webpage [62].

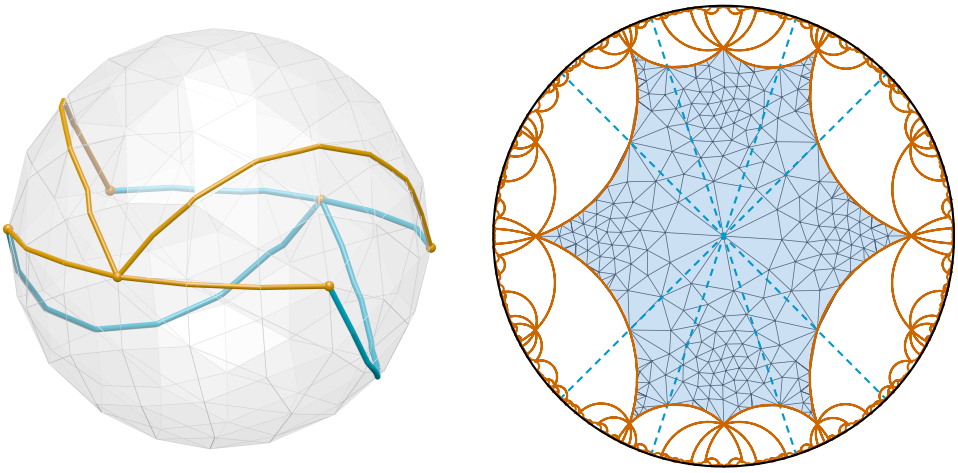
Hyperelliptic curve.

We uniformize the hyperelliptic curve $\mu^2 = \lambda^6 - 1$ as described in Section 1.7.2. The results are shown in Figure 1.35.

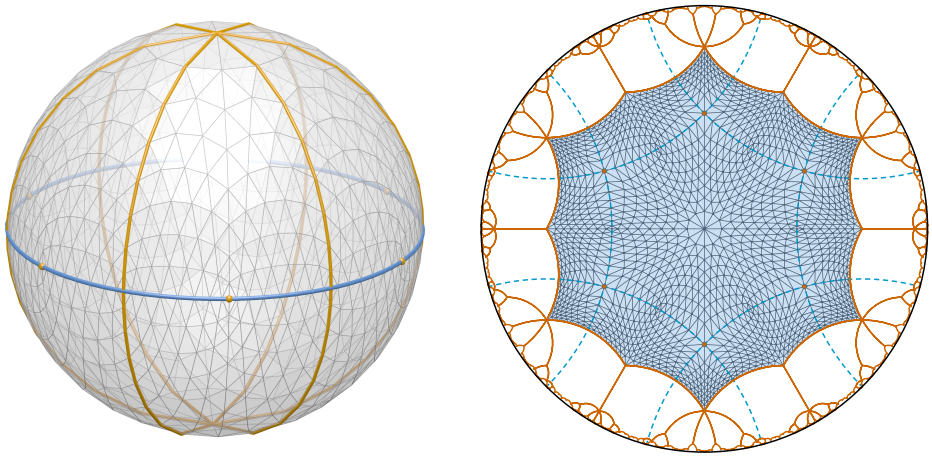
Canonical domain: For the canonical fundamental domain we pick a vertex at 0 on one of the sheets to be the vertex of the domain. We proceed as described in Section 1.8.1. Extra points are chosen such that the triangulation is almost regular on the sphere. The system of loops on the surface is chosen such that each loop starts at the base vertex, runs around two branch points, and returns, see the red curves in Figure 1.35, top-left.



data/lawson_curve/curve_canonical.xml



data/lawson_curve/curve_opposite.xml



data/lawson_curve/curve_cyclic.xml

Figure 1.35: Uniformization of the hyperelliptic curve $\mu^2 = \lambda^6 - 1$. Left: Triangulated double cover of the sphere branched at the 6th roots of unity, with the boundary of the fundamental domain shown in brown and the axes of generators shown in blue. Right: Fuchsian uniformization and fundamental polygons. Canonical polygon (top), polygon with opposite sides identified (middle), and 12-gon specially adapted to the six-squares surface (bottom).

For the visualization we truncate the triangles of the universal cover to fill the fundamental domain only. Figure 1.35 shows the triangulation on $\hat{\mathbb{C}}$ (left) and the universal cover (right). Red curves are the boundary of the fundamental domain. We show them also pulled back onto $\hat{\mathbb{C}}$. Blue curves are simple closed geodesics on the surface. They are the axis of hyperbolic motions identifying pairs of fundamental edges in the hyperbolic plane.

The branch points are clearly visible: The valence of the triangulation is doubled at these vertices. The blue geodesic loops connect two branch points each.

Opposite sides domain: The opposite sides domain (second row) is chosen as described in Section 1.8.1. All axes of hyperbolic motions intersect in a branch point of the curve. Red and blue geodesic arcs are dual to each other. For the example model we chose a coarse discretization of the underlying curve and truncate the triangulation to fit into the fundamental domain.

12-*gon:* This domain is chosen such that it corresponds to the domain suited for the square-tiled surface as described below. To understand the system of loops, take the canonical system of loops meeting at 0 (top row of the figure) and deform the loops until they touch at ∞. By this move we introduce two more vertices on the fundamental polygon, namely the two vertices corresponding to ∞ on the two-sheeted cover. The triangulation of the example model is created by a subdivision process. Starting from a triangulation with vertices at 0, ∞, at branch points, and the mid-points on the equator in between the branch points, we subdivide using a linear 4-to-1 split followed by a projection to the sphere. Thus the boundary of the fundamental domain is contained in the edge set of the triangulation. For the visualization of the universal cover we cut along these edges to draw the triangulation, Figure 1.35, bottom-right.

Lawson's surface.

Figure 1.36 shows Fuchsian uniformizations of Lawson's minimal surface in S^3. The triangulated surface model was kindly provided by Konrad Polthier [52].

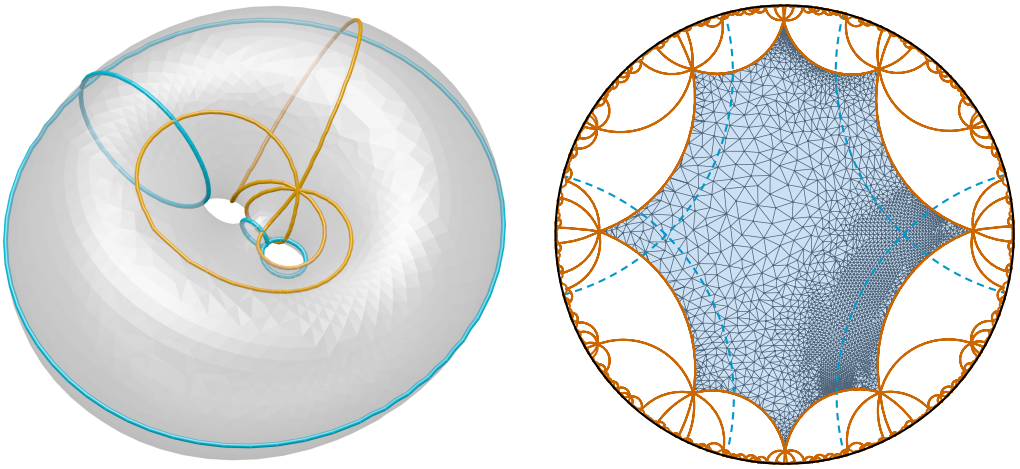
This model of the Lawson surface realizes the hyperelliptic involution as a euclidean rotational symmetry. Its symmetry axis meets the surface in six points. These fixed points of the hyperelliptic involution correspond to the branch points of the hyperelliptic curve representation. This allows us to uniformize the model with corresponding fundamental domains.

The fundamental domain where opposite sides are identified is chosen as described in Section 1.8.1. The vertex of this fundamental polygon has to be a point on the symmetry axis of the surface. As the system of loops forming the boundary of the polygon and their identifying axes are dual to each other, the common intersection of generator axes must lie on an intersection with the symmetry axis as well, see Figure 1.36, middle-left.

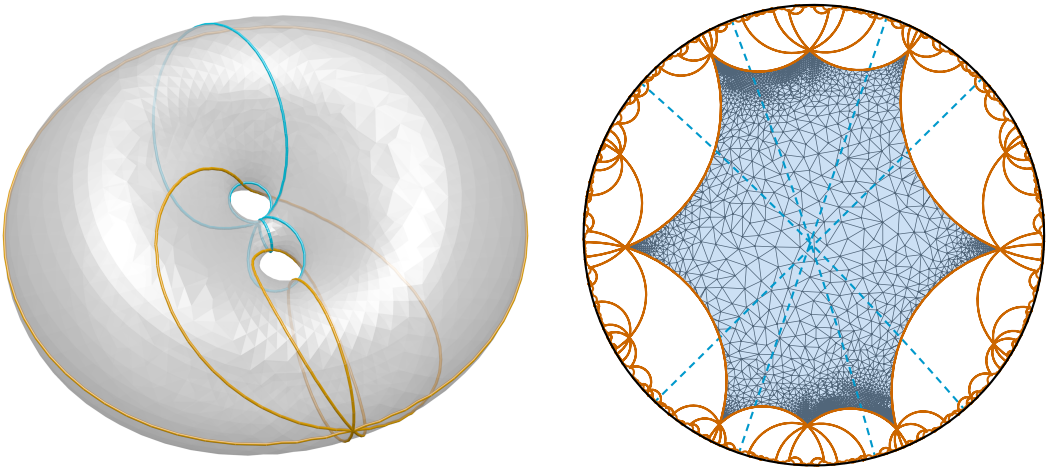
The third fundamental polygon we present for Lawson's surface corresponds to the square-tiled segmentation of the surface as described below. As in the uniformization of the hyperelliptic curve we have to deform the canonical polygon and introduce two new vertices corresponding to antipodal points of the current vertex on the curve. The hyperelliptic curve of the Lawson surface admits several anti-holomorphic involutions, one of which interchanges 0 and ∞. We also have these involutions on the embedded surface realized as reflections about symmetry planes containing the rotational symmetry axis. Hence we can find the new vertices of the fundamental polygon by reflection of the canonical root vertex about these symmetry planes, see Figure 1.36, bottom-left.

Six-squares surface.

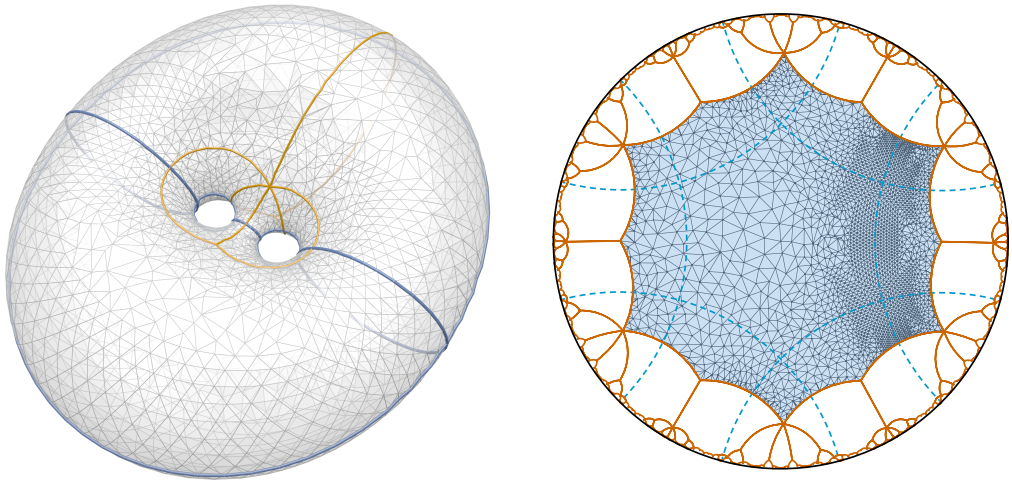
Figure 1.37 (left) shows a surface glued from six squares, which is conformally equivalent to Lawson's surface and the hyperelliptic curve. Edges with the same marking are glued together.



data/lawson_embedding/embedding5k_canonical.xml

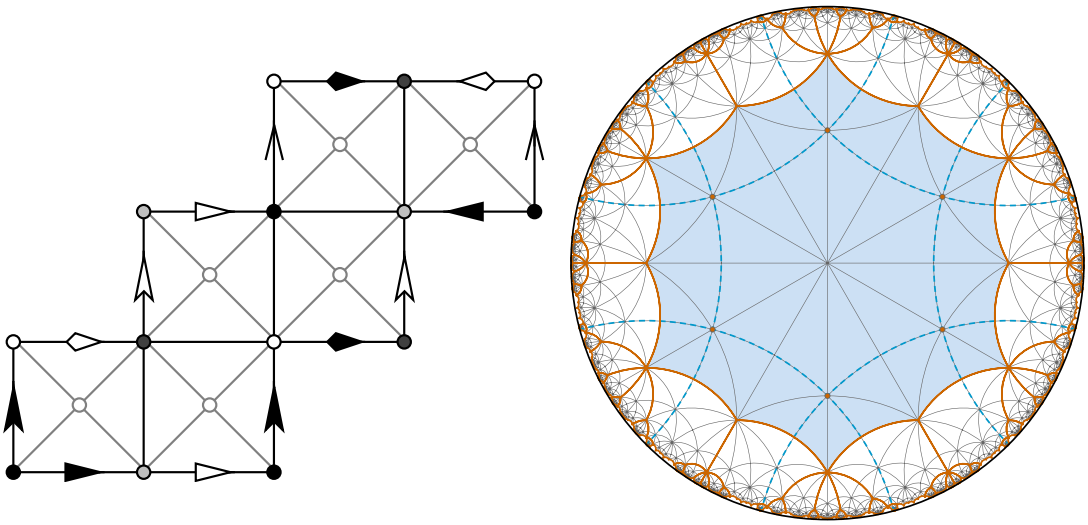


data/lawson_embedding/embedding5k_opposite.xml



data/lawson_embedding/embedding5k_cyclic.xml

Figure 1.36: Uniformization of Lawson's surface. Left: Triangulated model [52], with the boundary of the fundamental domain shown in brown and the axes of the generators shown in blue. Right: Fuchsian uniformizations and fundamental domains. Canonical domain (top), opposite sides domain (middle), and 12-gon (bottom).



data/lawson_squares/lawson_squares_branch.xml

Figure 1.37: Left: A surface glued from six squares. Right: Fuchsian uniformization and fundamental domain.

We calculate a uniformization using the triangulation with vertices added in the centers of the squares as shown. An adapted fundamental domain for this square-tiled translational surface arranges all squares around a single vertex, see Figure 1.37 (right). By comparison with Figure 1.35 (bottom) we see that the vertices in the center of the squares correspond to the branch points of the hyperelliptic representation of the surface. The black, gray, and white vertices correspond to the north and south pole of the hyperelliptic representation.

Part II

Applications

Chapter 2

Surface panelization using periodic conformal maps

This publication is joint work with Thilo Rörig, Agata Kycia, and Moritz Fleischmann. It was previously published in the proceedings of the conference "Advances in Architectural Geometry 2014" [57].

We present a new method to obtain periodic conformal parameterizations of surfaces with cylinder topology and describe applications to architectural design and rationalization of surfaces. The method is based on discrete conformal maps from the surface mesh to a cylinder or cone of revolution. It comes with a number of degrees of freedom on the boundary that can be used to obtain a variety of interesting panelizations. We illustrate different choices of parameters for NURBS surface designs. Further, we describe how our parameterization can be used to get a periodic boundary-aligned hex-mesh on a doubly-curved surface. We optimize this initial mesh to consist of a limited number of planar regular hexagons that panel the surface.

2.1 Introduction

The panelization of surfaces remains a challenge in architectural design. CAD software such as Rhino [56], delivers powerful NURBS surface modelling to the designer. Their ease of use have made them a de facto standard for the design of freeform (and other) shapes in architectural design, especially envelopes and facades of buildings.

The scale of buildings introduces challenges to surface-based modelling strategies: The large scale of building elements demands that they are divided into smaller elements. The cost of material and labor, standardized production lines, green building concepts, availability and redundancy during construction periods demands for a high degree of similarity of these elements. Yet, the inherent UV subdivision of NURBS surfaces offers limited control over the layout, shape and configuration of the panels. While strategies for the controlled and careful creation of freeform surfaces have been presented and realized [27], the tiling of true freeform surfaces through alternative algorithms is still a challenge.

The quality of a surface panelization solution can be defined in various ways. From an aesthetic point of view the shape of individual elements is important. Further there are global conditions

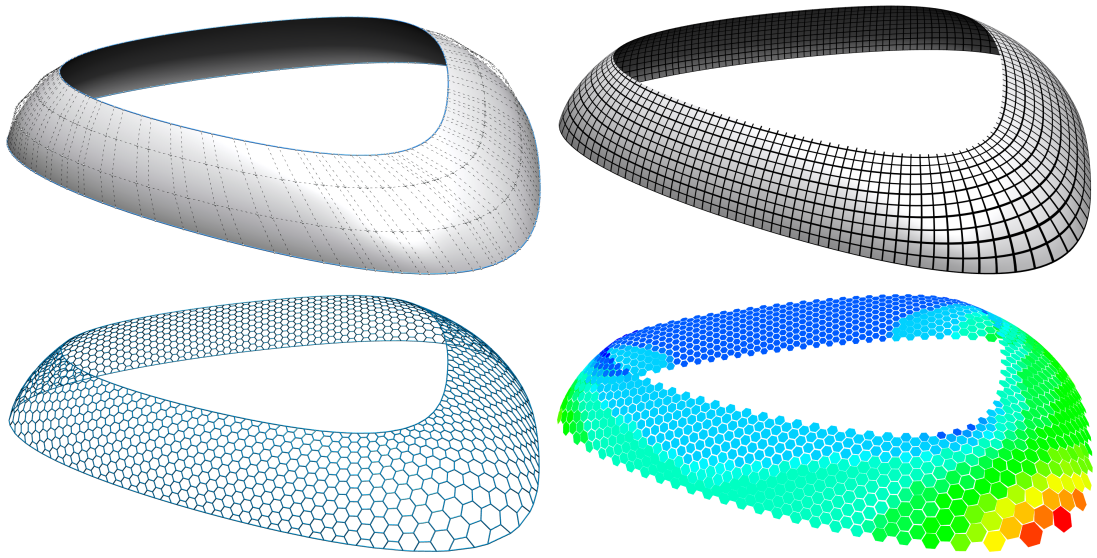


Figure 2.1: For a cylindrical NURBS surface (top-left) we create a seamless periodic conformal parameterization (top-right). A new mesh (bottom-left) is then rationalized and the panels are optimized for quantized regular hexagons (bottom-right).

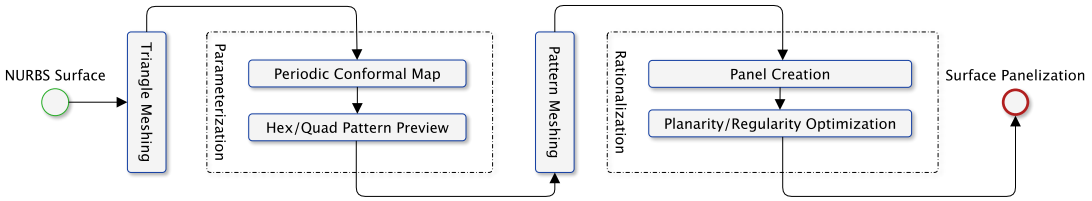


Figure 2.2: Flow diagram of the algorithms described in this paper. From a NURBS surface a triangle mesh is created. The parameterization part is described in Section 2.3. A pattern-mesh is created using this parameterization. The creation and optimization of panels is described in Section 2.5.

such as alignment with surface boundaries and smooth transition of element shape. From the standpoint of fabrication, the elements should be repetitive. The contributions of this chapter are:

- **Periodic discrete conformal parameterization**: We present an algorithm that maps a triangle mesh with cylinder topology conformally to a cylinder or a cone of revolution. This allows us to obtain seamless patterns on surfaces. It is a generalized version of the discrete conformal parameterization scheme described by Springborn, Schröder, and Pinkall [69].
- **Regular elements**: We show how the periodic discrete parameterization can be used to construct a panelization of the given periodic surface into a small number of repeating regular elements.
- **Applications to architectural design**: We present a case study that initiated the development and where the described methods have successfully been applied in the architectural design context.

The chapter is organized as follows: Section 2.2 describes various parameterization schemes

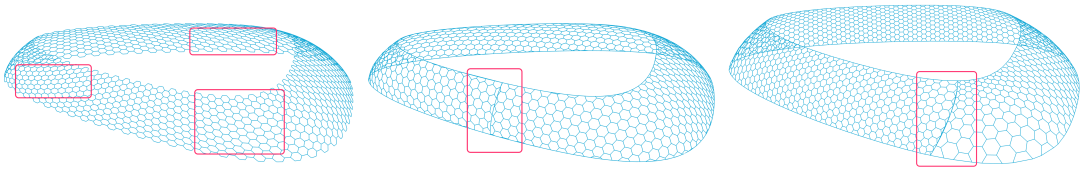


Figure 2.3: State of the art unroll methods can create patterns on a closed surface. The `ApplyCrv` command of Rhinoceros produces boundary alinged periodic patterns but introduces unacceptable non-isotropic stretch (left). The `SquishBack` method creates sufficiently regular elements but does not respect the periodicity of the surface (middle). Non-periodic conformal maps align with the boundary of a cut surface. Along the cut the map is not continuous (right).

available to architectural geometers. We briefly discuss the shortcomings of current methods and explain how we adress them with our method. Section 2.3 introduces the concept of periodic discrete conformal parameterizations. See Figure 2.2 for a schematic view of the proposed method. In Section 2.4 we present a case study that led to the development of this work. Section 2.5 deals with the optimization of panels to meet requirements arising during the case study. We give implementational details in Section 2.6 and close with an outlook on further research.

2.2 Related work

In this section we will review different methods used to unroll/parameterize surfaces that are accessible in the architectural design process on the example of Rhinoceros3D and 3rd party plugins. All available tools lack at least one of the key features of the proposed method:

- *Conformality* of the mapping avoids non-uniformly stretched panels
- *Periodicity* of the parameterization is needed to layout panels seamlessly on the surface
- *Boundary alignment* avoids irregular cutting of panels at the boundary of the surface

Rhinoceros CreateUVCrv/ApplyCrv.

A NURBS surface is naturally equipped with a parameterization, i.e., a UV mapping from a rectangle domain to the surface. For the surface of Figure 2.1 such a map can be used to project a pattern from this rectangle to the surface (`ApplyCrv` in Rhinoceros). The pattern can be constructed periodically as it is defined in the UV domain of the surface. However, in general this method does not produce satisfactory results in terms of quality of elements for complex freeform surfaces. The UV parameterization is not conformal and thus introduces non-isotropic stretch and shear preventing the elements to be regular on the surface, see Figure 2.3, left. This limitation exists even for developable surfaces.

Rhinoceros Squish/SquishBack.

The `Squish/SquishBack` command of Rhino maps a surface to the plane minimizing the amount of stretch. While this is geometrically not a conformal map it produces acceptable patterns on the surface. It is however not capable of calculating periodic maps to the surface. Thus it is not applicable in our situation, see Figure 2.3, middle.

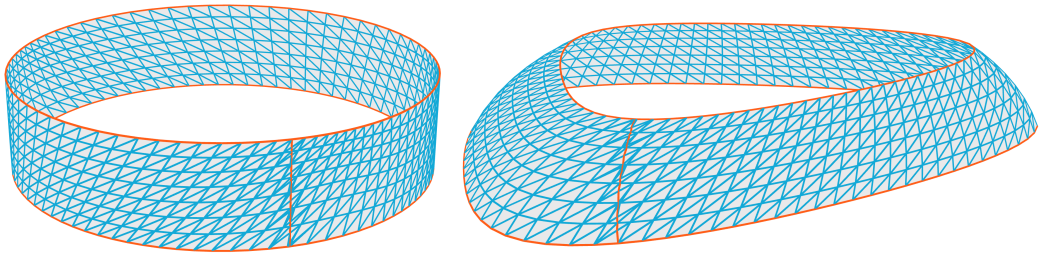


Figure 2.4: A discrete periodic map from a cylinder to a triangulated surface. On the cylinder all edges of the triangulation are geodesic arcs. If the cylinder is cut at the vertical orange path, then it can be unrolled to the plane creating a rectangular domain.

PanelingTools for Rhino.

The paneling tools of Rhinoceros use the UV parameterization of the underlying surface to populate grid-points over the surface. With the help of such a grid, panels are placed onto the surface [46]. The shapes of the panels depend heavily on the NURBS-parameterization. An example is shown in Figure 2.3, left.

Hexagonal tilings.

In the architectural context, hexagonal panelizations have been studied, e.g., by Zimmer *et al.* [76] and Troche [71]. These aproaches however do not include regularity or special boundary alignment as introduced in the current work. In the work of Schiftner *et al.* [58] the result of the panelization depends on the choice of an initial triangle mesh that is optimized towards touching incircles. This allows for a torsion free support structure of a non planar hex-mesh. Hexagonal tilings for triangulated surfaces also have been studied by Nieser *et al.* [50]. They focus on regularity but not on boundary conditions.

Mesh parameterizations.

There are a vast number of parameterization schemes for meshes. To elaborate on all methods is beyond the scope of this section and we describe only the most relevant results here. General purpose parameterization methods for triangle meshes produce high quality quad or hex meshes for unstructured input data [17, 3, 69]. They have been used with success in the architectural context, e.g., by Bo *et al.* [9] and Sechelmann *et al.* [64].

The basis of our method are conformally equivalent triangle meshes as described by Springborn *et al.* [69]. The straight forward method to map a surface with this approach is to cut it open and map it to a rectangle domain. This method yields boundary aligned conformal maps that however do not match along the introduced cut, see Figure 2.3, right. How to generalize this method to overcome this limitation is the content of the following section.

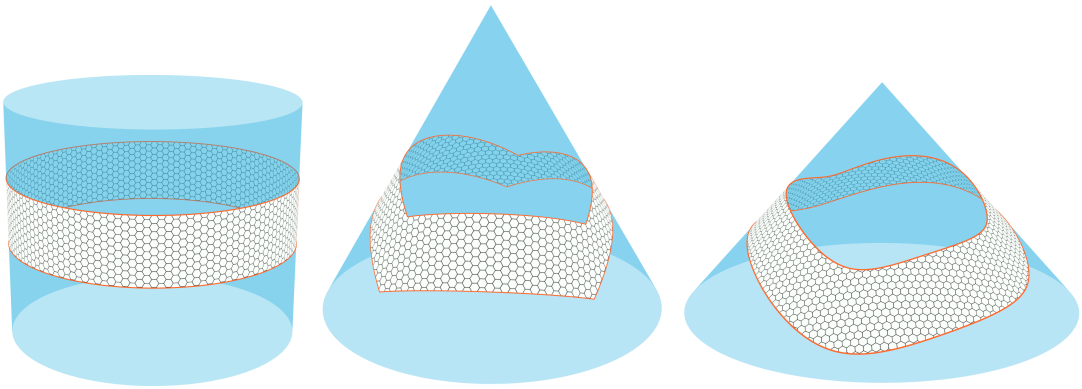


Figure 2.5: Periodic domains of parameterization of the surfaces shown in Figure 2.6. Left: Map to a cylinder with geodesic boundary curves. Middle: Map to a cone of revolution with hex-pattern-adapted angle. The domain is a polygon with quantized angles. Right: Isometric boundary on a cone with hex-pattern-adapted angle. Panelizations created with the help of these maps are shown in Figure 2.6.

2.3 Periodic conformal parametrization

In this section we describe our algorithm for the creation of periodic conformal maps for cylindrical meshes/surfaces, i.e., surfaces with the same topology as a cylinder. First we will review the discrete conformal maps of Springborn *et al.* [69]. Then we show how it can be generalized to yield periodic maps to cylinders or cones.

A smooth *conformal map* between two surfaces is a map that preserves angles. Intuitively, one can think of a conformal map as a map that preserves the shape but not the scale of small figures. For conformal surface parameterization, one looks for conformal maps from the plane to a surface and vice-versa. These can be used to map different patterns onto surfaces in a way that only isotropic stretch/uniform scaling is applied to the pattern elements. The method we build onto is a triangle mesh based discretization of conformal maps [69]. For each vertex v of the surface – interior or boundary – we may prescribe an angle θ_v that corresponds to the angle sum of adjacent triangles in the target mesh. Starting from an input mesh and target angles θ_v, the method calculates new edge lengths for the triangles of the target mesh such that the angle sums at the target vertices are as prescribed. This goal is achieved by minimizing a convex functional. The prescribed angles have to satisfy a Gauss-Bonnet type condition, i.e., the angles at interior vertices have to match the angles at the boundary vertices. We will state the condition for the special cases treated later in the article, see Equation (2.1).

For the parameterization problem, we want to construct a map from a surface to the plane. To get a planar target mesh, the target angles have to be set to 2π for all interior vertices, i.e., the angles of the triangles adjacent to every interior vertex sum up to 2π. Thus the computed target triangles can be laid out in the plane. At the boundaries there is still a certain degree of freedom, which allows to map the surface to different shapes, e.g., a rectangle or a more general polygon with prescribed angles. An alternative choice of boundary conditions yields a target mesh whose boundary edges have the same lengths as the original mesh. Then the control over the boundary angles is no longer possible.

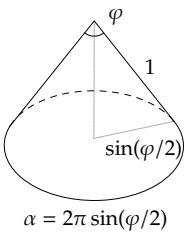
This method for the parameterization of triangle meshes can be generalized to triangle meshes with cylinder topology, see Figure 2.4. Instead of constructing a discrete conformal map from

the surface to the plane, we construct a map to a cylinder or cone, whose image is isometric to a polygonal region in the plane, see Figure 2.5. This works with an approach very similar to the previous one. We start with the definition of a periodic parameterization.

Definition 2.3.1. *Let $M = (V, E, F)$ be a mesh with cylinder topology and vertices V, edges E, and triangles F. Let $D \subset C$ be a region on a cone/cylinder of revolution. A continuous bijection $\Phi : D \to M$ is called a* discrete periodic parameterization. *D is called the* domain of parameterization.

In the latter we always assume that the preimages of edges of M are geodesic arcs on the cone/cylinder C. For panelization of periodic surfaces we need to make sure, that different patterns match around the cone or cylinder. This yields certain restrictions on the cone that serves as domain of parameterization.

Definition 2.3.2. *Let C be a cone with aperture φ and $\Phi : C \supset D \to M$ be a discrete periodic parameterization of a triangle mesh M with domain D. The map Φ is called* triangle adapted *if the cone angle α is a multiple of $\frac{\pi}{3}$ and* quad-pattern adapted *if it is a multiple of $\frac{\pi}{2}$.*



This definition ensures that either a quad-, triangle-, or hex-pattern fits seamlessly onto the surface after parameterization.

2.3.1 Periodic boundary conditions

If we want to construct periodic conformal maps we are allowed to specify angle sums θ_v at boundary vertices. The condition for the sums of boundary angles differs from the simply-connected case in the following way: The curvature at a boundary vertex v is given by $\kappa_v = \pi - \theta_v$, where θ_v is the angle sum of the adjacent triangles in the target mesh. For the two boundary loops $(v_1, \ldots, v_n)$ and $(w_1, \ldots, w_m)$ we have:

$$\sum_{i=1}^{n} \kappa_{v_i} + \sum_{j=1}^{m} \kappa_{w_j} = 0. \tag{2.1}$$

This condition makes sure that the two boundary curves "bend" the same amount and can hence be wrapped around a cone. We will now show how boundary conditions can be used to construct periodic patterns on the studied models. We start with a discrete conformal map of the doubly-curved model from Figure 2.1 to a standard cylinder.

Straight cylinder.

The simplest way to generate a map to the cylinder is to set the target angles for all boundary vertices to π. Hence the curvatures at the boundary vertices are zero and the two boundary loops are mapped to "straight" curves. In this case both angle sums of Equation (2.1) vanish and the target mesh can be wrapped around a cylinder, see Figure 2.5, left. The new edge lengths computed with the variational principle correspond to the lengths on a cylinder. This cylinder

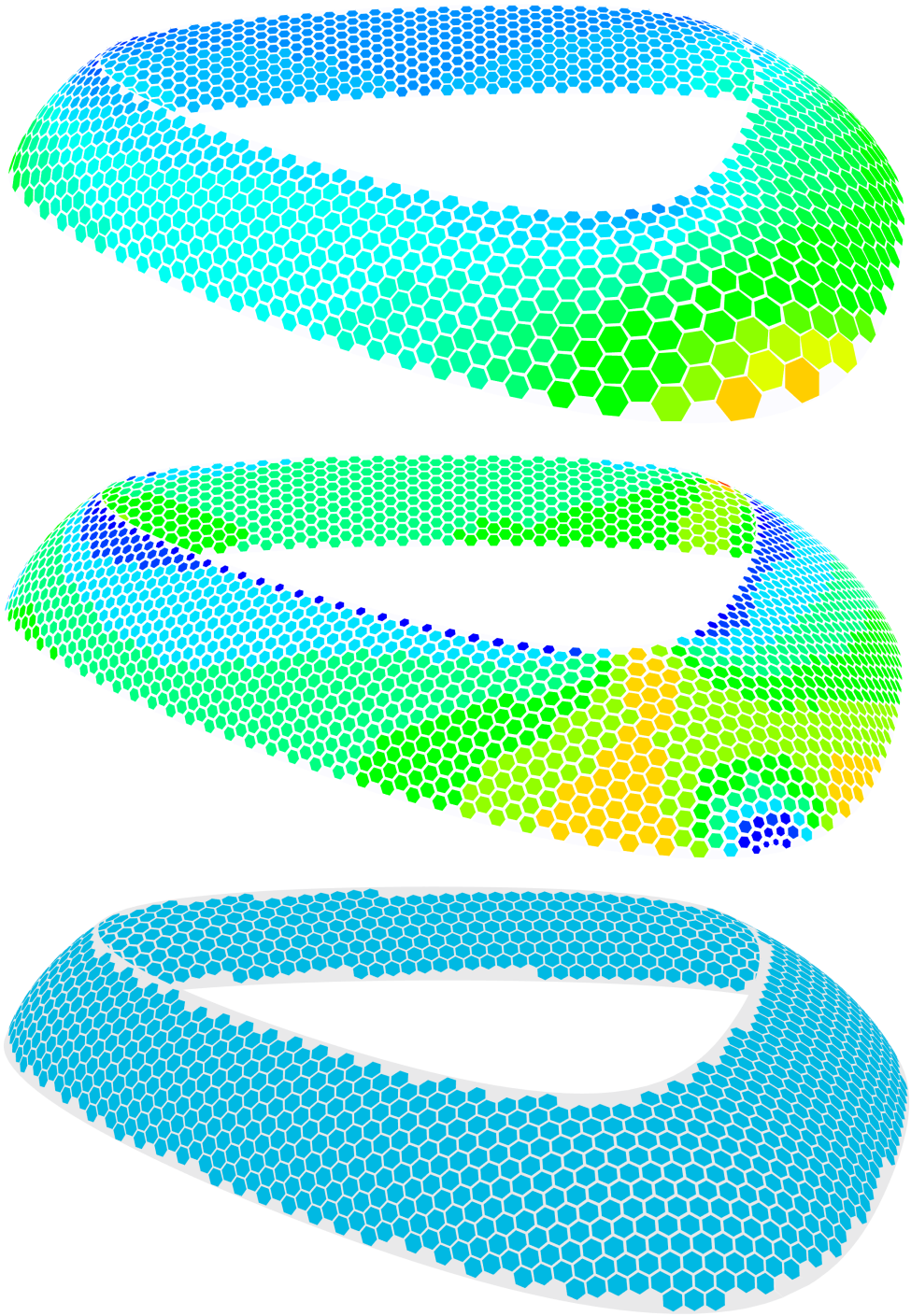


Figure 2.6: Quantized periodic hexagonal panelizations. Boundary conditions affect the amount of stretch in the interior of the surface. Top: Hexagonal pattern aligns with the boundary, a strong condition that produces large deviation of panel sizes. Middle: Map to a pattern-adapted polygon on a cone of revolution. The pattern contains exceptional points at the boundary. The stretch is minimized while at the same time the pattern alignes with the boundary. Bottom: Conformal map with the least stretch in the interior, pattern can be optimized to consist of congruent hexagons alone. In each image, panels with the same colors are congruent. The corresponding domains of parameterization are shown in Figure 2.5.

