Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.

http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.

http://www.ub.tuwien.ac.at/eng

TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Diplomarbeit

Modeling nearly developable Catmull-Clark surfaces

Ausgeführt am Institut für

Diskrete Mathematik und Geometrie

der Technischen Universität Wien

unter der Anleitung von

O.Univ.Prof. Dr.techn. Helmut Pottmann
Dr.rer.nat. Martin Kilian

durch

Patrick Kurt Smejkal, BSc

Stromstraße 28/12
1200 Wien

Wien, 13. März 2018

Unterschrift

Kurzfassung

Abwickelbare Flächen stehen im Interesse vielfältiger Anwendungen aus Architektur, Industrie und Computergrafik, weshalb eine Nachfrage nach Werkzeugen besteht, die das Design dieser Flächen erlauben. Ansätze zur Entwicklung solcher Werkzeuge bedienen sich typischerweise der Eigenschaft, dass abwickelbare Flächen Regelflächen sind, deren Tangentialebenen über den Verlauf der Erzeugenden konstant bleiben oder der Gauß'schen Krümmung der Flächen. Allerdings sind abwickelbare Flächen auch dadurch charakterisiert, dass sie sich ohne Verzerrung, das heißt isometrisch, in die Ebene abbilden lassen, was sich der Ansatz dieser Arbeit zu Nutze macht. Dabei wird ein vom Nutzer veränderbares Kontrollnetz mittels des Algorithmus von Catmull-Clark unterteilt und mit einem zweiten Netz selber Kombinatorik in der Ebene verglichen. Das Kontrollnetz soll dann so optimiert werden, dass die beiden Netze unter vorgegebenen Verzerrungsschranken möglichst isometrisch bleiben. Dadurch entsteht ein interaktives Design-Werkzeug, das näherungsweise abwickelbare Flächen erzeugt, dabei irreguläre Knoten berücksichtigt, und durch das Vergleichsnetz jederzeit eine Abwicklung liefert.

Abstract

Since developable surfaces are desired by architecture, industry, computer graphics and various other applications, there is a high demand for tools to design these surfaces. Implementation approaches for such tools typically use the fact that developable surfaces are special kinds of ruled surfaces with constant tangent planes over each ruling or their Gaussian curvature. However, developable surfaces can also be characterized as surfaces which can be mapped without stretching or tearing, or in other words isometrically, to the plane, what is exploited by this thesis. It uses a user-defined control net that is subdivided by the algorithm of Catmull-Clark and compared to a second net in the plane of same combinatorics. The control net will then be optimized such that the both nets stay as isometrically as possible under given distortion bounds. Thus there is realized an interactive design tool for nearly developable surfaces, that keeps irregular vertices in mind, and provides an unfolding at any time.

Danksagung

Die Erfahrung eine Diplomarbeit zu schreiben, war für mich eine interessante, aber große Aufgabe, und daher eine außerordentliche Herausforderung. Deshalb bin ich für die Unterstützung aus meinem Umfeld dankbar, was ich hier zum Ausdruck bringen möchte.

Zuerst möchte ich mich bei meinem Diplomarbeitsbetreuer Prof. Helmut Pottmann für seine Geduld und seinen inspirierenden Rat bedanken. Er hat mir stets sowohl aus fachlicher, aber auch aus organisatorischer und administrativer Sicht über das gesamte Projekt schnell und unkompliziert weitergeholfen. Für seinen technischen Rat zu Implementierungsfragen möchte ich Herrn Martin Kilian danken.

Meiner Familie bin ich für ihre unbeschwerte und lustige Art dankbar, die für mich ein Rückzugsort war und mich auf andere Gedanken gebracht hat. Allen voran möchte ich meinen Eltern Anna und Wolfgang für ihr offenes Ohr und jede nur denkbare Form der Unterstützung danken. Sie waren oft bereits für mich da, bevor mir selbst klar war, dass ich sie brauche. Meinem Bruder Martin danke ich für seine Ruhe und seinen Humor. Da ich hoffe, dass ich ihm genauso eine große Hilfe sein kann wie er mir, kann er jedenfalls auf mich zählen.

Abschließend möchte ich dem Computer Vision Lab der Fakultät für Informatik der TU Wien und dessen Vorstand Prof. Robert Sablatnig für ihr Verständnis zu meinem Studium, das ich neben meiner Tätigkeit als Techniker dort ausgeübt habe, danken. Als Student der Technischen Mathematik habe ich es sehr geschätzt, einen praktischen Ausgleich als Systemtechniker zu haben.

Contents

1 Introduction **1**
1.1 Structure of Work . . . 2

2 Related Work **3**
2.1 Ruled surface approaches . . . 3
2.2 Discrete net approaches . . . 6

3 Concepts Used **9**
3.1 Catmull-Clark surfaces . . . 9
3.2 Guided projection . . . 12
3.2.1 Implemented variations . . . 13

4 Approach **14**
4.1 Target Equations . . . 14
4.1.1 Developability . . . 14
4.1.2 Discretization . . . 16
4.1.3 Catmull-Clark surfaces . . . 18
4.1.4 Summary . . . 19
4.2 Fairness . . . 21
4.2.1 Developability . . . 21
4.2.2 Distance to old net . . . 22
4.2.3 Geodesic curvature . . . 22
4.2.3.1 Discrete unfolding . . . 22
4.2.3.2 Directly on discrete 3D net . . . 22
4.2.3.3 Irregular and special vertices . . . 23
4.3 Expectations . . . 25

5 Implementation **26**
5.1 Overview . . . 26
5.2 Engine . . . 27
5.3 Net classes . . . 28
5.3.1 Subdivision classes . . . 28
5.4 Guided projection solver . . . 28
5.5 Optimization approach . . . 29
5.6 Visualization . . . 30
5.6.1 Edge length . . . 30
5.6.2 Distortion . . . 31
5.6.3 Gauss map . . . 32
5.6.4 Registration . . . 33

6 Results **34**
6.1 Design of nearly developable surfaces 34
6.1.1 Allowed distortions between 0.9 and 1.1 36
6.1.2 Allowed distortions between 0.99 and 1.01 39
6.2 Variation of allowed mean distortion 41
6.3 Handling of irregular vertices 44
6.4 Influence of developability fairness 46

7 Appendix **48**
7.1 Examples 48
7.2 Used system 49

List of Figures **50**

List of Tables **52**

Bibliography **53**

Chapter 1

Introduction

Due to their importance in theory and practice developable surfaces are the topic of countless studies. They appear as special case of ruled surfaces with many useful properties and are used in industrial and geometric applications.

A surface is called developable, iff it can be mapped isometrically, i.e. without stretching or tearing, to a planar domain. There are many materials allowing only deformations under this assumption, for instance paper and many metals (without the use of deformation under heat), but also many fabrics behave nearly developable and allow only stretching within specific bounds. This leads to a demand of modeling and simulation tools and an underlying mathematical basis. Applications can be found in the design of architecture, ship hulls, car bodies and other industrial areas but also for instance in the design and modeling of origami.

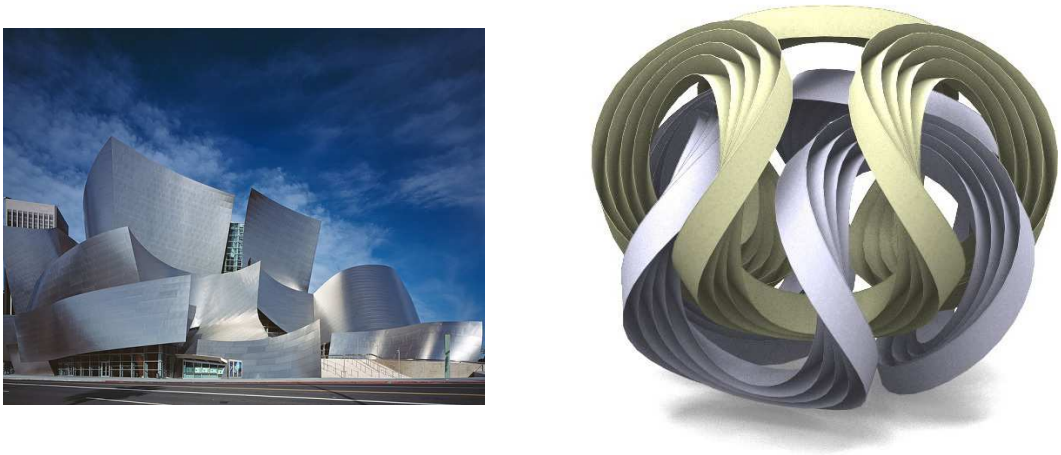


Figure 1.1: Walt Disney Concert Hall[1] and origami model[2]

In geometry in general, but especially in computer graphics, subdivision algorithms have become very important since the 1980s. A very popular example is the algorithm of Edwin Catmull and James Clark, first mentioned in their paper from 1978 (see [2]). Nowadays Catmull-Clark surfaces are widely used in many computer graphic design tools, giving the user an easy way to create smooth models via coarse control nets. The properties of Catmull-Clark surfaces are very desirable and well known (see for instance [3] and [4]).

[1] `https://pixabay.com/de/walt-disney-concert-hall-architektur-63133/`, June 28th 2017
[2] [1]

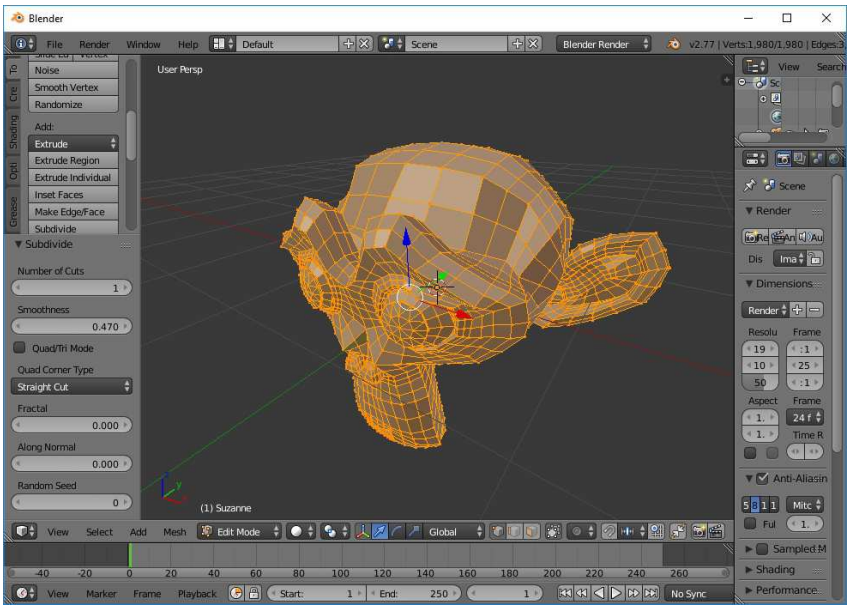


Figure 1.2: Blender modeling tool

Also developable surfaces are exploited in computer graphics. For instance, in texturization they provide an easy way to map a given texture on a 3D model. In fact, the mapping is given directly via the isometric mapping of the surface to its planar domain.

Due to the broad usage of Catmull-Clark surfaces and the applications of developable surfaces in industrial design and computer graphics, this thesis studies an approach to keep Catmull-Clark surfaces quasi-developable by the definition of conditions directly on the given meshes. This will help a user to design nearly developable surfaces by the implementation of a vertex movement based design tool, where the user designs the desired surface by dragging vertices, known from many CAD tools.

1.1 Structure of Work

In chapter 2 the thesis gives an overview over approaches related to the design of developable surfaces, whereas chapter 3 will give an introduction on concepts used by the studied approach and how they were implemented. The investigated approach is then described by chapter 4 in detail. Chapter 5 targets the implementation of the developed design tool, whereas results are shown in chapter 6.

Chapter 2

Related Work

This chapter will deal with approaches for the design of developable surfaces to give an overview of the related work. It will explain some details to make the approaches in the literature easily comparable to the approach of this thesis.

Basically there are two ways in literature how developable surfaces are targeted in design and optimization tools. The first way considers developable surfaces as a special case of ruled surfaces with constant tangent planes along each ruling. The second one tries to find discrete counterparts of developable surface properties from differential geometry and exploits them directly on discrete nets.

2.1 Ruled surface approaches

A ruled surface is a surface that can be described by a parametrization of the form

$$x : I \times [0,1] \to \mathbb{R}^3 : (u,v) \mapsto (1-v) \cdot a(u) + v \cdot b(u)$$

where $a(u)$ and $b(u)$ are curves in $\mathbb{R}^3$, the so called boundary curves. For every, but fixed, $u \in I$ a part of the line $r(u)$, spanned by $b(u) - a(u)$, lies in the surface x and is called a ruling.

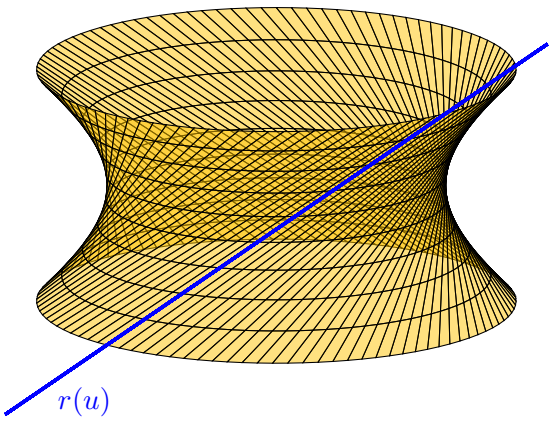


Figure 2.1: Ruled surface

The surface x is called developable, iff the tangent plane $T_{(u,v)}$ in $x(u,v)$ is constant along each ruling $r(u)$, or in other words

$$\forall u \in I : \forall v_1, v_2 \in [0,1] : T_{(u,v_1)} = T_{(u,v_2)} \tag{2.1}$$

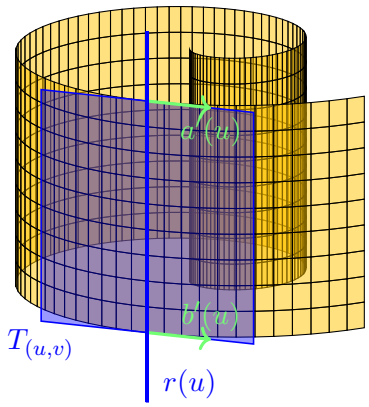


Figure 2.2: Developable surface

The vectors $a'(u)$, $b'(u)$ and $b(u) - a(u)$ have to be coplanar if the surface is developable. Furthermore, this is also sufficient for a developable surface and can therefore be used to constructively generate developable surfaces from a given boundary curve. For instance, the approach of Aumann [5] considers $a(u)$ as a given Bézier curve. It generates a new Bézier curve $b(u)$ from a given first control point, so that the resulting surface is developable.

Let $a_0, \ldots, a_N \in \mathbb{R}^3$ be the given control points of the Bézier curve $a(u)$ and $b_0 \in \mathbb{R}^3$ be the given first control point of the desired Bézier curve $b(u)$. Aumann points at the fact that the coplanarity of the vectors $a'(u)$, $b'(u)$ and $b(u) - a(u)$ leads to the coplanarity of the points a_i, a_{i+1}, b_i and b_{i+1}, what can be shown with a simple argument via the algorithm of de Casteljau. This leads to a simple recursive algorithm to compute the other control points of $b(u)$. The approach was also extended by Aumann by degree elevation techniques in [6] to solve a more general interpolation problem. There, instead of a given boundary curve, two rulings are given with their corresponding tangent planes. A wide study about conditions on developable Bézier surface patches is also given in [7].

Another similar approach was suggested in [8] by Bo and Wang where a curve $p(s)$ is given, where s is arc length, that is considered as a geodesic curve on the desired developable surface. In this case it was shown, that this curve defines a unique developable surface with the ruling direction

$$\frac{\ddot{p}}{\|\ddot{p}\|} \times \left(\frac{\ddot{p}}{\|\ddot{p}\|}\right)^{\cdot}.$$

$\ddot{p}$ is the second derivative of p in respect to arc length, or in other words the direction vector of the principal normal.

Let $U(s)$ be the tangent plane in $p(s)$ spanned by the tangent vector $\dot{p}$ and the binormal vector $\dot{p} \times \ddot{p}$ and let $n(s) := \frac{\ddot{p}}{\|\ddot{p}\|}$ be the unit normal vector on $U(s)$. Consider the line given by the intersection $U(s) \cap U(s+\Delta s)$, where Δs is small, which has the direction vector $n(s) \times n(s+\Delta s)$. By dividing this vector by Δs and forming the limit one gets

$$\lim_{\Delta s \to 0} \frac{n(s) \times n(s+\Delta s)}{\Delta s} = \lim_{\Delta s \to 0} n(s) \times \frac{n(s+\Delta s) - n(s)}{\Delta s} = n(s) \times \dot{n}(s).$$

These direction vectors form the rulings of a developable surface, the so called envelope of the tangent plane family $U(s)$. More details on this can be found in [9] and it is also strongly connected to the dual representation of developable surfaces mentioned in [10].

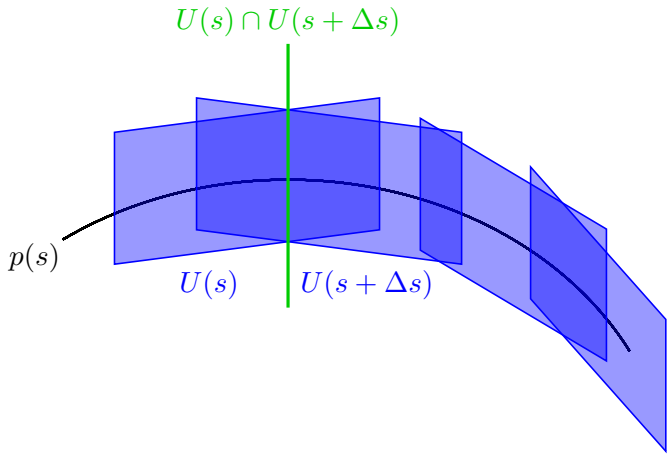


Figure 2.3: Envelop of a tangent plane family

The fact that $a'(u)$, $b'(u)$ and $b(u) - a(u)$ have to be coplanar can also be exploited for optimization like in the approach of Tang et al. [1]. The approach implements a suggested optimizer to create a tool that enables the user to design surfaces which stay developable at any time. To achieve this, it uses an optimization strategy called guided projection (see [11]) that will also be used by the approach of this thesis. It will be described in more detail in section 3.2.

Many authors allow the extension of the idea of developable surfaces by creased surfaces. These surfaces are only piecewise developable in the sense of figure 2.1, because there exist no tangent planes along the given creases (see [12], [13] and also [1]). Since creases are part of the boundary of a developable piece of the surface, they can occur as rulings or they are boundary curves. This is used for instance by [1] and [13] to implement special cases in their algorithms. For an example of a creased surface see the origami model in figure 1.1.

The approach of Solomon et al. in [13] is similar to the approach of [1] but it implements the design of developable surfaces in two stages. First the user is asked to add creases to a given domain and then these creases can be smoothed by a subdivision strategy and a special optimization approach based on a mean curvature energy term.

Pérez and Suárez use similar techniques in their paper [14] to apply developable surfaces in ship hull design. They approximate specific creases in ship hulls by B-Splines to obtain the boundary curves $a(u)$ and $b(u)$. To find a developable surface they choose parameters u_1 and u_2 and calculate two normals on the potential ruling spanned by $b(u_2) - a(u_1)$. If they are (close to) parallel, the ruling is accepted.

Another interactive modeling approach was given by Sun and Fiume in [15]. Every developable surface can locally be described by one of three surface types, namely cones, cylinders and tangent surfaces of space curves (see also figure 2.4). The approach subdivides larger surfaces into patches, which are then described by general cones. Each such cone consists of a Bézier curve, that handles its form, and an apex. Cylinders are handled as cones with an apex at infinity and the implemented forming tool via Bézier curves enables the user to model tangent surfaces.

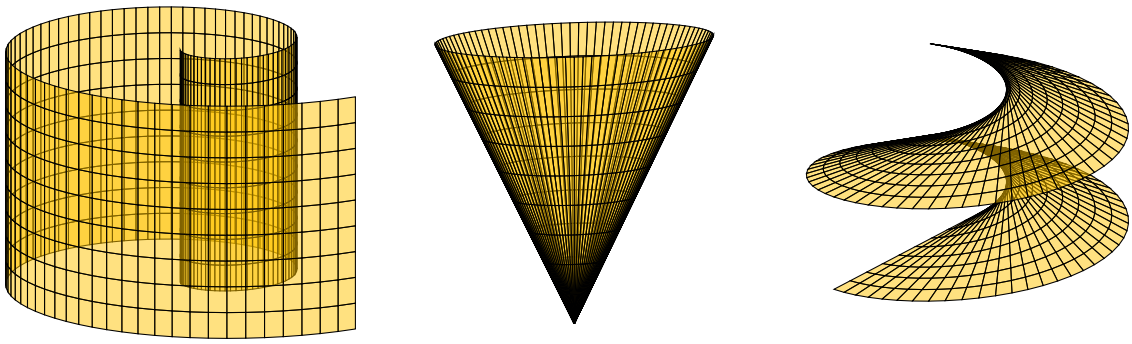


Figure 2.4: Types of developable surfaces

2.2 Discrete net approaches

Another view on developable surfaces is to classify them as surfaces with vanishing Gaussian curvature. This means that developable surfaces are only single curved, one of the principal curvatures in every point on the surface is zero.

The well-known Steiner formula of Jakob Steiner describes the volume $V^d(M)$ generated between a surface M and an offset with distance $d > 0$.

$$V^d(M) = d \cdot A(M) - d^2 \int H dO + \frac{d^3}{3} \int K dO$$

where $A(M)$ is the area, H is the mean curvature and K is the Gaussian curvature of M. All integrals are surface integrals on M.