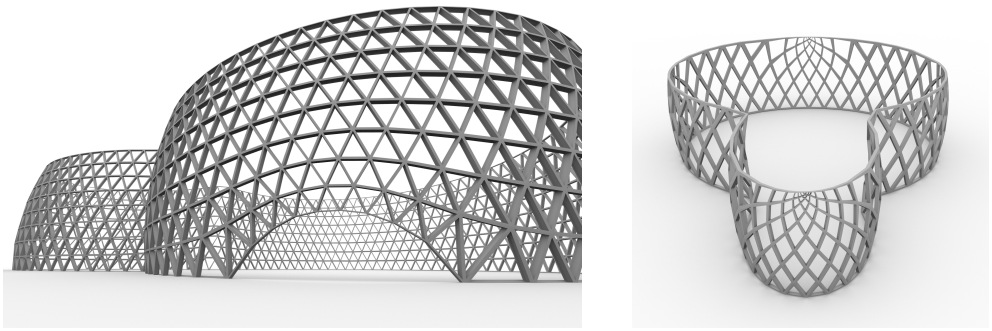


Figure 2.7: A periodic conformal map onto a cylinder with special vertices creates the opportunity to incorporate entrances (left) or concentration of support structure (right).

can be unrolled into the plane preserving angles and lengths. Therefore the two boundary polygons are mapped to straight lines in the plane. These two straight lines have to be parallel and of equal lengths. If the lengths of the boundary curves in the original model differ a lot, then a map to a cylinder induces a lot of conformal stretch. This stretch can be reduced by specifying special boundary conditions for a parameterization on a cone of revolution.

Cone of revolution.

As long as Equation (2.1) is satisfied we obtain a map to a general cone of revolution. In our case, we require that the periodic parameterization is adapted to the target pattern. This means that the two sums of Equation (2.1) need to be (the same) multiples of $\frac{\pi}{3}$ (triangle or hex) or multiples of $\frac{\pi}{2}$ (quad). We present two methods to achieve this requirement: a uniform distribution and a concentration of curvature.

If the boundary of the mesh should align with the pattern, then boundary angles need to be quantized, i.e., multiples of $\frac{\pi}{3}$ or $\frac{\pi}{2}$ need to be chosen as target angles. In Figure 2.5, middle, three vertices of the top and bottom boundary curve were manually assigned to $\frac{4}{3}\pi$ and $\frac{2}{3}\pi$, respectively. All other boundary angles are set to π, i.e., straight. Such a map can be used as a starting point to obtain a tesselation with quantized hexagons as described in Section 2.5.

It is known that a discrete confomal map that does not change the lengths of the boundary edges exhibits the least stretch in the interior of the surface. To obtain such a parameterization we first construct a periodic conformal mapping onto an arbitrary cone such that the lengths of the boundary edges are equal. The resulting angle sums at boundary vertices of the target mesh determine the cone angle φ of the map. The cone angle of a pattern adapted periodic parameterization is the closest multiple of the desired quantization. We distribute the difference to the closest quantized angle uniformly to the individual boundary vertices and recompute the map with these angle conditions. The obtained map is periodic and exhibits the lowest stretch of all periodic conformal maps (see Figure 2.5, right).

Design and structural opportunities.

Opposed to achieving a homogenous pattern distribution, as described previously, it is also possible to use special boundary conditions to support structural purposes or design requirements. If one aims for a panelization with boundary aligned patterns, then the target boundary angles

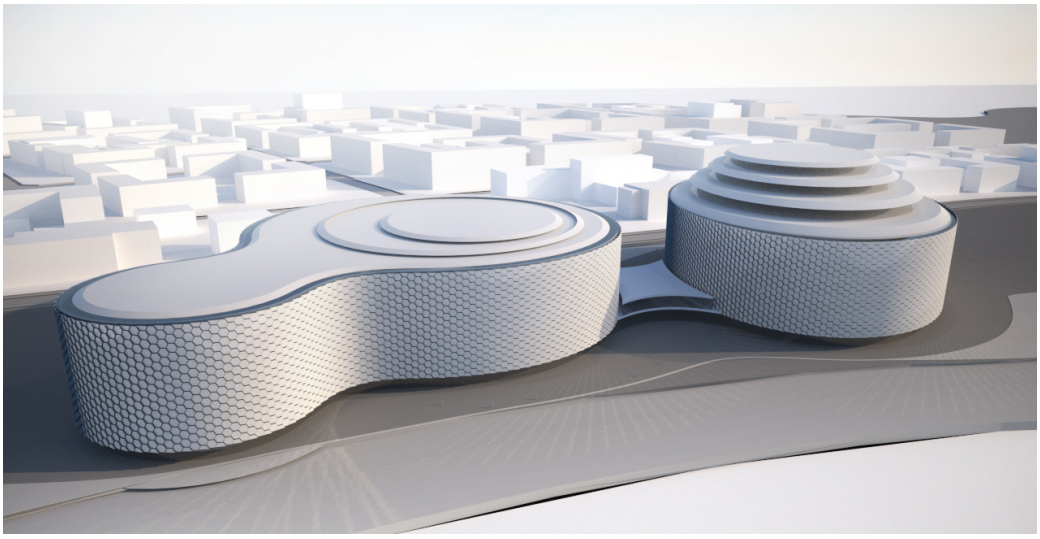


Figure 2.8: Rendering of Case Study - A project where the method of conformal mapping was utilized for the facade design.

must be quantized.

To include entrances in a facade it is possible to incorporate special boundary conditions. An example with special boundary vertices with domain angles $\frac{4}{3}\pi$ and $\frac{\pi}{3}$ is shown in Figure 2.7, left. In the remeshed surface, the lower boundary curve bends inside at the vertices with angle $\frac{4}{3}\pi$ around the vertex with angle $\frac{\pi}{3}$. This incorporates a natural entrance into the facade.

Another effect of such angle conditions is a densification of the pattern at the vertices with small angles. Such a concentration of elements can be used to enforce structural properties of a geometry. An example of a diagrid generated with such boundary conditions is shown in Figure 2.7, right.

2.4 Case study: Hexagonal surface panelization

Here we present the findings of a first case study, where the method of discrete conformal mappings was utilized for a real-world project. In this case study, a facade design, important questions concerning panel layout, similarity and therewith constructability had to be addressed at the early design stage.

Discrete conformal maps were used, as they allow the designer to explore alternative surface textures and surface panelizations with great design flexibility. This distinguishes the method from more constrained modelling techniques [27]. Through the method of conformal mapping, opposed to naive UV mapping, the density of the surface panelization varies across the entire surface, yet the shape of elements does not. This can be used for structural purposes, such as diagrid layouts, or design driven, such as window distribution, see Figure 2.7. The optimization of the surface panelization towards multiple criteria such as edge length and planarity was consequential.

For a commissioned competition entry we tested and developed the method of periodic conformal mappings. The project, which served as a case study, was highly constrained, as the

Figure 2.9: The data for the component-like construction of each panel was derived from the mesh.

architects were asked to propose an alternative facade design for an existing design proposal of a multifunctional exhibition center in China, see Figure 2.8. The massing was fixed, but there were 2 alternative massing options (1 single curved and 1 doubly-curved envelope) to be explored. Also, the client wanted a hexagonal tiling on the facade but only had a very limited budget of approximately 200 Euro / sqm for the entire facade including sub structure in mind.

These limitations, in combination with the very short timeframe of 2 weeks for the entire redevelopment of the facade including a feasibility study drove the development of the conformal mapping method. Especially, since existing solutions such as UV mapping led to unsatisfactory results producing anisotropic stretch and shear of some regions in the master surfaces. Some specific questions that had to be addressed for each massing option were:

- How many (different) panels would we need?
- Can we clad the entire surface with planar tiles?
- Can we equalize the edge lengths of each hexagon?
- Can we control the orientation of the panels?
- Can we achieve a regular pattern with a homogeneous visual appearance?

In the end, all the above questions were answered/solved.

The first step of development focused on achieving periodicity across the surface and alignment with the boundary. While the issue of periodicity directly addressed the last question, it is strongly related to the others as they could be achieved by successive optimization steps.

Already during the design phase a fully periodic tiling was achieved. In a following step the panels were planarized, grouped by dimension and their edge lengths were equalized. Finally, a control for the panel orientation based on the tangents of the NURBS master surface was implemented. This hexagonal pattern served as a base for the facade engineering team. Due to the high cost demands, a simple component system that served as a sub-structure for each panel was developed, see Figure 2.9.

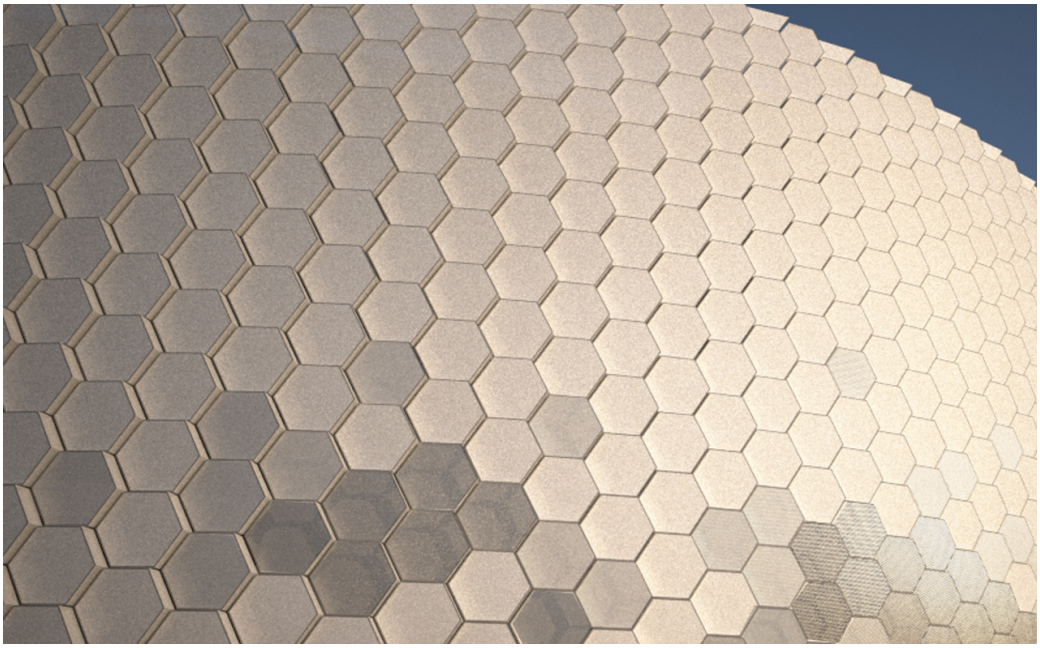


Figure 2.10: Close-up rendering of facade.

Unfortunately the given massing options for the building were not very challenging in terms of geometry. One massing option was a simple extrusion and the other had very little distortion. After the successful submission of the project, we decided to continue the development and test the method of discrete conformal mappings on more extreme base geometries.

During these tests a Grasshopper plug-in for VaryLab has been developed and refined [63]. We focused on tiling surfaces with a large distortion/stretch and double curvature. The main aim was to tile these surfaces without distorsion. This led to focus on the boundary conditions. The designer is now able to choose between an aligned mapping where the tile-pattern aligns with the underlying surface boundary. The trade-off being, that panels need to vary in sizes. Or one chooses a "homogeneous tiling", where all the tiles are the same, but do not align with the boundary. VaryLab's numerous optimization algorithms can be applied and combined with either of the two approaches, see Figure 2.6. During the development, we realized that through singularities and special boundary conditions, one is able to control the density and distribution of the pattern on the surface and along its boundaries, see Figure 2.7.

2.5 Rationalization: Hexagon optimization

Starting from the conformal parameterization we optimize the obtained hex-mesh to have identical regular hexagons. We use a global optimization approach and define energies to achieve *planarity*, *regularity*, and *equality*.

Planarity.

The planarity function is a simple adaptation of the usual energy used to planarize quad-meshes: A quadrilateral $\{A, B, C, D\}$ is planar if the volume of the tetrahedron $\{A, B, C, D\}$ is zero. So if we require the volume of all tetrahedra spanned by the vertices of a polygon to be zero we obtain a planar polygon. The planarity energy E_{pl} can easily be expressed in terms of determinants.

Regularity.

A regular planar polygon is characterized by having equal edge lengths and equal angles at all vertices. As for planarity we define an energy that is minimized in case of regular polygons. The interior angle at a vertex of a regular p-gon is $\frac{p-2}{p}\pi$. So an energy E_{reg} that is minimized for a regular p-gon with vertices $\{v_1, \ldots, v_p\}$ and corresponding angles $\{\alpha_1, \ldots, \alpha_p\}$ is

$$E_{reg}(P) = \lambda_P \, E_\alpha(P) + \mu_P \, E_\ell(P) \qquad \text{with}$$

$$E_\alpha(P) = \sum_{i=1}^{p} (\alpha_i - \frac{p-2}{p}\pi)^2 \qquad E_\ell(P) = \sum_{(v_i, v_{i+1})} (\|v_i - v_{i+1}\| - \ell_P)^2,$$

where ℓ_P is the desired target edge length for the polygon and λ_P and μ_P weights for the different energies. In a first step, the target length can be chosen to be the average edge length of the polygon or the shortest edge length among the edges to avoid overlap. Note that, the normalization of the angles already implies planarity of the polygons. Nevertheless, we consider the planarity energy since it increases the rate of convergence.

Starting from a cylindrical or conical periodic conformal parameterization we construct a hex-mesh that may or may not be aligned with the boundary. As a consequence of the conformality of the parameterization the angles of the hexagons are almost $\frac{2\pi}{3}$. In the following we do not work with a water tight mesh any more but split the surface into individual hexagonal panels. We optimize the edge lengths of the hexagons to be constant per face using E_ℓ. To avoid overlap we choose the length of the shortest edge of each face as target length. We add the planarity and angle regularity functionals E_{pl} and E_α to the optimization and obtain planar and regular hexagons. Each of the hexagons has its own constant edge length. Finally, we can rationalize the panelization further, by choosing a discrete set of edge lengths as target lengths for the polygons in the edge length functional E_ℓ. Due to the symmetry of the edge length functional for regular hexagons edge length optimization will not destroy the planarity and regularity of the hexagons. So it is possible to adjust the edge lengths using E_ℓ only. This quantization process is illustrated in Figure 2.11. The used method allows to include additional functionals into the optimization, e.g., functionals minimizing the distance to a reference surface or the distribution of gaps tangent and normal to the surface. We have not yet added these functionals due to the early phase of the presented project.

The range of lengths obtained depends on the initial hex-mesh constructed on the chosen target geometry. The effect of the different periodic conformal parameterizations on the quantization is shown in Figures 2.5 and 2.6.

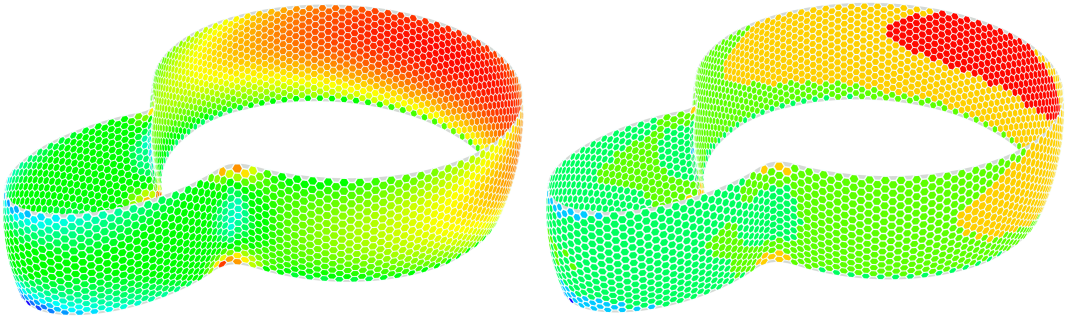


Figure 2.11: Panelization of a doubly curved design alternative of the case study shown in Figure 2.8. Left: Unquantized panelization. Right: Quantization to 11 panel sizes with edge lengths varying from 1.5m to 2m.

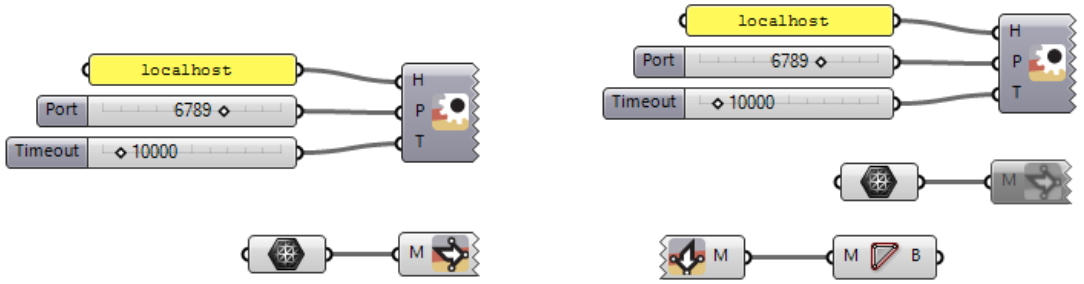


Figure 2.12: Grasshopper networks connecting Rhino and VaryLab. In the first step we send mesh data to VaryLab running on the same machine at `localhost:6789` (left). In the second step we collect the result from this VaryLab instance and create polygons from the mesh's faces (right).

2.6 Implementation

We use the software package VaryLab [63] in combination with Rhino's Grasshopper to calculate discrete conformal maps to the cylinder or cone as shown in Figure 2.5. Figure 2.12 shows the Grasshopper network used to connect Rhino and VaryLab. VaryLab uses the optimization package TAO/PETSc [8, 7, 68] to perform energy minimization. We calculated the exact gradients of the functionals for the optimization and used the conjugate gradient method of TAO.

2.7 Conclusion

The collaboration between architect and math department proved to be very satisfactory for both parties: The architects did provide specific questions related to real world projects whilst the mathematicians were able to translate these questions into mathematical formulas and provided meaningful results that could not have been achieved alternatively. A common design framework such as Rhino and the basic knowledge of NURBS geometry and modeling techniques proved to be of essential importance for the successful collaboration between the teams.

As a result of this collaboration, we presented a method for homogeneous periodic panelization

of NURBS surface geometry of cylinder type. A natural future development is the design of suitable support structures possibly with torsion free nodes. This can be derived from the panel layout by intersecting the panel planes. Furthermore the quantization of edge lengths of regular hexagonal panels is not yet fully explored. A denser distribution of quantized edge lengths in regions with great edge length variance can possibly improve the layout and number of different panels. For a later stage of the project one could add optimization for the gaps between the panels, which has not been incorporated yet. It would also be interesting to look at more extreme examples, even if those might not fit the architectural context.

We also started to look into how patterns can be applied across multiple surface patches, such that the pattern aligns at the crease where the patches meet. This would necessitate to develop methods for more general mappings and the applications of multiple boundary conditions; A field that is yet to be developed.

While the method presented does require a certain level of expertise from the user, we see the potential of this post-panelization strategy of freeform surfaces in the architectural design process: Opposed to carefully constrained (parametric) modelling approaches this method allows to realize even distributed, homogenous panelizations on arbitrary surfaces with cylinder topology at early design stages - something a lot of designers are interested in.

Chapter 3

Quasiisothermic mesh layout

This chapter is joint work with Thilo Rörig and Alexander Bobenko. Most of the material presented here was published previously in the proceedings of the conference "Advances in Architectural Geometry 2012" [64]. We added Section 3.5 on isometric deformations of s-isothermic surfaces and Section 3.6 about a discrete ellipsoid.

In architecture the quality of a quad-mesh depends on the shape of the individual quadrilaterals. The ideal shape from an architectural point of view is the planar square or rectangles with fixed aspect ratio. A parameterization that divides a surface into such shapes is called isothermic, i.e., angle-preserving and curvature-aligned. Such a parameterization exists only for the special class of isothermic surfaces. We extend this notion and introduce quasiisothermic parameterizations for arbitrary triangulated surfaces.

We describe an algorithm that creates quasiisothermic meshes. Interestingly many surfaces appearing in architecture are close to isothermic surfaces, namely those coming from form finding methods and physical simulation. For those surfaces our method works particularly well and gives a high quality and robust mesh layout. We show how to optimize such meshes further to obtain disk packing representations. The quadrilaterals of these meshes are planar and possess touching incircles.

3.1 Introduction

A key problem in architectural geometry is to convert surfaces created by form finding methods, physical simulation, or manual modeling to quadrilateral meshes, which are preferred for glass-steel structures. There are many possible quad-meshes that approximate a given shape and we study those that consist of principle-curvature-aligned conformal squares, see Figure 3.2. Not all surface shapes can be approximated by such meshes. A smooth analog of a surface with this property is called an isothermic surface. These surfaces admit conformal curvature line parameterizations, i.e., angle-preserving parameterizations aligned with the principle curvature directions. Their discrete counterpart are so-called s-isothermic meshes. These meshes have the additional property that neighboring quadrilaterals possess touching incircles, see Figure 3.7.

The class of isothermic surfaces comprises, for example, constant mean curvature surfaces. Roofs that act shell-like turn out to have almost constant mean curvature. These are the kinds

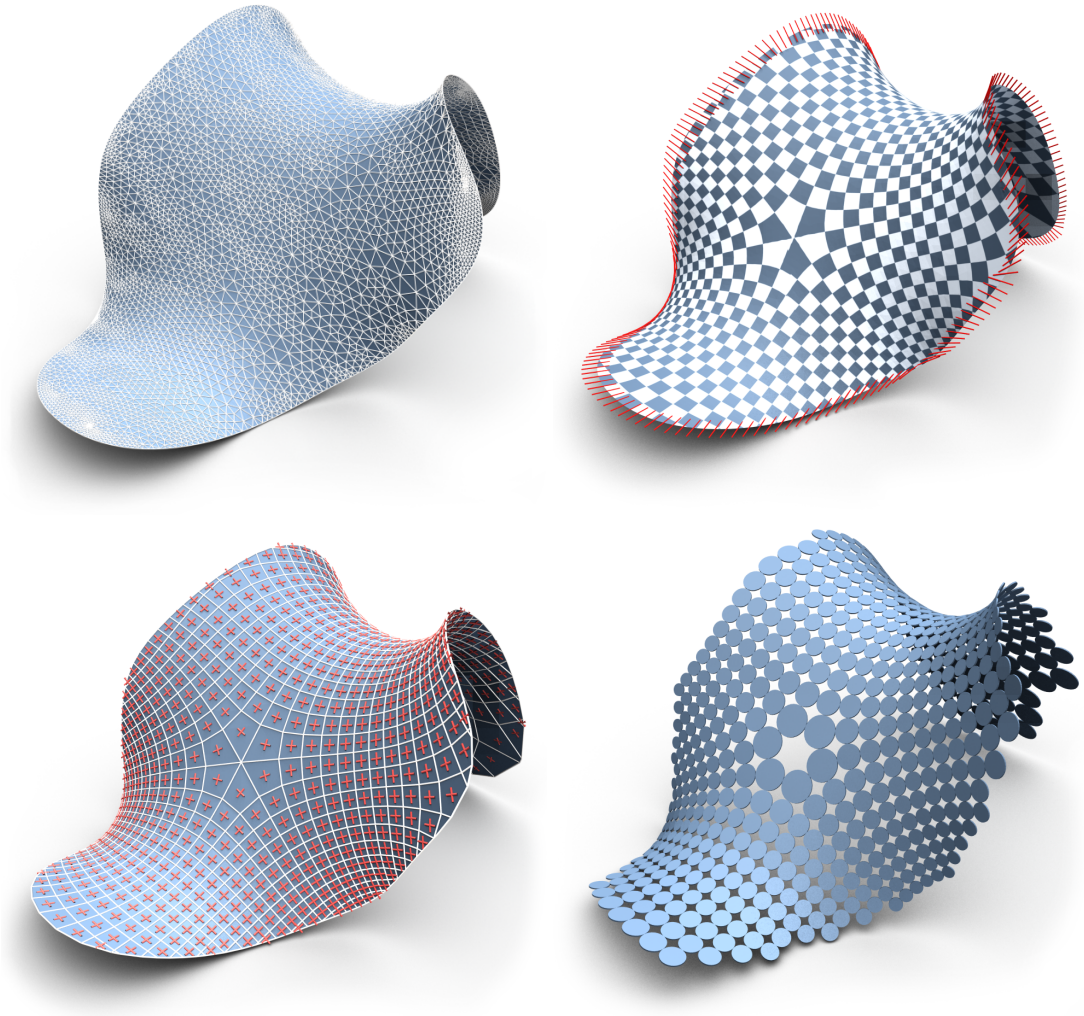


Figure 3.1: The algorithmic steps of this paper: For a triangulated surface we calculate texture coordinates by solving a boundary value problem for principle curvature directions on boundary edges (checker board texture and red directions). The edges of the corresponding quad mesh align with the curvature directions (red crosses). The mesh is then optimized towards planar quads with touching incircles.

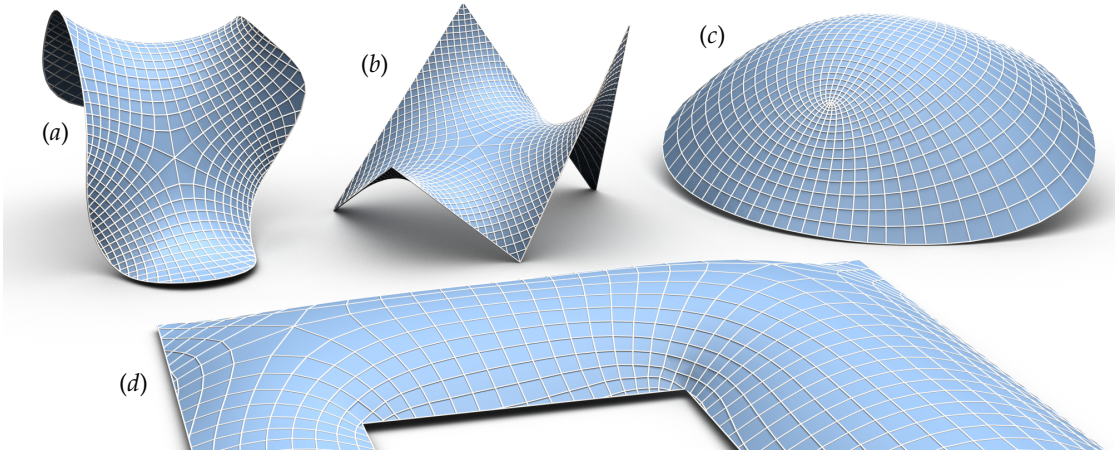


Figure 3.2: The surface examples of this paper. All have been parameterized and remeshed. (*a*) The Teaser surface is the minimizer of a spring energy with a smooth fixed boundary curve. (*b*) A Minimal surface with polygonal boundary curve. (*c*) Dome: Part of a NURBS surface exhibiting positive curvature and two curvature field singularities. (*d*) Roof structure with planar boundary curve and regions of positive and negative curvature.

of surfaces that initiated our study of conformal curvature line parameterizations in the architectural context. Both conformality and alignment with curvature lines are favorable properties for meshes.

The contributions of this paper are:
Definition of quasiisothermic parameterizations: We propose a definition of quasiisothermic parameterizations of triangle meshes. It is based on angles between curvature directions and edges of a triangle mesh. We define the quasiisothermic modulus that measures how isothermic a parameterization is. If this modulus is zero we obtain discrete isothermic parameterizations in the sense of our definition.

Parameterization Algorithm: We give an algorithm that creates quasiisothermic parameterizations based on discrete conformal maps of triangle meshes to the plane. This approach is built on top of the conformal mapping technique of Springborn *et al.* [69]. We inherit the speed and superior projective mapping properties of their parameterizations.

Variational principle for circle packing quad-meshes: The obtained parameterizations are used for remeshing and we optimize quadrilaterals to have touching incircles by minimizing a novel energy. These s-isothermic meshes have been studied in discrete differential geometry and possess some remarkable properties, e.g., minimal s-isothermic surfaces may be deformed isometrically retaining the same Gauß map.

The rest of the paper is organized as follows: Section 3.2 gives an overview of existing parameterization schemes and their relation to our approach. We also give reference to the related mathematical literature in discrete differential geometry. In Section 3.3 we define quasiisothermic parameterizations and a corresponding quality measure. In Section 3.4 we describe an algorithm to obtain quasiisothermic parameterizations with small modulus. We describe the connection to discrete conformal maps and discuss how we deal with singularities. A variational principle to generate s-isothermic meshes is presented in Section 3.5. At the end of the section we show the effect of our optimization on several examples from different classes of surfaces. In the final Section 3.7 we sum up the results and propose extensions and enhancements subject

to further research.

3.2 Related work

There has been considerable work on conformal parameterizations as well as on curvature line parameterizations related to our quasiisothermic scheme. We can only give a selection of previous work here. For a general background on mesh parameterization we refer to the surveys by Floater and Hormann [24] and Sheffer *et al.* [67].

Our algorithmic approach is based on the discrete conformal equivalence of triangle meshes as introduced by Springborn *et al.* [69] (see the work of Bobenko *et al.* for the mathematical background [13]). The convex functional optimized by Springborn *et al.* constructs a conformally equivalent flat mesh for specified boundary conditions and singularities. Our method is related to work on angle based flattening by Sheffer and de Sturler [66]. They aim for conformal parameterizations and express this by the additional constraint, that triangle angles have to be close to the original ones on the surface. For discrete isothermic parameterizations the definitions coincide.

Parameterizations aligning with lines of principle curvature were constructed by Alliez *et al.* [4]. Their method involves the integration of curvature vector fields and does not include an optimization towards conformality. Global parameterizations following arbitrary frame fields (including in particular principle curvature fields) are constructed in the work of Kälberer *et al.* [35]. They use discrete Hodge decomposition and harmonic vector fields to obtain a globally consistent parameterization. Their QuadCover algorithm can deal with surfaces of arbitrary genus and treats singularities using a suitable branched cover. Both algorithms cover arbitrary triangulated surfaces and implementations are highly complex.

The use of variational principles to enforce desired properties such as planarity of quadrilateral faces has been successfully used in architectural geometry. Liu *et al.* [44] propose an algorithm to optimize a quadrilateral mesh to become planar and even conical. Pottmann *et al.* [55] use functionals to approximate freeform surfaces with single curved panels. The energy minimized in Section 3.5 is a combination of a new functional with an energy recently described by Schiftner *et al.* [58]. They construct circle packing triangle meshes that approximate a given surface by minimizing a combination of energies.

Discrete s-isothermic minimal surfaces are defined in terms of their Gauß map by Bobenko *et al.* [11]. This Gauß map is a Koebe polyhedron with edges tangent to a sphere. These Koebe polyhedra also occur in the study of edge offset meshes by Pottmann *et al.* [54], who again use a variational approach to obtain support structures. Another parametrization technique creating quad-dominant meshes guided by conjugate parameter directions is given by Zadravec *et al.* [75]. Their algorithm includes a level set approach to circumvent the integration of a vector field.

The notion of discrete s-isothermic meshes was introduced in the mathematical context by Bobenko and Pinkall as a special class of quad meshes [12]. The mathematical theory of these meshes has since then been an active field of research in discrete differential geometry. A good overview of the recent development and literature can be found in the book by Bobenko and Suris [16].

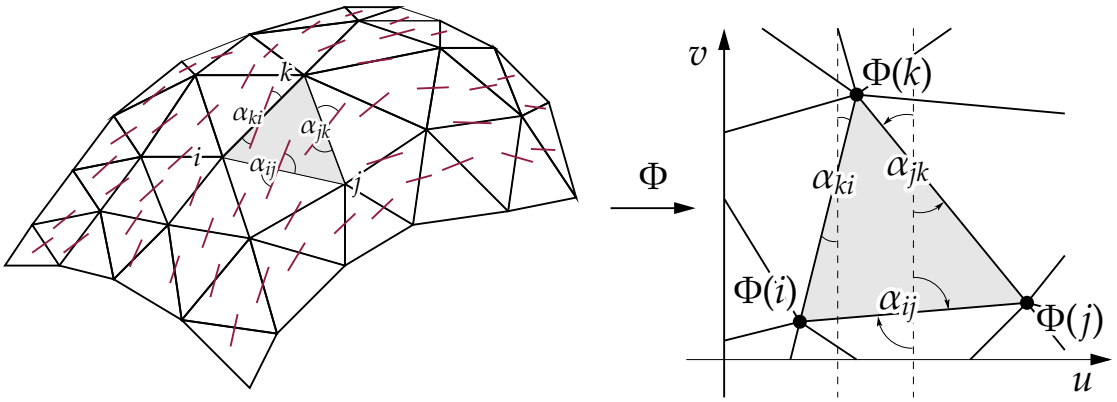


Figure 3.3: A discrete isothermic parameterization. Angles between triangle edges and a curvature direction family are preserved by the map.

3.3 Discrete quasiisothermic parameterization

In this section we introduce the notion of quasiisothermic parameterizations and the corresponding quality measure.

Discrete parameterizations

Let $M = (V, E, F)$ be a triangle mesh. The elements of V are the vertices of the mesh denoted by simple indices $i \in V$. Edges are denoted by double indices $ij \in E$, and faces are denoted $ijk \in F$. A triangulated surface is a map $S : V \to \mathbb{R}^3$, $i \mapsto (x_i, y_i, z_i)$. We call a map $\Phi : V \to \mathbb{R}^2$, $i \mapsto (u_i, v_i)$ a *discrete parameterization* of the surface S. We only consider orientable surfaces and parameterizations that preserve the orientation of the triangles with respect to the canonical orientation of $\mathbb{R}^2$.

The next definition connects arbitrary parameterizations with certain directions tangent to the surface S, e.g., curvature directions. Such a direction is encoded as an angle per edge.

Definition 3.3.1. *Let $\alpha : E \to]-\frac{\pi}{2}, \frac{\pi}{2}]$, $ij \mapsto \alpha_{ij}$ be a map that assigns an angle to each edge. A discrete parameterization $\Phi : V \to \mathbb{R}^2$, $i \mapsto (u_i, v_i)$ is called a* discrete parameterization with α *if*

$$\tan \alpha_{ij} = \frac{u_i - u_j}{v_i - v_j} \tag{3.1}$$

for all edges of the mesh.

In other words, in a parameterization with α the image of an edge $ij \in E$ under the map Φ encloses the prescribed angle α_{ij} with the v-axis of the parameter space. One could equally use the u-axis here.

Quasiisothermic parameterizations

Our main example of a parameterization with an angle function α comes from α defined by the curvature directions of a surface $S : V \to \mathbb{R}^3$ (see Fig. 3.3). For a triangulated surface

curvature directions and magnitudes can be calculated and assigned to edges. This is usually done by averaging curvature information over neighborhoods of points on the surface [22]. A discrete parameterization with angle function α stemming from the curvature direction field is then called a *discrete isothermic parameterization*. Indeed, the latter is just a curvature line parameterization. In our case the curvature directions are mapped to the coordinate directions in the (u, v)-plane. The map can be treated as conformal: Angles between edges and curvature directions are preserved.

Generic surfaces do not allow for isothermic parameters (those admitting isothermic parameterizations are called isothermic surfaces). Therefore we do not expect a parameterization with given α to exist in general. To be able to deal with arbitrary surfaces we introduce the notion of discrete quasiisothermic parameterization. The idea is to obtain a parameterization with angles $\tilde{\alpha}$ as close to the curvature directions α as possible. Let

$$Q^{\alpha}(ij) = \left|\alpha(ij) - \tilde{\alpha}(ij)\right| \tag{3.2}$$

where $\tilde{\alpha}(ij)$ is angle the between the v-axis and the edge ij in parameter space.

Definition 3.3.2. *We call a discrete parameterization Φ with angle function α* quasiisothermic *with modulus $Q \in \mathbb{R}_+$ if*

$$Q^{\alpha}(ij) \leq Q \tag{3.3}$$

for all edges $ij \in E$.

The motivation for this measure of quasiisothermicity is the following: If Q is small the directions of principle curvature on the surface are almost tangent to the parameter lines of the parameterization. For a modulus of zero we have an in a sense angle preserving map where edges enclose the same angles with the coordinate axes in parameter space as with curvature directions on the surface. We will now create parameterizations that have small Q.