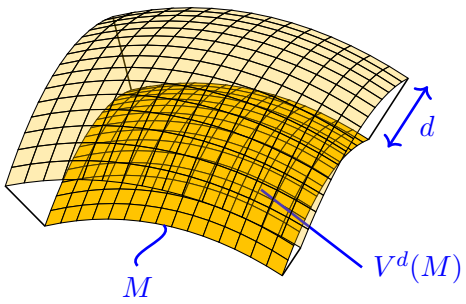


Figure 2.5: Offset of a surface

Analogous, like presented by Cohen-Steiner and Morvan in their paper [16], this volume can be calculated on the offset of a discrete net with planar faces. Let $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a discrete net with vertices $\mathcal{V}$, edges $\mathcal{E}$ and faces $\mathcal{F}$ (more details on the definition of discrete nets can be found in section 3.1). Let further $A(f)$ be the area of the face $f \in \mathcal{F}$, $l(e)$ be the length of the edge $e \in \mathcal{E}$, α_e the angle between the face normals adjacent to e, and β_v^i be the inner angles

between two edges connected to the vertex $v \in \mathcal{V}$. Then one will get

$$\begin{aligned} V^d(f) &= d \cdot A(f) \\ V^d(e) &= d^2 \frac{\alpha_e}{2} l(e) \\ V^d(v) &= \frac{d^3}{3}\left(2\pi - \sum_{i=1}^{n_v} \beta_v^i\right) \end{aligned}$$

where n_v is the number of edges connected to v. This gives a discrete version of the Steiner formula on discrete nets:

$$\begin{aligned} V^d(M) &= \sum_{f\in F} V^d(f) + \sum_{e\in E} V^d(e) + \sum_{v\in V} V^d(v) \\ &= d \cdot \sum_{f\in F} A(f) + d^2 \sum_{e\in E} \frac{\alpha_e}{2} l(e) + \frac{d^3}{3} \sum_{v\in V} \left(2\pi - \sum_{i=1}^{n_v} \beta_v^i\right) \end{aligned}$$

Finally, one gets the definition of the discrete Gaussian curvature in a vertex $v \in V$ by

$$K(v) := 2\pi - \sum_{i=1}^{n_v} \beta_v^i.$$

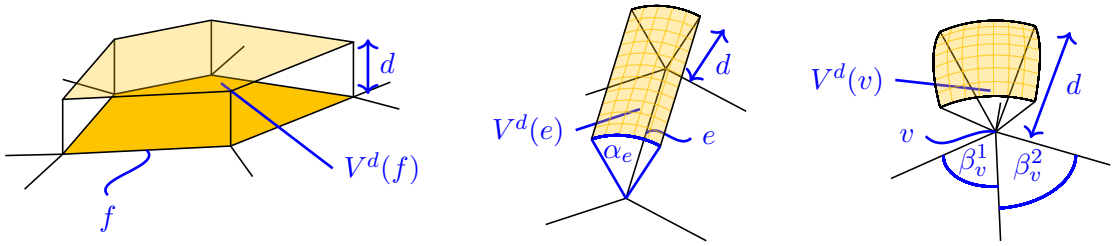
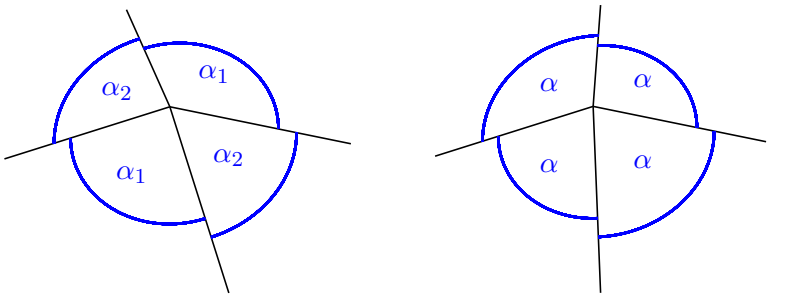


Figure 2.6: Offset of discrete nets

For instance, the approaches of Wang and Tang [17] or Tang and Chen [18] use this discrete Gaussian curvature to optimize a discrete net directly. They generate a nearly developable surface by implementing the condition $\sum_{i=1}^{n} \beta_v^i = 2\pi$ for every vertex in their optimizers. Notice that discrete net offsets leads to the need of planar faces, so both approaches work on triangle nets. However, subdivision algorithms, like the algorithm of Catmull-Clark discussed in section 3.1, often deliver non-planar discrete nets. Also notice that [18] uses an optimizer based on linearization and least squares very similar to the guided projection solver of [11].

Since a smooth developable surface is described by an isometric mapping, it can be parametrized by orthogonal geodesics. This is motivated and proven in the approach of Rabinovich, Hoffmann and Sorkine-Hornung [19] and used by defining a discrete orthogonal geodesic net. A quadrilateral net is identified there as a discrete orthogonal geodesic net if for every vertex all angles between adjacent edges are equal (see figure 2.7). This definition can be used to develop an optimization tool to generate developable surfaces directly via discrete nets. Notice that the used nets there are quadrilateral nets with regular vertices only, i.e. all inner vertices have 4 adjacent vertices. Subdivision algorithms like Catmull-Clark do not yield such nets in general and they do not require such nets as control nets.



Discrete geodesic net Discrete orthogonal geodesic net

Figure 2.7: Angles in discrete geodesic nets

Other approaches related to discrete nets subdivide the given net to generate (nearly) developable strips. These strips can then be glued together to form a piecewise developable model or surface (see Mitani and Suzuki [20]). Alternatively, they can be modeled by planar-quad strips (PQ strips) as discrete analogue of developable surfaces (see Liu et al. [21]). The fact that PQ strips are a discrete version of developable surfaces can also be motivated by the observations on Bézier surfaces from the last section 2.1 mentioned for instance in [5] or [7].

Another interesting approach related to this thesis is [22] of Rohmer et al. The approach takes a polygonal 2D pattern and a 3D contour representing a 3D positioning of the 2D pattern as inputs. The approach separates the 2D pattern by lines what generates curves in the 3D contour with the goal to make the newly generated parts of the 3D contour more isometric to their 2D pattern counterparts. So this approach makes direct use of the idea of mapping developable surfaces to a planar domain.

Chapter 3

Concepts Used

This chapter will target the used techniques to achieve the goals of the approach. It will discuss the ideas of the creators of the concepts and explain in detail how they are used in the context of this thesis.

3.1 Catmull-Clark surfaces

The subdivision surface algorithm of Edwin Catmull and James Clark, first presented in [2], is one of the most popular subdivision algorithms in discrete geometry and various applications. Let's first fix the definition of a discrete net.

Definition 3.1.1 (Discrete net)**.** The triple $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ is called a discrete net iff

1. $\mathcal{V} \subset \mathbb{R}^3$ is a set of points in $\mathbb{R}^3$. These points are called vertices.
   (a) A set $e = \{v_1, v_2\} \subset \mathcal{V}$ of two vertices is called an edge.
   (b) Two edges e_1, e_2 are called adjacent to each other, if they have exactly one vertex $v \in \mathcal{V}$ in common, or in other words $e_1 \cap e_2 = \{v\}$.
   (c) A set of edges $\{e_1, \ldots, e_m\}$ is called a closed ring, if the edges e_i and e_{i+1} are adjacent for every $1 \leq i \leq m-1$. Additional e_1 and e_m have to be adjacent edges.
2. $\mathcal{E}$ is a set of edges.
3. $\mathcal{F}$ is a set of closed rings of edges. The elements of $\mathcal{F}$ are called faces.
4. Let $f \in \mathcal{F}$ be an arbitrary face and $\{v_1, v_2\} \in f$ be an edge of the face. Then $\{v_1, v_2\} \in \mathcal{E}$.
5. If $e \in \mathcal{E}$, there has to exist a face $f \in \mathcal{F}$ such that $e \in f$.

The vertices shared by the adjacent edges of a face $f \in \mathcal{F}$ in definition 3.1.1 are called the adjacent vertices of the face. It is easy to see, that the face $f = \{e_1, \ldots, e_m\}$ with m adjacent edges has also m adjacent vertices.

Let $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be an arbitrary discrete net with vertex set $\mathcal{V}$, edge set $\mathcal{E}$ and face set $\mathcal{F}$. A subdivision algorithm should provide a recursive way to produce finer nets $(\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$, where i indicates the step of iteration (so $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ can also be called $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$). If i grows towards infinity, the nets $(\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$ should converge to a so called limit surface x, which should possess specific properties linked to smoothness. In addition the algorithm should of course be intuitive in a sense that the resulting limit surface should be of "similar" shape like the discrete starting net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. This would give the user a tool to describe smooth surfaces by defining only coarse nets.

The algorithm of Catmull-Clark starts with an arbitrary net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ but produces in every case and in every step i a new discrete net $(\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$ with only quadrilateral faces $f \in \mathcal{F}^i$. But it may yield subdivided nets including vertices with other valence than 4, so called irregular vertices. It produces such vertices in exactly two cases:

1. If the input net $(\mathcal{V}^{i-1}, \mathcal{E}^{i-1}, \mathcal{F}^{i-1})$ contains an irregular vertex.
2. If the input net $(\mathcal{V}^{i-1}, \mathcal{E}^{i-1}, \mathcal{F}^{i-1})$ contains a non-quadrilateral face. Since every output net of Catmull-Clark is a quadrilateral net, this can only happen in case of $i = 1$.

This can easily be verified by a closer look on the description of the algorithm below.

Another property of the algorithm that is useful to keep in mind is, that it produces a new vertex $v \in \mathcal{V}^i$ for every vertex of $\mathcal{V}^{i-1}$, every edge of $\mathcal{E}^{i-1}$ and every face of $\mathcal{F}^{i-1}$. So one will get

$$\forall i \in \mathbb{N} : |\mathcal{V}^i| = |\mathcal{V}^{i-1}| + |\mathcal{E}^{i-1}| + |\mathcal{F}^{i-1}|.$$

In this sense the new net $(\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$ of every iteration is finer than the net $(\mathcal{V}^{i-1}, \mathcal{E}^{i-1}, \mathcal{F}^{i-1})$ was before.

Given an input net $(\mathcal{V}^{i-1}, \mathcal{E}^{i-1}, \mathcal{F}^{i-1})$ the algorithm of Catmull-Clark performs the following steps:

1. For every face $\phi \in \mathcal{F}^{i-1}$ a new vertex
$$v_\phi := \frac{1}{|n_v(\phi)|} \sum_{\nu \in n_v(\phi)} \nu \quad \in \mathcal{V}^i$$
is calculated, where $n_v(\phi) \subset \mathcal{V}^{i-1}$ is the set of adjacent vertices of ϕ (the neighbors of ϕ). See figure 3.1a for an example.
2. For every edge $\epsilon = \{\nu_1, \nu_2\} \in \mathcal{E}^{i-1}$, that connects the vertices $\nu_1, \nu_2 \in \mathcal{V}^{i-1}$, a new vertex $v_\epsilon \in \mathcal{V}^i$ is calculated.
   (a) If ϵ is not a boundary edge (such an edge is called an inner edge), then
   $$v_\epsilon := \frac{1}{4}(\nu_1 + \nu_2 + v_{\phi_1} + v_{\phi_2})$$
   where $\phi_1, \phi_2 \in \mathcal{F}^{i-1}$ are the adjacent faces of the edge ϵ. So v_{ϕ_1} and v_{ϕ_2} are the previously calculated vertices for the faces ϕ_1 and ϕ_2 from step 1. Compare figure 3.1b.
   (b) If ϵ is a boundary edge, then
   $$v_\epsilon := \frac{1}{2}(\nu_1 + \nu_2)$$
   like shown in figure 3.1c.
3. For every vertex $\nu \in \mathcal{V}^{i-1}$ a new vertex $v_\nu \in \mathcal{V}^i$ is calculated.
   (a) If ν is not a boundary vertex (so ν is an inner vertex), then
   $$v_\nu := \frac{|n_v(\nu)| - 2}{|n_v(\nu)|}\nu + \frac{1}{|n_v(\nu)|^2} \sum_{\mu \in n_v(\nu)} \mu + \frac{1}{|n_v(\nu)|^2} \sum_{\phi \in n_f(\nu)} v_\phi$$
   where $n_v(\nu) \subset \mathcal{V}^{i-1}$ are the adjacent vertices of ν, also called neighbors of ν or one-ring of ν. $n_f(\nu) \subset \mathcal{F}^{i-1}$ are the adjacent faces of ν, so v_ϕ are again vertices from step 1. This is shown in figure 3.1d.