3.4 Minimization of the functional

In this section the surface M is a triangulated surface with one boundary component. We will now construct a function Φ that has zero approximation error Q^{α} at boundary edges and is a discrete conformal map in the sense of Springborn *et al.* [69] in the interior of the surface. We argue under which circumstances this leads to nice behaviour in the interior. We start by briefly introducing discrete conformal maps and the boundary conditions we need for our purposes.

Discrete conformal maps

We recall the definition by Springborn *et al.* of discrete conformal maps via conformal equivalence of triangle meshes. It is stated in terms of lengths of the surface edges and corresponding parameter edges in the (u, v)-plane.

Definition 3.4.1. *A discrete parameterization Φ is* conformal *if there exists a function $\mu : V \to \mathbb{R}$, $i \mapsto \mu_i$ such that the following condition for the edge lengths l_{ij} on the surface and $\tilde{l}_{ij} = \|\Phi(i) - \Phi(j)\|$ in parameter space holds*

$$\tilde{l}_{ij} = \mu_i \mu_j l_{ij}. \tag{3.4}$$

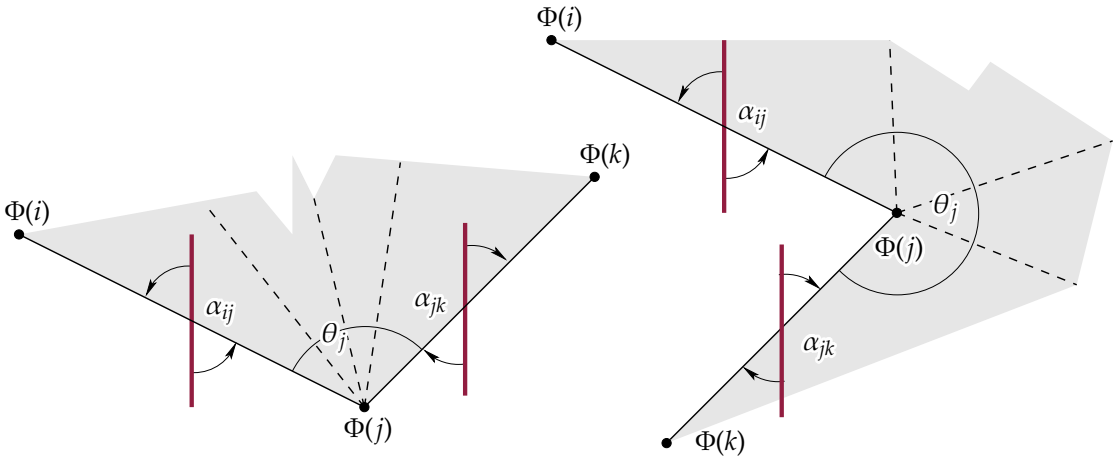


Figure 3.4: Curvature boundary conditions: In parameter space the interior angle θ_j at the vertex $\Phi(j)$ has to be chosen such that curvature directions given by angles α_{ij} and α_{jk} align with the v-axis. This choice is unique up to addition of $k\pi$. The above pictures show two possible layouts in the parameter plane depending on the interior angle sum on the surface.

For a given triangle mesh there is a unique solution μ that retains the boundary edge lengths. Another unique solution can be obtained by fixing the angles between consecutive boundary edges. These angles have to be chosen consistently obeying the Gauß-Bonnet relation (see the section about singularities).

The function μ for a given triangle mesh can be found as the minimizer of a convex functional. Thus its computation is efficient. The resulting parameterization is created by a breadth-first layout that enumerates all triangles and assigns texture coordinates. In addition to boundary angles one can ask for solutions that contain special interior vertices where the sum of triangle angles is not equal to 2π (see Fig. 3.1). These so-called cone points will be inserted at singularities of the parameterization.

Curvature boundary conditions

Let $\alpha : E \to]-\frac{\pi}{2}, \frac{\pi}{2}]$ be an angle function derived from numerical curvature directions and the given surface orientation. Let $ij \in E$ and $jk \in E$ be two consecutive boundary edges with common vertex j and let $\tilde{\theta}_j$ be the sum of interior triangle angles on the surface at this vertex. The direction of the edge $\Phi(ij)$ (resp. $\Phi(jk)$) is determined by the angle α_{ij} (resp. α_{jk}), since the curvature directions encoded by the α's should align with the v-axis. The orientation of the surface (resp. the boundary edges) defines the angle θ_j between the edges $\Phi(ij)$ and $\Phi(jk)$ up to addition of $k\pi$ (see Fig. 3.4). To make the angle unique we require that the difference between the original surface angle $\tilde{\theta}_j$ and the angle in the parameter plane θ_j is as small as possible, i.e., choose $k > 0$ as small as possible such that

$$|\underbrace{\angle(\Phi(ij), \Phi(jk)) + k\pi}_{\theta_j} - \tilde{\theta}_j| \leq \pi/2.$$

See Figure 3.4 for an illustration of the alignment in the parameter plane. The angles θ_j at boundary vertices serve as boundary conditions for the discrete conformal parameterization. A

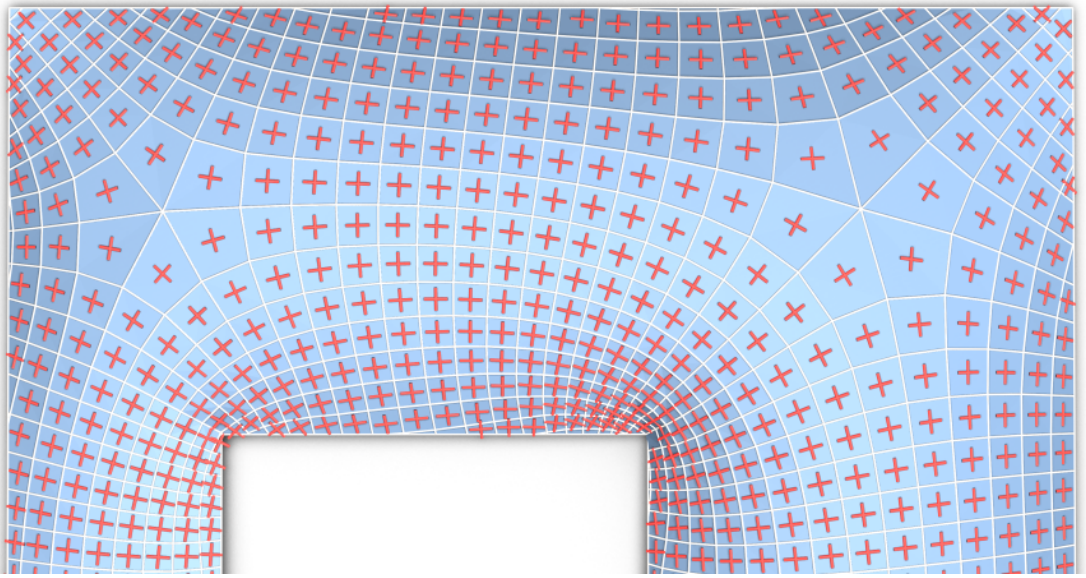


Figure 3.5: Parameterization of the Roof model. The discrete curvature lines approximate curvature directions with high quality. See Section 3.4 for a discussion.

conformal parameterization with these boundary conditions will then have perfect alignment of curvature directions at the boundary.

Singularities

There exists an analog of the smooth Gauß-Bonnet theorem for discrete surfaces that relates the Gaussian and the boundary curvature to the Euler characteristic. During parameterization we construct a metric that is flat everywhere except for cone singularities where positive or negative curvatures are introduced. For the purpose of curvature line parameterizations we can only have cone points with discrete curvatures of π, 0, or $-k\pi$ at singularities of the curvature direction field. The Gaussian curvature κ_i at interior vertices $i \in V_I$ is the angle defect, i.e., $\kappa_i = 2\pi - \theta_i$, where θ_i is the sum of the angles at the vertex i. For a boundary vertex $j \in V_B$ the corresponding geodesic curvature is defined by $\kappa_j^g = \pi - \theta_j$. So if we split the vertex set $V = V_B \cup V_I$ into boundary vertices V_B and interior vertices V_I the discrete Gauß-Bonnet theorem becomes:

$$\sum_{i \in V_I} \kappa_i + \sum_{j \in V_B} \kappa_j^g = 2\pi\chi, \tag{3.5}$$

where χ is the Euler characteristic of the surface ($\chi = 1$ for disks). Since all curvature directions at boundary edges in parameter space become parallel, the boundary curvature adds up to a multiple of π. If this sum happens to differ from 2π, there must be singularities in the curvature field and we have to compensate the deficit at interior vertices to satisfy Equation (3.5). In Figure 3.5 the boundary curvature sum of the domain is 4π. So by inspection of the curvature field in the interor of the surface we picked two singularities each of curvature $-\pi$ to satisfy the Gauß-Bonnet equation. They correspond to cone points with angle 3π in the parameterization.

Implementation

The algorithm to compute a quasiisothermic parameterization with vanishing modulus at the boundary of the domain is the following:

(1) Generate curvature directions and compute α at the boundary
(2) Calculate boundary angles θ and pick singularities
(3) Compute conformal parameterization with given θs
(4) Perform remeshing
(5) Remove cone point cuts

To estimate principle curvature fields we use the method of Cohen-Steiner and Morvan [22], where the curvature tensor is averaged over a disk of a given radius centered at edge midpoints. Together with a fixed orientation of the surface this defines the angle function α.

We deduce the angles θ for the boundary vertices as described in Section 3.4. These angles are the boundary curvatures we plug into the algorithm of Springborn *et al.* to obtain a conformal parameterization. If necessary, we pick singularities for the curvature field at vertices and prescribe corresponding cone angles by inspection of the curvature direction field on the interior of the surface. A consistent singularity choice can easily be checked using Equation (3.5). By construction we can only process curvature fields with isolated singularities.

We layout the new edges in the parameter plane such that an arbitrary boundary edge $\Phi(ij)$ intersects the v-axis in the desired angle α_{ij}. By construction the intersection angles coincide with the prescribed α's for all boundary edges. The domain of parameterization can contain singularities, which are modeled as cone points with prescribed curvature. Therefore we have to cut along paths from the cone points to the boundary of the mesh. The layout overlaps if singularities with negative curvature are used. To create seamless parameter lines we use the rectification approach described by Springborn *et al.* [69].

Finally, we create a new mesh based on a regular (u, v)-grid in $\mathbb{R}^2$. The remeshing process is carried out as a subdivision step followed by some cleanup and regluing: We use the projective interpolation in the texture domain to increase the quality of the result. Previously cut paths from singularities to the boundary are sewed up to obtain the final remesh.

Examples and quality

With the quasiisothermic modulus Q^α on edges introduced in Equation (3.2) we are now able to measure the quality of our parameterizations.

There are two kinds of examples to consider: The first class of meshes stems from smooth surfaces that admit conformal curvature line parameterizations, i.e., triangulations approximating isothermic surfaces. The second class consists of arbitrary non-isothermic surfaces. For almost isothermic surfaces we expect our parameterization to reconstruct the isothermic coordinates up to numerical precision and hence Q^α to be reasonably small. For non-isothermic surfaces we achieve the correct directions on the boundary but lack accuracy in the interior. In Table 3.1 we summarize the numerical results obtained from the surfaces of Figure 3.2.

Isothermic surfaces. The class of smooth isothermic surfaces contains surfaces of constant mean curvature, surfaces of revolution, and conic sections. We use the MINIMAL example as an instance of an isothermic surface with mean curvature zero (Figure 3.2b). As expected this surface exhibits the highest curvature line quality of all tested meshes. The error however cannot

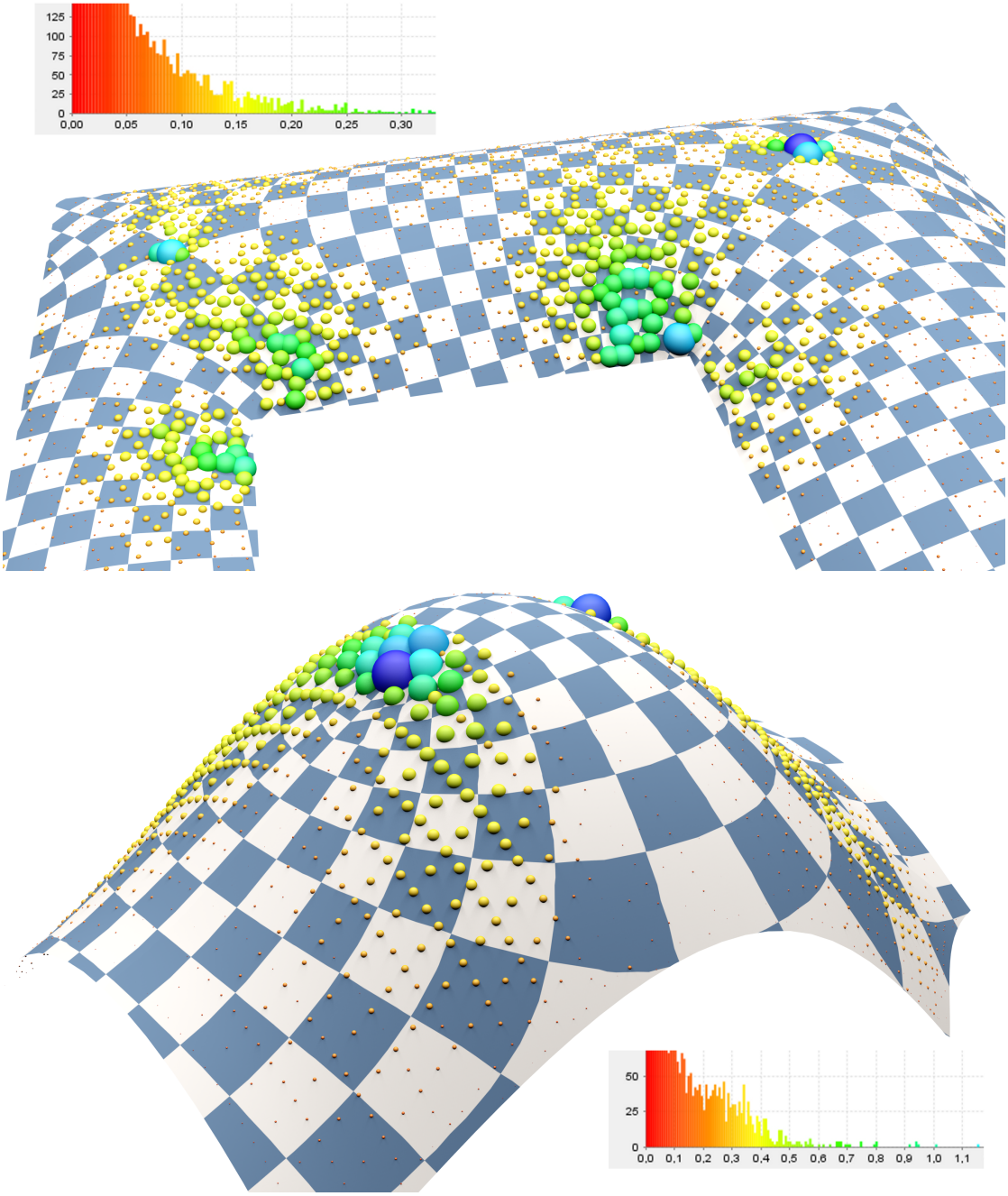


Figure 3.6: The quality of the parameterization is measured in radians per edge of the underlying triangulation. The checkerboard texture indicates the parameter lines of the map. Small (red and yellow) beads represent good curvature direction quality, big beads (green and blue) represent high deviation. The color of the histogram corresponds to the color of the beads. Note that the mean error of the Roof surface (top) is half the error of the Dome. See also Table 3.1 for detailed quality measures of the other surfaces.

	$\#E$	$\#\partial E$	Q^{α}_{mean}	$Q^{\alpha}_{\max}$	Q^{α}_{σ}
Minimal	6260	450	0.036	0.603	0.033
Teaser	17550	1000	0.057	1.20	0.066
Roof	3766	470	0.051	0.610	0.059
Dome	1900	350	0.133	1.52	0.157

Table 3.1: The curvature line approximation quality of the examples. $\#\partial E$ are the number of boundary edges. Q^{α}_{σ} is the standard deviation of Q^{α}.

vanish completely since the surface's curvature field contains singularities. In the vicinity of these points the numerical curvature directions contain significant amounts of noise.

Non-isothermic surfaces. Non-isothermic surfaces are surfaces that do not admit a parameterization with conformal curvature lines. We investigate the properties of surfaces (a), (c), and (d) displayed in Figure 3.2.

The Teaser surface was created as a minimizer of a spring functional fixing the boundary and modelling interior edges as springs of rest length zero. It is not far away from a minimal surface with the same boundary. The curvature line pattern however differs substantially as it contains singularities whereas the minimal surface with this boundary curve does not. The quality of the curvature line pattern is also very high. The mean angle error of the numerical directions is 3.2 degrees. Note that the deviation Q_{σ} from the mean value is also very low. For this surface the coordinates generated by our algorithm are a globally good approximation to conformal curvature lines.

A quality plot of the Roof surface (Figures 3.2d and 3.5) is shown in Figure 3.6. Surprisingly, the quality of the curvature lines is as high as in the Teaser or the Minimal case. This suggests that a slight variation of the surface yields an isothermic surface. See also Figure 3.5 for a visual impression of the quality of the curvature lines.

The Dome model (Figure 3.2c) is created from a NURBS surface. The quality plot (Figure 3.6) reveals areas of high angle deviation especially around the singularities. Other areas, in particular those near the boundary, are of high curvature line quality. The distance to the nearest isothermic surface is expected to be larger than in the previous examples. More evidence for this is given in Section 3.5.

Discussion. Our parameterization scheme works well for surfaces that are not too far away from surfaces that possess isothermic coordinates. In the case of surfaces stemming from minimal or constant mean curvature surfaces we get almost perfect approximation quality of curvature lines. These are surfaces that are particularly interesting when designing beam layouts for roof structures that where form-found. For other surfaces the parameterization is conformal and the parameter line pattern captures the combinatorics of the curvature line pattern while approximating the curvature line geometry. There are of course surfaces for which our method is not applicable. If the boundary is too short compared to the overall size of the surface we cannot expect the solution to follow curvature lines as the distance to the boundary increases.

3.5 Discrete s-isothermic surfaces

Starting with a quasiisothermically parameterized mesh with low modulus we now aim to create discrete s-isothermic surfaces that stay in the vicinity of the input surface. S-isothermic surfaces

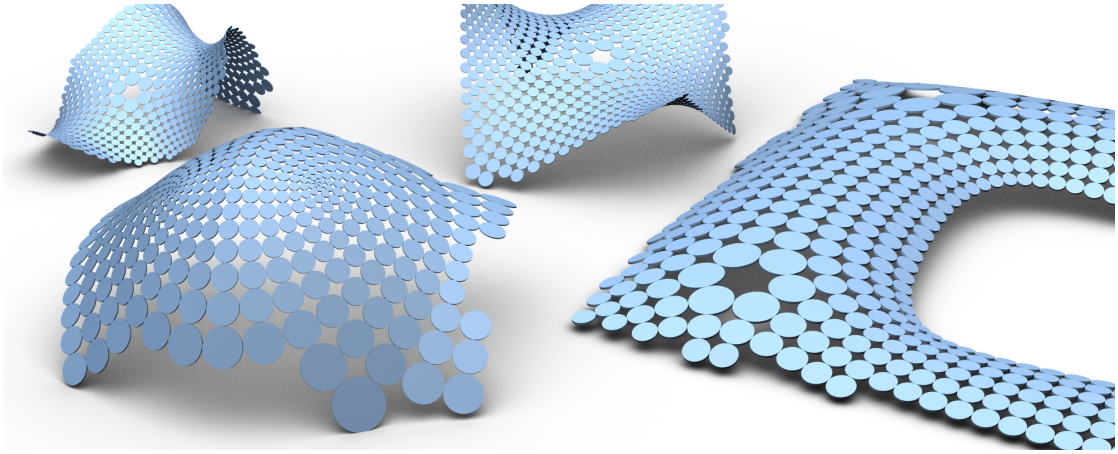


Figure 3.7: S-isothermic meshes created from the models presented in Figure 3.2. The inner quadrilaterals are optimized towards touching incircles. A series of touching circles in a row can be interpreted as discrete curvature line.

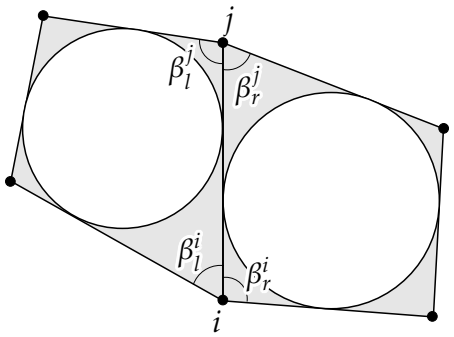


Figure 3.8: Labels for the touching-circle functional at an edge. The circles touch if the ratio $\cot\left(\beta^i/2\right) / \cot\left(\beta^j/2\right)$ is equal on both sides of the edge.

were introduced by Bobenko and Pinkall [12]: A quadrilateral mesh is *s-isothermic* if (i) all the quadrilaterals are planar, (ii) all faces have incircles, and (iii) the incircles of adjacent quadrilaterals touch. Figure 3.7 displays s-isothermic surfaces derived from our parametrizations shown in Figure 3.2.

Variational Principle

In this section we introduce an energy whose minimizers are s-isothermic surfaces. We denote quadilaterals by $ijkm \in F$ where the indices are in cyclic order. The s-isothermic energy E_S consists of three parts:

$$E_S := \lambda_1 E_{\text{planar}} + \lambda_2 E_{\text{incircle}} + \lambda_3 E_{\text{touch}} \tag{3.6}$$

The planarity energy E_{planar} penalizes non-planar quadrilateral faces. For each quad it can be defined either by the distance of the diagonals (an idea attributed to Peter Schröder [55]) or the volume of the tetrahedron spanned by the four vertices. We give the formula for the former

here.

$$E_{\text{planar}} = \sum_{ijkm \in F} \frac{\left\langle \Delta_{ji}, \Delta_{mj} \times \Delta_{ki} \right\rangle^2}{\|\Delta_{mj} \times \Delta_{ki}\|^2} \tag{3.7}$$

Here Δ_{ij} is the vector pointing from vertex i to j.

For E_{incircle} we use the energy defined by Schiftner *et al.* [58] based on the fact that the sum of opposite edge lengths must be equal for a planar quad to possess an incircle.

$$E_{\text{incircle}} = \sum_{ijkm \in F} \left(l_{ij} + l_{km} - l_{jk} - l_{mi} \right)^2 \tag{3.8}$$

The energy E_{touch} is a new energy that enforces touching incircles if faces are planar and possess incircles. It is defined per edge, see Figure 3.8 for the exact labeling of the angles at one edge. For an interior edge $ij \in E$ we define

$$E_{\text{touch}}(ij) = \left(\cot \frac{\beta_l^j}{2} \cot \frac{\beta_r^i}{2} - \cot \frac{\beta_r^j}{2} \cot \frac{\beta_l^i}{2} \right)^2 . \tag{3.9}$$

On boundary edges the energy is zero. All energies can be formulated in terms of the vertex coordinates and the derivatives can be calculated explicitly.

Since all energies are in general non-convex we need a good initial guess to find meaningful minimizers of E_S. S-isothermic minimal surfaces converge to isothermic parameterizations of smooth minimal surfaces [11]. For general s-isothermic surfaces this is an open conjecture. The parameterizations obtained in Section 3.3 are good candidates to start from with the optimization of the functional. We use the non-linear optimization package PETSc/TAO [7, 8] and its java binding [68] to find minimizers of E_S. Figure 3.10 shows convergence plots of E_S for the four models that were discussed in the previous section.

As seen in the quality analysis of Section 3.4, the Teaser, the Minimal, and the Roof models are quasiisothermic surfaces with low modulus. For these models the corresponding s-isothermic surface is also close to the input surface. Figure 3.10 shows the energy during the optimization. Here the three close-to-isothermic meshes start with a lower energy than the Dome model. After the Dome has passed some iterations it exhibits convergence properties similar to the other models. As this surface converges against a discrete s-isothermic surface, we observe a considerable change in shape during the first iterations especially around the singularities.

Isometric Deformations of Minimal Surfaces

Every smooth minimal surface possesses a 1-parameter (associated) family of non-trivial isometric deformations. All surfaces in this family have the same Gauß map. For discrete s-isothermic minimal surfaces this construction is discretized by Bobenko *et al.* [11]: Edge lengths and the conformality of the parameterization are preserved. We need to introduce the concept of dual surface to construct the family of isometric surfaces.

The Dual Surface. In differential geometry for isothermic surfaces there is a notion of a dual surface. This dual or Christoffel transform is also an isothermic surface. Both surfaces are parameterized with isothermic coordinates. This setup can be discretized using the definition of discrete s-isothermic surfaces. The dual surface can be constructed using the incircle structure of s-isothermic meshes. We introduce consistent signs on edges on a discrete s-isothermic surface

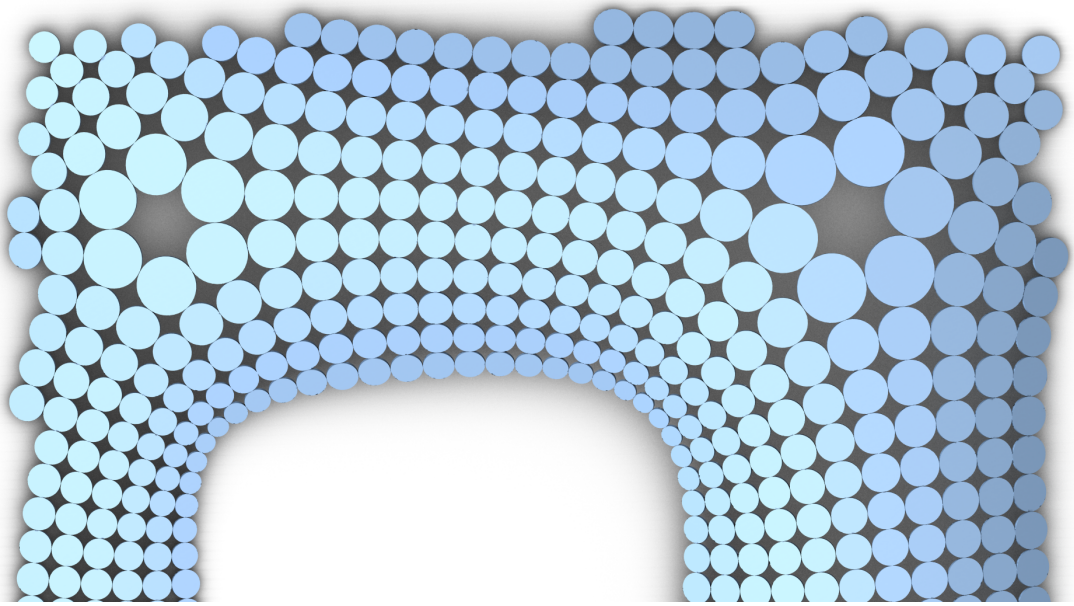


Figure 3.9: The s-isothermic circle packing on the Roof model in detail.

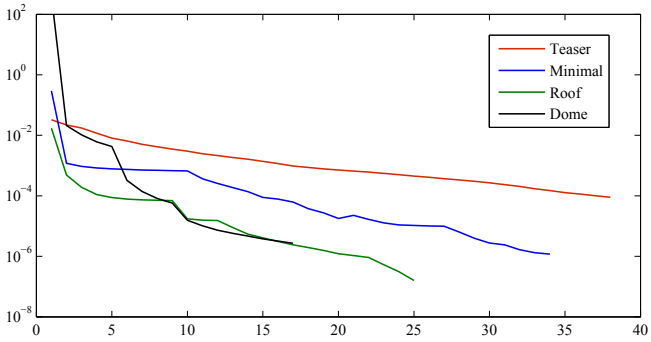


Figure 3.10: Convergence behavior of E_S during optimization. We use the meshes displayed in Figure 3.2 as initial guesses for the minimization. The convergence of the Teaser geometry is slower due to the high complexity.

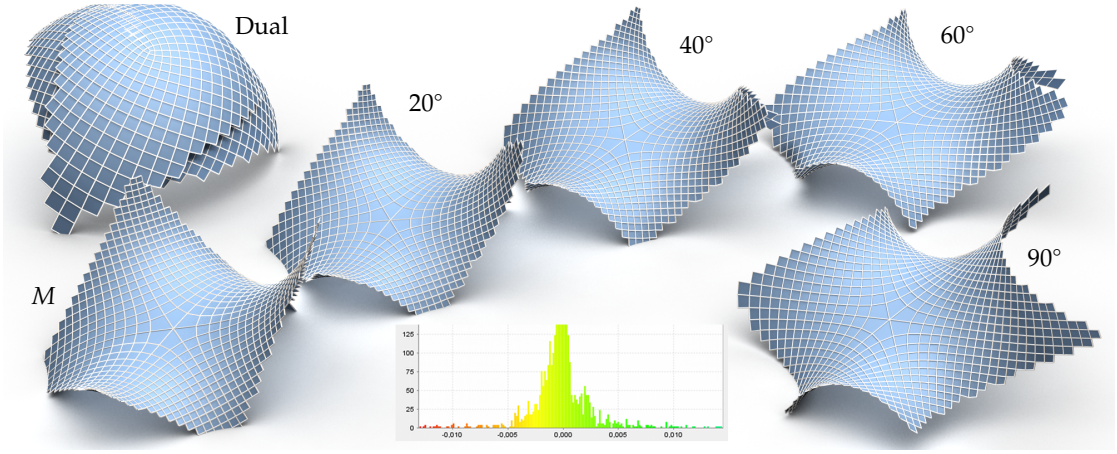


Figure 3.11: Non-trivial isometric deformations of a minimal surface. The edges of the dual surface are rotated to create a 1-parameter family of isometrically deformed surfaces with period 2π. The histogram shows the edge length error of the 90° model when compared with the initial surface. The mean edge length deviation of this model is 1.28%.

such that opposite sides of the quads have the same sign and consecutive edges in a quad have different signs. The dual mesh of a mesh M will have parallel edges calculated as follows: Let $e_{ij} = v_j - v_i$ be an edge vector of M. Then the dual edge vector e^*_{ij} is defined as

$$e^*_{ij} = \pm \frac{1}{r_i \cdot r_j} e_{ij}$$

where r_i and r_j are the distances from the vertex v_i and v_j to the touching point of the incircles of incident faces at the edge e_{ij}. For a given discrete s-isothermic mesh we can easily calculate the dual mesh by enumerating the vertices along a spanning tree of edges. In Figure 3.11 the dual surface of the MINIMAL model is shown in the upper left hand corner. Via the dual surface we can now construct isometric deformations of minimal surfaces.

Deformation. The dual of a smooth minimal surface coincides with its Gauß map which is a part of a sphere. This sphere may be multiply covered. For discrete s-isothermic minimal surfaces this Gauß map is a part of a Koebe polyhedron, i.e., a polyhedral surfaces with edges tangent to a sphere. On the Koebe polyhedron every edge is rotated by a fixed angle in the tangent space of the sphere at the points of tangency of the edge. The resulting edge vectors again form closed quads and can be dualized. This dual surface has the same edge lengths as the initial minimal surface.

For minimal quad meshes with touching incircles that were created using our parameterization and the optimization step, the dual surface will be close to a Koebe polyhedron. We use a least-squares-sphere to define a consistent tangent space at the touching points of incircles with the edges. The resulting deformation of a given minimal surface is then close to isometric. To distribute the isometry error on the edges we average over different roots of the layout spanning tree. Surfaces that do not possess exact isometric deformations are deformed approximative.

We apply this procedure to the MINIMAL model (Figure 3.2a). Figure 3.11 shows the surface together with its dual and isometrically deformed versions with different turning angles.

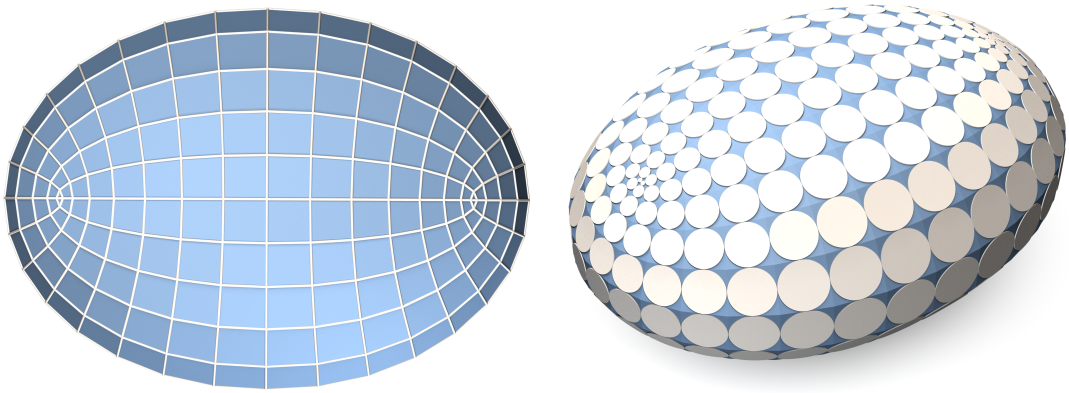


Figure 3.12: Discrete s-isothermic ellipsoid

3.6 A discrete s-isothermic ellipsoid and its dual surface

Here we describe the construction of a discrete s-isothermic ellipsoid and its dual surface. (See Hertrich-Jeromin [32, p. 202] for the smooth analog). The rough construction outline is as follows. We start with a discretized version of the smooth isothermically parameterized ellipsoid. We obtain this by calculating a quasiisothermic parameterization of half of a triangulated ellipsoid. After remeshing, the parameter lines are approximate curvature lines. We complete the surface by reflection at the boundary. Optimization of the s-isothermic functional yields the final discrete ellipsoid. Its dual surface is constructed as in Section 3.5.

The connectivity at singularities of the discrete ellipsoid is not unique. If there is a discrete curvature line in the symmetry plane of the ellipsoid, we have degenerate quads that have vertices with angle $\approx \pi$. The singularity is located at this valence 2 vertex. If there is no discrete curvature line at the symmetry plane then the singularity is located at the mid point of the edge connecting two valence three vertices. We have observed that the convergence behavior of the s-isothermic functional is considerable better in the former model of singularities. We therefore like to call only the discrete ellipsoid with curvature lines at symmetry planes a discrete ellipsoid, see Figure 3.12 and 3.13.

3.7 Conclusions and future research

The main contributions of this article are, on the one hand, the definition of quasiisothermic parameterizations together with a new algorithm to compute parameterizations of surfaces that optimizes the corresponding quality measure. On the other hand, we have defined a new energy for meshes with touching incircles.

We see the main advantage of the algorithm presented in Section 3.3 in its simplicity and its applicability to shell-like roof structures which arise in architectural models. Since these models often have almost constant mean curvature and thus allow for an almost isothermic parameterization, our algorithm performs particularly well on these examples.

The new energy described in Section 3.5 is closely related to the work of Schiftner *et al.* [58] dealing with circle packing meshes. They explicitly do not treat quadrilateral meshes since

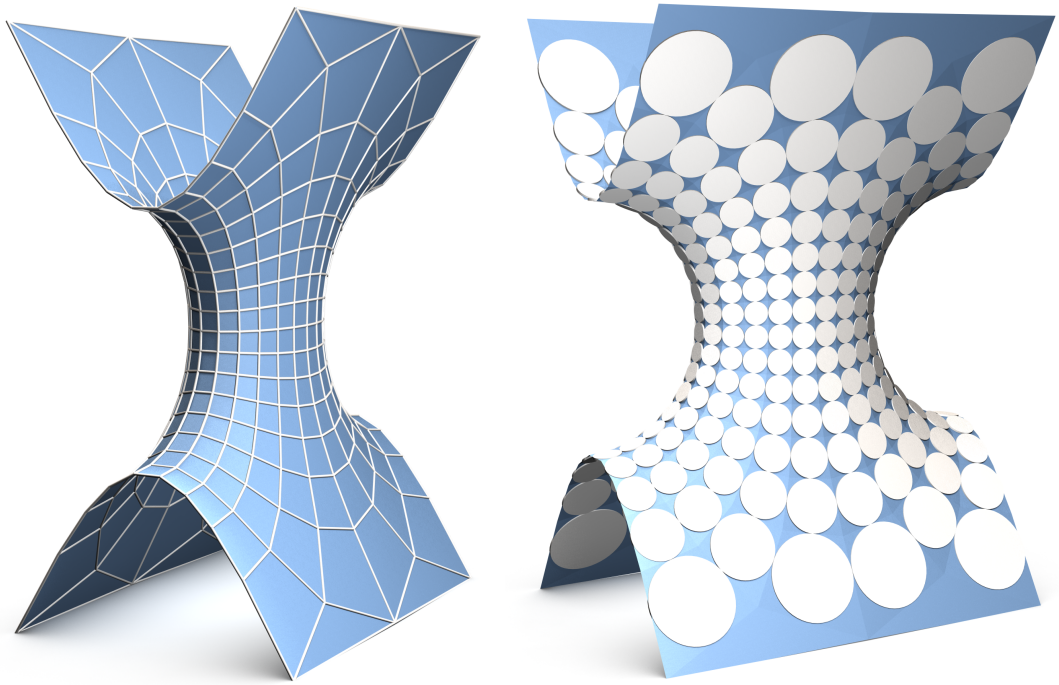


Figure 3.13: Dual surface of discrete s-isothermic ellipsoid of Figure 3.12.

they are aware of the shape restrictions and focus on triangle meshes instead. The shape restriction lies in the core of isothermic surfaces but did not influence our results dramatically for the surfaces in our focus. How to approximate arbitrary surfaces by isothermic surfaces is unknown and will be subject to future research. The results of Section 3.5 suggest that this might be possible using related methods.

Our new functional generates quad circle packing meshes in the sense of Schiftner and co-authors. For surfaces arising in architectural context (in particular for shell-like roofs) we are able to construct aesthetically pleasing quad meshes supporting a circle packing.

Chapter 4

Optimization of Regular and Irregular Elastic Gridshells

This publication is joint work with Elisa Lafuente Hernández, Thilo Rörig, and Christoph Gengnagel and was previously published in the proceedings of the conference "Advances in Architectural Geometry 2012" [41] under the title *Topology optimisation of Regular and Irregular Elastic Gridshells by means of a Non-linear Variational Method*.

Gridshells composed of elastically-bent profiles offer significant cost and time advantages during the production, transport, and construction processes. Nevertheless, the shaping of the initially flat grid imposes significant bending stresses on the structure, thus reducing the bearing capacity against external loads. An optimization of the grid's geometry in order to minimize the curvature of the profiles and, with it, the initial stresses is therefore crucial. In this work a non-linear variational method for optimizing the geometry of elastic gridshells with regular and irregular meshes is presented. Different case studies of doubly-curved gridshells show the advantages and capacity of this method.

4.1 Introduction

Elastic gridshells make use of the principle of active bending [5], since their final geometry results from the elastic deformation of initially flat grids. This construction principle has the advantage of reducing costs and time during the production, transport, and construction. Nevertheless, the shaping of the profiles induces significant stresses on the grid, thus reducing the bearing capacity against external loads.

In order to diminish the initial stresses, profiles with low sections and materials with low modulus of elasticity are usually chosen. However, this leads to a reduction of the global stiffness of the gridshell that can result in stability problems. With an optimization of the geometry (orientation and arrangement of the grid profiles), a minimization of the profile's curvature can be obtained and the load-bearing capacity of the gridshells can be improved [40].

In 2009, M. Kuijvenhoven proposed a design methodology for elastic gridshells based on particle-spring models [39]. In this method, the gridshell geometry results from an iterative process, where the initially flat grid is progressively approached, vertically, towards the refer-

ence surface by shaping springs until achieving the maximum allowable curvature on the grid. Material and sectional properties of the grid profiles are given as input information. Dynamic relaxation is used here to calculate the equilibrium of forces on the grids.

Bouhaya *et al.* [18], Laboratoire Navier of the Paris-Est University, present an optimization method based on the geometric compass method described by Frei Otto's Institute for Lightweight Surface Structures [53], combined with genetic algorithms. This method consists of mapping grids, differing in the orientation and angle between the crossing profiles, on an imposed surface as in the compass method and selecting the one with lowest curvature using stochastic genetic algorithms.

In this work, a non-linear variational method for optimizing topologies of regular and irregular elastic gridshells is proposed. The optimization parameters *mesh size*, *reference surface*, and *profiles curvature* are defined as penalizing energies (the difference to the desired values will be considered) with corresponding weighting factors. The resulting grid definition is calculated by minimizing the linear combination of these three energies. In the context of discrete differential geometry, a mesh with constant edge lengths is called a discrete Chebyshev net. So we aim for meshes with the Chebyshev property that approximate a given surface with low curvature in the parameter curves.

The advantage of this method is that the grid is not required to stay on the reference surface and displacements of the grid nodes are possible in all directions, so that a further optimization of the grid can be achieved. Moreover, different grid configurations can be calculated by defining priorities between the optimization parameters. For example, a higher reduction of the profile's curvature can be achieved by tolerating a larger distance from the reference surface or variation on the mesh size (irregular meshes). Several double-curved surfaces with regular and irregular meshes have been optimized with the variational method and the results are presented in the following sections.

4.2 Optimization

Let $M = (V, E, F)$ be a quad-mesh. The vertices of M are denoted by $v_i \in V$, the edges are $e_{ij} \in E$, and the quadrilaterals are denoted by $f_{ijkm} \in F$.

4.2.1 Energies

We use a linear combination of energies to enforce desired properties on the optimized mesh. Our energy consists of three parts.

$$E(M) = \lambda_1 E_{\text{ref}} + \lambda_2 E_{\text{len}} + \lambda_3 E_{\text{cur}} \tag{4.1}$$

The energy E_{ref} penalizes the distance of vertices from a reference surface. This surface can be anything that gives a distance function, e.g., a triangulated surface or a NURBS-surface. The energy and its gradient are given by

$$\begin{aligned} E_{\text{ref}}(M) &= \sum_{v_i \in V} \langle v_i - cp_i, v_i - cp_i \rangle \\ \frac{\partial E_{\text{ref}}}{\partial v_i} &= 2\,(v_i - cp_i) \end{aligned}$$

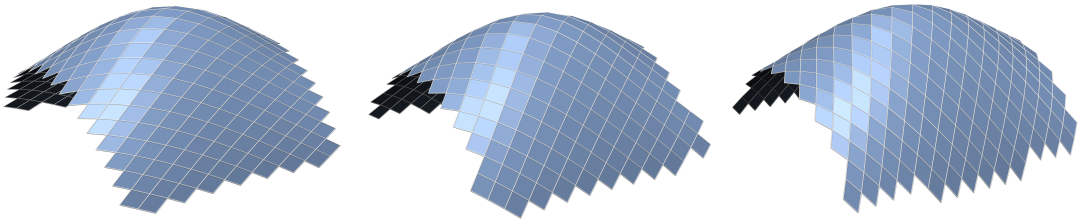


Figure 4.1: Different initialization shear angles for a conformal re-mesh. 30° left, 0° middle, and −30° right.

Here v_i is a vertex of the optimized quad-mesh and cp_i a closest point on the reference surface measured from vertex v_i. The functional E_{len} measures edge length deviation from a given reference length L. Its derivative and energy is given as

$$\begin{aligned} E_{\text{len}}(M) &= \sum_{e_{ij} \in E} \left(\|v_i - v_j\| - L \right)^2 \\ \frac{\partial E_{\text{len}}}{\partial v_i} &= \sum_{e_{ij} \in \text{star}(v_i)} \left(2 - \frac{2L}{\|v_i - v_k\|} \right) (v_i - v_k) \end{aligned}$$

The sum in the derivative is taken over all edges incident to vertex v_i, called the edge-star of vertex v_i. The third energy is a fairing term that penalizes a notion of curvature of curves on the surface. As we only deal with quad-meshes with $\mathbb{Z}^2$ combinatorics every interior vertex has four adjacent edges. The energy E_{cur} and its gradient is defined as:

$$\begin{aligned} E_{\text{cur}}(M) &= \sum_{v_i \in V} (\pi - \angle(e_{i1}, e_{i3}))^2 + (\pi - \angle(e_{i2}, e_{i4}))^2 \\ \frac{\partial}{\partial v_j} \angle(e_{ij}, e_{ik}) &= \frac{-1}{\|e_{ij}\|} \left(e_{ik} - e_{ij} \frac{\langle e_{ik}, e_{ij} \rangle}{\langle e_{ij}, e_{ij} \rangle} \right) \left\| e_{ik} - e_{ij} \frac{\langle e_{ik}, e_{ij} \rangle}{\langle e_{ij}, e_{ij} \rangle} \right\|^{-1} \\ \frac{\partial}{\partial v_i} \angle(e_{ij}, e_{ik}) &= -\left(\frac{\partial}{\partial v_j} \angle(e_{ij}, e_{ik}) + \frac{\partial}{\partial v_k} \angle(e_{ij}, e_{ik}) \right) \end{aligned}$$

Here and e_{i1}, e_{i2}, e_{i3}, and e_{i4} are the adjacent edges of v_i in cyclic order. An edge e_{ij} is also used in the role of a vector pointing from vertex v_i to vertex v_j. $\angle(e_{ij}, e_{ik})$ is the angle spanned by the vectors e_{ij} and e_{ik} From the angle derivatives with respect to the vertices the gradient can be computed efficiently.

4.2.2 Initialization and parameters

The energy $E(M)$ is in general non-convex. That means there can be many local optima, and the solution found by some gradient descent depends on the initialization. We propose to use a conformal re-mesh of the reference surface as initialization. The conformality of the parameterization gives us control over the angles between edges of the quadrilaterals. We can introduce shear to the parameterization and modify this angle globally. By this we start with meshes that have almost constant edge angle, see Figure 4.1.

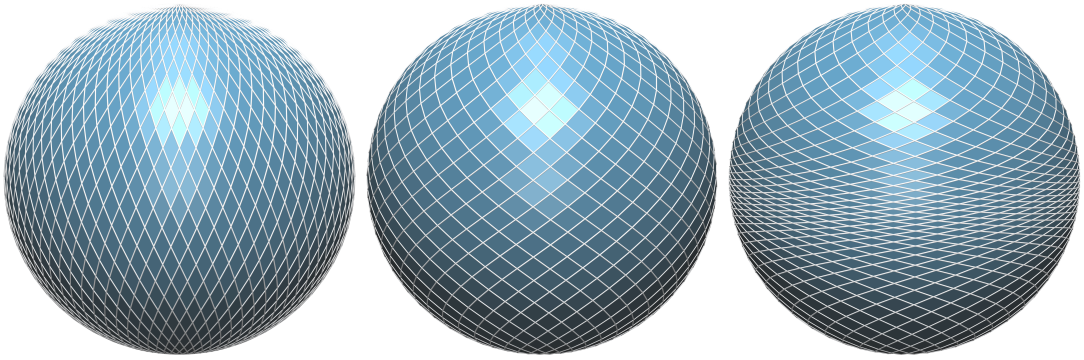


Figure 4.2: Explicitly parameterized spheres with equal edges length $l = 0.11$. The parameter k is equal to 0.4 (left), 0.8 (middle), and 0.99 (right).

4.2.3 Implementation

We use the conformal mapping algorithm by Springborn *et al.* [69] to create the initial mesh. To minimize the energy $E(M)$ we use the non-linear optimization package PETSc/TAO [7, 8] and its java binding [68]. We have made good experiences with normalizing the energies to have gradient length one before optimization. Then we start with all λs equal to one and modify them on the way if needed. If one encounters degenerate configurations during optimization one can drop the length energy term for a few iterations.

4.3 Case studies regular gridshells

4.3.1 The sphere

A simple test of our method is the meshing of a part of a sphere. We will compare our results with a reference mesh that we obtain from a special smooth parameterization of the sphere. Namely there is a smooth parameterization of the sphere that has the property that the lengths of partial derivatives are constant throughout the surface. That means that for small discretization steps we can produce meshes with equal edge lengths from such a parameterization. The formula for this unit sphere can be found in the work of Voss [72]:

$$\begin{aligned} x(u,v) &= \operatorname{sn}(u+v,k)\cdot\cos(k\cdot(u-v)) \\ y(u,v) &= \operatorname{sn}(u+v,k)\cdot\sin(k\cdot(u-v)) \\ z(u,v) &= \operatorname{cn}(u+v,k). \end{aligned}$$

Here sn and cn are the Jacobi elliptic functions with modulus k. For different $k \in]0,1[$ we get spheres with equal edges lengths and different shapes of parameter curves (see Fig. 4.2).

We measure qualitative curvature of these curves like in our energy E_{cur} as $(\pi - \measuredangle(e,\tilde{e}))^2$. Where e and $\tilde{e}$ are opposite edges at a vertex of the quad mesh. The curvature mean of the parameter curves is decreasing for k approaching zero, see Figure 4.3. Using our optimization scheme from the previous section we can reproduce the mesh shapes obtained for different k. The initial mesh is here a sheared conformal re-mesh of a part of the unit sphere, see Figure 4.4. As the sphere suggests there might be solutions with low curvature in the parameter curves that are

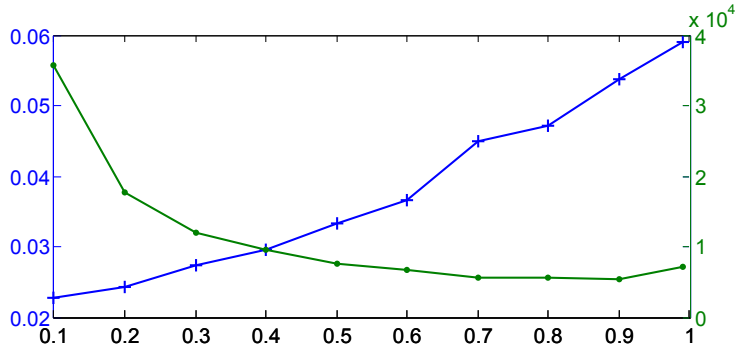


Figure 4.3: The mean value of $(\pi - \angle(e, \tilde{e}))^2$ (blue) on parameter curves of the explicit sphere parameterization is plotted against the parameter k. The green curve indicates the number of edges in the corresponding mesh.

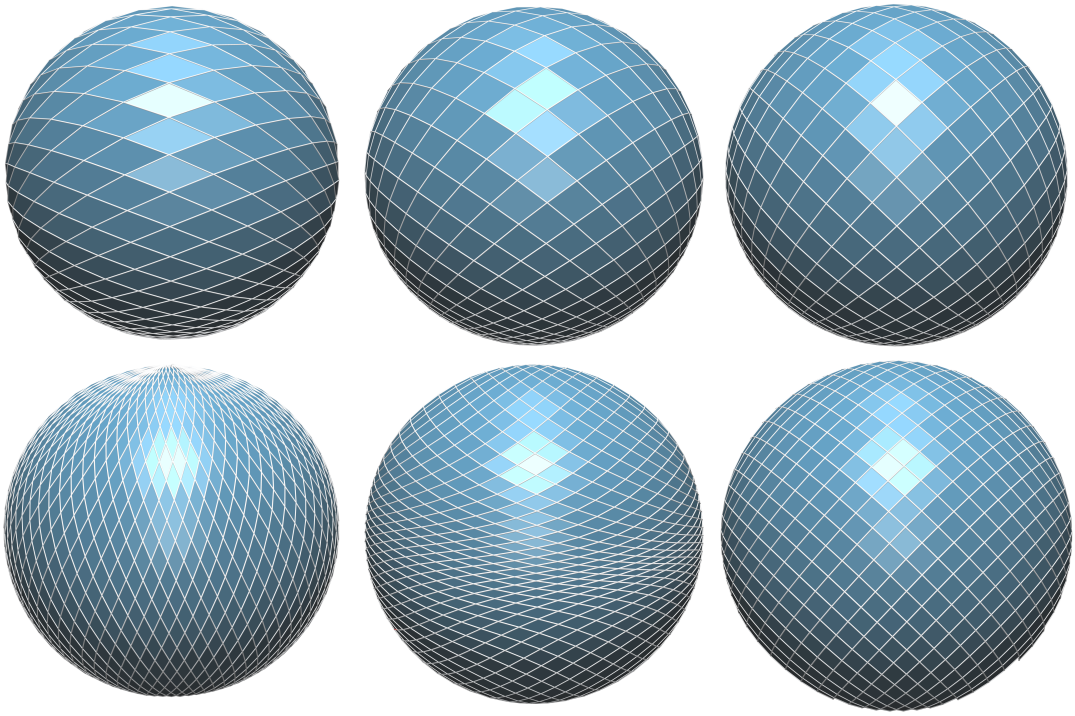


Figure 4.4: Initialization meshes (top) and optimized geometries (bottom). The obtained geometry depends on the initial shear angle and on the boundary shape of the mesh. This leads to geometries that correspond visually to $k \approx 0.4$ (Fig. 4.2 left), $k \approx 0.9$ (Fig. 4.2 middle). For an orthogonal init mesh we obtain a solution that is not contained in the family of smooth parameterizations defined by Voss [72]. The edge lengths are constant and equal to 0.11 in all solutions.

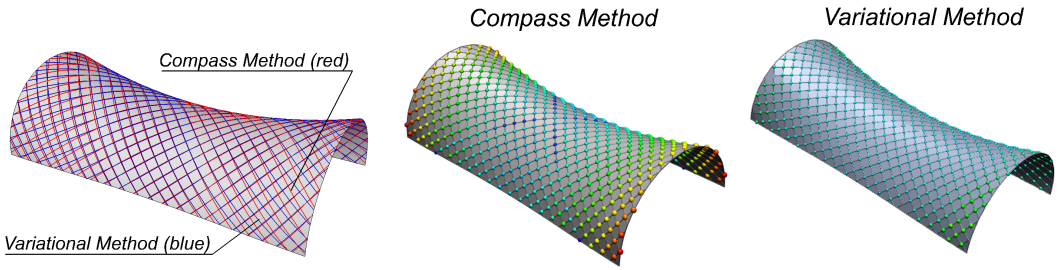


Figure 4.5: Comparison between grid topologies for an anticlastic gridshell. The grid topology resulting from the compass method presents extreme curvature values (large red nodes) on the corners of the lateral edges. The configuration obtained with the variational method shows a more homogeneous curvature distribution and lower curvature values (smaller nodes).

not of use. In our case two angles of the quadrilaterals tend to zero with decreasing curvature. At the same time the number of edges needed for the mesh is increasing. The shear angle of the start mesh gives the family of parameterizations for the sphere and one can easily obtain a good trade-off between number of edges and curvature of the parameter curves.

4.3.2 Comparison with the compass method

Three double-curved gridshells with different types of curvature (anticlastic, synclastic and a combination of both) have been analyzed with the variational method and the results compared with the grid definitions obtained with the classic compass method. The anticlastic gridshell is between 5 and 7.5m high, 14 and 15m wide and 30m long. The synclastic gridshell is between 7.5 and 10m high, 14 and 15m wide and 30m long. Finally, the gridshell with anticlastic and synclastic curvatures, analogue to the Downland Museum gridshell in Sussex, Great Britain (2002), has a height between 7.35 and 9.50m, a width between 12.5 and 16m and a length of 50m.

The mesh size of all three grids is 1m and the starting angle between crossing directions in the centre of the gridshells is 90°. In the compass method, the starting angle corresponds to the angle between the initial curved axes [53], and in the variational method to the angle between crossing segments. A high weighting factor of the E_{ref} energy has been chosen so that a distance from the reference surface lower than 1/500 of the span length can be maintained.

In the following pictures the grid topologies resulting from both methods are shown and compared for the three gridshell structures. The curvatures of the profiles have been calculated as the reciprocal of the radius of the circles defined by three consecutive grid nodes. The curvature distributions, calculated by the variational method in terms of E_{cur}, have been also illustrated through colored points. The size of the points is proportional to the curvature. The maximum and minimum curvatures correspond to the red and blue colors, respectively.

On the case of the anticlastic gridshell, see Figure 4.5, the main difference between the grid topologies is located on the corners of the lateral edges. There, the topology resulting from the variational method tends to go more transversally to the front sides. Also there, the critical curvature of the grid given by the compass method is to be found. The variational method provided a grid topology with a more homogeneous curvature distribution and a maximum curvature value reduced to 87%.

On the case of the synclastic gridshell, see Figure 4.6, slight differences can be found on the whole

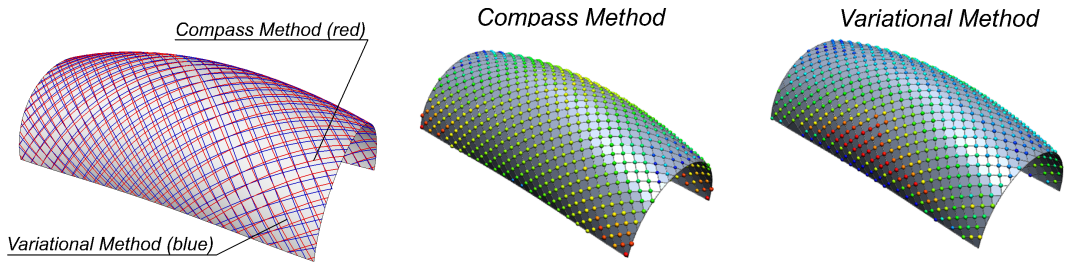


Figure 4.6: Comparison between grid topologies for a synclastic gridshell. The grid topology resulting from the compass method presents again extreme curvature values (large red nodes) on the corners of the lateral edges. The configuration obtained with the variational method owns lower maximum and mean curvature values.

lateral sides between the grid topologies obtained with both methods. The grid configuration given by the compass method presents extreme curvature values on the corners of the lateral edges and on the gridshell crown. The variational method provided a grid configuration with lower curvature values on the top and higher on the bottom of the gridshell, the maximum profiles curvature could be reduced to 90% compared to the compass method.

On the case of the gridshell analogue to the Downland Museum, see Figure 4.7, differences between the grid topologies increase when approaching to the face sides. In both methods, higher curvature values are to be found on the crowns and lower on the valleys. By the grid resulting from the variational method, extreme curvature values are less concentrated as in the compass method configuration. The maximum profiles curvature could be minimized to 88%.

With the variational method, grid topologies with lower and more homogeneously distributed profiles curvatures than by the compass method could be obtained. A further optimization could be achieved by using another starting mesh with different edge angles, by tolerating a higher distance from the reference surface or by allowing variation on the segment lengths. In the following chapters the weighting factors of the it reference surface and it segments length energies have been minimized in order to achieve a higher reduction of the grid curvature.

4.3.3 Further optimization by allowing more distance to reference surface

The anticlastic, synclastic and Downland-like gridshells have been further optimized by reducing the weighting factor of the *reference surface* energy and with it allowing a spacing between grid and target surface up to 0.6 m. Depending on the curvature distribution, the grids have been deformed above or below the reference surface.

On the case of the anticlastic gridshell, the corners of the lateral sides deform outside reducing here the maximum curvature values up to 45% and obtaining a more homogeneous distribution on the centre of the gridshell. The mean curvature of the profiles could be reduced up to 51%. By the synclastic gridshell, the lateral edges tend to distort outwards in the middle and the crown of the grid slightly upwards. The maximum and mean profiles curvatures were reduced up to 79% and 78%, respectively. By the gridshell with anticlastic and synclastic curvatures, the crowns deform inwards and the valleys outwards getting a flatter surface. The maximum and mean curvatures of the profiles could be reduced here up to 76% and 64%, respectively, see

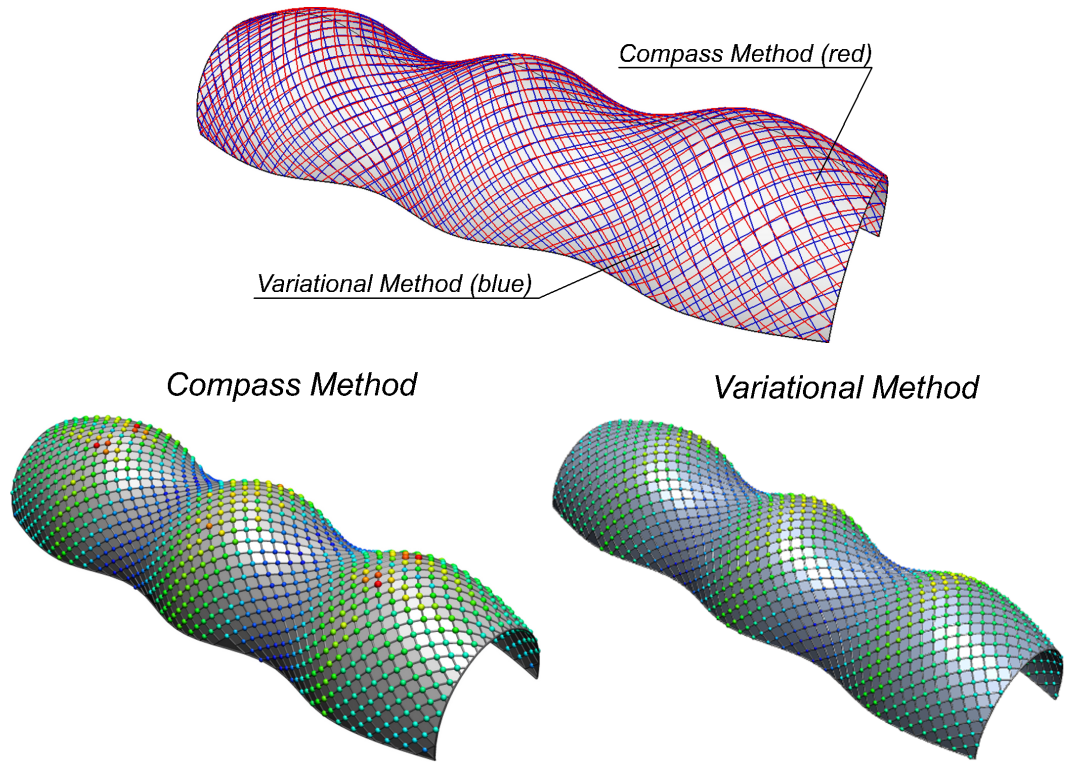


Figure 4.7: Comparison between grid topologies for the Downland-like gridshell. In both methods, higher curvature values are located on the crowns (red nodes) and lower on the valleys (blue nodes). With the variational method, a higher distribution of the extreme profiles curvatures could be obtained.

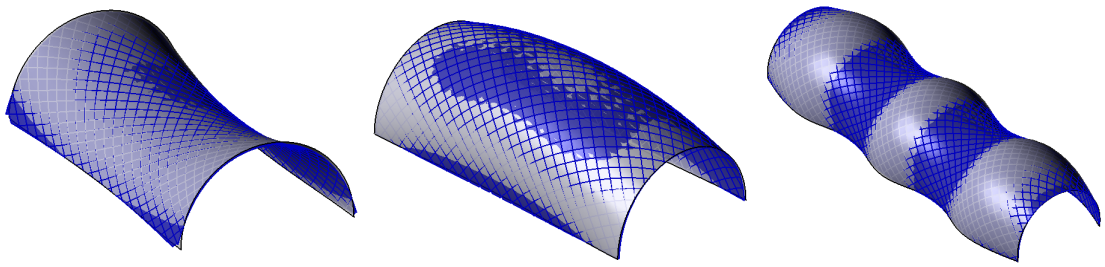


Figure 4.8: Deformation of the grids by allowing more distance from the reference surface

Figure 4.8).

The diagram shown in Figure 4.9 outlines the optimization results achieved with the variational method in comparison to the compass method. The maximum and mean curvatures are illustrated for the three gridshells. The color of the bars represents the distance from the reference surface. Generally, a higher reduction of the mean curvature is achieved, as the energy E_{cur} to be minimized corresponds to the sum of all curvature values.

4.4 Case studies irregular gridshells

4.4.1 Further optimization allowing variation on the segment lengths

A further optimization can also be achieved by allowing the distance between grid nodes to vary, reducing thus the curvature on the grid locally and globally. A spherical calotte of 15m diameter and 10m height has been firstly optimized, with constant segment length (regular gridshell) and a starting angle between profiles of 67.5°, and afterwards by letting the segment lengths progressively differ (irregular gridshell).

On the case of the calotte with regular mesh, due to the characteristic polar singularities of the spheres, strong local concentrations of curvature can be observed. By letting the mesh size vary, the segment lengths become shorter on the poles and the also typical alignment in S of the grid tends to disappear, see Figure 4.10.

4.4.2 Practical application: The Flying Dome

Elastic gridshells offer great advantages on temporary structures since the initially straight and afterwards elastically shaped profiles composing the grid allow rapid and cost-efficient production, transport and erection processes. For a temporary hanging 3D Projection Hemisphere (Flying Dome) of 10m diameter, an irregular elastic gridshell has been designed, see Figure 4.11. The project is a cooperation between the UdK Berlin, the TU Berlin, the Fraunhofer Institut FIRST and industrial partners and is planned to be built in Berlin in October 2012.

The structure consists in a hybrid construction composed of an irregular elastic gridshell between a double-layer membrane stabilized by underpressure (vacuum) of 0.08mbar. The profiles of the gridshell are made of GFK and have a tubular section of 20mm diameter and 3mm thickness.A third layer of profiles assures the bracing of the grid and activates its shear-bearing capacity. A PVC-coated polyester fabric and a PVC projection foil have been planned for the outer and inner membranes, respectively. The extremities of the bent profiles are fixed on a steel box ring of 100x100x4mm. The hemisphere hangs from the roof through four cables of 6mm diameter and is horizontally stabilized by other four cables of 3mm diameter. The total weight of the structure is approximately 1.3 tons.

An irregular mesh was chosen in order to minimize the profiles curvature (the maximum curvature could be reduced up to 80% compared to the regular gridshell) and to obtain a more interesting arrangement of the grid pattern. Physical modeling was used to analyze the visual effects, see Figure 4.12.

Contrary to regular grids, grids with irregular meshes cannot be completely deployed and can only be partially pre-assembled. The two profile layers will be joined and bent in a progressive process in order not to exceed the maximum allowable curvature during the erection of the

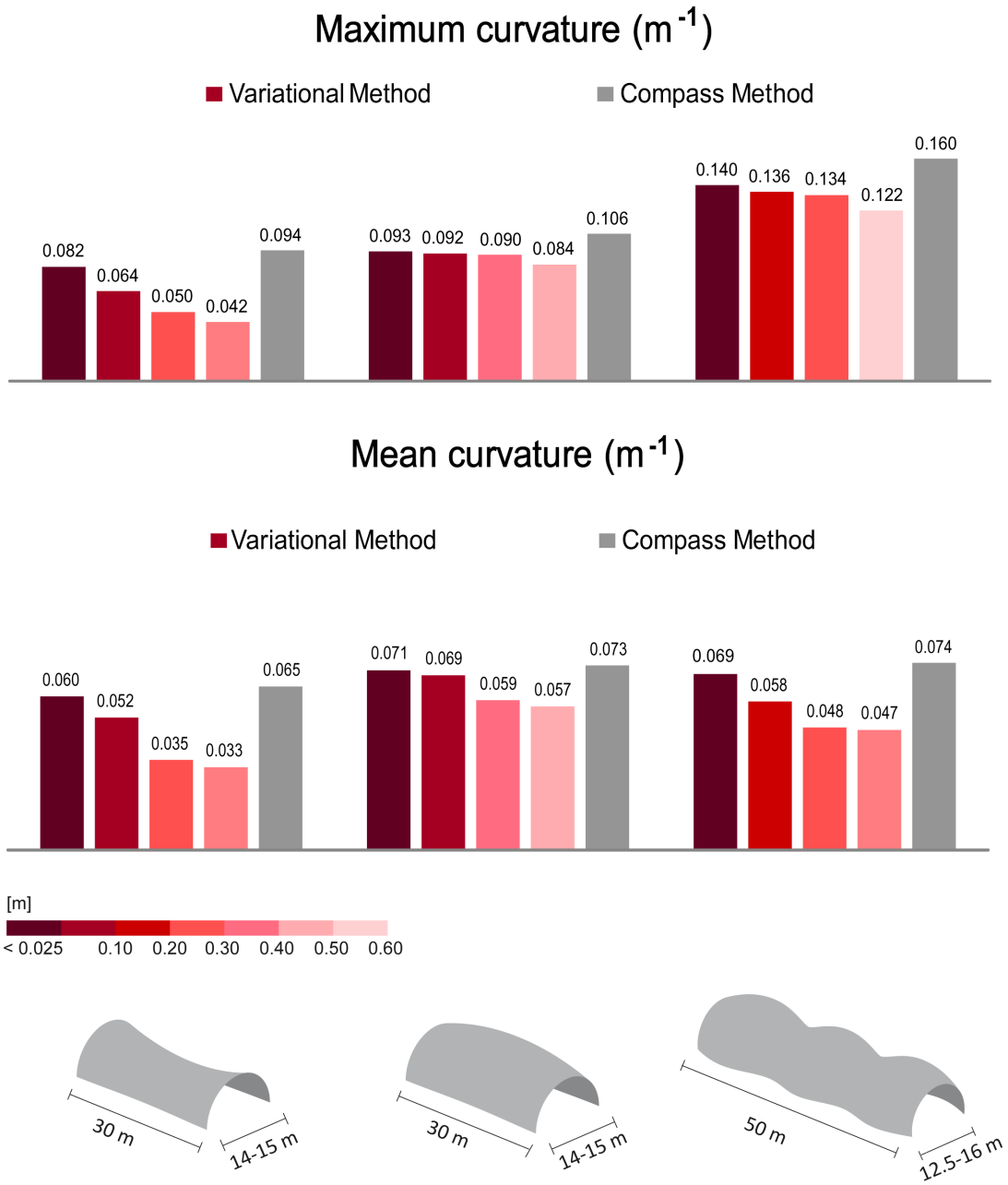


Figure 4.9: Comparison of the maximum and mean profile curvatures of the anticlastic, synclastic and Downland-like grid topologies resulting from the variational and compass methods

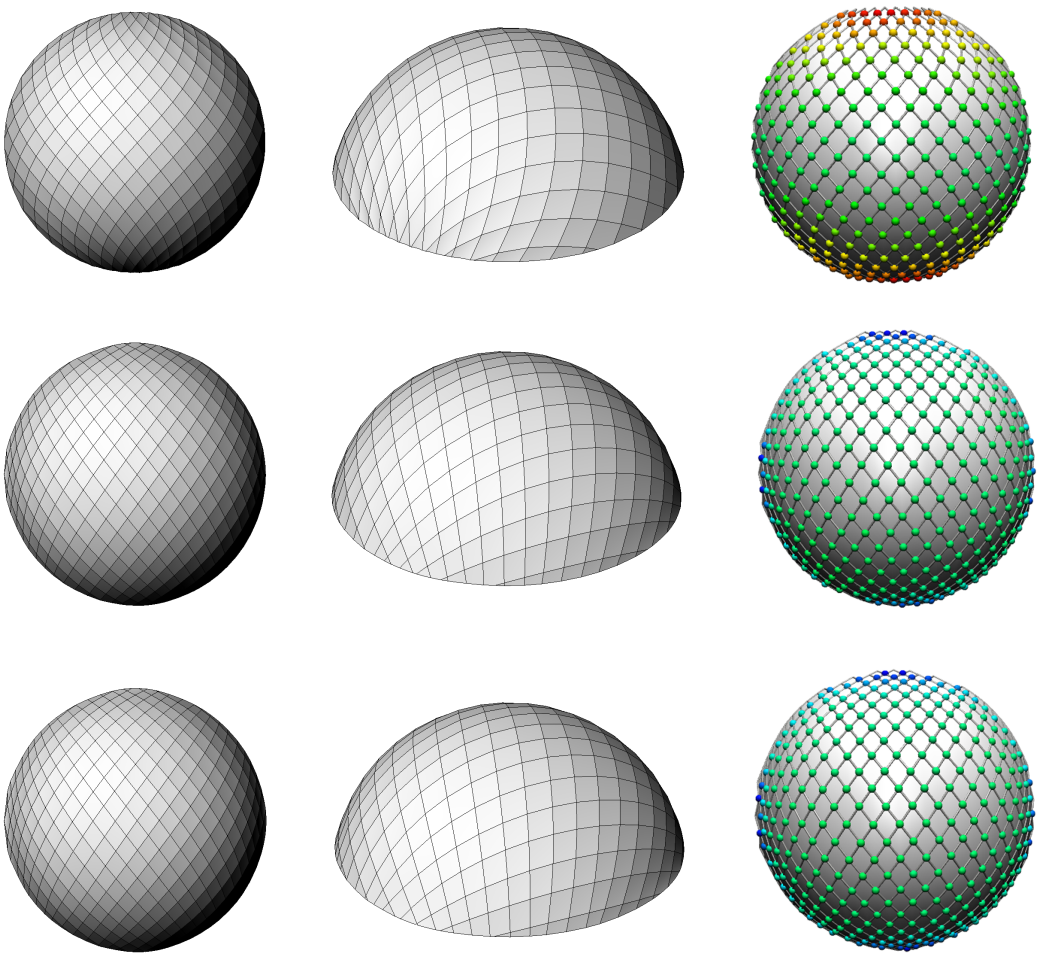


Figure 4.10: optimization of a spherical calotte with regular and irregular meshes. With a maximum variation of the segment length of 0.40m (middle) and 0.50m (bottom), the mean curvature of the grid could be reduced up to 83% and 77% respectively.