(b) If ν is a boundary vertex, then it can be marked as a corner vertex. In that case, it should remain fixed, so $v_\nu := \nu$. If it is not a corner vertex, then there are two neighbor vertices $\nu_1, \nu_2 \in \mathcal{V}^{i-1}$ of ν which are also boundary vertices. Then

$$v_\nu := \frac{3}{4}\nu + \frac{1}{8}\nu_1 + \frac{1}{8}\nu_2$$

like in figure 3.1e.

4. Let $\phi \in \mathcal{F}^{i-1}$ be a face of the old net, $\nu \in n_v(\phi) \subset \mathcal{V}^{i-1}$ an adjacent vertex of ϕ and $\epsilon_1, \epsilon_2 \in \mathcal{E}^{i-1}$ the two edges which have ν in common and are adjacent to ϕ. Then $\{\{v_\phi, v_{\epsilon_1}\}, \{v_{\epsilon_1}, v_\nu\}, \{v_\nu, v_{\epsilon_2}\}, \{v_{\epsilon_2}, v_\phi\}\} \in \mathcal{F}^i$ forms a face of the new net.

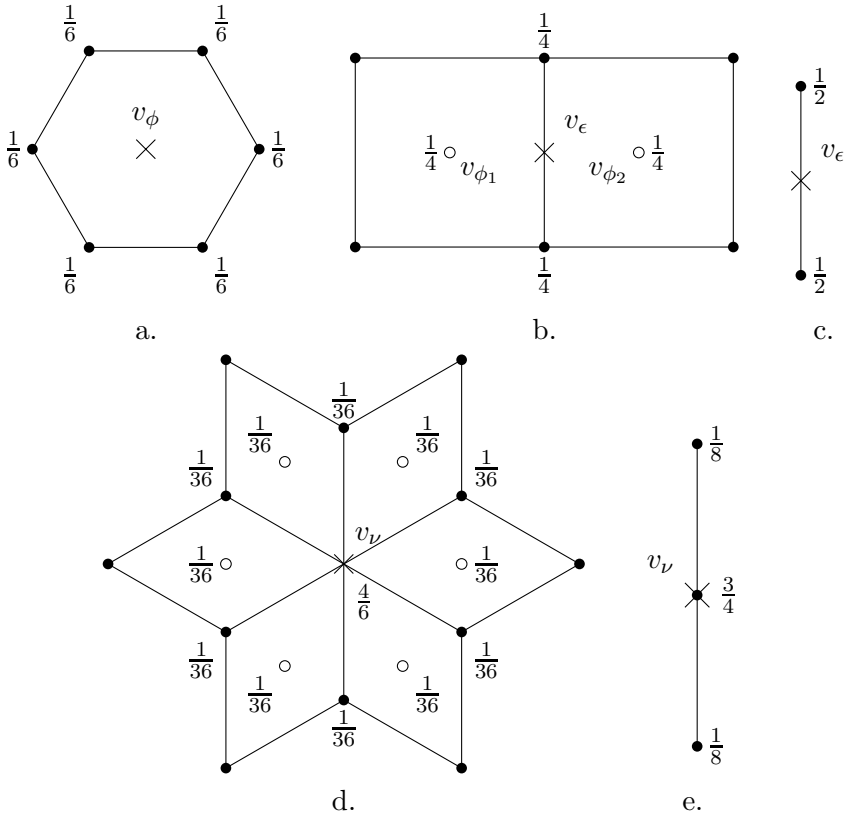


Figure 3.1: Algorithm of Catmull-Clark

The algorithm of Catmull-Clark is implemented in the approach of this thesis like presented above. Step 3b refers to marked corner vertices. This is implemented as a Boolean property in the net data structure or in other words, every saved vertex of a discrete net has a Boolean property called `isCorner` that marks it as corner vertex. In a design tool it maybe makes sense to let the user decide which vertices of the control net are corners. However, for the sake of simplicity the implemented demonstration tools will provide control nets $(\mathcal{V}, \mathcal{E}, \mathcal{F}) = (\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$ of fixed combinatorics and corner vertices. Another Boolean vertex marker to improve the implementation is `allowGeodesic` mentioned in section 4.2.3.3.

3.2 Guided projection

The presented approach will give a list of quadratic equations which have to be solved. This problem is a very common one targeted in various theses and papers in literature like for instance in [11] of Tang et al. To solve it, they've used a technique called guided projection.

Let $\phi_i(x) : \mathbb{R}^n \to \mathbb{R}, i \in \{1, \ldots, N\}$ be quadratic functions and assume that one searches for a solution $x \in \mathbb{R}^n$ that satisfies all equations $\phi_i(x) \stackrel{!}{=} 0$. Since the functions ϕ_i are quadratic, they can be noted as

$$\phi_i(x) = \frac{1}{2} x^T A_i x + b_i^T x + c_i$$

where $A_i \in \mathbb{R}^{n \times n}$ are constant matrices, $b_i \in \mathbb{R}^n$ are constant vectors and $c_i \in \mathbb{R}$ are constant scalars. The basic idea of the guided projection solver is to linearize these equations and find a solution recursively. Then one gets

$$\phi_i(x_{n+1}) \approx \phi_i(x_n) + \nabla\phi_i(x_n)(x_{n+1} - x_n) \stackrel{!}{=} 0.$$

To find a new solution iteration $x_{n+1} \in \mathbb{R}^n$ from the old iteration x_n one has to solve the linear system

$$Hx_{n+1} = r \tag{3.1}$$

where

$$H = \begin{pmatrix} \nabla\phi_1(x_n)^T \\ \vdots \\ \nabla\phi_N(x_n)^T \end{pmatrix}$$

and

$$r = \begin{pmatrix} -\phi_1(x_n) + \nabla\phi_1(x_n)^T x_n \\ \vdots \\ -\phi_N(x_n) + \nabla\phi_N(x_n)^T x_n \end{pmatrix}.$$

Fortunately, the evaluation of the $\nabla\phi_i(x) = A_i x + b_i$ is easy, since $\phi_i(x)$ are quadratic functions.

The list of equations $\phi_i(x) = 0$ can be considered as a wish list of desired properties. It often leads to redundant equations or to an underdetermined system 3.1. If the system 3.1 has a solution at all, numerical errors could lead to a small solution space due to the redundant equations. To avoid this, the system has to be enriched by regularization terms.

Let $(k_i^T x - s_i)^2$ be so called fairness or energy terms, where $i \in \{1, \ldots, M\}$, $k_i \in \mathbb{R}^n$ are constant vectors and $s_i \in \mathbb{R}$ are constant scalars. Together they can be noted as $\|Kx - s\|^2$ where

$$K = \begin{pmatrix} k_1^T \\ \vdots \\ k_M^T \end{pmatrix}$$

and

$$s = \begin{pmatrix} s_1 \\ \vdots \\ s_M \end{pmatrix}.$$

To avoid the numerical problems and regularize the system 3.1, one searches for a solution that keeps the energy $\|Kx - s\|^2$ small. In addition, the term $\|x_{n+1} - x_0\|^2$ is added for further regularization to keep the new iterations near the starting vector x_0. In the end the guided projection algorithm solves the problem

$$\|Hx_{n+1} - r\|^2 + \epsilon^1\|Kx_{n+1} - s\|^2 + \epsilon^2\|x_{n+1} - x_0\|^2 \to \min \tag{3.2}$$

where $\epsilon^1, \epsilon^2 \in \mathbb{R}$ are small weights. Please note that these weights can vary for each iteration x_n. The strategy of the presented approach for flexible weights during runtime can be found in section 5.5.

3.2.1 Implemented variations

In addition to the original method from Tang et al. [11] there will be implemented a small variation to get more control over different parts of the energy terms. The designed optimizer doesn't cluster all energy terms $(k_i^T x - s_i)^2$ in one group defined by the matrix K and the vector s. Instead, it forms multiple fairness groups by matrices K_j and vectors s_j where $j \in \{1, \dots, m\}$ and $m \in \mathbb{N}$ is the number of fairness groups. So there are $1 < i_1 < i_2 < \cdots < i_m = M$ such that

$$K_1 = \begin{pmatrix} k_1^T \\ \vdots \\ k_{i_1}^T \end{pmatrix}, \quad K_2 = \begin{pmatrix} k_{i_1+1}^T \\ \vdots \\ k_{i_2}^T \end{pmatrix}, \quad \dots \quad K_m = \begin{pmatrix} k_{i_{m-1}+1}^T \\ \vdots \\ k_M^T \end{pmatrix}$$

and

$$s_1 = \begin{pmatrix} s_1 \\ \vdots \\ s_{i_1} \end{pmatrix}, \quad s_2 = \begin{pmatrix} s_{i_1+1} \\ \vdots \\ s_{i_2} \end{pmatrix}, \quad \dots \quad s_m = \begin{pmatrix} s_{i_{m-1}+1} \\ \vdots \\ s_M \end{pmatrix}.$$

This has the advantage that different fairness groups can be weighted by different weights $\epsilon_j^1 \in \mathbb{R}$. Instead of 3.2 there will be solved

$$F(x_{n+1}) := \|Hx_{n+1} - r\|^2 + \sum_{j=1}^{m} \left(\epsilon_j^1 \|K_j x_{n+1} - s_j\|^2\right) + \epsilon^2\|x_{n+1} - x_0\|^2 \to \min. \tag{3.3}$$

To solve the problem 3.3 one searches for the zeros of the gradient of the function that should be minimized. Since the function is quadratic this leads to a linear system, namely

$$\left(H^T H + \sum_{j=1}^{m}(\epsilon_j^1 K_j^T K_j) + \epsilon^2 I\right) x_{n+1} = \left(H^T r + \sum_{j=1}^{m}(\epsilon_j^1 K_j^T s_j) + \epsilon^2 x_0\right). \tag{3.4}$$

The matrix of this system is clearly positive semidefinite and also positive definite, if the energy terms regularize the matrix sufficiently. So a Cholesky decomposition can be used for high performance solving.

To make the used optimization more stable, $F(x_{n+1})$ is compared to $F(x_n)$ after solving, where F is the objective function from equation 3.3. If $F(x_{n+1}) > F(x_n)$, then a smaller step size $0 < \delta < 1$ and

$$\hat{x}_{n+1} := x_n + \delta \cdot (x_{n+1} - x_n)$$

is used as the new vector instead of x_{n+1}.