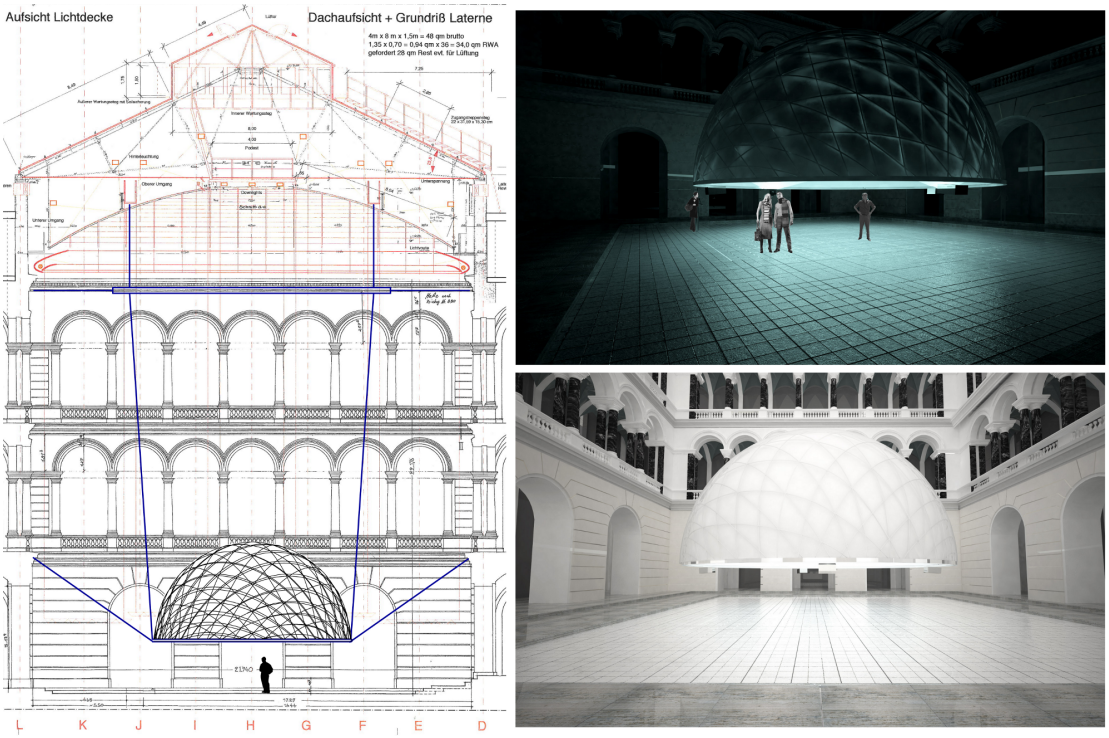


Figure 4.11: Front view and renderings of the Flying Dome

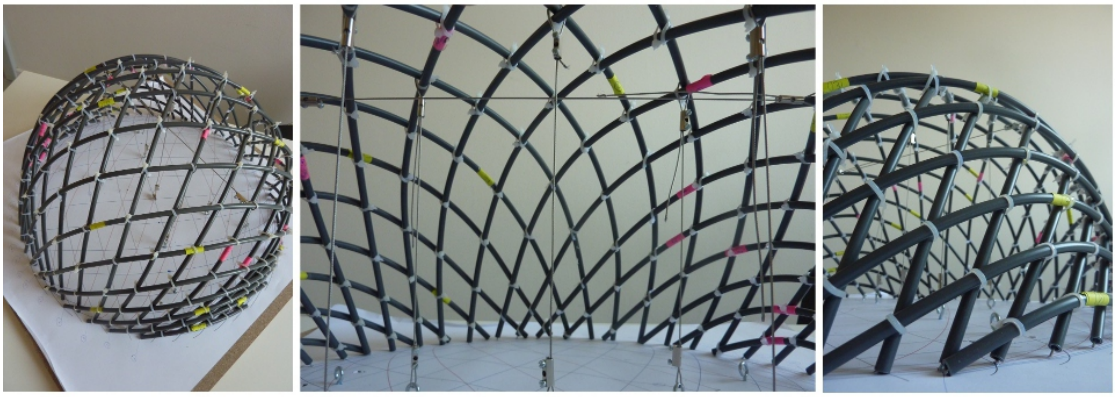


Figure 4.12: Physical modeling of the Flying Dome to analyze the visual effects of the irregular mesh

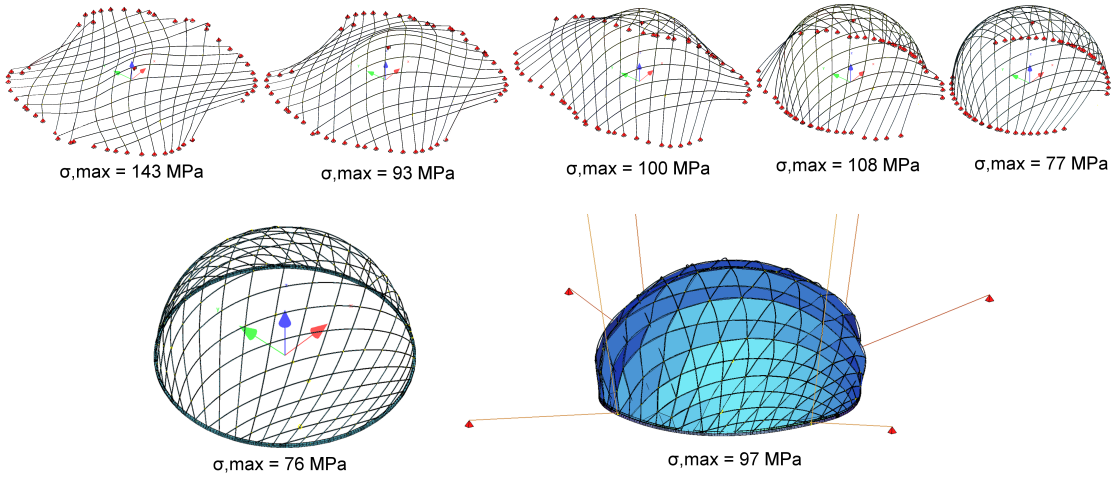


Figure 4.13: Simulation of the erection process and loading of the Flying Dome and analysis of the maximum stresses by means of FEA

structure. By means of finite element analysis, the assembling process of the hemisphere has been simulated and the maximum stresses during the shaping of the grid and by underpressure loading have been controlled, see Figure 4.13.

4.5 Conclusion

A non-linear variational method for optimizing topologies of regular and irregular elastic gridshells is presented in this paper. The design parameters *mesh size*, *reference surface* and *profiles curvature* are defined as penalizing energies with corresponding weighting factors. The resulting grid configuration is calculated by a non-linear algorithm by minimizing the linear combination of these three energies. The advantage of the variational method compared to other existing ones is that spacing between grid and target surface can be allowed and displacements of the grid nodes are possible in all directions, thus a further optimization of the grid topology can be achieved. Moreover, by defining different priorities between the design parameters through the energy weighting factors, diverse grid configurations can be generated and the resulting gridshell design can be adapted to specific structural and architectural requirements.

Part III

Implementation

Chapter 5

Introduction

In all chapters of Part III, words printed in SmallCapitals are names of software packages; words printed in `TeleType` are names of Java classes, methods, or fields.

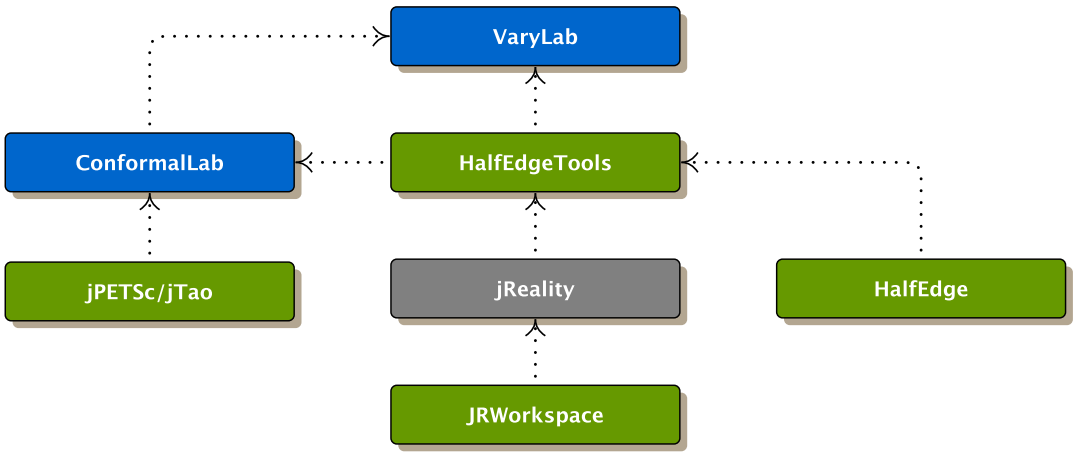


Figure 5.1: Software architecture and dependencies of the DDG Framework. Jtem library packages (green), application packages (blue). JReality is used mainly for 3d-visualization (grey).

In the field of Discrete Differential Geometry (DDG) there is a special need for experiments with the help of computer software. Especially if the methods of DDG are applied to problems in computer graphics, geometry processing, or architecture, algorithms have to be implemented and convincing examples have to be presented. Additionally, a suitable visualization of the results has to be included in a state-of-the-art publication.

There is a growing knowledge of software development in the mathematical community. This is partly due to the curricula of universities, which started to include programming courses for undergraduate students with an emphasis on mathematical visualization [26, 21]. This knowledge enables students to extend their abilities of creating visualizations and mathematical software, where former generations of students solely used the visualization abilities of standard computer algebra packages like Mathematica or MatLab.

The audience of the following chapters is two-fold. On the one hand it is students creating visualizations of surfaces and developing algorithms. On the other hand it is researchers in the field of DDG who need a stable data structure and programming infrastructure for their work.

This part of the thesis is the description of a set of software packages written in the programming language Java. They are specifically designed for the creation of custom interactive software for experiments with algorithms and geometric objects treated within DDG. Its main components JRworkspace, HalfEdge, and HalfEdgeTools are part of the collection of mathematical software libraries Jtem [34]. They are currently being used for research projects within the geometry group as well as for teaching mathematical visualization courses at TU-Berlin [26]. Representing the core of VaryLab, they are being deployed to users from the field of architectural geometry via the online version of VaryLab [63].

Chapter 6 introduces the JRworkspace library of the Jtem project [34]. It is the foundation of any application created with the framework. It is also the user interface basis of Jreality, a mathematical visualization library that uses JRworkspace as plug-in and user interface tool [20]. Chapter 7 introduces the HalfEdge and HalfEdgeTools package. They implement a half-edge data structure and various user interface tools and algorithms for interaction and editing. In Chapter 8 we describe the software ConformalLab. This implements the algorithms described in Chapter 1. Chapter 9 introduces VaryLab, the software implementation of the methods described in Chapters 2, 3, and 4. The chapters about ConformalLab and VaryLab should enable the reader to use the software to reproduce the results presented in the respective chapters of this work.

Figure 5.1 visualizes the dependencies of the different software packages. Every package depends on the plug-in functionality implemented in JRworkspace. It is the basis of the Jreality plug-in system. HalfEdgeTools uses Jreality for visualization and is built on top of the Jtem project HalfEdge. ConformalLab and VaryLab use Jpetsc/Jtao to perform numerical optimization. Their algorithms are implemented as JRworkspace plug-ins.

The development of the described software is joint work with Thilo Rörig (HalfEdgeTools, VaryLab), the Jreality members [20], Hannes Sommer (Jpetsc/Jtao) [68], Ulrich Pinkall and Paul Peters (JRworkspace), and Boris Springborn (Halfedge).

Chapter 6

JRworkspace - Java API for modular applications

JRworkspace is part of the Jtem family of software projects [34]. It defines a simple API to create modular Java applications. This API consists of three basic classes (Listings 6.2, 6.3, and 6.4). The project contains a reference implementation that supports the creation of Java Swing applications using the JRworkspace API. This implementation is used in all applications described in this work.

6.1 Plug-ins and the controller

In a JRworkspace application a feature is implemented as a plug-in and the corresponding Java class extends the abstract class Plugin (Listing 6.2). The idea is that a plug-in can be installed by the controller calling its `install` method or uninstalled via the `uninstall` method. It can be thought of as a feature added to the program. In particular, there is no more than one instance of a plug-in class in a JRworkspace application.

A plug-in has a life-cycle during the runtime of the program that includes these basic steps:

instantiation	1	set default plug-in state
`restoreStates`	2	load plug-in state from `Controller`
`install`	3	calls `getPlugin` to obtain dependent plug-ins
-	4	program execution
`storeStates`	5	stores state values to the `Controller`
(`uninstall`	6	clean up)

Step 1 instantiates a plugin and initializes its default properties. In step 2 the controller calls the `restoreStates` method. Step 3 is the actual installation of the plug-in. During runtime of the application the plug-in can interact with a potential user interface created during installation or offer services to other plug-ins. Before program termination or before uninstall, the `storeStates` method is called. The plug-in is supposed to store its state values by calling the `storeProperty` method of the controller. Inter-plug-in-communication is done via the `getPlugin` method of the controller. A plug-in should call `getPlugin` from within the `install` method to obtain

the unique instance of a dependent plug-in. The `getPlugin` method always returns the same instance of a plug-in so its result can be stored by the `install` method for later reference, see for example Listing 6.1. Step 6 `uninstall` is only used with dynamic plug-ins that support this operation. An implementation of `Controller` may not support uninstallation of plug-ins.

We describe the basic API usage from a programmer's point of view by giving an example plug-in in Listing 6.1 and the source code of the three basic API classes `Plugin`, `Controller`, and `PluginInfo` in Listings 6.2, 6.3, and 6.4.

```
1  public class MyPlugin extends Plugin {
2      private DependentPlugin dependency = null;
3      private double doubleState = 0.0;

5      public void helloPlugin() {
6          String depName = dependency.getPluginInfo().name;
7          System.out.println("I am a plug-in. I depend on " + depName);
8      }
9      @Override
10     public void storeStates(Controller c) throws Exception {
11         c.storeProperty(MyPlugin.class, "doubleState", doubleState);
12     }
13     @Override
14     public void restoreStates(Controller c) throws Exception {
15         doubleState = c.getProperty(MyPlugin.class, "doubleState", 1.0);
16     }
17     @Override
18     public void install(Controller c) throws Exception {
19         dependency = c.getPlugin(DependentPlugin.class);
20     }
21 }
```

Listing 6.1 A simple plug-in class. It depends on a plug-in called `DependentPlugin` and has the property `doubleState`. It provides the method `helloPlugin()` that prints some message. In the `storeStates` method the value of `doubleState` is written to the controller. The class `MyPlugin` is used as context class. The name of this class is used as a namespace to avoid property name ambiguities. The value of `doubleState` is read from the controller in the `restoreStates` method using the same context class and property name as in `storeStates`. If there is no value with the given context and name, the default value `1.0` is returned by the `getProperty` method.

```
1  public abstract class Plugin {

3      public PluginInfo getPluginInfo() {
4          return PluginInfo.create(getClass());
5      }

7      public void install(Controller c) throws Exception{}
8      public void uninstall(Controller c) throws Exception {}
9      public void restoreStates(Controller c) throws Exception {}
10     public void storeStates(Controller c) throws Exception {}

12     @Override
13     public String toString() {
14         if (getPluginInfo().name == null) {
15             return "No Name";
16         } else {
17             return getPluginInfo().name;
18         }
19     }
```

```
21      @Override
22      public boolean equals(Object obj) {
23          if (obj == null) {
24              return false;
25          } else {
26              return getClass().equals(obj.getClass());
27          }
28      }
```

Listing 6.2 The `Plugin` base class (excerpt). Note that plug-ins are equal if their classes are. It is not supported to have multiple instances of the same plug-in class installed.

```
1  public interface Controller {

3      public <T extends Plugin> T getPlugin(Class<T> clazz);
4      public <T> List<T> getPlugins(Class<T> pClass);
5      public Object storeProperty(Class<?> context, String key, Object property);
6      public <T> T getProperty(Class<?> context, String key, T defaultValue);
7      public <T> T deleteProperty(Class<?> context, String key);
8      public boolean isActive(Plugin p);

10 }
```

Listing 6.3 The `Controller` interface. A plug-in can obtain other plug-in instances by calling `getPlugin`, which returns the unique instance of the given plug-in class. The semantics of the `getPlugins` methods is different. It returns all plug-ins that are already known to the controller so no new dependencies are created by calling `getPlugins`. Property handling is done via the `xxProperty` methods. Note that any `Object` can be used as property value. This requires the controller to use generic serialization to store data. It is strongly discouraged to use other classes than official java API classes as stored values, as deserialization may fail if the class geometry changes.

```
1  public class PluginInfo {

3      public String name = "unnamed";
4      public String vendorName = "unknown";
5      public String email = "unknown";
6      public Icon icon = null;
7      public URL documentationURL = null;
8      public boolean isDynamic = true;

10     public PluginInfo() {
11     }

13     public PluginInfo(String name) {
14         this.name = name;
15     }

17     public PluginInfo(String name, String vendor) {
18         this(name);
19         this.vendorName = vendor;
20     }

22     public static PluginInfo create(Class<?> pluginClass) {
23         PluginInfo pi;
24         if (pluginClass == null) {
25             pi = new PluginInfo();
```

```
26          } else {
27              pi = new PluginInfo(pluginClass.getSimpleName());
28          }
29          if (pluginClass != null &&  pluginClass.getPackage() != null) {
30              pi.vendorName = pluginClass.getPackage().getImplementationVendor();
31          }
32          return pi;
33      }

35  }
```

Listing 6.4 The plug-in meta data class (excerpt). Instances are returned by the `getPluginInfo` method of any plug-in. The value of the `name` field is a plaintext name that could be shown in a user interface, as well as the `vendorName` and `email` information. An optional `icon` and a `documentationURL` can be given. The flag `isDynamic` is evaluated by controller implementations that support deinstallation of plug-ins. A dynamic plug-in can be installed or uninstalled during application runtime. A non-dynamic plug-in must be installed at startup and remains installed until program termination. The static `create` method returns a default `PluginInfo` instance for the given plug-in class.

6.2 Reference implementation

This section describes a reference implementation of the JRworkspace plug-in API. It was started as a user interface framework for Jreality [20]. It implements the `Controller` interface in a class called `SimpleController`. This name is historic and did not change as the features evolved from simple into quite complex. `SimpleController` implements a Java Swing® framework for the creation of complex modular applications based on the JRworkspace API. It defines various plug-in flavors that define user interface features. The implementation does not support dynamic plug-ins.

In the remainder of this section we describe the basic and most interesting features of this implementation. For a complete API reference see the documentation on the Jtem website [34].

Perspective Flavor

A plug-in implementing the interface `PerspectiveFlavor` provides the base for a program's user interface. It implements the method `getCenterComponent` that returns an AWT `Component` that is placed in the main frame of the application. The main program window itself is created and managed by the controller. A reference implementation of this flavor is the `SideContainerPerspective`. It layouts its content using a `BorderLayout` and places slots in the north, south, east, and west of the main window. These slots can contain `ShrinkPanel`s that can be moved between slots by drag-and-drop. A `ShrinkPanel` behaves like a `JPanel` and has a title bar that resizes the panel when the user clicks with the mouse.

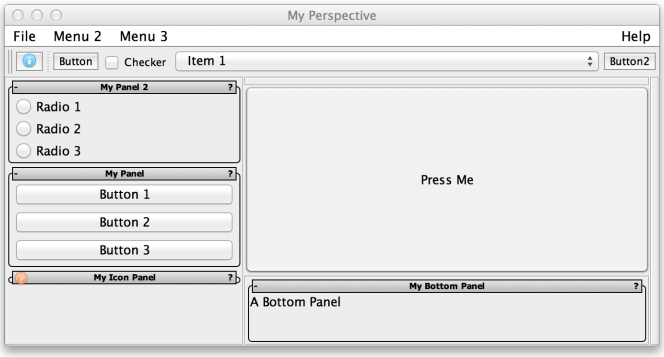


Figure 6.1: The `SideContainerPerspective` implementation uses slots to layout panels at the side of the main window. The left slot contains three `ShrinkPanels`, the top and right slots are empty. The menu bar and tool bar are created by respective plug-in flavors.

Menu Flavor

A plug-in implementing the `MenuFlavor` interface provides JAVA SWING® menu components that are placed at the top of the main window. A reference implementation of this flavor is the plug-in `MenuAggregator` that manages menu entries by contexts and menu paths. Its API provides four methods to add and remove menus, menu items, and separators. A typical method signature is

```
public void addMenu(Class<?> ctx, double priority, JMenu m, String... path)
```

where the plug-in stores the menu item with the given context class. This context is used to bulk-remove menus from a menu aggregator. Menus are sorted ascending by their priority. The menu item appears at the end of the given menu path. See Listing 6.5 and Figure 6.2.

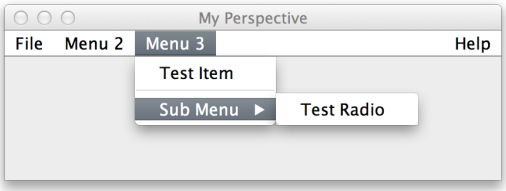


Figure 6.2: The menu created by Listing 6.5

```
2       @Override
3       public void install(Controller c) throws Exception {
4           super.install(c);
5           addMenu(MyMenuBar.class, 0.0, new JMenu("File"));
6           addMenu(MyMenuBar.class, 1.0, new JMenu("Menu 2"));
7           addMenuItem(MyMenuBar.class, 0.0, new QuitAction(), "File");
8           addMenuItem(MyMenuBar.class, 0.0, new JCheckBoxMenuItem("Test Checker"), "
                Menu 2");
9           addMenuItem(MyMenuBar.class, 0.0, new JRadioButtonMenuItem("Test Radio"), "
                Menu 3", "Sub Menu");
10          addMenuItem(MyMenuBar.class, 0.0, new JMenuItem("Test Item"), "Menu 3");
11          addMenuSeparator(MyMenuBar.class, 1.0, "Menu 3");
12      }
```

Listing 6.5 Usage of the `MenuFlavor` interface and the `MenuAggregator` implementation.

Tool Bar Flavor

Plug-ins implementing this flavor interface create a JAVA SWING® tool bar at the top of the main window. There can be more than one plug-in implementing this interface to create multiple tool bars. The API method signatures are similar to the signatures of the menu aggregator flavor. As a tool bar does not have a hierarchy, there is no path parameter. The signature of an API method is, e.g.,

```
public void addAction(Class<?> context, double priority, Action a).
```

The tool bar aggregator implementation can handle `Actions`, `Components`, and tool bar separators. See Listing 6.6 and Figure 6.3.

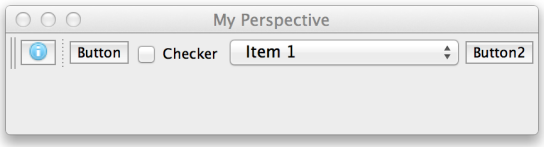


Figure 6.3: The tool bar created by Listing 6.6. Elements are sorted according to their priority.

```
2       @Override
3       public void install(Controller c) throws Exception {
4           addAction(MyToolBar.class, 0.0, new MyAction());
5           addTool(MyToolBar.class, 2.0, new JButton("Button"));
6           addSeparator(MyToolBar.class, 1.0);
7           addTool(MyToolBar.class, 3.0, new JCheckBox("Checker"));
8           addTool(MyToolBar.class, 4.0, new JComboBox<Object>(testItems));
9           addTool(MyToolBar.class, 5.0, new JButton("Button2"));
10          super.install(c);
11      }
```

Listing 6.6 Usage of the `ToolFlavor` interface and the `ToolBarAggregator` implementation.

The API of `SimpleController`

A plug-in implementation is independent of the concrete implementation of the `Controller`. To create an application with the `SimpleController` we need to register plug-ins we want to use and then invoke the startup sequence. A typical `main` method is, e.g.,

```
1       public static void main(String[] args) throws Exception {
2           UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
3           SimpleController c = new SimpleController("My Application");
4           c.setManageLookAndFeel(false);
5           c.setPropertiesMode(StaticPropertiesFile);
6           c.setStaticPropertiesFile(new File("MyApp.xml"));
7           c.registerPlugin(MyPerspective.class);
8           c.registerPlugin(MyMenuBar.class);
9           c.registerPlugin(MyToolBar.class);
10          c.registerPlugin(MyShrinkPanel.class);
11          c.registerPlugin(MyShrinkPanel2.class);
12          c.registerPlugin(MyShrinkPanel3.class);
13          c.registerPlugin(MyShrinkPanel4.class);
14          c.startup();
15      }
```

Listing 6.7 `Main` method of a program created with the `SimpleController` implementation. The result is shown in Figure 6.1.

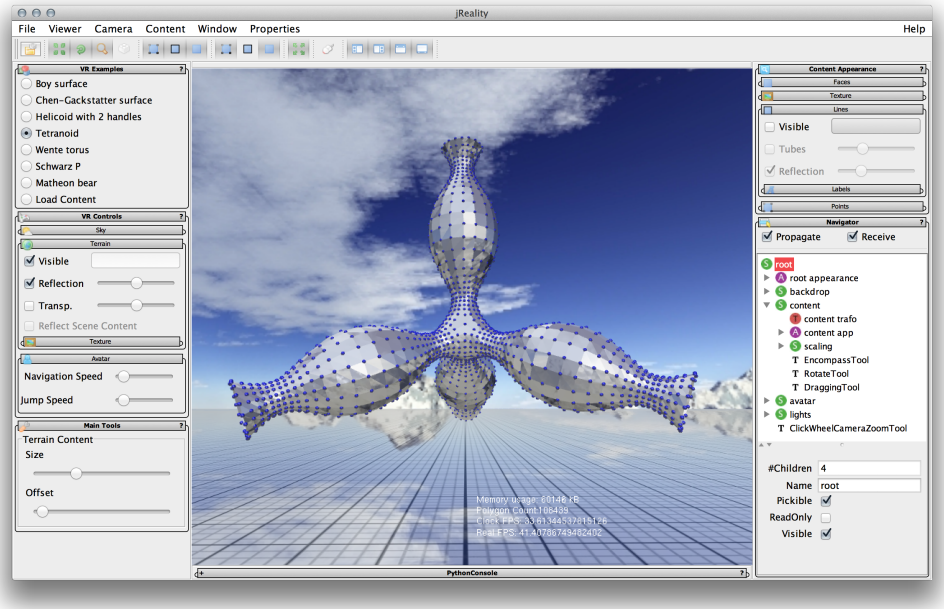


Figure 6.4: The JREALITY user interface. A `SideContainerPerspective` with `ShrinkPanels` is used. The tool bar and menu bar are created by the aggregators described in Section 6.2. This application uses a set of predefined user interface features and virtual reality components. In a custom application the developer usually registers only a subset of these features.

We set the look and feel to be the system look and feel, in this case the Mac OS style. Then we create a `SimpleController` and set the `magageLookAndFeel` property to `false`. Plug-in properties are saved to a file called `MyApp.xml`. There are two property modes defined by the `SimpleController`: `StaticPropertiesFile` and `UserPropertiesFile`. In static mode, there is only one file location. In user mode, the predefined location can be altered by the user of the program. This user decision is then stored as a JAVA preference. In this example we use the static properties.

6.3 JRWORKSPACE and JREALITY

JREALITY [20] is a scene graph and visualization library written in JAVA. The user interface of JREALITY is based on the JRWORKSPACE API and reference implementation. A custom JREALITY application can be based on the central `JRViewer` class. This class uses `SimpleController` to manage plug-in registration and start-up. The design of the `JRViewer` application makes intensive use of the plug-in concept. The architecture of the application is divided into separate modules, which in turn are realized as plug-ins. The core functionality is provided by plug-ins implementing the content, the scene, the view, and the tool system. Then there are plug-ins providing the tool bar and menu bar as well as the overall layout of shrink panels as described above.

Chapter 7

The JTEM libraries HALFEDGE and HALFEDGETOOLS

This chapter describes the JAVA implementation of a half-edge data structure and a set of tools combined in the package HALFEDGETOOLS. Both packages are part of the JTEM project [34]. This is joint work with Boris Springborn (HALFEDGE), Thilo Rörig, Felix Knöppel, and Kristoffer Josefsson (HALFEDGETOOLS), and other contributers to the JTEM project.

This implementation of a half-edge data structure was inspired by an implementation contained in the CGAL library [37]. This specific implementation, however, differs from existing libraries, as it uses the generic programming concept of JAVA to achieve compatibility and flexibility.

The code presented in this section is compatible with JAVA version 1.7 or later.

7.1 The HALFEDGE data structure

Half-edge data structures are used primarily to represent cell decompositions of oriented surfaces. We say "primarily" because half-edge data structures can be used to represent somewhat more general combinatorial structures, such as, for example, a checker board surface with white squares removed.

Here, surface means two-dimensional manifold, possibly with boundary; and a cell decomposition of a surface is a graph embedded in the surface such that the complement of the graph is (topologically) a disjoint union of open disks. The term map on a surface means the same. Thus, a cell decomposition decomposes a surface into vertices, edges, and faces.

Regular and strongly regular

A cell decomposition of a surface is called regular if it has no loops (edges with the same vertex on both ends) and if the boundary of a face contains an edge or vertex at most once. It is called strongly regular if two edges have at most one vertex in common, and if two faces have at most one edge or one vertex in common. A strongly regular cell decomposition is usually called a mesh.

This half-edge data structure implementation consists of different types of objects representing vertices, half-edges, and faces. The term half-edge can and should be thought of as synonymous

with oriented edge or directed edge.

Every half-edge object holds references to:

- its oppositely-oriented companion edge
- the next edge in the boundary of the face on its left-hand side
- the previous edge in the boundary of the face on its left-hand side
- the face on its left-hand side
- the vertex it points to.

The face and vertex objects hold back references to a half-edge referencing them. Finally, there is the class `de.jtem.halfedge.HalfEdgeDataStructure` representing a whole half-edge data structure. It acts as a container for (and sort of factory of) its vertices, half-edges, and faces.

Use of generics

Typically, one wants to equip vertices, edges, and faces with additional properties or functionality. For example, vertices may have coordinates associated with them, edges may have weights, and faces may have colors.

Our half-edge data structure facilitates this by using generic classes as abstract base classes for vertex, edge, and face types: The classes `de.jtem.halfedge.Vertex`, `de.jtem.halfedge.Edge`, `de.jtem.halfedge.Face` are all parameterized with the associated vertex, edge, and face types.

Example

To create a half-edge data structure with vertices that have 2D coordinates, proceed as follows.

- Step 1. Define appropriate subclasses of `de.jtem.halfedge.Vertex`, `de.jtem.halfedge.Edge`, and `de.jtem.halfedge.Face`, for example:

```
public class MyVertex extends Vertex<MyVertex, MyEdge, MyFace> {
    public Point2D p;
}
public class MyEdge extends Edge<MyVertex, MyEdge, MyFace> { }
public class MyFace extends Face<MyVertex, MyEdge, MyFace> { }
```

  Of course one might make the property `p` of `MyEdge` private and provide getter and setter methods, etc. Note that one always has to subclass `de.jtem.halfedge.Vertex`, `de.jtem.halfedge.Edge`, and `de.jtem.halfedge.Face`, even if there is no additional functionality or properties.
- Step 2. Instantiate a `de.jtem.halfedge.HalfEdgeDataStructure`:

```
HalfEdgeDataStructure<MyVertex, MyEdge, MyFace> heds = new HalfEdgeDataStructure
    <>(MyVertex.class, MyEdge.class, MyFace.class);
```

  The parameters of the constructor serve as run-time type tokens. Alternatively one can create a subclass of `de.jtem.halfedge.HalfEdgeDataStructure` and in turn create an instance of this:

```
public class MyHDS extends HalfEdgeDataStructure<MyVertex, MyEdge, MyFace> {
    public MyHDS() {
        super(MyVertex.class, MyEdge.class, MyFace.class);
    }
}
...
MyHDS mds = new MyHDS();
```

- Step 3. Instantiate vertices, edges, and faces using the `addNewVertex`, `addNewEdge`, and `addNewFace` methods, like this:

```
MyVertex v = heds.addNewVertex();
```

```
MyEdge e = heds.addNewEdge();
MyFace f = heds.addNewFace();
```

Linkage and incidence.

Node classes provide methods to create a valid data structure. Most of the methods are bidirectional linkage methods that affect the target and the caller. The corresponding methods are:

- `E.linkNextEdge` sets the `nextEdge` property of this edge as well as the `previousEdge` property of the given edge.
- `E.linkOppositeEdge` sets the `oppositeEdge` property of this edge and of the given edge accordingly.
- `E.linkPreviousEdge` sets `previousEdge` and `nextEdge` properties of this and the given edge.
- `E.setLeftFace` sets the `leftFace` property of this edge and the `boundaryEdge` property of the given face.
- `E.setTargetVertex` sets the `targetVertex` property of this edge and the `incomingEdge` property of the given vertex.

Node indices.

During algorithm development it is often necessary to access consistent indices on vertices, half-edges, or faces. We support this operation by the `index` field of nodes accessible via the `getIndex()` method of all node types. Nodes are required to have indices $0, \ldots, \#V$, $0, \ldots, \#E$, and $0, \ldots, \#F$, respectively. Note that $\#E$ is the number of half-edges in the data structure. The running time of a `getIndex` operation is $O(1)$ if no `remove` operation has invalidated the indices. Otherwise it is $O(n)$ where n is the number of nodes in the data structure, e.g., $n = \#V$.

Iterating and directly accessing nodes.

Nodes can be directly accessed via the corresponding `get` method. It takes the node index and returns the node with the given index. It is guaranteed that for a given $i \in \{0, \ldots, \#N\}$ it is `getNode(`i`).getIndex()`$= i$. Nodes in the data structure can be iterated over using one of the methods `getVertices`, `getEdges`, or `getFaces`. These return unmodifiable instances of `java.util.List` containing the respective node instances.

Undirected edges

A half-edge has the `positive` property. In a pair of opposite edges it is required that the boolean value of this property is different, i.e., `e.isPositive() ! = e.getOppositeEdge().isPositive()`. Using this property it is possible to iterate over undirected edges, i.e., the positive or the negative half-edges. The methods `getPositiveEdges` and `getNegativeEdges` of the data structure implement this.

Removing nodes from a data structure.

A node can be deleted from the data structure using the `removeVertex/Edge/Face` methods. The running time of a remove operation is always $O(1)$. Node indices however will get invalidated and reindexed on the next access of any index-related operation that in turn will then have running time $O(n)$.

Half-Edge utilities.

To work efficiently with half-edge data, a set of algorithms is implemented as static utility methods in the class `de.jtem.halfedge.util.HalfEdgeUtils`. It contains methods that help iterating over incoming edges at a vertex, or iterating over the boundary edges of a surface or a face, and many more.

7.2 Data, algorithms, and tools

A set of algorithms and tools is implemented in the JTEM project HALFEDGETOOLS [34].

Many algorithms in the library are purely combinatorial. This means there is no extra data involved during algorithm execution. Such an algorithm is thus generic by definition. The method signature could look like this:

```
public static <
    V extends Vertex<V, E, F>,
    E extends Edge<V, E, F>,
    F extends Face<V, E, F>,
    HDS extends HalfEdgeDataStructure<V, E, F>
> void triangulate(HDS hds){
    ...
}
```

This method works on any half-edge data structure that is either an instance of `de.jtem.halfedge.HalfEdgeDataStructure` or an instance of a sub-class. This method signature makes the algorithm code itself look very clean. For instance iterating over all vertices amounts to:

```
for (V v : hds.getVertices()) {
    E e = v.getIncomingEdge();
    ...
}
```

On the other hand, when designing a generic algorithm that needs certain data associated with nodes, we have basically two options. Option 1 requires the generic node classes to implement the required interfaces:

```
public static <
    V extends Vertex<V, E, F> & HasCoordinate3D,
    E extends Edge<V, E, F>,
    F extends Face<V, E, F>,
    HDS extends HalfEdgeDataStructure<V, E, F>
> void convexHull(HDS hds){
    ...
    Coordinate3D x = v.getCoordinate3D();
}
```

This forces the Vertex implementations that use this algorithm to implement an interface called `HasCoordinate3D`. It leads to explicit and clean code of the algorithm. A drawback of this is that an existing implementation that should use this algorithm has to be adapted to implement the potentially many interfaces required by the algorithm. This is not a feasible solution when is comes to a modular application where algorithms come as plug-ins without the chance to change the data structure.

AdapterSet and Adapters

The second option uses the concept of adapters and is implemented in the package `de.jtem.halfedgetools.adapter`. An adapter defines a map from nodes to a data type supported by the adapter. We first show how this concept works when designing algorithms and then describe the implementation of the required adapters.

In this next example we calculate the discrete Dirichlet energy of a double-valued function with double-valued weights on edges.

```
public static <
```

```
3       V extends Vertex<V, E, F>,
4       E extends Edge<V, E, F>,
5       F extends Face<V, E, F>,
6       HDS extends HalfEdgeDataStructure<V, E, F>
7   > double computeDirichlet(HDS hds, AdapterSet a){
8       double energy = 0.0;
9       for (E e : hds.getPositiveEdges()) {
10          V s = e.getStartVertex();
11          V t = e.getTargetVertex();
12          double fStart = a.get(FunctionValue.class, s, Double.class);
13          double fTarget = a.get(FunctionValue.class, t, Double.class);
14          double w = a.getDefault(Weight.class, e, 1.0);
15          double d = fStart - fTarget;
16          energy += w * d * d;
17      }
18      return energy;
```

Listing 7.1 Algorithm that uses data from the `AdapterSet`. The `get` method of the `AdapterSet` takes the data class type to find a matching adapter (Line 12 and 13). The corresponding `getDefault` method takes a default value that is returned if no matching adapter is found (Line 14). It uses `@FunctionValue` and `@Weight` annotated adapters to acquire the data needed for calculation of the output value.

This method requires the `AdapterSet` to contain adapters that provide `@FunctionValue` data on vertices (Line 12 and 13) and `@Weight` data on half-edges (Line 14).

The classes `FunctionValue` and `Weight` are runtime annotation classes, e.g.,

```
1   @Retention(RetentionPolicy.RUNTIME)
2   @Target(ElementType.TYPE)
3   public @interface FunctionValue {}
```

An adapter class annotated with this annotation serves as `@FunctionValue` data adapter when called for as in Line 11-13 of Listing 7.1. The meaning of adapter annotations is an agreement between the algorithm using it and the implementor of the adapter. It is best-practice to implement the adapters such that they output data that fits their annotation names.

There are three basic classes that could serve as the base class of an adapter:

- `Adapter` - The abstract base class of all adapters. Should never be subclassed directly.
- `AbstractAdapter` - An adapter class that knows about the supported data type and implements all getter and setter methods. Here only the needed methods can be overwritten. Node type checking is done manually and thus allows for the creation of generic adapters.
- `AbstractTypedAdapter` - If one knows which half-edge node classes the adapter is supposed to work with this is the adapter base class one should use. Node type checking and casting is done in the super class.

An adapter implementation using the `AbstractAdapter` is for instance:

```
1   @FunctionValue
2   public class MyAbstractAdapter extends AbstractAdapter<Double> {
3       private Map<Vertex<?, ?, ?>, Double> valueMap = null;

5       public MyAbstractAdapter() {
6           super(Double.class, true, false);
7       }

9       @Override
10      public <
```

```
11          N extends Node<?, ?, ?>
12      > boolean canAccept(Class<N> nodeClass) {
13          return Vertex.class.isAssignableFrom(nodeClass);
14      }

16      @Override
17      public <
18          V extends Vertex<V, E, F>,
19          E extends Edge<V, E, F>,
20          F extends Face<V, E, F>
21      > Double getV(V v, AdapterSet aSet) {
22          return valueMap.get(v);
23      }

25      @Override
26      public <
27          V extends Vertex<V, E, F>,
28          E extends Edge<V, E, F>,
29          F extends Face<V, E, F>
30      > void setV(V v, Double value, AdapterSet aSet) {
31          valueMap.put(v, value);
32      }
33  }
```

Listing 7.2 Adapter implementation using the `AbstractAdapter` as base class and a map as storage concept for `Double` values on generic vertices. It is annotated with a `@FunctionValue` annotation to serve as provider for data in the `AdapterSet` (Line 1). The super class `AbstractAdapter` is parameterized with the data type of the implementation; in this case `Double` (Line 2). The super class constructor is invoked with the class object of this type and flags that tell the adapter if get and/or set operations are permitted (Line 6). The method `canAccept` decides whether the adapter can work with the given node class; in this case the adapter can accept any `Vertex` object (Line 12). Vertex getter and setter methods are generic methods (Line 16 to 32).

When writing the adapter for concrete node classes we have a more concise description:

```
2   @FunctionValue
3   public class MyTypedAdapter extends AbstractTypedAdapter<VV, VE, VF, Double> {
4       public MyTypedAdapter() {
5           super(VV.class, null, null, Double.class, true, true);
6       }

8       @Override
9       public Double getVertexValue(VV v, AdapterSet aSet) {
10          return v.value;
11      }

13      @Override
14      public void setVertexValue(VV v, Double value, AdapterSet aSet) {
15          v.value = value;
16      }
```

```
17 }
```

Listing 7.3 An adapter using `AbstractTypedAdapter` as base class. Also annotated with the `@FunctionValue` annotation. This super class is parameterized with a set of node class implementations and the adapter data type. Here the super constructor takes the node class objects or `null` if a node type is not supported, the data type class object, and the getter/setter flags. The vertex getter and setter methods are not generic and the corresponding casting is done in the super class. Adapters working on edges or faces implement the corresponding `get/setEdgeValue` or `get/setFaceValue` methods.

Using this concept of typed adapters the usage of an algorithm amounts to the implementation of required data adapters and the creation of a suitable `AdapterSet`.

```
2 public static double calculate() {
3     VHDS hds = new VHDS();
4     HalfEdgeUtils.addDodecahedron(hds);
5     AdapterSet a = new AdapterSet();
6     a.add(new MyTypedAdapter());
7     return computeDirichlet(hds, a);
8 }
```

Listing 7.4 Usage example of the algorithm presented in Listing 7.1. In this example an empty data structure is created, filled with dodecahedron combinatorics, and then processed by the algorithm. The `AdapterSet` contains the `@FunctionValue` annotated adapter to provide `Double` values on vertices. In this example a `@Weight` adapter is not needed, since the `computeDirichlet` algorithm uses the `getDefault` method to obtain weight data for edges.

Generic adapters

There is a set of generic adapters implemented in the package `de.jtem.halfedgetools.adapter.generic`. Here generic means that the adapter does not provide the data itself. Instead it asks for other adapters to provide data and converts, merges, or recalculates its output based on the data acquired from other adapter implementations.

The most important example of pre-defined generic adapters are position and texture position adapters. These adapters come in three variants, i.e., `@Position2D`, `@Position3D`, and `@Position4D`. The implementations, e.g, `de.jtem.halfedgetools.adapter.generic.Position3dAdapter`, obtain position data from the `AdapterSet` using an adapter annotated with `@Position` and convert the resulting data arrays if their lengths is not equal to 3. As a result an algorithm using generic adapters is free of data conversion code.

Error handling

If a data adapter that is used by an algorithm is not found, the `AdapterSet` throws exceptions stating the name of the required annotation and the node type that is expected.

```
Exception in thread "main" de.jtem.halfedgetools.adapter.AdapterException: Adapter
     "FunctionValue" for node VV not found
    at de.jtem.halfedgetools.adapter.AdapterSet.get(Unknown Source)
    at de.sechel.thesis.MyUtility.computeDirichlet(MyUtility.java:25)
    at de.sechel.thesis.MyUtility.calculate(MyUtility.java:39)
    at de.sechel.thesis.MyUtility.main(MyUtility.java:44)
```

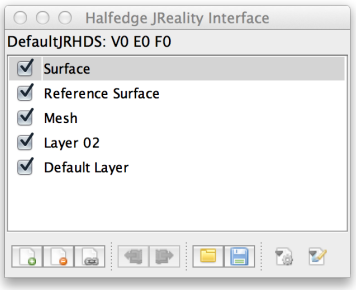


Figure 7.1: The user interface created by the `HalfedgeInterface` plug-in. It manages layers that contain different instances of half-edge data structures and the corresponding visualizations. Layers can be merged, `OBJ` files can be exported and imported, and visualization options can be adjusted.

7.3 HalfEdgeTools and JReality

The HalfEdgeTools package contains utility classes for the visualization of half-edge data with JReality [20]. It is part of the Jtem project [34]. See also Section 6.3 for the interaction between JReality and JRworkspace.

Half-edge Interface

The plug-in `de.jtem.halfedgetools.plugin.HalfedgeInterface` plays a central role. This plug-in works as a converter between the half-edge data structure and the `IndexedFaceSet` data structure used internally by JReality. It creates a user interface that contains GUI elements for layer management, import, export, undo, redo, and more features, see Figure 7.1.

The API of the `HalfedgeInterface` supports `set` and `get` methods that convert to and from JReality. During conversion, data is read from an `AdapterSet` that is managed by the half-edge interface. The plug-in supports various data on node types. We give a list of annotation types and their purposes. All conversion adapters work with `double[]` or `double` data types. All supported annotation types are located in the package `de.jtem.halfedgetools.adapter.type`.

- **@Position** - Positions of vertices can have lengths 2, 3, or 4.
- **@Color** - Colors either on vertices, edges, or faces.
- **@Normal** - Normals usually for vertices or faces.
- **@TexturePosition** - Texture coordinates of length 2, 3, or 4.
- **@Label** - Text annotations that appear next to the node.
- **@Radius** - Radii of vertex sphere representations or edge cylinders when rendered as spheres or tubes.
- **@Size** - Size in pixels of vertex points or edge lines when rendered as points or lines.

Internally the conversion is done using the classes from the package `de.jtem.halfedgetools.jreality`. Conversion from and to a JReality `IndexedFaceSet` is implemented in `ConverterHeds2JR` and `ConverterJR2Heds`.

Visualization Interface

A second important plug-in is the `VisualizationInterface`. It defines a plug-in API and various implementations for data visualization with the half-edge data structure.

Every `Adapter` that is managed by the `HalfedgeInterface` is available as data source. In

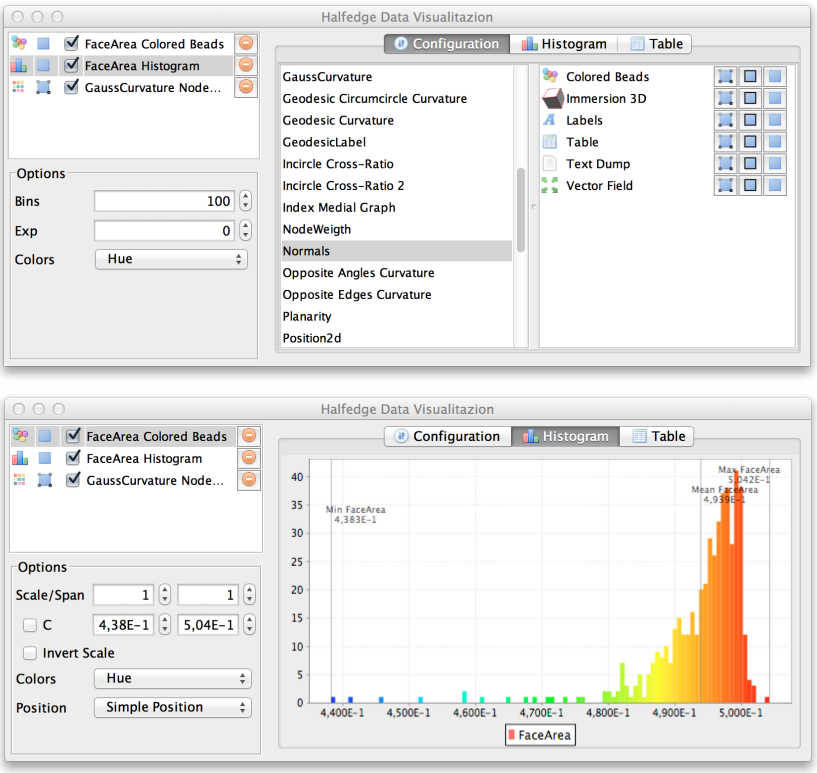


Figure 7.2: The half-edge visualization user interface. Selection of a visualization plug-in for a data adapter (top). A histogram view for scalar data on half-edge nodes (bottom).

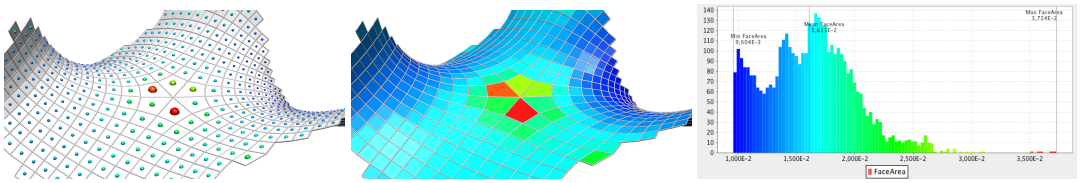


Figure 7.3: Different visualization plug-ins for scalar data on faces. Colored beads (left), node colors, and histogram view.

addition to this, every plug-in extending `de.jtem.halfedgetools.plugin.data.DataSourceProvider` is asked for data adapters. A list of these data sources is displayed in the user interface, see Figure 7.2 (top). A visualization of the selected data source is created by a corresponding `DataVisualizer` plug-in. For scalar data on vertices, edges, or faces we provide the colored beads or colored nodes visualizer, see Figure 7.3. There is a histogram that displays a density plot for scalar data on nodes, Figure 7.3 (right) and Figure 7.2 (bottom). For vector-valued data there is a visualizer that creates arrows starting at half-edge nodes.

Chapter 8

ConformalLab - Conformal maps and uniformization

This chapter enables the reader to reproduce examples contained in this work that involve discrete uniformization of Riemann surfaces, see Chapter 1, and applications of conformal mappings of Part II. The author has written Java software, ConformalLab, to calculate the corresponding data. The software and the data is included on the accompanying CD. Its structure is described in the Appendix.

The software ConformalLab has been developed in Project A01 "Discrete Riemann Surfaces" of the SFB Transregio 109 - Discretization in Geometry and Dynamics [65]. It is designed to be used via a graphical user interface, see Figure 8.1. To store resulting data, an XML format is used. We describe this format in Section 8.1. The graphical user interface is described in Section 8.2. ConformalLab uses JRworkspace, see Chapter 6, to implement its user interface. This allows the user interface to be divided into panels which serve separate purposes. ConformalLab uses HalfEdge and HalfEdgeTools, see Chapter 7, to work with discrete surfaces.

8.1 XML data format

To store and process data, ConformalLab uses an XML data format. All examples presented in Chapter 1 are stored in this format. All XML data is contained in the XML namespace `http://www.varylab.com/conformallab/types`.

Schottky data

A Riemann surface can be given by Schottky data, see Section 1.8.2. An example is shown in Listing 8.1. `SchottkyData` can include one or more `SchottkyGenerators`. A `SchottkyGenerator` defines fix points A and B, the complex number μ, and a `Circle`. This circle is required to contain A and must not contain B. If there is more that one generator, then the circles and their images must not intersect.

```
<SchottkyData name="Schottky">
    <SchottkyGenerator>
        <A re="-1.0" im="0.0"/>
        <B re="1.0" im="0.0"/>
        <Mu re="0.25" im="0.0"/>
```

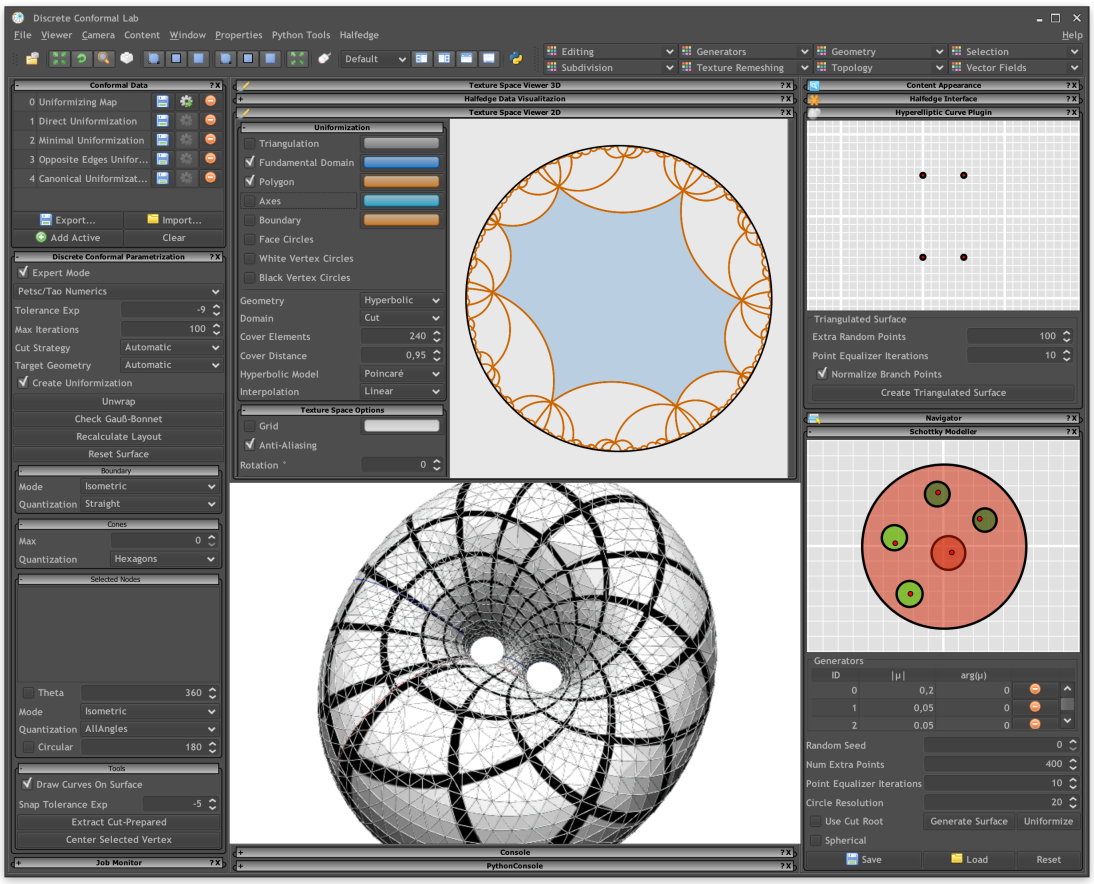


Figure 8.1: The user interface window of ConformalLab.

```
            <Circle radius="1.3">
                <Center re="-1.6" im="0.0"/>
            </Circle>
        </SchottkyGenerator>
    </SchottkyData>
```

Listing 8.1 A torus given by Schottky data

Hyperelliptic data

`HyperEllipticAlgebraicCurves` as used in the examples of Section 1.7.2 and 1.8.3 are given by the location of their branch points in the complex plane. All points must be distinct. Listing 8.2 shows the XML representation of an elliptic algebraic curve defining a Riemann surface of genus 1. The point at infinity cannot be specified and is implicitly given if an odd number of branch points is listed.

```
<HyperEllipticAlgebraicCurve name="Curve g1">
    <BranchPoint re="-0.5" im="-1.0"/>
    <BranchPoint re="0.5" im="-1.0"/>
    <BranchPoint re="1.0" im="0.0"/>
    <BranchPoint re="-1.0" im="0.0"/>
</HyperEllipticAlgebraicCurve>
```

Listing 8.2 A torus given as hyperelliptic data

Fuchsian data

A Riemann surface given by Fuchsian data (`UniformizationData`) is represented by generators of the corresponding Fuchsian group and their inverse elements as elements of $PSL(2,\mathbb{R})$ (`UniformizingGroup` and `IsometryPSL2R` XML nodes). In addition to the group a fundamental polygon is defined by its vertices and edges. In Listing 8.3 the uniformizing group of euclidean motions defines the structure of a flat torus. The `FundamentalPolygon` is a parallelogram. There is only one `FundamentalVertex` since all vertices of the parallelogram are identified. There are four `FundamentalEdges` each of which is identified with its opposite edge of the parallelogram. Each `FundamentalEdge` defines a `StartPosition` in $\mathbb{C}$ that is interpreted as an element of $\mathbb{R}P^2$. The polygon is positively oriented.

```
<UniformizationData name="Direct Uniformization">
    <UniformizingGroup>
        <IsometryPSL2R
            m11="1.0" m12="0.0" m13="1.0"
            m21="0.0" m22="1.0" m23="0.0"
            m31="0.0" m32="0.0" m33="1.0"
        />
        <IsometryPSL2R
            m11="1.0" m12="0.0" m13="0.0"
            m21="0.0" m22="1.0" m23="1.0"
            m31="0.0" m32="0.0" m33="1.0"
        />
        <IsometryPSL2R
            m11="1.0" m12="0.0" m13="-1.0"
            m21="0.0" m22="1.0" m23="0.0"
            m31="0.0" m32="0.0" m33="1.0"
        />
        <IsometryPSL2R
            m11="1.0" m12="0.0" m13="0.0"
            m21="0.0" m22="1.0" m23="-1.0"
            m31="0.0" m32="0.0" m33="1.0"
        />
```

```
    </UniformizingGroup>
    <FundamentalPolygon>
        <FundamentalVertex index="0"/>
        <FundamentalEdge index="0" nextEdge="1" previousEdge="3" identifiedEdge="2"
             startVertex="0">
            <StartPosition re="0.0" im="0.0"/>
        </FundamentalEdge>
        <FundamentalEdge index="1" nextEdge="2" previousEdge="0" identifiedEdge="3"
             startVertex="0">
            <StartPosition re="1.0" im="0.0"/>
        </FundamentalEdge>
        <FundamentalEdge index="2" nextEdge="3" previousEdge="1" identifiedEdge="0"
             startVertex="0">
            <StartPosition re="1.0" im="1.0"/>
        </FundamentalEdge>
        <FundamentalEdge index="3" nextEdge="0" previousEdge="2" identifiedEdge="1"
             startVertex="0">
            <StartPosition re="0.0" im="1.0"/>
        </FundamentalEdge>
    </FundamentalPolygon>
</UniformizationData>
```

Listing 8.3 A torus given by its Fuchsian uniformizing group and a corresponding fundamental polygon. The elements of the group are either euclidean motions or hyperbolic motions given as elements of $PSL(2, \mathbb{R})$.

Discrete data

To specify discrete data we use the concept of a half-edge data structure, see also Chapter 7. Listing 8.4 shows the structure of a `HalfedgeEmbedding`. `Vertices` are assigned coordinates in $\mathbb{R}P^3$ and their index. `Halfedge` nodes specify indices of incident faces, edges, and vertices as defined in Section 7.1. In addition to the data needed to specify the combinatorics of the half-edge data structure, one can specify identifications of vertices and edges. An `Identification` node contains all vertices that are identified. For example, consider a parallelogram fundamental domain of a flat torus. In the parallelogram all four vertices are identified to form the topological torus. Also the edges of the parallelogram are identified. An `EdgeIdentification` node defines the four half-edges that belong to the pair of half-edges on the torus.

```
<HalfedgeEmbedding name="Uniformizing Map_domain">
    <Vertex x="0.0" y="0.0" z="0.0" w="1.0" index="0"/>
    <Vertex x="1.0" y="0.0" z="0.0" w="1.0" index="1"/>
    <Vertex x="1.0" y="1.0" z="0.0" w="1.0" index="2"/>
    <Vertex x="0.0" y="1.0" z="0.0" w="1.0" index="3"/>
    <Identification>
        <Vertex>0</Vertex>
        <Vertex>1</Vertex>
        <Vertex>2</Vertex>
        <Vertex>3</Vertex>
    </Identification>
    <Halfedge left="0" target="1" next="1" opposite="4" index="0"/>
    <Halfedge left="0" target="2" next="2" opposite="5" index="1"/>
    <Halfedge left="0" target="3" next="3" opposite="6" index="2"/>
    <Halfedge left="0" target="0" next="0" opposite="7" index="3"/>
    <Halfedge left="-1" target="0" next="7" opposite="0" index="4"/>
    <Halfedge left="-1" target="1" next="4" opposite="1" index="5"/>
    <Halfedge left="-1" target="2" next="5" opposite="2" index="6"/>
    <Halfedge left="-1" target="3" next="6" opposite="3" index="7"/>
    <EdgeIdentification edge1="0" edge2="4" edge3="2" edge4="6"/>
    <EdgeIdentification edge1="1" edge2="5" edge3="3" edge4="7"/>
```

```
        <Face index="0"/>
    </HalfedgeEmbedding>
```

Listing 8.4 A torus given as `HalfedegeEmbedding` with identified edge pairs and vertices.

The `HalfedgeEmbedding` data type can be used to define discrete maps between discrete embeddings. A discrete map consists of a domain `HalfedgeEmbedding` and the corresponding image. Both nodes must define the same number of vertices. The map is defined via vertex indices. A vertex i from the domain is mapped to the vertex with index i in the image data structure, see Listing 8.5.

```
<HalfedgeMap name="Uniformizing Map">
    <Domain name="Uniformizing Map_domain">
        ...
    </Domain>
    <Image name="Uniformizing Map_image">
        ...
    </Image>
</HalfedgeMap>
```

Listing 8.5 A discrete map is given by a pair of `HalfedgeEmbeddings`, the `Domain` and `Image` of the map. Both are of type `HalfedgeEmbedding`.

A way of defining a discrete Riemann surface is to specify its discrete metric. A `DiscreteMetric` defines non-oriented `MetricEdges` and their lengths. `MetricTriangles` are glues along those edges. All triangles are oriented by the order of edges defined by attributes `edge1`, `edge2`, and `edge3` respectively. By that we can only encode oriented discrete Riemann surfaces using a `DiscreteMetric`. A simple example featuring a Wente torus is shows in Listing 8.6.

```
<DiscreteMetric name="Wente Torus">
    <MetricEdge length="1.0" index="0"/>
    <MetricEdge length="1.0" index="1"/>
    <MetricEdge length="1.0" index="2"/>
    <MetricTriangle edge1="0" edge2="1" edge3="2"/>
    <MetricTriangle edge1="0" edge2="1" edge3="2"/>
</DiscreteMetric>
```

Listing 8.6 A Wente torus given by a discrete metric. Vertices are given implicitly by following the order of triangle glueing.

Data lists

Each of the data nodes as described above can appear in an element of type `ConformalDataList`. A typical XML file representing the data of an example of Chapter 1 contains input data, e.g., `SchottkyData` and `DiscreteMetric`, as well as the resulting output, i.e., the surface and uniformization data represented as `HalfedgeEmbedding` and `UniformizationData`.

```
<ConformalDataList>
    <SchottkyData name="Schottky Data g1">
        ...
    </SchottkyData>
    <DiscreteMetric name="Input Schottky Data Metric">
        ...
    </DiscreteMetric>
    <HalfedgeEmbedding name="Output Euclidean Embedding">
        ...
    </HalfedgeEmbedding>
    <HalfedgeMap name="Uniformizing Map">
```

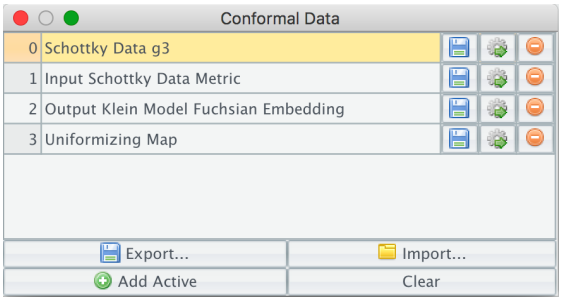


Figure 8.2: Data import and export interface works with XML data files as described in Section 8.1.

```
        <Domain name="Uniformizing Map_domain">
            ...
        </Domain>
        <Image name="Uniformizing Map_image">
            ...
        </Image>
    </HalfedgeMap>
    <UniformizationData name="Direct Uniformization">
        ...
    </UniformizationData>
    <UniformizationData name="Minimal Uniformization">
        ...
    </UniformizationData>
</ConformalDataList>
```

Listing 8.7 A list of data XML nodes as the result of an algorithm calculating the Fuchsian uniformization of a genus 1 Riemann surface given by Schottky data.

8.2 Uniformization and conformal mappings

The user interface of ConformalLab is divided into panels that serve separate purposes in the workflow of performing experiments with discrete Riemann surfaces and conformal mappings.

Data import and export

The data import and export panel, see Figure 8.2, provides functions to import and export XML data as described in Section 8.1. Data files can be loaded into memory via the Import button. A table lists the entries of the loaded file. Loadable are `ConformalDataList` as well as single data instances. The entries of a `ConformalDataList` are listed in the table of the panel.

Each of the rows contains buttons to save the data to disk (blue disk), load it into the program (gear with green arrow), or delete it from the list (red circle). The function of the load button depends on the data type. A `HalfedgeEmbedding` is loaded as geometry and is displayed in the 3D viewer. A `HalfedgeMap` defines geometry together with texture coordinates and boundary identifications. If suitable boundary identification is given, a uniformizing group is calculated and visualized. `HyperEllipticAlgebraicCurves` are loaded into the hyperelliptic curve panel of the user interface, see Figure 8.3, right-top. `SchottkyData` is loaded into the Schottky modeler panel, see Figure 8.3, right-bottom.

Hyperelliptic curves

The hyperelliptic curve panel (Figure 8.3 right-top) is used to create branched triangulations of the doubly-covered sphere. Hyperelliptic curves are defined via their branch points, see Section 1.8.3 of Chapter 1. The user can add branch points by double-clicking in the graph paper view. Double-clicking an existing point brings up a coordinate editor. The lower section of the panel defines parameters of mesh creation. The user can choose to add extra equally-distributed random points to the triangulation. These points can be further optimized to form a more regular triangulation. The number of point equalizer iterations defines the number of gradient decent steps in the optimization of the functional defined in Section 1.7.3. The location of the branch points can be normalized via a Möbius transformation on the sphere such that the center of mass of the points is the center of the sphere. For the details see Section 1.6.3.

Schottky modeler

The Schottky modeler panel (Figure 8.3 right-bottom) generates Schottky data for Riemann surfaces, see Section 1.8.2.

In the graph-paper view, the circles and fix points of the generators are shown and can be modified. Moving a fix point of a generator transforms the corresponding circle as the image of the circle around the other fix point. Moving a circle causes the image circle to be recalculated. New generators can be added from the right-click menu. The parameter μ of each generator can be edited in the table below. Similar to the generation of hyperelliptic data, additional points can be added and modified by a regularization process. In addition to this, the number of points that discretize the circles can be given.

The *Generate Surface* button creates a triangulated surface together with a metric calculated from the Schottky data. This surface can then be uniformized via the main interface. The *Uniformize* button directly creates a uniformization and introduces cuts along the circles of the Schottky data. The corresponding fundamental domain is then bounded by the pre-images of the circles and curves connecting them. The examples of Section 1.8.2 use this fundamental domain.

Main interface

The main interface of ConformalLab offers all functions to create the conformal mapping examples described in this work (see Figure 8.3 left). It is divided into sections that we cover briefly from top to bottom.

The user interface comes in two modi, normal mode and expert mode, that can be switched with the *Expert Mode* check box. In normal mode we show only user interface elements that are needed by users of VaryLab. It supports basic surface parameterization and remeshing of discrete surfaces, see Chapter 9. All other elements are hidden in normal mode.

ConformalLab uses two numerical libraries to implement energy minimization. They can be chosen from the drop-down box at the top of the panel. *Petsc/Tao Numerics* selects the PETSc/-TAO C++ library, see [7, 8]. We use the Newton Trust Region (NTR) method as implemented by PETSc/TAO whenever we have calculated the Hessian matrix for the corresponding energy. Otherwise we use the LMVM method with the default configuration. *Java/MTJ Numerics* implements a version of Newton's method using the linear algebra package MTJ [31]. We implement the backtracking line-search as described by Boyd and Vandenberghe [19, pp. 464].

The *Tolerance Exp* and *Max Iterations* options are used during energy minimization. Tolerance means absolute tolerance as defined by PETSc/TAO. The minimizer converges if the Frobenius norm of the gradient is less than ten to the selected exponent. The *Max Iterations* value means the maximum number of Newton steps carried out by the optimization library.

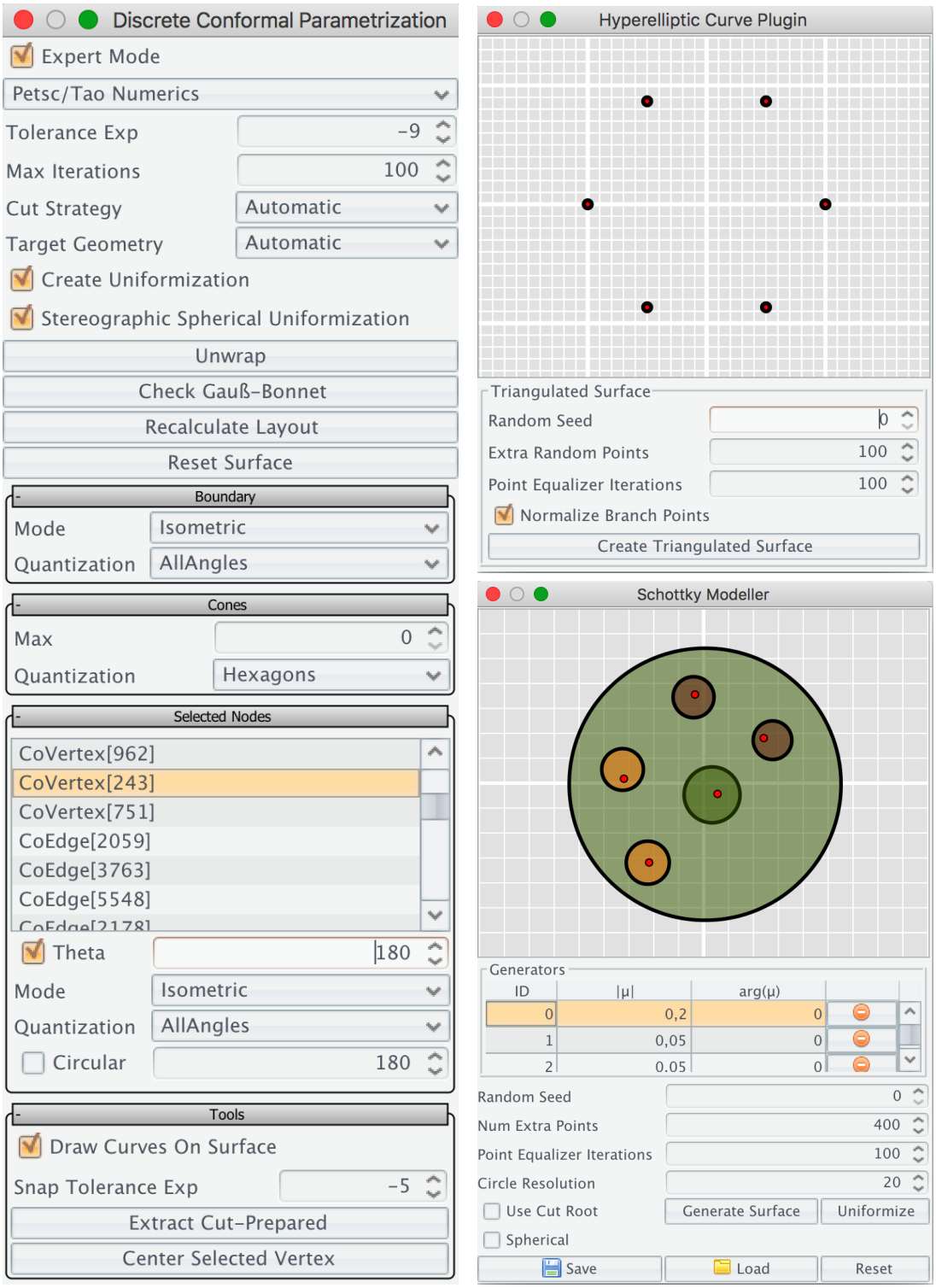


Figure 8.3: The main interface of ConformalLab (left). Hyperelliptic curve interface (right-top). Schottky modeler user interface (right-bottom).