More information about the technical details of the implementation are presented in section 5.4.

Chapter 4

Approach

This chapter describes the idea of the approach of this thesis. The goal of the approach is to get a method to interactively model a surface by dragging control points of a subdivided net but keeping the resulting surface nearly developable.

In chapter 2 developability is achieved by considering surfaces as special ruled surfaces or via keeping the Gaussian curvature of the surface small. In the approach of this chapter it is directly used that developable surfaces are surfaces which can be isometrically mapped to a planar domain. The connection between a surface in $\mathbb{R}^3$ and an isometric counterpart in $\mathbb{R}^2$ is important for the approach and will give an unfolding of the modeled surface at any time, similar to the approach [22].

Since the used nets will be subdivided by the algorithm of Catmull-Clark (see section 3.1), they can contain irregular vertices. Hence the presented approach, in contrast to most existing approaches, has to be able to handle such vertices.

Additionally, to get more control over the term "nearly developable", section 4.1.1 of the approach will introduce upper and lower bounds for allowed relative distortions of the surface in $\mathbb{R}^3$ in comparison to the net in the plane.

4.1 Target Equations

This section describes the used target equations ϕ_i of the approach for the implemented guided projection solver from section 3.2.

4.1.1 Developability

Let $x : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ be a regular surface. Since x is regular, $\frac{\partial x}{\partial u}$ and $\frac{\partial x}{\partial v}$ exist for every point $(u, v)^T \in U$ and are linearly independent. The information about the distortion of x is contained in the matrix of its first fundamental form

$$I = \begin{pmatrix} \frac{\partial x}{\partial u}^2 & \frac{\partial x}{\partial u}\frac{\partial x}{\partial v} \\ \frac{\partial x}{\partial u}\frac{\partial x}{\partial v} & \frac{\partial x}{\partial v}^2 \end{pmatrix} = \begin{pmatrix} x_u^2 & x_u x_v \\ x_u x_v & x_v^2 \end{pmatrix} = \begin{pmatrix} E & F \\ F & G \end{pmatrix}.$$

Let $a = (a_1, a_2)^T$ be a unit vector fixed in a point $(u, v)^T \in U$ in the parameter domain. Then x maps this vector to the vector

$$t = a_1 x_u + a_2 x_v = \begin{pmatrix} x_u & x_v \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}$$

fixed in the point $x(u,v)$. The length of t compared to the length of a (which is 1) is the distortion of x in the direction of a. So one will observe

$$t^2 = \begin{pmatrix} a_1 & a_2 \end{pmatrix} \begin{pmatrix} x_u^2 & x_u x_v \\ x_u x_v & x_v^2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \end{pmatrix} I \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}. \tag{4.1}$$

The maximum and minimum value of the length of t are the principal distortions of x in $(u,v)^T$. To get these quantities, one has to find the extrema of the quadratic form t^2 under the constraint $\|a\|^2 = 1$, what leads to a simple eigenvalue problem.

Let $x : U \subset \mathbb{R}^2 \to \mathbb{R}^3$ now be a developable surface and $\bar{x} : U \to \mathbb{R}^3$ its unfolding, hence $\bar{x}$ lies in a plane in $\mathbb{R}^3$. Since x can be mapped without stretching or tearing to the surface $\bar{x}$,

$$I = \bar{I} \tag{4.2}$$

must hold in every point $(u,v) \in U$, where $\bar{I} = \left(\begin{smallmatrix} \bar{E} & \bar{F} \\ \bar{F} & \bar{G} \end{smallmatrix} \right)$ is the first fundamental form of $\bar{x}$.

To sharpen the term "nearly developable" the approach should provide upper and lower bounds for the distortion of x in relation to the distortion of $\bar{x}$. This relative distortion can be obtained by proceeding similar to the absolute distortion motivated above. But instead of searching the extrema of 4.1 under the constraint

$$\|a\|^2 = \begin{pmatrix} a_1 & a_2 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = 1$$

one uses the constraint

$$a^T \bar{I} a = \begin{pmatrix} a_1 & a_2 \end{pmatrix} \begin{pmatrix} \bar{x}_u^2 & \bar{x}_u \bar{x}_v \\ \bar{x}_u \bar{x}_v & \bar{x}_v^2 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = 1.$$

This leads to the generalized eigenvalue problem $\det(I - \lambda \bar{I}) = 0$ with the squared principal distortions $\sigma_{1,2} = B \pm \sqrt{B^2 - A}$ as solutions where

$$\begin{aligned} A &= \frac{EG - F^2}{\bar{E}\bar{G} - \bar{F}^2} \\ B &= \frac{\bar{E}G + E\bar{G} - 2F\bar{F}}{2\left(\bar{E}\bar{G} - \bar{F}^2\right)}. \end{aligned}$$

Let $0 < \gamma_1^2 < \gamma_2^2$, then one can claim $\gamma_1^2 \le \sigma_{1,2} \le \gamma_2^2$ to determine bounds for the relative distortion of x compared to its unfolding $\bar{x}$. By squaring the inequalities $\sqrt{B^2 - A} \le B - \gamma_1^2$ and $\sqrt{B^2 - A} \le \gamma_2^2 - B$ this would lead to

$$0 \le \gamma_1^4 \left(\bar{E}\bar{G} - \bar{F}^2\right) - \gamma_1^2 \left(\bar{E}G + E\bar{G} - 2F\bar{F}\right) + \left(EG - F^2\right) \tag{4.3}$$
$$0 \le \gamma_2^4 \left(\bar{E}\bar{G} - \bar{F}^2\right) - \gamma_2^2 \left(\bar{E}G + E\bar{G} - 2F\bar{F}\right) + \left(EG - F^2\right). \tag{4.4}$$

If in contrast the inequalities 4.3 and 4.4 hold, it is necessary to additionally claim $B - \gamma_1^2 \ge 0$ and $\gamma_2^2 - B \ge 0$ or

$$0 \le -2\gamma_1^2 \left(\bar{E}\bar{G} - \bar{F}^2\right) + \left(\bar{E}G + E\bar{G} - 2F\bar{F}\right) \tag{4.5}$$
$$0 \le 2\gamma_2^2 \left(\bar{E}\bar{G} - \bar{F}^2\right) - \left(\bar{E}G + E\bar{G} - 2F\bar{F}\right) \tag{4.6}$$

to satisfy $\gamma_1^2 \le \sigma_{1,2} \le \gamma_2^2$.

4.1.2 Discretization

So far all ideas about the approach were made on continuous surfaces. Let now $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a discrete net with vertices $\mathcal{V}$, edges $\mathcal{E}$ and faces $\mathcal{F}$ like in definition 3.1.1, but let $\mathcal{F}$ only contain quadrilateral faces. This constraint may first seem to be restrictive, but it will satisfy the requirements of this approach, since the designed optimization will operate on Catmull-Clark iterations. Like described in section 3.1, every iteration in the algorithm of Catmull-Clark yields a quadrilateral net.

To map the ideas about developability from subsection 4.1.1 to discrete nets, one has to find a discrete version of the quantities E, F and G of the first fundamental form of the discrete surface described by the given net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. In comparison to the derivatives x_u and x_v in section 4.1.1 one can use the vectors given by the edges $\mathcal{E}$ of the discrete net. So one can define

Definition 4.1.1. Let $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a discrete net that contains only quadrilateral faces.

1. Let $e = \{v_0, v_1\} \in \mathcal{E}$ be an edge of the discrete net. Then

$$E_e := (v_1 - v_0)^2$$

2. Let $f = \{e_0, e_1, e_2, e_3\}$ be a face of the discrete net with $e_0 = \{v_0, v_1\}, e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}, e_3 = \{v_3, v_0\}$. Then

$$\begin{aligned} F_{f,0} &:= (v_1 - v_0)(v_3 - v_0) \\ F_{f,1} &:= (v_1 - v_0)(v_2 - v_1) \\ F_{f,2} &:= (v_2 - v_1)(v_2 - v_3) \\ F_{f,3} &:= (v_3 - v_0)(v_2 - v_3) \end{aligned}$$

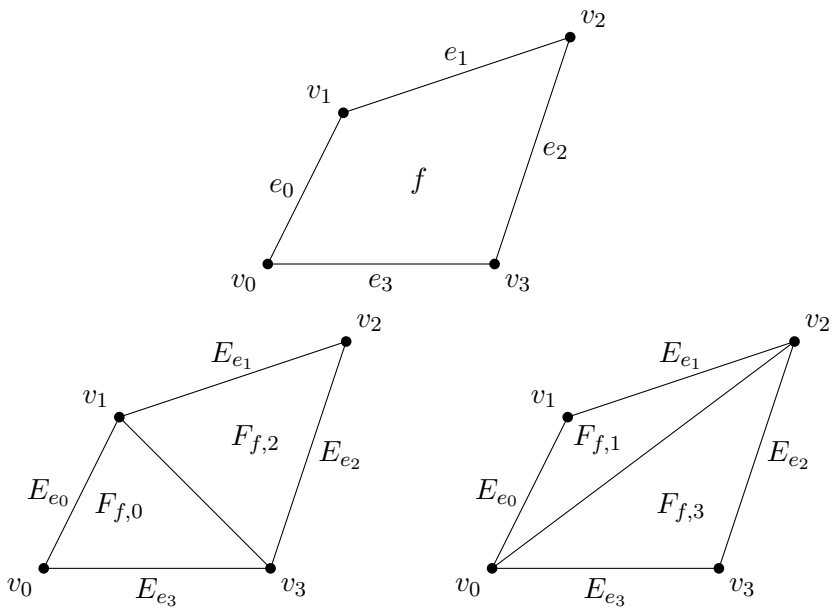


Figure 4.1: Indices in a quad of a quadrilateral net

In addition to definition 4.1.1 one has to get an idea of the unfolding of a discrete net.

Definition 4.1.2 (Discrete Unfolding)**.** Let $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a discrete net that contains only quadrilateral faces. A discrete net $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ is called an unfolding of $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ iff

1. All vertices of $\bar{\mathcal{V}}$ lie in a plane.

2. $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ is of the same combinatorics as $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. This means there is a bijection $\sigma : \mathcal{V} \to \bar{\mathcal{V}}$ with

   $$\forall v_0, v_1 \in \mathcal{V} : \{v_0, v_1\} \in \mathcal{E} \Leftrightarrow \{\sigma(v_0), \sigma(v_1)\} \in \bar{\mathcal{E}}$$

   and a bijection $\tau : \mathcal{E} \to \bar{\mathcal{E}} : \{v_0, v_1\} \mapsto \{\sigma(v_0), \sigma(v_1)\}$ with

   $$\forall e_0, e_1, e_2, e_3 \in \mathcal{E} : \{e_0, e_1, e_2, e_3\} \in \mathcal{F} \Leftrightarrow \{\tau(e_0), \tau(e_1), \tau(e_2), \tau(e_3)\} \in \bar{\mathcal{F}}$$

3. There is an isometric mapping between $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ and $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$. So let $\{v_0, v_1, v_2, v_3\} \subset \mathcal{V}$ be the adjacent vertices of a face $f \in \mathcal{F}$. Then there has to hold

   $$\forall v, w \in \{v_0, v_1, v_2, v_3\} : \|v - w\|^2 = \|\sigma(v) - \sigma(w)\|^2$$

Especially condition 2 of definition 4.1.2 means that the unfolding of a quadrilateral discrete net is again a quadrilateral discrete net. Although definition 3.1.1 determines that the vertices of an unfolding are elements in $\mathbb{R}^3$ they will often be identified with points in $\mathbb{R}^2$, since they lie in a plane due to condition 1.