For surfaces with genus 1 or higher a *Cut Strategy* can be chosen from a drop-down menu. *Automatic* creates cuts along precomputed edge cycles such that the resulting surface is simply connected. At the same time the number of edges that have been cut is small. *Selection* uses user-selected edges as cutting cycles. This is primarily used to create manual cutting curves. *NoCuts* does not cut the mesh. This is needed, e.g., in the workflow of creating periodic maps as described in Chapter 2.

The *Target Geometry* defines the geometry for the domain of parameterization. A value of *Automatic* selects spherical geometry for surfaces of genus 0, euclidean geometry for genus 1, and hyperbolic geometry for surfaces of higher genus.

The *Create Uniformization* check box defines whether a uniformization is calculated when pressing the *Unwrap* button. The program manages two instances of the surface: The input surface and the "unwrapped" surface. Pressing the *Reset* button reloads the input surface.

To check boundary conditions and internal cone angles, pressing *Check Gauß-Bonnet* prints the integrals over boundary curvature and Gaussian curvature to the console.

$$\sum_{v \in \partial S} (\pi - \theta_v) + \sum_{v \in \hat{S}} (2\pi - \theta_v)$$

By default all interior angles are set to 2π unless otherwise defined by manual settings in the *Selected Nodes* section. The angle at a boundary vertex is defined either manually or calculated via settings in the *Boundary* section.

The *Boundary* section contains the settings that define boundary conditions at vertices of the mesh. The default mode is *Isometric*. In this mode boundary edges on the surface and in texture space have the same length. If boundary angles on the surface are not far away from desired angles in the domain, then *Quantized Angles* mode can be used to define boundary angles via angles on the surface and a quantization function selected from the *Quantization* drop-down:

$$\begin{aligned}
Q_{\text{AllAngles}}(\theta) &= \theta \\
Q_{\text{Quads}}(\theta) &= \text{round}\left(\theta, \frac{\pi}{4}\right) \\
Q_{\text{QuadsStrict}}(\theta) &= \text{round}\left(\theta, \frac{\pi}{2}\right) \\
Q_{\text{Hexagons}}(\theta) &= \text{round}\left(\theta, \frac{\pi}{3}\right) \\
Q_{\text{Triangles}}(\theta) &= \text{round}\left(\theta, \frac{\pi}{6}\right).
\end{aligned}$$

Here round(x, y) maps x to the nearest multiple of y. *Conformal Curvature* boundary mode defines angles via a vector field on boundary edges as described in Chapter 3. The *Circle* option creates a map to the unit circle. Using this option we invoke the method described in Sections 1.4.2 and 1.5.1. In *Read Isometric Angles* mode the program performs conformal parameterization with isometric boundary. Additionally, resulting boundary angles are read into manual boundary conditions for later use, e.g., with *QuantizedAnglePeriods* boundary conditions, which modify boundary angles as described in Section 2.3.1.

The *Cones* section specifies automatically placed cones as introduced by Springborn *et al.* [69]. One can specify the number of cones and the corresponding quantization function as defined above.

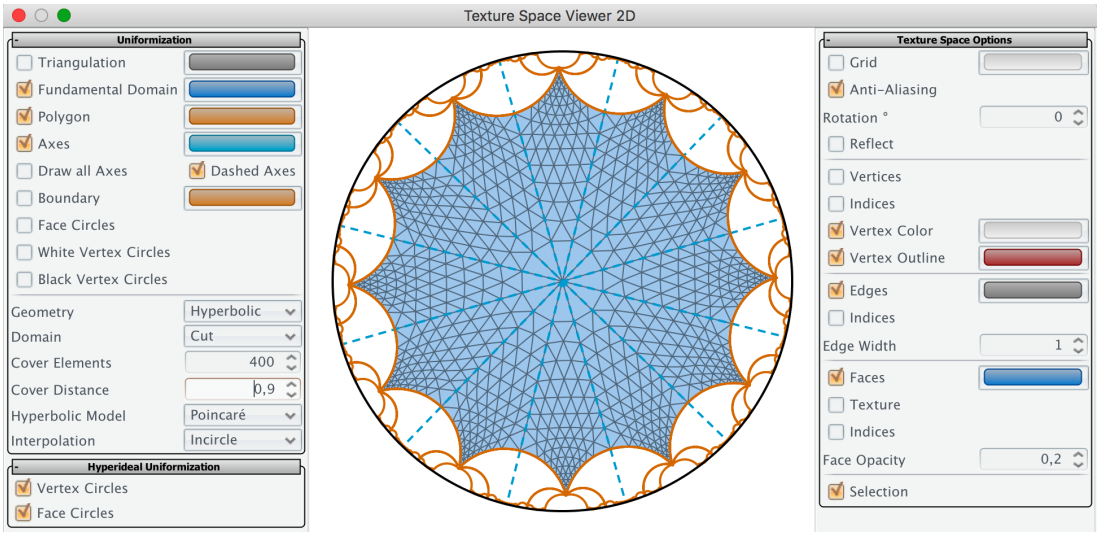


Figure 8.4: Visualization interface. It shows the domain of parameterization for euclidean and hyperbolic geometry. For hyperbolic images the Klein model, Poinaré disk model, and the half-plane model are supported.

Manual boundary conditions can be specified in the *Selected Nodes* section. Here the currently selected vertices and edges are listed. Selected vertices can have manually prescribed angles θ or boundary modes as described above. Edges can be set circular with a certain intersection angle of circum-circles, see Section 1.4.2 or Section 1.5.1. The default angle is 180°, i.e., the two adjacent triangles form a circular quadrilateral.

The *Tools* section includes commands that operate on a readily computed uniformization of a surface of higher genus $g \geq 2$. The *Draw Curves On Surface* check box calculates, if activated, images of the edges of fundamental domains. These curves are displayed on the surface, see, e.g., Figure 1.36 left. The *Extract Cut-Prepared* button creates a new surface as a copy of the current input data including edges of a fundamental polygon. One can use this to create uniformized surfaces where the triangulation fits perfectly into the fundamental polygon. The *Center Selected Vertex* button applies a hyperbolic motion to the texture coordinates such that the selected vertex becomes the center of the unit disk.

Visualization

The *Visualization Interface*, see Figure 8.4, shows the domain of uniformization. In addition, depending on the geometry, one can visualize extra structures like fundamental polygons, universal cover, or identification maps. It is divided into two sections. The generic *Texture Space Options* define options for the mapping of the mesh.

The *Uniformization* section defines the visualization of the current uniformization. *Triangulation* activates the visualization of the mesh across the universal cover of the surface using as many group elements as specified in the *Cover Elements* and *Cover Distance* input fields. The fundamental polygon corresponding to the identity group element can be visualized with the *Fundamental Domain* check box. Its axes of identification can be shown with the *Axes* check box. The *Polygon* box enables the visualization of polygon edges for all group elements defined by *Cover Elements* and *Cover Distance*. If the triangulation exhibits a circle pattern, face circles through vertices of faces can be shown.

The *Geometry* option is set automatically during calculation of uniformization but can be adjusted from the check box. The automatic value depends on the genus of the surface alone and may not be the correct choice. The program is able to calculate four different fundamental domains that can be selected from the *Domain* drop-down, see Section 1.8.1. If the selected geometry is *Hyperbolic*, one can choose the corresponding hyperbolic model. ConformalLab supports visualization of *Poicaré*, *Klein*, and *HalfPlane* model.

The *Interpolation* option lets you select how textures are interpolated on the current model. If a texture image is active, the 3D visualization is updated accordingly.

Chapter 9

VaryLab - Discrete surface optimization

9.1 Introduction

In this chapter we introduce the Java software VaryLab. It is a software developed at Berlin Institute of Technology by the author of this thesis, Thilo Rörig, and others. The software has been developed in the Projects A01 "Discrete Riemann Surfaces" and A08 "Discrete Geometric Structures Motivated by Applications in Architecture" of the SFB Transregio 109 - Discretization in Geometry and Dynamics [65]. It is designed to be an extensible and modular tool for experiments with discrete surfaces in pure mathematics and applications in industrial geometry. The purpose of this chapter is to enable the reader to reproduce the results presented in Part II of this work.

We start with a general description of VaryLab and its features in Section 9.2. Section 9.6 introduces the user interface which is based on JRWorkspace, see Chapter 6. Section 9.7, gives details on the use of VaryLab when calculating the examples of Chapter 2. Section 9.8 explains the steps needed to reproduce results presented in Chapter 3. Finally, in Section 9.9 we present the capabilities of VaryLab when calculating gridshell nets as explained in Chapter 4.

9.2 Non-linear discrete surface optimization

In its core, VaryLab is a solver for non-linear optimization problems on the coordinates of a given 3D discrete surface. That means, given a surface S and functionals $f_1, \ldots, f_n : S \to \mathbb{R}$ we (try to) minimize the combined functional

$$f(S) = \sum_{i=1}^{n} \lambda_i f_i(S) \tag{9.1}$$

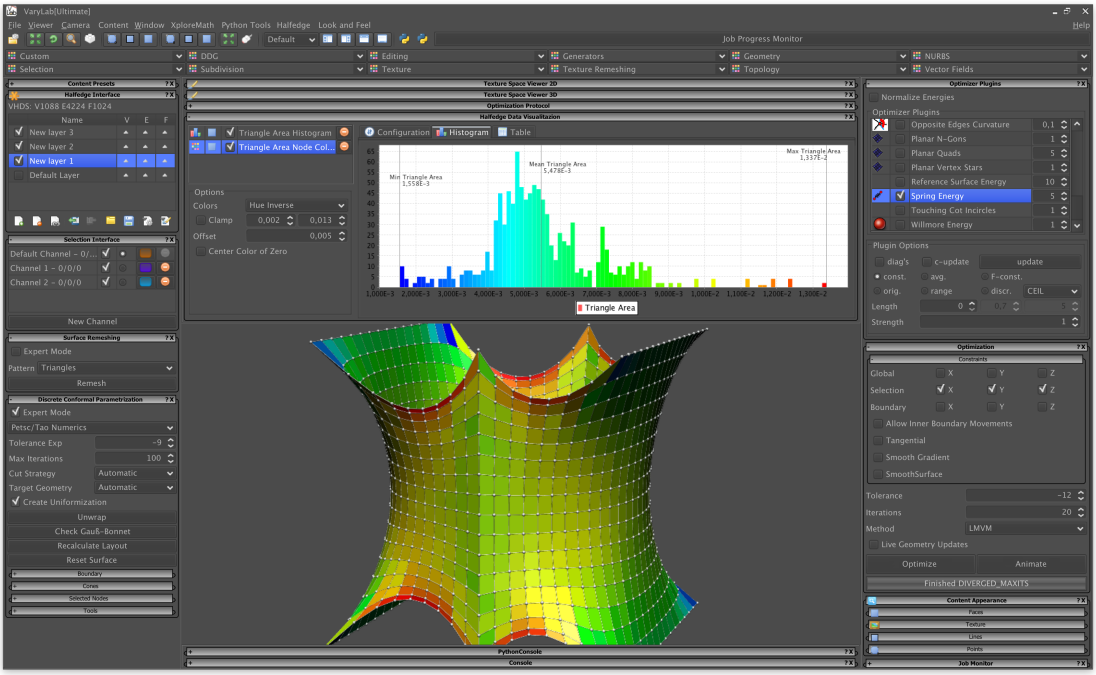


Figure 9.1: VARYLAB main user interface window. Visualization interface (top), energy configuration and optimization tools (right), layers, selections, etc. (left).

where $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ are user-defined weights. Correspondingly the first and second derivatives of $f_i(S)$ are weighted by λ_i

$$\nabla f(S) = \sum_{i=1}^{n} \lambda_i \nabla f_i(S), \quad \nabla\nabla f(S) = \sum_{i=1}^{n} \lambda_i \nabla\nabla f_i(S). \tag{9.2}$$

VARYLAB uses the numerical library PETSc/TAO [6, 7, 8] and the corresponding JAVA bindings [68] for computations. To run optimization methods we need at least an implementation of the functional's value. Other methods need the gradient or Hessian of the functional. The most important methods are

f	$f, \nabla f$	$f, \nabla f, \nabla\nabla f$
`NM` Nelder-Mead	`LMVM` Limited-Memory, Variable-Metric	`NLS` Newton Line-Search
	`CG` Conjugate Gradient	`NTR` Newton Trust-Region.

In VARYLAB a functional can choose to implement just the value. Additionally it can implement the gradient and the Hessian of S. In principle all methods can be used with all functionals even if those do not implement all data needed for the algorithm. VARYLAB approximates the values of the gradient or the Hessian if they are missing.

VARYLAB has the option to normalize the energies before optimization and calculates a μ_i for each of the energies such that $\left\|\mu_i \nabla f_i(S)\right\| = 1$ for each functional f_i and for the current mesh

geometry. Then it uses a modified energy

$$\hat{f}(S) = \sum_{i=1}^{n} \mu_i \lambda_i f_i(S). \tag{9.3}$$

for optimization.

VaryLab can handle constraints on vertex positions. We implement this by modifying the gradient and Hessian of the selected energies to be zero at the constraint variables. Even if the derivative does not match the energy anymore, we can still calculate solutions for the rest of the variables using conjugate gradient methods. Here the line search step of the algorithm minimizes the energy over the remaining variables in the direction of the gradient, i.e., the update will not include constraint variables.

9.3 Built-in functionals

VaryLab implements a number of energy functionals that are implemented as JRworkspace plug-ins and can be selected in the *Optimizer Plug-ins* panel.

- *Circular Quadrilaterals Energy* - Defines an energy functional that is minimal for planar circular quads. Since we are using an angle criterion, the convergence to planarity is relatively slow. If the *Planar Quadrilaterals* energy is added to the optimization, the geometry converges more quickly.
- *Conical Quadrilaterals Energy* - This energy implements an angle criterion for conical meshes. In combination with *Planar Quadrilaterals* or *Planar n-gons* it optimizes a mesh to have the property that faces adjacent to a vertex are tangent to a cone of revolution.
- *Cotangent Dirichlet Energy* - Dirichlet energy for the three coordinate functions of vertices of a triangle mesh. Edge weights are defined as $\omega_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \alpha_{ji})$ where α_{ij} and α_{ji} are the angles opposite of the edge e_{ij}.
- *Equal Edge Lengths* - This energy penalizes the deviation of edge lengths from their mean value. As a result it is minimal, if all edges have the same length.
- *Incircles* - The property for a quadrilateral to possess an incircle tangent to its sides is that the two sums of opposite side lengths are equal, i.e., $a + c = b + d$. Planarity is not included in this functional, so to get planar quadrilaterals with inscribed incircles one needs to add planarity to the optimization.
- *Touching Incircles* - In a quad-mesh with incircles, the incircles need not touch. In combination with the *Incircles* and *Planarity* energies one can create a meshes with touching incircles.
- *Opposite Edges Curvature* - This energy penalizes the deviation of a parameter poly-line from a straight line. Using this energy alone, will move the mesh towards a discrete ruled surface. Used together with, e.g., a *Reference Mesh* energy, this energy smoothes the parameter lines of a quad-mesh.
- *Opposite Angles Curvature* - This curvature is based on the intrinsic geometry of the surface. Let α, β, γ, and δ denote the angles in the adjacent quads at a node in cyclic order. Then the optimal mesh satisfies $\alpha + \beta = \gamma + \delta$ and $\beta + \gamma = \delta + \alpha$, i.e., the parameter lines are straight from an intrinsic point of view.
- *Planar Quadrilaterals* - Energy that enforces planarity of quadrilaterals. Implemented either as the distance of the diagonals or as the determinant of the four homogeneous vertex coordinates.

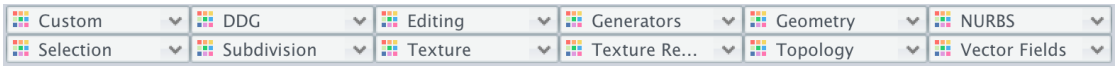


Figure 9.2: VaryLab tools tool bar.

- *Planar n-gons* - Generalization of the energy for planar quadrilaterals. Implemented as the sum of all choices of four vertices in each face.
- *Planar Vertex Stars* - This energy is dual to the planar faces energy. It computes the volume spanned by a node and its neighbors. Minimization yields meshes such that each node lies in a plane with its neighbors. If used together with face planarity, the initial mesh is mapped to a plane.
- *Reference Mesh* - Given a reference mesh we compute the closest point to a node and add a spring force between each node and its projection. The projection point is recomputed in each step of the optimization. If combined with other energies, it keeps the optimized mesh close to a reference mesh.
- *Spring Energy* - The spring energy is computed by adding springs to all the edges of the mesh. These springs can have user-specified target lengths and strengths that can be specified by various options.

9.4 Geometry processing

The geometry processing core of VaryLab is based on HalfEdge and HalfEdgeTools (Chapter 7), i.e., all geometry processing is done with the help of the half-edge data structure and algorithms that run on top of it. Frequently used geometry processing features include

- Mesh Generators - Regular Planar Meshes, Convex Hull, Primitive Meshes such as cube, cylinder, sphere, etc.
- Mesh Editing - Vertex/Edge/Face operations based on user selection.
- Subdivision - Catmull-Clark, Doo-Sabin, Loop, Sqrt3, etc.
- Remeshing - Quadrilaterals, Triangles, Singularities.

All tools are available via the tools tool bar at the top of the main window, see Figure 9.1 and 9.2.

9.5 Data visualization

VaryLab adds data sources corresponding to the energies of the optimization core. Their data can be visualized on the surface using the *Halfedge Data Visualization* interface, see Section 7.3.

Example: analyzing edge length distribution.

Load a mesh geometry. In the *Halfedge Data Visualization* select the *Edge Length* data source. It provides scalar data for edges of the mesh. Select the *Histogram* and *Node Colors* visualizers to create colored edges and a corresponding histogram for the edge lengths of the mesh, see Figure 9.3.

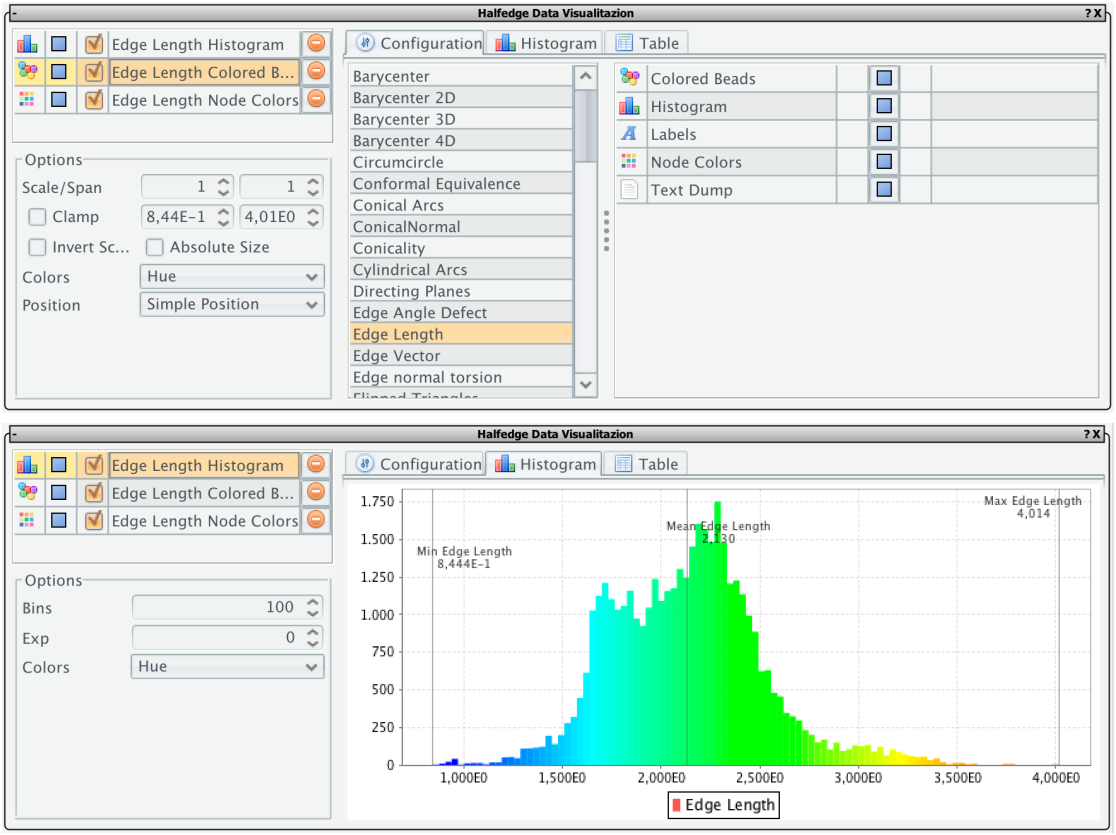


Figure 9.3: Visualization interface (top), Visualizing mesh edge length data as node colors on a surface and as histogram (bottom).

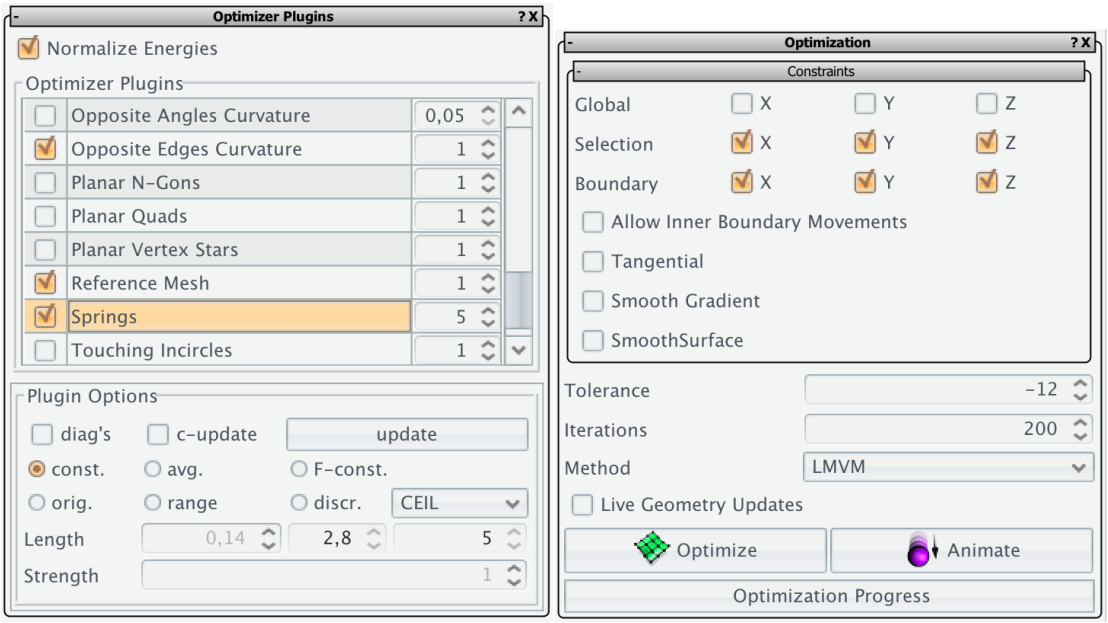


Figure 9.4: The main user interface panels of VARYLAB. List of optimization functional plug-ins and their options (left). Main optimization controls with global constraints and minimizer settings (right).

9.6 User interface

The user interface of VARYLAB, see Figure 9.1, is based on JRWORKSPACE, Chapter 6. Thus it inherits the ability for the user to freely move around all interface components of the main window, see Figure 9.1. The three most important interface components are shown in Figure 9.4 and 9.5. It is the list of energies, the optimization controls, and the optimization protocol.

Optimizer Plug-ins Panel

The *Optimizer Plugins* panel, see Figure 9.4 (left), has a list with all currently loaded plug-ins that implement energies for the use with the non-linear optimization core. For each energy one can adjust the coefficient λ in the sum of functionals, see Equation (9.1). The checkbox *Normalize Energies* activates normalization of energies, see Equation (9.3). For a selected energy, options are displayed right under the table.

Optimization Panel

In the *Optimization Panel*, see Figure 9.4 (right), we can configure constraints and run the optimization. Vertices can be fixed either globally in one or all coordinate directions, by selection, or as boundary constraint. Constraints are handled as described in Section 9.2

The check box *Allow Inner Boundary Movement* is used in conjunction with boundary constraints. The corresponding gradient part is projected onto an adjacent boundary edge thus any conjugate

Figure 9.5: The *Optimization Protocol* panel shows the progress of the optimization for each activated energy in the *Optimizer Plug-ins* panel, see Figure 9.4 (left).

gradient update will move the vertex along this direction. This mode is meant to be used with straight boundaries, otherwise the behavior is undefined.

Analog to the inner boundary movement constraint, the *Tangential* constraint projects the gradient of a vertex onto the tangent plane at this vertex for current mesh geometry. With this option, vertices stay close to the initial surface if the surface is sufficiently smooth but can move freely along tangent directions of the surface.

The options *Smooth Gradient* and *Smooth Surface* apply Laplacian smoothing to either the gradient or the coordinate values of the mesh. We use a graph Laplacian, i.e., all weights are equal to 1.

The values for *Tolerance* and *Iterations* define stop criteria for the optimization core. If the Frobenius norm of the gradient drops below the tolerance or if the maximum number of iterations are performed, the optimizer stops.

The numerical *Method* can be selected from the drop down field, see Section 9.2 for explanation.

The check box *Live Geometry Updates* updates the coordinates of the surface during optimization if selected. Any visualization is updated accordingly.

Optimization Protocol Panel

During optimization VaryLab logs the energy of each functional as well as the Frobenius norm of the gradient and plots them to the optimization protocol, see Figure 9.5. The energy value is drawn as a green curve, the norm of the gradient is a purple curve. By default the display is logarithmic in the values and linear in time.

9.7 Periodic conformal maps with VaryLab

In this section we describe how the methods of Chapter 2 are implemented in VaryLab. We split the process in two parts. Part one describes the creation of periodic triangle, quad, or hexagonal meshes from an initial unstructured triangle mesh. Part two deals with optimization of panels created from a mesh in part one.

9.7.1 Periodic parameterization

The parameterization part of the work is carried out via the ConformalLab main user interface.

(0) We load a surface with two boundary components. We can map the surface to a pattern-adapted cone of revolution using the three methods described in Chapter 2: mapping to a cylinder (a), polygonal map to a cone of revolution (b), and isometric boundary mapping(c).

(1a) For the map to the cylinder we create a conformal parameterization with straight boundary using the *Discrete Conformal Parameterization* panel and *Quantized Angles/Straight* boundary mode. A cut is introduced automatically to uniquely define the map.

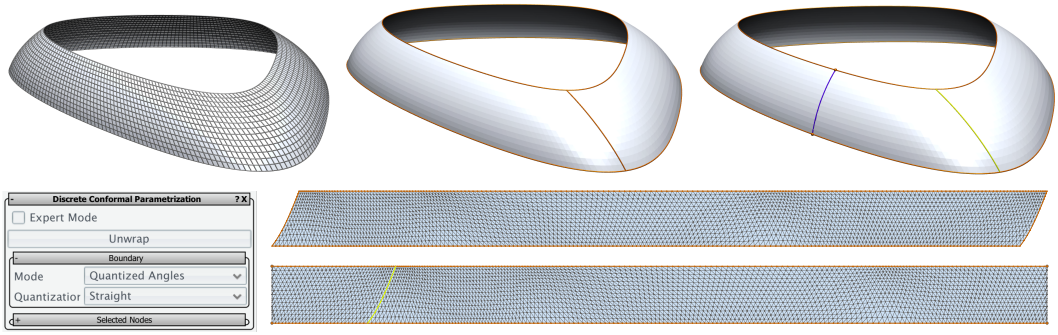


Figure 9.6: Map to cylinder. Start with an automatically cut mesh (top-middle and upper domain image) and modify the cut (top-right and lower domain image) such that the remeshing algorithm can handle the complete boundary.

(1b) For a polygonal map to a cone of revolution, we select boundary vertices to specify boundary angles for the domain. We use multiples of $\frac{\pi}{3}$ for triangle/hexagonal panels and multiples of $\frac{\pi}{4}$ for quadrilaterals.

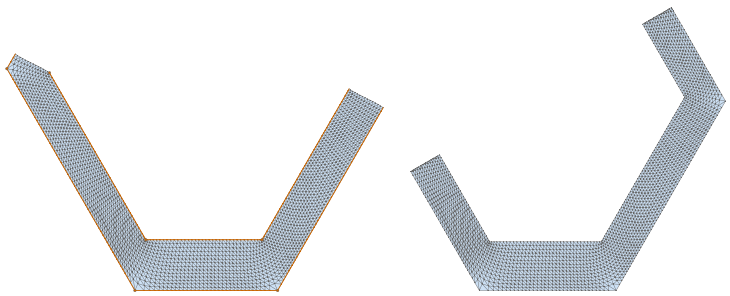


Figure 9.7: Mapping the surface from Figure 9.6 to a hex pattern adapted domain with polygonal boundary curve. Cut orthogonal to the boundary to create a domain that can easily be meshed with boundary-aligned hexagons. The domain is identified along the cut via a rotation by π.

(1c) We create an isometric boundary and a map to cone of revolution using the *Read Isometric Angles* boundary mode and the *NoCuts* cut strategy. It creates a map to an arbitrary cone of revolution and reads off the resulting boundary angles. Those are set as new boundary conditions (red vertex selection). In a second step we choose the *Quantized Angle Periods* boundary mode and the desired quantization for the final map to the pattern adapted cone of revolution.

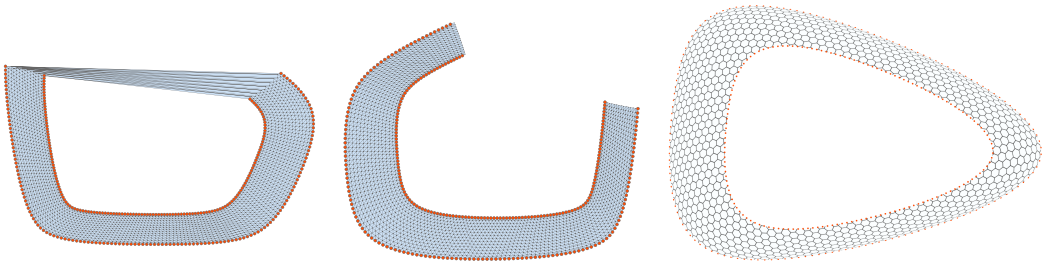


Figure 9.8: Map to cylinder with isometric boundary. Create a map with isometric boundary without cutting the mesh (left). Modify the resulting boundary angles to support a periodic pattern (middle). Periodic hexagonal pattern on the surface (right).

(2) We use the *Cut and Glue Texture Domain* command and select *Orthogonal to Boundary* to create a map to a rectangle (a) or a map to a polygonal region (b). In the isometric boundary case (c) we do not need a polygon as boundary curve.

(3) Select a predefined texture for quadrilateral, triangle, or hexagonal mesh preview from the *Content Appearance→Texture* panel. Adjust the texture scale to a reasonable value. In the isometric case (c) we close the period with the *Texture Transformation* user interface manually. For cases (a) and (b), periods will be closed automatically by the remeshing algorithm.

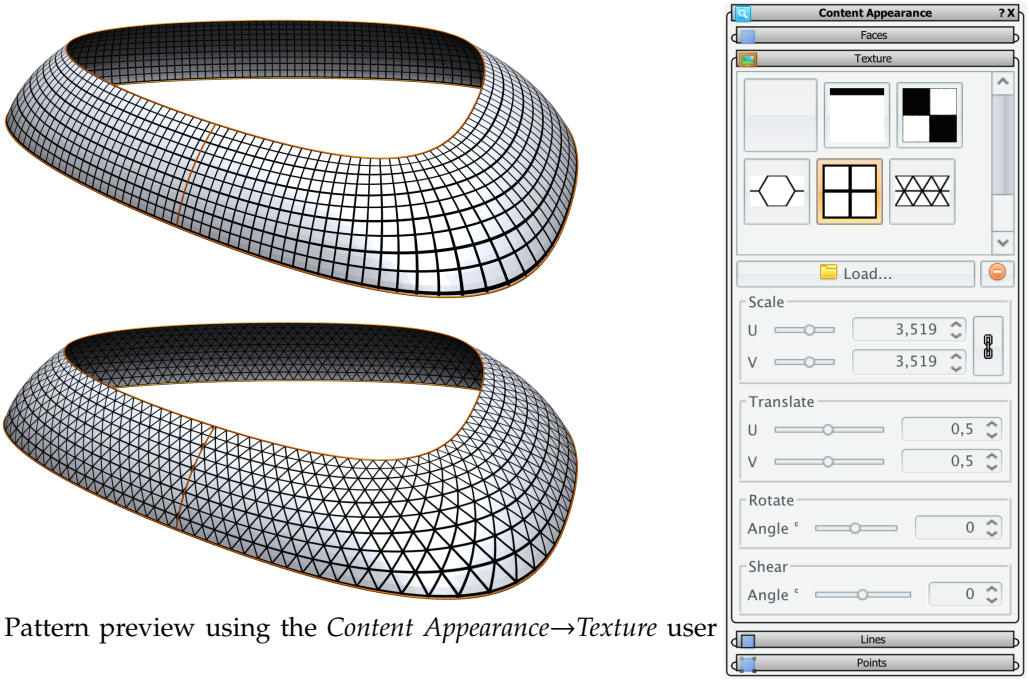


Figure 9.9: Pattern preview using the *Content Appearance→Texture* user interface.

(4) Perform remeshing in cases (a) and (b) either for *Boundary Aligned Triangles* or *Boundary Aligned Quads* using the *Surface Remeshing* panel. For non-boundary-aligned parameteriza-

tion (c) we use the *Quads With Singularities/Triangles With Singularities* remeshing mode.

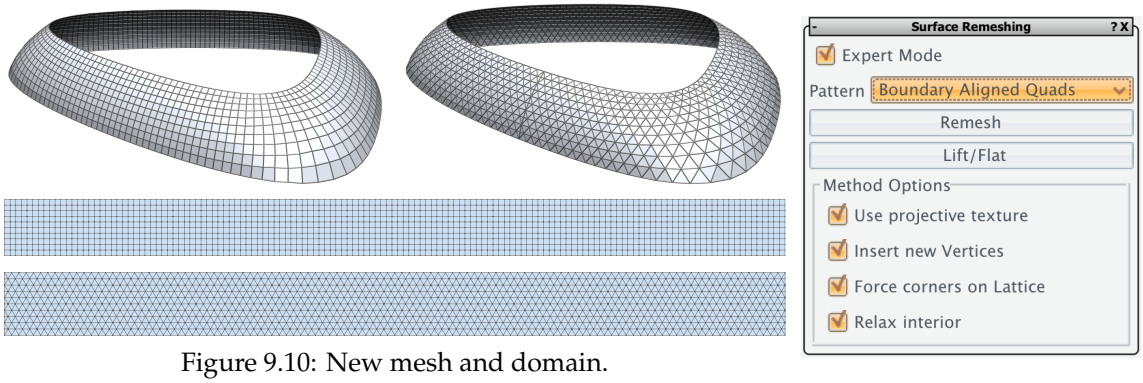


Figure 9.10: New mesh and domain.

(5) We create a watertight mesh using the *Watertight Mesh Generator* and remove extra edges and vertices. In the isometric case (c) we use a combination of *Topology→Stitch* and *Topology→Stitch Cut Path* to remove the cut path from the mesh.
(6) For hexagonal mesh creation we select from the periodic triangle mesh all centers of hexagons using the *Selection→Lattice* command. The *Remove Vertex and Fill* command then creates a periodic hexagonal mesh.

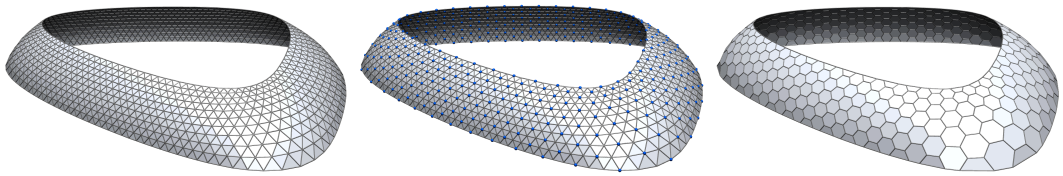


Figure 9.11: Creating a periodic hexagonal mesh from a triangle mesh.

9.7.2 Panel optimization

(7) *Topology→Explode* creates separate faces. We use a *Mean Face Edge Length* histogram to show the density of edge lengths. If we want planar panels we should planarize them now.

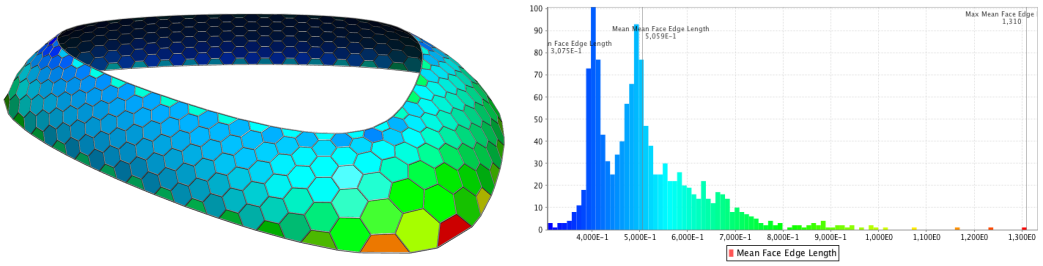


Figure 9.12: Measuring panel sizes and density with the *Halfedge Data Visualization* facility.

(8) We equalize the edge lengths per face using the *Springs* Energy and *F-const* option. Use the *Floor* rounding method. Press *Update* to set target lengths per face.

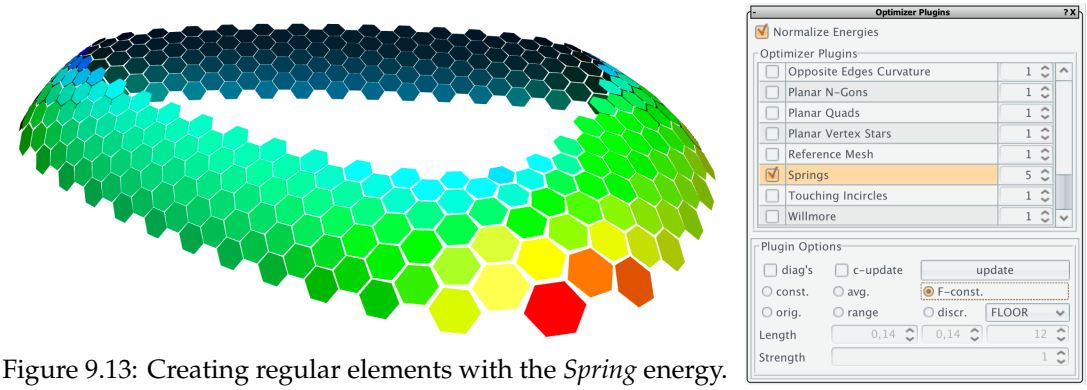


Figure 9.13: Creating regular elements with the *Spring* energy.

(9) From the histogram, read off the smallest and largest edge lengths and transfer those into the *Springs* energy UI. We select the *discr.* option and the number of discrete steps. Optimize the surface to consist of a limited number of panel sizes.

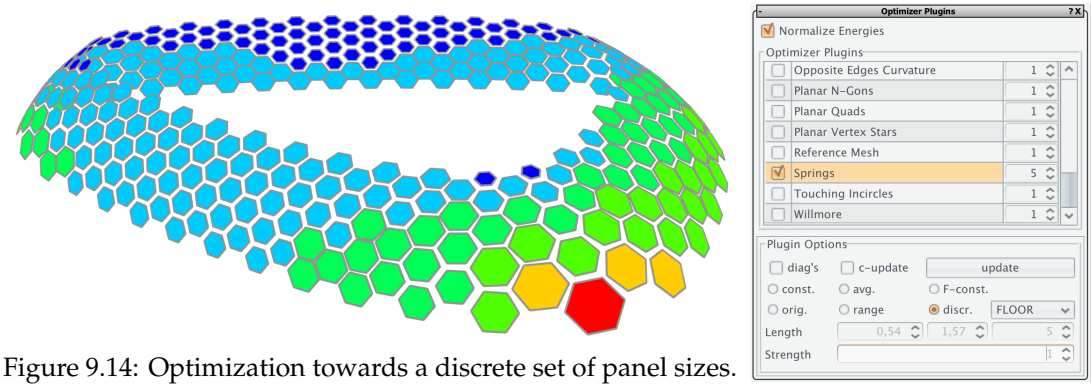


Figure 9.14: Optimization towards a discrete set of panel sizes.

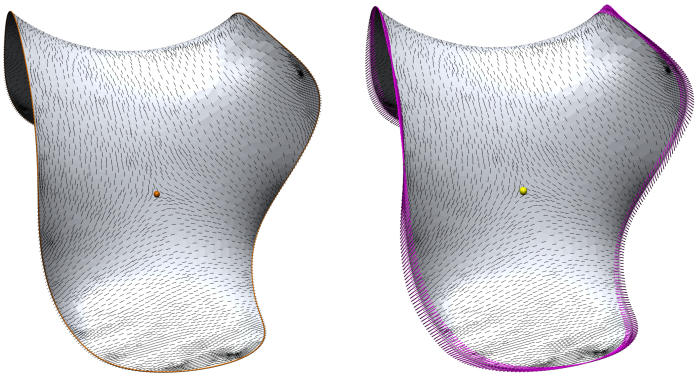
9.8 Quasiisothermic meshes with VARYLAB

The methods of Chapter 3 can be divided into two parts. The Parameterization part where we create the coordinates of the triangle mesh based on curvature direction data on the boundary of the mesh. And the optimization part where a quadrilateral mesh from part 1 is optimized towards touching incircles, the defining property of discrete s-isothermic meshes.

9.8.1 Quasiisothermic paramerization

(0) We load a genus-0 surface with one boundary component.
(1) Calculate curvature direction estimates on interior vertices to find singularity locations and indices with the *Vector Field→Curvature Vector Fields* command. We visualize directions using the *Halfedge Data Visualization* interface. The data is called, e.g., *Kmax Vec V* for maximum curvature direction with respect to the surface normal.
(2) We select singularity vertices and assign corresponding cone angles in the *Selected Nodes* panel of the *Discrete Conformal Parameterization* panel.
(3) Calculate curvature direction estimates on boundary edges of the surface, again using the *Vector Field→Curvature Vector Fields* command. We check singularity indices with the *Check*

Gauß-Bonnet button in the *Discrete Conformal Parameterization* panel. It prints the left side of the Gauß-Bonnet equation to the console. It should give 2π for a genus 0 surface in this case.



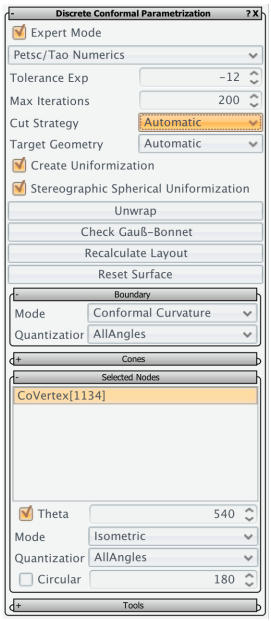


Figure 9.15: Inspect the curvature direction field on the surface (left) and specify boundary conditions (right, magenta) and singularities (right, yellow).

(4) Create a discrete conformal parameterization. Use *Conformal Curvature* as boundary mode. Select suitable cone angles at singularities in the *Selected Nodes* panel. Select a quad texture to visualize the parameterization on the surface. Rotate the texture to match the prescribed boundary directions.

(5) Move the texture such that singularities lie either in the middle or at a corner of a quad of the texture. Use the *Texture Remeshing→Transform Texture* command to transform the texture such that two selected vertices lie on $(0, 0)$ and $(1, 0)$ respectively. This method works for one or two singularities. We do not implement methods to distort the mapping to match more that two singularities.

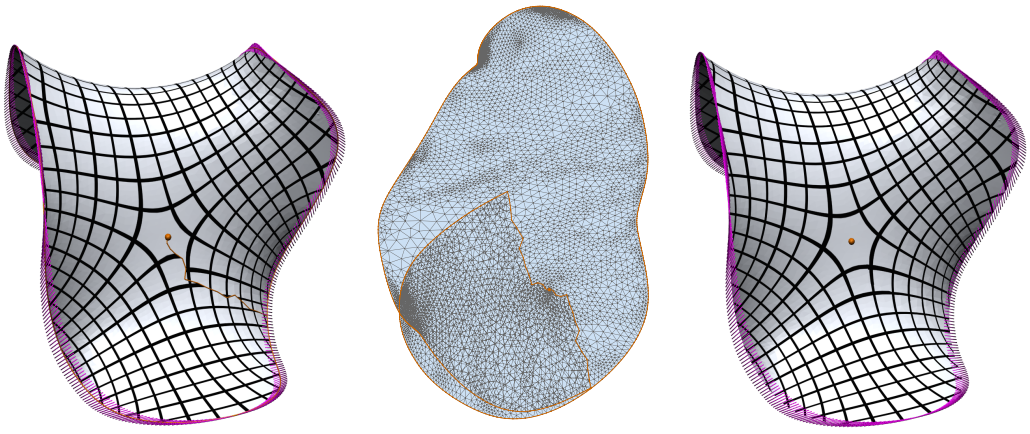


Figure 9.16: A quasiisothermic parameterization and its domain (left and middle). Move the singularity to a symmetry point of the pattern to close the parameter lines on the surface.

(6) Create a subdivision quad-mesh using the *Surface Remeshing* panel and the *Quads With Singularities* mode. This mode is available in the *Expert* mode of the panel. Press the *Lift/Flat* button to lift the subdivision to the surface.
(7) Use the *Texture Remeshing→TextureGeometry* command to extract a quad mesh from the subdivision mesh. Disable the layer of the original mesh as we will continue to work with the quad-mesh.

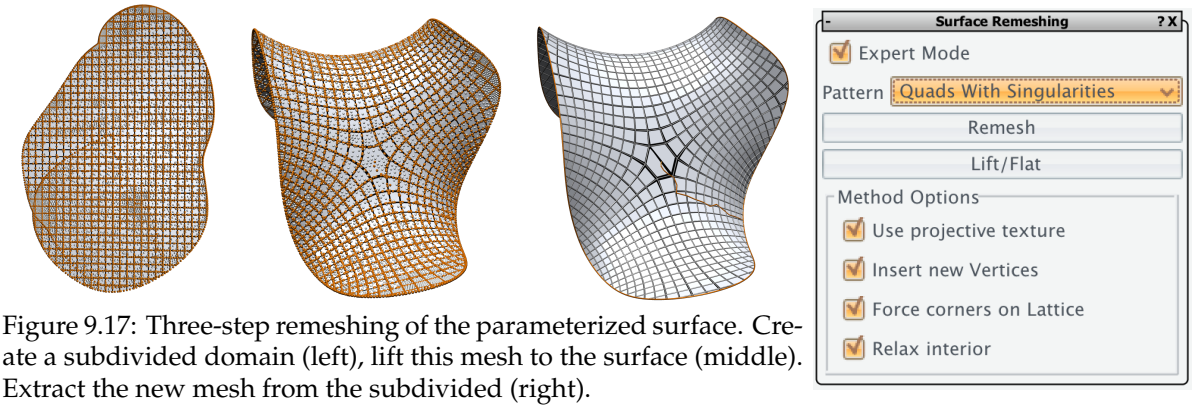


Figure 9.17: Three-step remeshing of the parameterized surface. Create a subdivided domain (left), lift this mesh to the surface (middle). Extract the new mesh from the subdivided (right).

(8) Sew up the path from the boundary to the singularity using the *Topology→ Stitch Cut Path* command. Select the two boundary vertices and a connected edge on the path.
(9) Remove extra vertices with the *Texture Remeshing→Collapse 1,2-Valent Vertices* command.
(10) Clean up the mesh from extra edges and vertices with the *Selection→Geodesic*, *Edit→Remove Edge And Fill*, and *Texture Remeshing→Collapse 1,2-Valent Vertices* commands.

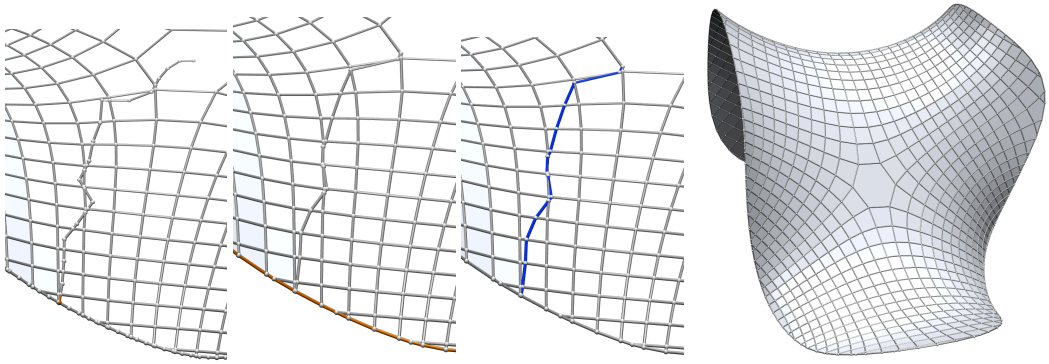


Figure 9.18: Removing the cut from a singularity to the boundary. Stitch the cut path by selecting the two vertices on the boundary and one adjacent edge on the path (left). Remove resulting vertices of valence 2 (second to left). Select the path and remove extra edges.

9.8.2 Optimization towards touching incircles

In Chapter 3, Section 3.5, we propose to further optimize a new mesh to possess touching incircles, i.e., being a discrete s-isothermic mesh by definition. We propose an energy, implemented as *Touching Incircles* energy, that enforces planar quadrilaterals that possess incircles to additionally possess touching incircles. Thus we superpose three energies to achieve the desired effect. Activate *Incircles*, *Planar Quads*, and *Touching Incircles* to define the required energy.

9.9 Gridshells with VaryLab

VaryLab supports the creation of gridshell meshes, i.e., quadrilateral meshes with equal edge lengths. In this section we describe how the results of Chapter 4 were created using the features of VaryLab.

(0) Design the target surface. The surface must be a topological disk.
(1) Create an extended reference surface.

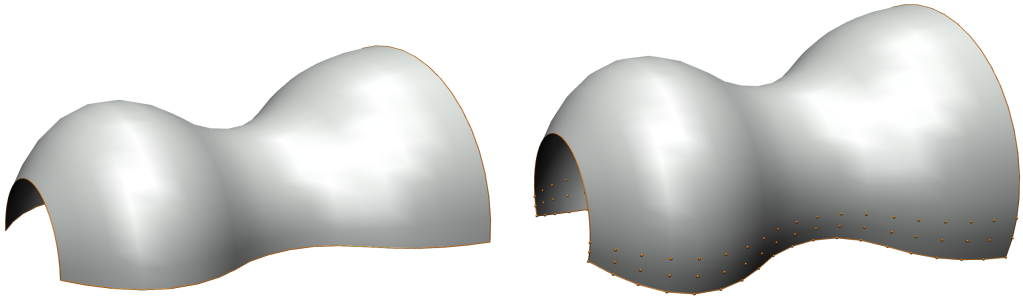


Figure 9.19: The target shape design (left) and the extended surface used as reference surface (right).

(2) Discrete conformal parameterization using isometric boundary.

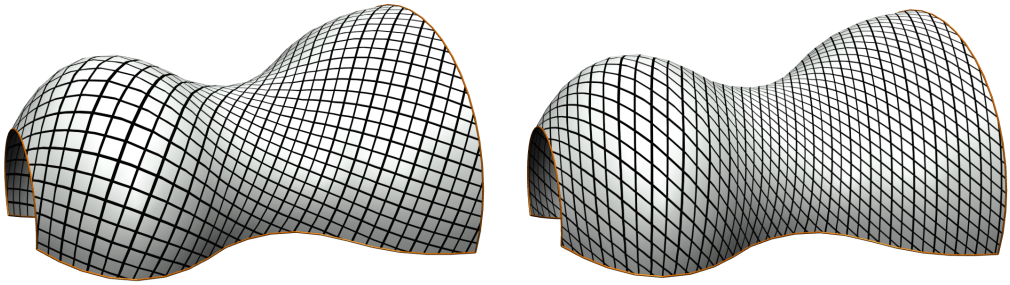


Figure 9.20: Conformal parameterization with orthogonal parameter lines visualized with a quadrilateral texture (left). Parameter lines sheared by 40° (right).

(3) Create a new quadrilateral mesh using the parameterization from step 2. We are free to shear the meshing pattern in the domain by an angle α. From the *Surface Remeshing* panel we choose the *Quads* mode and press *Remesh*. Alternatively the three-step remeshing method can be performed, see Figure 9.17.

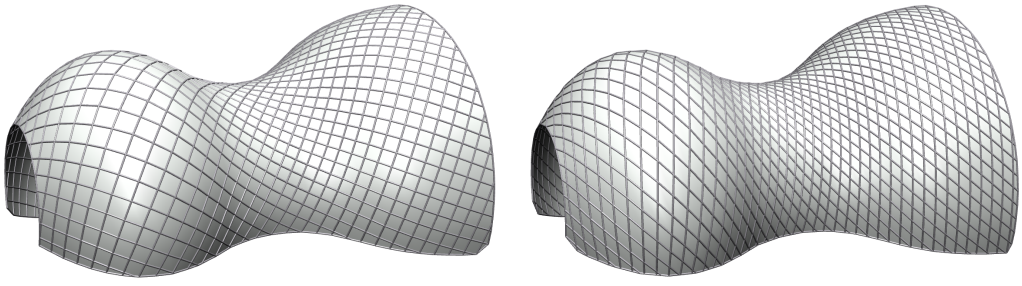


Figure 9.21: New quadrilateral meshes with orthogonal parameter lines (left) and parameter lines sheared by 40° (right).

(4) Remove all boundary vertices and analyze the remaining edge lengths with node colors and histogram.

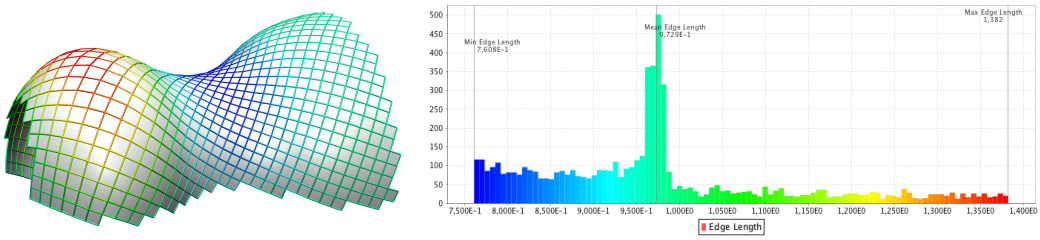


Figure 9.22: Edge length analysis. Target edge length should be below the mean edge length.

(5) Configure the optimization using the *Reference Surface* energy with the surface from step 1. Add the *Springs* energy with a constant edge length depending on the size of the target surface. Finally add the *Opposite Edges Curvature* energy as a fairing term and to control the local curvature of curves.

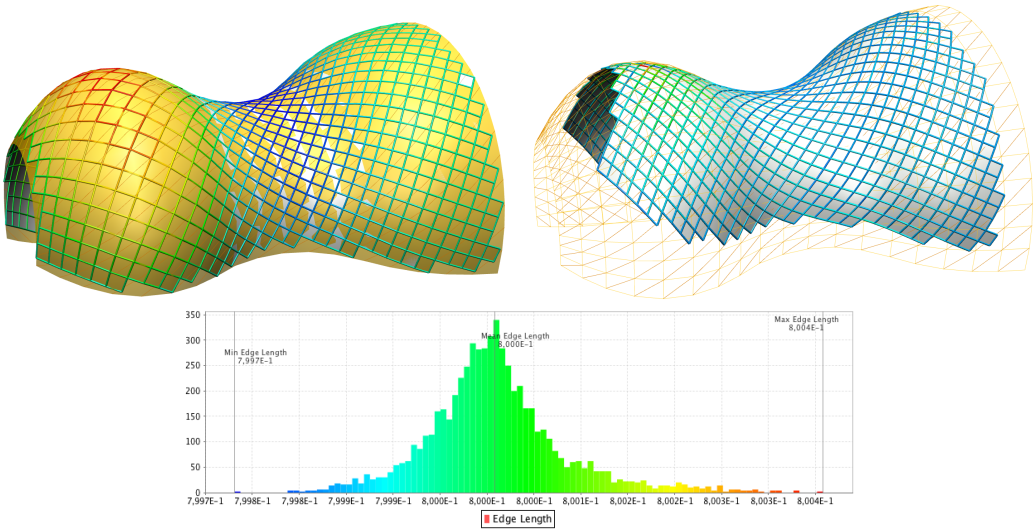


Figure 9.23: Reference surface visualization (left). Optimized mesh (right). Optimized histogram (bottom).

(6) Run optimization. It has been proven successful to switch the reference surface energy off for a few iterations to let the mesh relax towards equal edge length and smooth parameter curves. Switching it back on will then project the mesh back onto the reference surface retaining most of the lengths and curvatures. Since the energy is non-convex there are lots of local minima and the solution depends heavily on the solver in use. The *CG*, conjugate gradient, solver converges slower but yields smoother results than, e.g., the *LMVM* solver.

Bibliography

[1] U. Abresch. Constant mean curvature tori in terms of elliptic functions. *J. Reine Angew. Math.*, 374:169–192, 1987.

[2] V. E. Adler, A. I. Bobenko, and Y. B. Suris. Classification of integrable equations on quad-graphs. The consistency approach. *Comm. Math. Phys.*, 233(3):513–543, 2003.

[3] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 157–164, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[4] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003.

[5] H. Alpermann, E. Lafuente Hernández, and C. Gengnagel. Case-studies of arched structures using actively-bent elements. In *Conference Proceedings IASS Symposium 2012*, 2012.

[6] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[7] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2014.

[8] S. Benson, L. C. McInnes, J. Moré, T. Munson, and J. Sarich. TAO user manual (revision 1.9), 2007. `http://www.mcs.anl.gov/tao`.

[9] P. Bo, H. Pottmann, M. Kilian, W. Wang, and J. Wallner. Circular arc structures. *ACM Trans. Graphics*, 30:#101,1–11, 2011. Proc. SIGGRAPH.

[10] A. I. Bobenko. All constant mean curvature tori in $\mathbf{R}^3$, S^3, H^3 in terms of theta-functions. *Math. Ann.*, 290(2):209–245, 1991.

[11] A. I. Bobenko, T. Hoffmann, and B. Springborn. Minimal surfaces from circle patterns: geometry from combinatorics. *Ann. of Math. (2)*, 164(1):231–264, 2006.

[12] A. I. Bobenko and U. Pinkall. Discretization of surfaces and integrable systems. In A. I. Bobenko and R. Seiler, editors, *Discrete integrable geometry and physics*, volume 16 of *Oxford Lecture Ser. Math. Appl.*, pages 3–58. Oxford Univ. Press, New York, 1999.

[13] A. I. Bobenko, U. Pinkall, and B. Springborn. Discrete conformal maps and ideal hyperbolic polyhedra. *Geometry & Topology*, 19(4):2155–2215, 2015.

[14] A. I. Bobenko, S. Sechelmann, and B. Springborn. Discrete conformal maps: Boundary value problems, circle domains, Fuchsian and Schottky uniformization. In A. I. Bobenko, editor, *Advances in Discrete Differential Geometry*. Springer, 2016.

[15] A. I. Bobenko and Y. B. Suris. Integrable systems on quad-graphs. *Int. Math. Res. Not.*, (11):573–611, 2002.

[16] A. I. Bobenko and Y. B. Suris. *Discrete differential geometry – Integrable structure*, volume 98 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2008.

[17] D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 77:1–77:10, New York, NY, USA, 2009. ACM.

[18] L. Bouhaya, O. Baverel, and J.-F. Caron. Optimisation structurelle des gridshells. In *10ème Colloque national en calcul des structures (CSMA)*, Giens, France, May 2011.

[19] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. `http://stanford.edu/~boyd/cvxbook/`.

[20] P. Brinkmann, C. Gunn, T. Hoffmann, P. Peters, U. Pinkall, S. Sechelmann, and S. Weissmann. JREALITY. `http://www.jreality.de`.

[21] Caltch Discretization Center. DDG lecture notes and assignments. `http://brickisland.net/cs177/`.

[22] D. Cohen-Steiner and J.-M. Morvan. Restricted Delaunay triangulations and normal cycle. In *Symposium on Computational Geometry*, pages 312–321, 2003.

[23] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *Symposium on Computational Geometry*, pages 244–253, 2002.

[24] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 157–186. Springer, Berlin, Heidelberg, 2005.

[25] W. Floyd, B. Weber, and J. Weeks. The achilles' heel of $o(3,1)$. *Experimental Mathematics*, 11(1):91–97, 2002.

[26] Geometry Group@TU-Berlin. Mathematical visualization undergraduate course. `http://www3.math.tu-berlin.de/geometrie/Lehre/{WS08-SS13}/MathVis/`.

[27] J. Glymph, D. Shelden, C. Ceccato, J. Mussel, and H. Schober. A parametric strategy for free-form glass structures using quadrilateral planar facets. *Automation in Construction*, 13(2):187 – 202, 2004. Conference of the Association for Computer Aided Design in Architecture.

[28] X. Gu, F. Luo, J. Sun, and T. Wu. A discrete uniformization theorem for polyhedral surfaces. `arXiv:1309.4175`, 2013.

[29] X. Gu, F. Luo, J. Sun, and T. Wu. A discrete uniformization theorem for polyhedral surfaces II. `arXiv:1401.4594`, 2014.

[30] M. Heil. *Numerical tools for the study of finite gap solutions of integrable systems*. PhD thesis, TU-Berlin, 1995.

[31] B.-O. Heimsund. MTJ matrix toolkits for java, 2011. `https://github.com/fommil/matrix-toolkits-java`.

[32] U. Hertrich-Jeromin. *Introduction to Möbius Differential Geometry*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2003.

[33] J. Jost. *Compact Riemann Surfaces*. Universitext (En ligne). Springer Berlin / Heidelberg, 2007.

[34] JTEM Development Team. JTEM – Java Tools for Experimental Mathematics, 2015. `http://www.jtem.de`.

[35] F. Kälberer, M. Nieser, and K. Polthier. Quadcover - surface parameterization using branched coverings. *Comput. Graph. Forum*, 26(3):375–384, 2007.

[36] L. Keen. Canonical polygons for finitely generated fuchsian groups. *Acta Mathematica*, 115(1):1–16, 1966.

[37] L. Kettner. Halfedge data structures. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.4 edition, 2000.

[38] H. Kouřimská, L. Skuppin, and B. Springborn. A variational principle for cyclic polygons with prescribed edge lengths. In A. I. Bobenko, editor, *Advances in Discrete Differential Geometry*. Springer, 2016.

[39] M. Kuijvenhoven. A design method for timber grid shells, 2009. Master's thesis, Delft University of Technology.

[40] E. Lafuente Hernández, C. Gengnagel, S. Sechelmann, and T. Rörig. On the materiality and structural behaviour of highly-elastic gridshell structures. In C. Gengnagel, A. Kilian, N. Palz, and F. Scheurer, editors, *Computational Design Modeling: Proceedings of the Design Modeling Symposium Berlin 2011*, pages 123–135. Springer, 2011. `http://dx.doi.org/10.1007/978-3-642-23435-4_15`.

[41] E. Lafuente Hernández, S. Sechelmann, T. Rörig, and C. Gengnagel. Topology optimisation of regular and irregular elastic gridshells by means of a non-linear variational method. In L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 147–160. Springer, 2012. `http://dx.doi.org/10.1007/978-3-7091-1251-9_11`.

[42] J. Lawson, H. Blaine. Complete minimal surfaces in $\mathbb{S}^3$. *Annals of Mathematics*, 92(3):pp. 335–374, 1970.

[43] G. Leibon. Characterizing the Delaunay decompositions of compact hyperbolic surfaces. *Geom. Topol.*, 6:361–391, 2002.

[44] Y. Liu, H. Pottmann, J. Wallner, Y.-L. Yang, and W. Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graph.*, 25(3):681–689, 2006.

[45] F. Luo. Combinatorial Yamabe flow on surfaces. *Commun. Contemp. Math.*, 6(5):765–780, 2004.

[46] McNeel. Paneling tool documentation. http://wiki.mcneel.com/labs/panelingtools, 2015.

[47] J. Milnor. Hyperbolic geometry: the first 150 years. *Bull. Amer. Math. Soc. (N.S.)*, 6(1):9–24, 1982.

[48] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, Apr. 1959.

[49] Z. Nehari. *Conformal mapping*. McGraw-Hill Book Co., Inc., New York, Toronto, London, 1952.

[50] M. Nieser, J. Palacios, K. Polthier, and E. Zhang. Hexagonal global parameterization of arbitrary surfaces. *IEEE Trans. Vis. Comput. Graph.*, 18(6):865–878, 2012.

[51] F. Nijhoff and H. Capel. The discrete Korteweg-de Vries equation. *Acta Appl. Math.*, 39(1-3):133–158, 1995.

[52] B. Oberknapp and K. Polthier. An algorithm for discrete constant mean curvature surfaces. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics*, pages 141–161. Springer Berlin Heidelberg, 1997.

[53] F. Otto, E. Schauer, J. Hennicke, and T. Hasegawa. *Gitterschalen: Bericht über das japanisch-deutsche Forschungsprojekt STI, durchgeführt von Mai 1971 bis Mai 1973 am Institut für Leichte Flächentragwerke.* Seibu Construction Company / Institut für Leichte Flächentragwerke / Krämer, 1974.

[54] H. Pottmann, Y. Liu, J. Wallner, A. Bobenko, and W. Wang. Geometry of multi-layer freeform structures for architecture. *ACM Trans. Graph.*, 26(3), July 2007.

[55] H. Pottmann, A. Schiftner, P. Bo, H. Schmiedhofer, W. Wang, N. Baldassini, and J. Wallner. Freeform surfaces from single curved panels. *ACM Trans. Graph.*, 27(3):#76, 1–10, 2008.

[56] Robert McNeel & Associates. Rhino 3D, 2015. https://www.rhino3d.com.

[57] T. Rörig, S. Sechelmann, A. Kycia, and M. Fleischmann. Surface panelization using periodic conformal maps. In P. Block, J. Knippers, N. Mitra, and W. Wang, editors, *Advances in Architectural Geometry 2014*, page 365. Springer, 2014. http://dx.doi.org/10.1007/978-3-319-11418-7_13.

[58] A. Schiftner, M. Höbinger, J. Wallner, and H. Pottmann. Packing circles and spheres on surfaces. *ACM Trans. Graphics*, 28(5):#139,1–8, 2009. Proc. SIGGRAPH Asia.

[59] P. Schmutz Schaller. Geometry of riemann surfaces based on closed geodesics. *Bull. Amer. Math. Soc.*, 35:193–214, 1998.

[60] P. Schmutz Schaller. Teichmüller space and fundamental domains of fuchsian groups. *L'Enseignement Mathématique*, 45:169–187, 1999.

[61] O. Schramm. Circle patterns with the combinatorics of the square grid. *Duke Math. J.*, 86:347–389, 1997.

[62] S. Sechelmann, A. Bobenko, and B. Springborn. Lawson's surface uniformization. DGD Gallery, 2015. https://gallery.discretization.de/models/lawsons_surface_uniformization.

[63] S. Sechelmann and T. Rörig. VaryLab web page, 2015. http://www.varylab.com.

[64] S. Sechelmann, T. Rörig, and A. I. Bobenko. Quasiisothermic mesh layout. In L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 243–258. Springer, 2012. http://dx.doi.org/10.1007/978-3-7091-1251-9_20.

[65] SFB Transregio 109 - Discretization in Geometry and Dynamics. Webpage, 2015. https://www.discretization.de.

[66] A. Sheffer and E. de Sturler. Parameterization of faceted surfaces for meshing using angle-based flattening. *Engineering with Computers*, 17:326–337, 2001.

[67] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, 2(2):105–171, 2006.

[68] H. Sommer. JpetscTao, JNI wrapper, 2010. http://jpetsctao.zwoggel.net/.

[69] B. Springborn, P. Schröder, and U. Pinkall. Conformal equivalence of triangle meshes. *ACM Trans. Graph.*, 27(3):77:1–77:11, Aug. 2008.

[70] B. A. Springborn. A unique representation of polyhedral types. centering via möbius transformations. *Mathematische Zeitschrift*, 249(3):513–517, 2005.

[71] C. Troche. Planar hexagonal meshes by tangent plane intersection. In *Advances in Architectural Geometry 2008*, pages 57–64, 2008.

[72] A. E. Voss. Über ein neues Princip der Abbildung krummer Oberflächen. *Mathematische Annalen*, 19:1–26, 1881.

[73] R. Walter. Constant mean curvature tori with spherical curvature lines in non-Euclidean geometry. *Manuscripta Math.*, 63(3):343–363, 1989.

[74] H. C. Wente. Counterexample to a conjecture of H. Hopf. *Pacific J. Math.*, 121(1):193–243, 1986.

[75] M. Zadravec, A. Schiftner, and J. Wallner. Designing quad-dominant meshes with planar faces. *Computer Graphics Forum*, 29(5):1671–1679, 2010. Proc. Symp. Geometry Processing.

[76] H. Zimmer, M. Campen, R. Herkrath, and L. Kobbelt. Variational tangent plane intersection for planar polygonal meshing. In L. Hesselgren, S. Sharma, J. Wallner, N. Baldassini, P. Bompas, and J. Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 319–332. Springer Vienna, 2013.

CD Content

The data disk that accompanies this work contains data and source code. See Figure 24 for an overview of the directory structure of the data. We include the data and images for most of the objects investigated in Chapter 1. For many of the examples we include more instances of the same experiment varying in genus or discretization resolution.

For example the folder `/data/schottky_g2` contains data for a Riemann surface of genus 2 given by Schottky data, see Section 1.8.2. It contains the files `genus2.xml`, `genus2_fine2.xml`, and `genus2_fine2.xml` differing in the number of vertices on the Schottky circles and in the number of vertices approximating the metric in the interior of the fundamental domain.

Data belonging to chapters of the applications part is contained in the respective folders.

Licensing

All software packages belonging to the JTEM (www.jtem.de) collection and the corresponding source code are licensed under the BSD 2-Clause License. This includes the libraries JRWORKSPACE (Chapter 6), HALFEDGE, and HALFEDGETOOLS (Chapter 7).

CONFORMALLAB is licensed as follows (copied from creativecommons.org): CONFORMALLAB by Stefan Sechelmann is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

VARYLAB and its source code does not have a license yet. The source code and binaries are included on the data disk. If you want to use VARYLAB please contact the author or Thilo Rörig or visit the webpage at `www.varylab.com`.

/ ..... Root folder of the data disk
- applications
  - gridshells ..... Models and images of Chapter 4
  - periodic ..... Models and images treated in Chapter 2
  - quasiisothermic ..... Models and images of Chapter 3
- data ..... Example data for Chapter 1
  - algorithm
  - bear_torus
  - branched_euclidean_genus_2
  - branched_genus_1
  - ...
- java ..... Java source files of the listings of Chapter 6
- publications ..... Preprint versions of the original articles of Part I and Part II
- software ..... Source code and binaries of the software packages described in Part III
  - jtem
    - halfedge ..... HalfEdge
    - halfedgetools ..... HalfEdgeTools
    - jrworkspace ..... JRworkspace
  - conformallab ..... ConformalLab
  - varylab ..... VaryLab
- Thesis.pdf ..... This document in PDF file format

Figure 24: Directory structure of the data provided with this thesis.

Acknowledgements

This research was supported by DFG SFB/TR 109 “Discretization in Geometry and Dynamics”.