Now one can fix quadratic target equations for a guided projection solver from section 3.2 to yield a discrete net that is nearly developable. Let $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ be a discrete net and $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ its unfolding. Let for now x be a vector that gathers all unknown quantities of the optimization. For a detailed listing of all unknowns consider section 4.1.4.

From definition 4.1.1 one gets the equations

- $\phi_{I,e}(x) = (v_1 - v_0)^2 - E_e$ for every edge $e = \{v_0, v_1\} \in \mathcal{E}$.
- $\phi_{I,\bar{e}}(x) = (\bar{v}_1 - \bar{v}_0)^2 - \bar{E}_{\bar{e}}$ for every edge $\bar{e} = \{\bar{v}_0, \bar{v}_1\} \in \bar{\mathcal{E}}$.
- $\phi_{I,f,0}(x) = (v_1 - v_0)(v_3 - v_0) - F_{f,0}$ for every face $f \in \mathcal{F}$ with adjacent vertices $\{v_0, v_1, v_2, v_3\}$ and analogous equations $\phi_{I,f,1}(x)$, $\phi_{I,f,2}(x)$, and $\phi_{I,f,3}(x)$ for the quantities $F_{f,1}$, $F_{f,2}$, and $F_{f,3}$, respectively.
- $\phi_{I,\bar{f},0}(x) = (\bar{v}_1 - \bar{v}_0)(\bar{v}_3 - \bar{v}_0) - \bar{F}_{\bar{f},0}$ for every face $\bar{f} \in \bar{\mathcal{F}}$ with adjacent vertices $\{\bar{v}_0, \bar{v}_1, \bar{v}_2, \bar{v}_3\}$ and analogous equations $\phi_{I,\bar{f},1}(x)$, $\phi_{I,\bar{f},2}(x)$, and $\phi_{I,\bar{f},3}(x)$ for the quantities $\bar{F}_{\bar{f},1}$, $\bar{F}_{\bar{f},2}$, and $\bar{F}_{\bar{f},3}$, respectively.

To use the inequalities 4.3-4.6 with a guided projection solver, one has to transform these inequalities to classic equations. The idea to achieve this is, that if a given quantity $a \in \mathbb{R}$ is greater or equal to zero, it is equal to the square of another quantity $b \in \mathbb{R}$:

$$(a \geq 0) \Leftrightarrow (\exists b \in \mathbb{R} : a = b^2) \tag{4.7}$$

So let $\delta_{f,i,j} \in \mathbb{R}, i \in \{0,\dots,3\}, j \in \{1,\dots,4\}$ be auxiliary variables. These variables help to transform the inequalities from section 4.1.1 to normal equations, namely

$$\begin{aligned}
\phi_{Q,f,i,1}(x) &= \gamma_1^4 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\
&- \gamma_1^2 \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\
&+ \left(E_{e_i}E_{e_{i\ominus 1}} - F_{f,i}^2\right) \\
&- \delta_{f,i,1} \quad \text{and} \\
\phi_{Q,f,i,2}(x) &= \gamma_2^4 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\
&- \gamma_2^2 \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\
&+ \left(E_{e_i}E_{e_{i\ominus 1}} - F_{f,i}^2\right) \\
&- \delta_{f,i,2} \\
\phi_{Q,f,i,3}(x) &= -2\gamma_1^2 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\
&+ \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\
&- \delta_{f,i,3} \\
\phi_{Q,f,i,4}(x) &= 2\gamma_2^2 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\
&- \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\
&- \delta_{f,i,4}
\end{aligned}$$

for every $i \in \{0,\dots,3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$. Please note that $\bar{e} = \{\sigma(v_0), \sigma(v_1)\} \in \bar{\mathcal{E}}$ and $\bar{f} = \{\tau(e_0), \tau(e_1), \tau(e_2), \tau(e_3)\} \in \bar{\mathcal{F}}$ due to definition 4.1.2 and $i \ominus 1 := (i-1) \mod 4$. To get a better feeling about the many indices in the last equations it may help to consult figure 4.1.

4.1.3 Catmull-Clark surfaces

Like mentioned in previous sections the goal of the approach will be to give the user a modeling tool for nearly developable surfaces. To achieve this, a control net of a Catmull-Clark surface from section 3.1 is used to model a surface. The user then drags the vertices of a coarse net to create a specific surface. To link this idea to the approach so far, there is a need for more target equations.

Let $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$ be an arbitrary discrete net, that does not necessarily contain only quads, whose purpose is to define a control net. Let further $(\mathcal{V}, \mathcal{E}, \mathcal{F}) = (\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$ be the resulting discrete net after $i > 0$ iterations of the algorithm of Catmull-Clark on the net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$. Due to the properties of Catmull-Clark $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ is now a quadrilateral discrete net and the ideas of the approach so far can be applied.

Every vertex $v \in \mathcal{V}$ is a convex combination of vertices $\{V_0, \dots, V_m\} \subset \mathcal{V}_0$, so there holds

$$v = \sum_i \alpha_{v,i} V_i \tag{4.8}$$

where $\alpha_{v,i} \in \mathbb{R}$ is the weight of vertex V_i. This leads to new target equations.

Let $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ be the discrete unfolding of $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. Then there exists an unfolded control net $(\bar{\mathcal{V}}^0, \bar{\mathcal{E}}^0, \bar{\mathcal{F}}^0)$, such that $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ is the i-th Catmull-Clark iteration of it. Let further be $0 \le j \le i$ and $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$ the j-th Catmull-Clark iteration of the unfolded control net. This net will be called the intermediate unfolded net or just intermediate net because it lies "between"

the nets $(\bar{\mathcal{V}}^0, \bar{\mathcal{E}}^0, \bar{\mathcal{F}}^0)$ and $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$. Again the vertices $\bar{v} \in \bar{\mathcal{V}}$ are convex combinations of the vertices $\bar{V} \in \bar{\mathcal{V}}^j$, since $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ is the $(i-j)$-th Catmull-Clark iteration of the intermediate net. Let $\alpha_{v,i} \in \mathbb{R}$ and $\bar{\alpha}_{\bar{v},i} \in \mathbb{R}$ be the respective weights of these convex combinations like in equation 4.8. Then one gets the target equations

- $\phi_{S,v}(x) = v - \sum_i \alpha_{v,i} V_i$ for every vertex $v \in \mathcal{V}$.
- $\phi_{S,\bar{v}}(x) = \bar{v} - \sum_i \bar{\alpha}_{\bar{v},i} \bar{V}_i$ for every vertex $\bar{v} \in \bar{\mathcal{V}}$.

Please notice that the above equations are 5 equations in total, since the vertices of $\mathcal{V}$ are elements in $\mathbb{R}^3$ and the vertices $\bar{\mathcal{V}}$ of the unfolding are elements in $\mathbb{R}^2$.

The idea of the intermediate net seems to appear here unmotivated. But it will provide a tool to easily extend the degrees of freedom of the optimization. So far there appear a great bunch of unknowns in the optimization (as seen in detail in section 4.1.4). In addition

- the vertices $V = (V_0, V_1, V_2) \in \mathcal{V}^0 \subset \mathbb{R}^3$ and
- the vertices $\bar{V} = (\bar{V}_0, \bar{V}_1) \in \bar{\mathcal{V}}^j \subset \mathbb{R}^2$

appear from now on as unknowns. But the degrees of freedom of the optimization are very limited because of the great amount of target equations defined.

To give the user more control over the design process, it will be handy to have a tool to fix vertices. This can be achieved easily by the target equations

$$\phi_{H,V}(x) = V - W_V$$

where $W_V \in \mathbb{R}^3$ are fixed points to hold the specified vertex $V \in \mathcal{V}^0$ in place. The number of equations of this type for the optimization depends on the number of vertices held by the user. Also notice that, similar like before, this equation is in actual fact 3 equations in total.

4.1.4 Summary

The setup of the approach is a discrete quadrilateral net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ which is the i-th Catmull-Clark iteration of the control net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$, where $0 < i \in \mathbb{N}$. Furthermore, let $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ be the discrete unfolding of $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ is the i-th Catmull-Clark iteration of the net $(\bar{\mathcal{V}}^0, \bar{\mathcal{E}}^0, \bar{\mathcal{F}}^0)$ and the $(i-j)$-th iteration of the intermediate net $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$, where $0 \leq j \leq i$.

The unknowns of the optimization are

- the vertices $v = (v_0, v_1, v_2)^T \in \mathcal{V} \subset \mathbb{R}^3$ of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$,
- the vertices $\bar{v} = (\bar{v}_0, \bar{v}_1)^T \in \bar{\mathcal{V}} \subset \mathbb{R}^2$ of the unfolding $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$,
- the vertices $V = (V_0, V_1, V_2)^T \in \mathcal{V}^0 \subset \mathbb{R}^3$ of the control net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$,
- the vertices $\bar{V} = (\bar{V}_0, \bar{V}_1)^T \in \bar{\mathcal{V}}^j \subset \mathbb{R}^2$ of the intermediate net $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$,
- the quantities E_e for every edge $e \in \mathcal{E}$,
- the quantities $\bar{E}_{\bar{e}}$ for every edge $\bar{e} \in \bar{\mathcal{E}}$,
- the quantities $F_{f,0}, F_{f,1}, F_{f,2}, F_{f,3}$ for every face $f \in \mathcal{F}$,
- the quantities $\bar{F}_{\bar{f},0}, \bar{F}_{\bar{f},1}, \bar{F}_{\bar{f},2}, \bar{F}_{\bar{f},3}$ for every face $\bar{f} \in \bar{\mathcal{F}}$ and

- some auxiliary variables $\delta_{f,i,j} \in \mathbb{R}, i \in \{0,\dots,3\}, j \in \{1,\dots,4\}$ for every face $f \in \mathcal{F}$ for the inequalities 4.3-4.6

which are gathered in a single vector x. The approach delivers the target equations

1. **Discrete first fundamental form**
   (a) $\phi_{I,e}(x) = (v_1 - v_0)^2 - E_e$ for every edge $e = \{v_0, v_1\} \in \mathcal{E}$
   (b) $\phi_{I,\bar{e}}(x) = (\bar{v}_1 - \bar{v}_0)^2 - \bar{E}_{\bar{e}}$ for every edge $\bar{e} = \{\bar{v}_0, \bar{v}_1\} \in \bar{\mathcal{E}}$
   (c) $\phi_{I,f,0}(x) = (v_1 - v_0)(v_3 - v_0) - F_{f,0}$ for every face $f \in \mathcal{F}$ with adjacent vertices $\{v_0, v_1, v_2, v_3\}$ and $\phi_{I,f,1}(x)$, $\phi_{I,f,2}(x)$, and $\phi_{I,f,3}(x)$ for $F_{f,1}$, $F_{f,2}$, and $F_{f,3}$.
   (d) $\phi_{I,\bar{f},0}(x) = (\bar{v}_1 - \bar{v}_0)(\bar{v}_3 - \bar{v}_0) - \bar{F}_{\bar{f},0}$ for every face $\bar{f} \in \bar{\mathcal{F}}$ with adjacent vertices $\{\bar{v}_0, \bar{v}_1, \bar{v}_2, \bar{v}_3\}$ and $\phi_{I,\bar{f},1}(x)$, $\phi_{I,\bar{f},2}(x)$, and $\phi_{I,\bar{f},3}(x)$ for $\bar{F}_{\bar{f},1}$, $\bar{F}_{\bar{f},2}$, and $\bar{F}_{\bar{f},3}$.

2. **Distortion and Quality**
   (a) $$\begin{aligned}\phi_{Q,f,i,1}(x) = {} & \gamma_1^4 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\ & - \gamma_1^2 \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\ & + \left(E_{e_i}E_{e_{i\ominus 1}} - F_{f,i}^2\right) \\ & - \delta_{f,i,1}\end{aligned}$$
   for every $i \in \{0,\dots,3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$
   (b) $$\begin{aligned}\phi_{Q,f,i,2}(x) = {} & \gamma_2^4 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\ & - \gamma_2^2 \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\ & + \left(E_{e_i}E_{e_{i\ominus 1}} - F_{f,i}^2\right) \\ & - \delta_{f,i,2}\end{aligned}$$
   for every $i \in \{0,\dots,3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$
   (c) $$\begin{aligned}\phi_{Q,f,i,3}(x) = {} & -2\gamma_1^2 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\ & + \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\ & - \delta_{f,i,3}\end{aligned}$$
   for every $i \in \{0,\dots,3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$
   (d) $$\begin{aligned}\phi_{Q,f,i,4}(x) = {} & 2\gamma_2^2 \left(\bar{E}_{\bar{e}_i}\bar{E}_{\bar{e}_{i\ominus 1}} - \bar{F}_{\bar{f},i}^2\right) \\ & - \left(\bar{E}_{\bar{e}_i}E_{e_{i\ominus 1}} + E_{e_i}\bar{E}_{\bar{e}_{i\ominus 1}} - 2F_{f,i}\bar{F}_{\bar{f},i}\right) \\ & - \delta_{f,i,4}\end{aligned}$$
   for every $i \in \{0,\dots,3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$

3. **Catmull-Clark subdivision**
   (a) $\phi_{S,v}(x) = v - \sum_i \alpha_{v,i} V_i$ for every vertex $v \in \mathcal{V}$
   (b) $\phi_{S,\bar{v}}(x) = \bar{v} - \sum_i \bar{\alpha}_{\bar{v},i} \bar{V}_i$ for every vertex $\bar{v} \in \bar{\mathcal{V}}$

4. **Held control vertices**
   (a) $\phi_{H,V}(x) = V - W_V$ for every fixed vertex $V \in \mathcal{V}^0$

for the guided projection solver.

An early version of the implementation of the approach also used the target equations

5. **Developability**

   (a) $\phi_{D,e}(x) = E_e - \bar{E}_{\bar{e}}$ for every edge $e = \{v_0, v_1\} \in \mathcal{E}$

   (b) $\phi_{D,f,i}(x) = F_{f,i} - \bar{F}_{\bar{f},i}$ for every $i \in \{0, \ldots, 3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$

yielded by equation 4.2 of section 4.1.1. Although these equations improve the optimizer to deliver very good developable surfaces, it is also a goal of the approach to get more control over the term "nearly developable" via the target equations 2 above. Since the results showed, that the developability equations completely overrule the equations for distortion, they were shifted to the fairness terms in section 4.2.1. However, the influence of these equations can be seen in section 6.4.

4.2 Fairness

Like described in section 3.2 the direct solving of the found target equations would likely lead into numerical troubles and the solution would be bad. To avoid this the guided projection solver can be enriched with fairness terms $(k_i^T x - s_i)^2$ which influence the optimization with a small weight. This section describes the fairness terms used by the approach.

Please notice that all used symbols in this section refer to the above summary in section 4.1.4.

4.2.1 Developability

As mentioned in section 4.1.4, the previous target equations for developability were implemented as fairness terms to make the distortion target equations more important to the optimizer. This should enable the optimizer to handle a determined "degree" of developability by the user via distortion bounds.

The implemented fairness terms are

$$\left\| E_e - \left(\frac{\gamma_1 + \gamma_2}{2} \right)^2 \bar{E}_{\bar{e}} \right\|^2$$

for every edge $e = \{v_0, v_1\} \in \mathcal{E}$ of the discrete net in $\mathbb{R}^3$ and

$$\left\| F_{f,i} - \left(\frac{\gamma_1 + \gamma_2}{2} \right)^2 \bar{F}_{\bar{f},i} \right\|^2$$

for every $i \in \{0, \ldots, 3\}$ and face $f = \{e_0, e_1, e_2, e_3\} \in \mathcal{F}$. Again, like in section 4.1.2, $\bar{e} = \{\sigma(v_0), \sigma(v_1)\} \in \bar{\mathcal{E}}$ and $\bar{f} = \{\tau(e_0), \tau(e_1), \tau(e_2), \tau(e_3)\} \in \bar{\mathcal{F}}$ due to definition 4.1.2. γ_1 and γ_2 are again the chosen distortion bounds like in section 4.1.1.

In contrast to the implementation of the equations $E = \bar{E}$ and $F = \bar{F}$ suggested in section 4.1.1, the implementation of the fairness terms above will also work if the allowed mean distortion $\frac{\gamma_1+\gamma_2}{2}$ is not 1. For instance if the user wants to create a surface with distortions between 1.06 and 1.1, the fairness terms would else work "against" the distortion equations (number 2 in section 4.1.4).

4.2.2 Distance to old net

An easy and often used fairness term is the distance of the new solution to the old net. Although this is weakly implemented by the term weighted with $\epsilon^2 \in \mathbb{R}$ of guided projection as seen in equation 3.3, it may be desired to associate a stronger weight to specific parts of the solution vector x. In detail this approach uses the fairness term

$$\|v - v_{v,\text{old}}\|^2$$

for every vertex $v \in \mathcal{V}$ of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. Note that $v_{v,\text{old}} \in \mathbb{R}^3$ is the location of the vertex v before the solving by guided projection. This means that $v_{v,\text{old}}$ is updated before the solver is started.

4.2.3 Geodesic curvature

It is an often required property of designed discrete nets that their edge polylines form "discrete geodesics". Such nets avoid unnecessary zigzags and look more regular. Hence they are desired by multiple computer graphics applications.

4.2.3.1 Discrete unfolding

One way to keep the discrete geodesic curvature of the discrete net small is to observe the edge polylines of the discrete unfolding. So let $v \in \bar{\mathcal{V}}$ be a regular inner vertex of the discrete quadrilateral unfolding. This means that v is no boundary vertex and has 4 adjacent vertices denoted as $v_0, v_1, v_2, v_3 \in \bar{\mathcal{V}}$ like in figure 4.2a.

To keep the curvature of the edge polylines small, one might use the fairness terms $\|\frac{v_0+v_2}{2} - v\|^2$ and $\|\frac{v_1+v_3}{2} - v\|^2$ like shown in figure 4.2b. However, assume a discrete unfolding that contains a vertex with adjacent opposite edges, like $\{v, v_0\}$ and $\{v, v_2\}$ in figure 4.2a, of different length. Even if the given discrete net of the unfolding would describe a completely developable surface, the optimizer would try to move the vertices of the net, to get a result, where all edges are of equal length. To avoid this, one can use the normalized vectors instead of $v_0 - v$ and $v_2 - v$ like shown in figure 4.2c. This leads to the fairness terms

$$\left\| \frac{v_0 - v}{2\|v_0 - v\|} + \frac{v_2 - v}{2\|v_2 - v\|} \right\|^2$$

and

$$\left\| \frac{v_1 - v}{2\|v_1 - v\|} + \frac{v_3 - v}{2\|v_3 - v\|} \right\|^2 .$$

However, to use these terms for guided projection like described in section 3.2, they have to be linear in the unknowns v, v_0, v_1, v_2 and v_3. To achieve this, the implementation calculates the terms $\|v_0 - v\|$, $\|v_1 - v\|$, $\|v_2 - v\|$ and $\|v_3 - v\|$ every time before it calls the guided projection solver. They are then assumed as constant during solving, just like $v_{v,\text{old}}$ in section 4.2.2.

4.2.3.2 Directly on discrete 3D net

Another way to bound the discrete geodesic curvature of the discrete net is to observe the curvature vector $c_v := \frac{v_0 - v}{2\|v_0 - v\|} + \frac{v_2 - v}{2\|v_2 - v\|}$ from section 4.2.3.1 directly on the discrete net. But if one would use the fairness terms from above on a discrete net in 3D, it would not only minimize the geodesic curvature of the net, but also the normal curvature that should be allowed. To

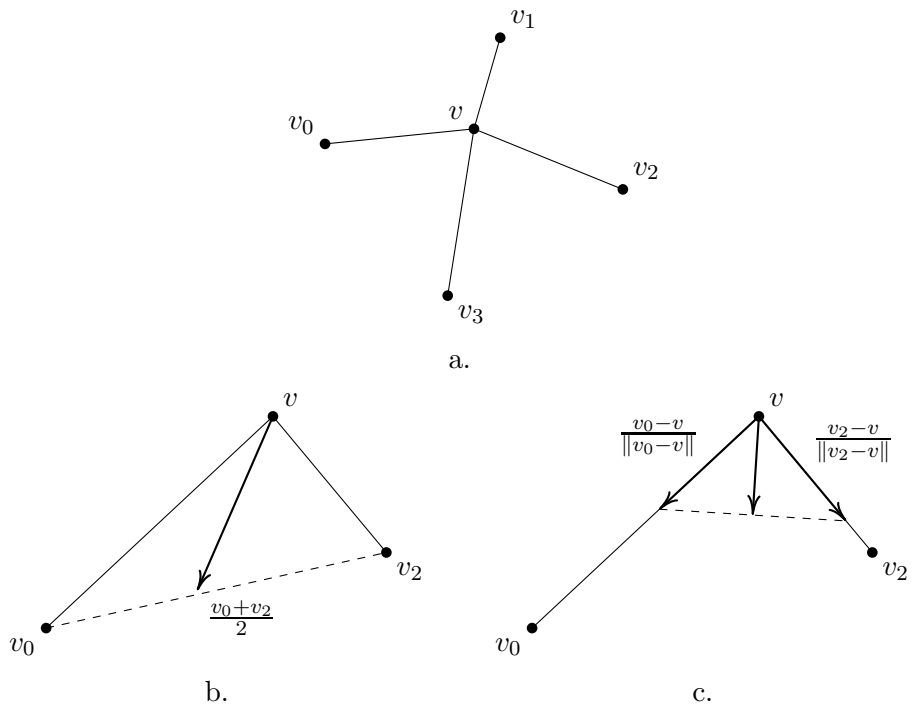


Figure 4.2: Discrete geodesic curvature

avoid this, the vector c_v has to be projected on an estimated tangent plane before it is used as fairness.

Let $v \in \mathcal{V}$ now be a regular inner vertex of the discrete quadrilateral 3D net and let $v_0, v_1, v_2, v_3 \in \mathcal{V}$ be again its adjacent vertices, arranged like in figure 4.2a. Then the projection of c_v on an estimated tangent plane with normal n_v would be

$$\begin{aligned} g_v &:= c_v - \langle c_v, n_v \rangle n_v \\ &= c_v - (n_v \cdot n_v^T) c_v \\ &= (I - n_v \cdot n_v^T) c_v \end{aligned}$$

where I is the 3-times-3 unit matrix. A way to get the normal n_v can be found in the description of the Gauss map implementation in section 5.6.3.

The vector g_v can be used as fairness term $\|g_v\|^2$ for each regular vertex $v \in \mathcal{V}$ if it can again be assumed as linear in the optimization unknowns v, v_0, v_2. So the matrix $I - n_v \cdot n_v^T$ has to stay fixed during the optimization, hence n_v needs to be refreshed before every guided projection solving. Additionally, this fairness term can also be applied on the curvature vector $\frac{v_1-v}{2\|v_1-v\|} + \frac{v_3-v}{2\|v_3-v\|}$.

4.2.3.3 Irregular and special vertices

So far every target equation and fairness term defined is compatible with irregular vertices but, like seen above, the used geodesic fairness terms cannot be applied on such vertices. They have to be skipped when the implementation defines these terms. But there are also potential problematic regular vertices for the geodesic fairness terms. Let for instance $v \in \bar{\mathcal{V}}$ be a regular vertex of the discrete unfolding, and let $v_0, v_1, v_2, v_3 \in \bar{\mathcal{V}}$ be again its adjacent vertices. Assume that the angle of two opposite adjacent edges of v, for instance $\{v, v_1\}$ and $\{v, v_3\}$, is not π,

like in figure 4.3. The fairness terms will then lead to a deformation of the net to achieve this angle, even if the net forms a completely developable surface.

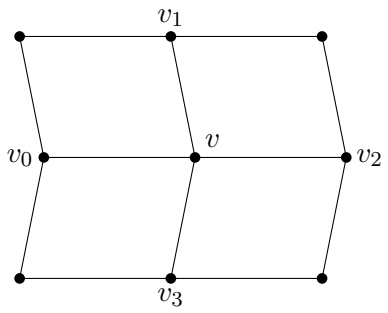


Figure 4.3: Special regular vertex

To avoid problems with such vertices, they can be marked in the net data structure by the Boolean property `allowGeodesic`, similar to the property `isCorner` mentioned in section 3.1. Marked regular vertices will then be skipped for the geodesic fairness terms like irregular vertices. To use this feature, subdivided discrete nets have to inherit marked vertices from the control net, where users can manually mark them. So let $(\mathcal{V}^{i-1}, \mathcal{E}^{i-1}, \mathcal{F}^{i-1})$ be the input net for a step of the algorithm of Catmull-Clark. Then a subdivided vertex $v \in \mathcal{V}^i$ will be marked if

1. it is derived from an old face $\phi \in \mathcal{F}^{i-1}$ (see step 1 in section 3.1) and ϕ has a marked adjacent vertex.

2. it is derived from an old edge $\epsilon \in \mathcal{E}^{i-1}$ (see step 2 in section 3.1) and ϵ has a marked adjacent vertex.

3. it is derived from an old vertex $\nu \in \mathcal{V}^{i-1}$ (see step 3 in section 3.1) and ν is marked.

For an example consider figure 4.4.

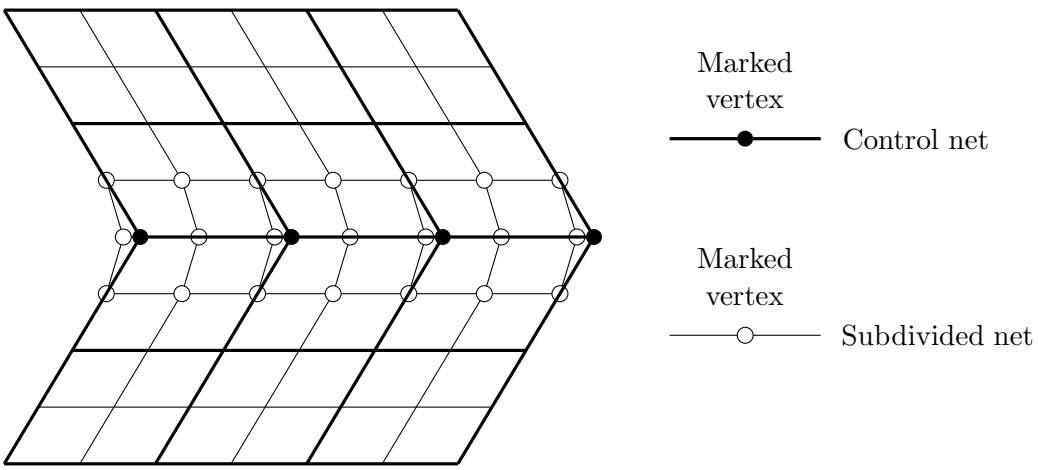


Figure 4.4: Mark special vertices

4.3 Expectations

Let's analyze the approach so far and summarize what one can expect from the designed optimization.

All target equations and fairness terms of the presented approach were defined with arbitrary control nets in mind. The algorithm of Catmull-Clark subdivides these nets and yields quadrilateral nets with vertices which may be irregular. Most parts of the approach are completely compatible with this scenario, except the fairness terms to avoid geodesic curvature from section 4.2.3. Irregular vertices and special kinds of regular vertices will then be ignored by these fairness terms, as described in detail in the regarding section. Altogether this enables the presented approach to handle more general surfaces than many other approaches in literature, often focused on B-Spline surfaces.

Other approaches in literature, for example [1] or [13], allow the explicit definition of creases for the design of developable surfaces. This can be a key feature to implement design tools for arbitrary developable surfaces. As mentioned in section 2.1, each developable surface can be locally classified as one of three types namely cylinders, cones and tangent surfaces. Globally a developable surface then changes from one type to another over its domain. If such a change should be smooth, i.e. there exist at least tangent planes at the crease, it can only appear at a ruling. Since tangent planes stay constant along rulings, they provide a matching constraint for the two connected types. For an example for a smooth connection of a cylinder and a cone see figure 4.5. If creases don't have to be smooth, type changes can also appear along boundary curves, like in [12], but these surfaces are out of scope of the presented approach.

The approach of this thesis has no tool for the definition of creases and hence no control over the change of type or the creation of rulings. It creates a surface that is "as developable as possible" under given constraints (for instance, the held vertices by the user). But its design leads, in combination with the distortion bounds from section 4.1.1, to the chance to control the "grade of developability" of modeled surfaces.

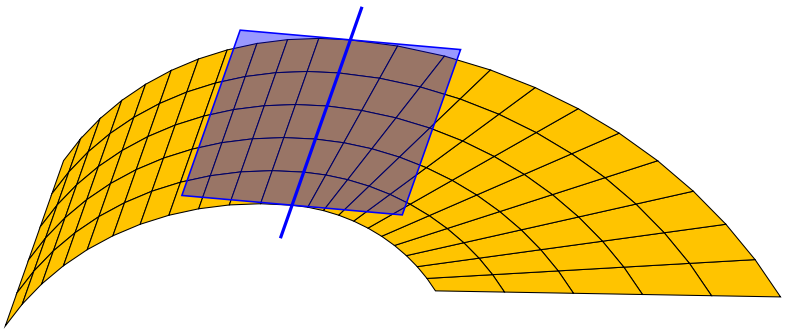


Figure 4.5: Type change

Chapter 5

Implementation

This chapter targets the implementation of the approach to get a design tool for nearly developable surfaces as desired in chapter 1. In addition to the realization of this tool, this chapter should document the details of the implementation to make the results of chapter 6 reproducible. The used system and libraries are summarized in chapter 7.2, all code of this implementation was written in C++.

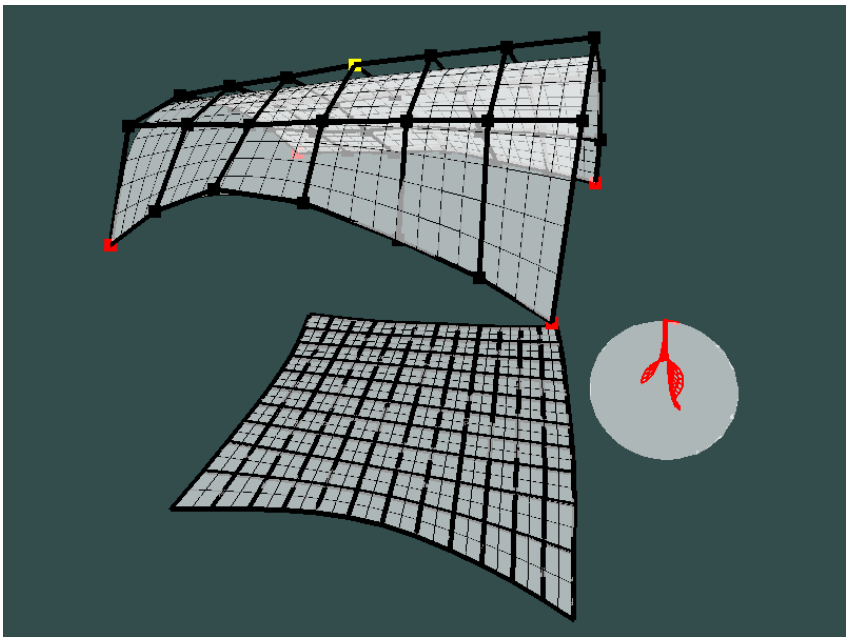


Figure 5.1: Implemented design tool

5.1 Overview

The developed implementation is written directly on top of the OpenGL window toolkit GLFW and the loader library GLAD. To realize this, there was the demand for a small graphics engine. Like most graphic engines the core element of the engine is the main loop executed after the initialization until the program is closed. The initialization builds the control net with its combinatorics and creates all other nets and objects. The user will then be able to move or fix the vertices of the control net. All other nets and objects depend on these changes and cannot

be altered directly. So the main loop performs the following steps:

1. **Process events:** Events are things done by the user. For instance, a mouse movement to initialize a drag of a vertex or a key stroke to demand the program to move the camera in space. The events are polled and processed in the code, for instance by changing the saved location of the dragged vertex or the camera.

2. **Optimization:** The current location of the vertices of all nets is loaded and the optimization approach of chapter 4 is performed by a guided projection solver. This step is not executed in every iteration of the main loop, due to performance reasons. It is also not required, because the vertices of the control net will not move too far per iteration round.

3. **Movement:** In this stage the vertices of all objects are moved to the new locations calculated by the optimization. Moved objects can also be implemented visualizations covered by section 5.6.

4. **Calculate view:** This is related to the camera and calculates how the scene has to be shown in the current state of the program.

5. **Rendering:** Tell OpenGL to render the scene.

Most steps of the main loop are related to graphics programming and well known, so they are covered in short in section 5.2. The most interesting part for this thesis is step 2 targeted in section 5.4 and 5.5.

5.2 Engine

The written engine builds the basis of the implemented tool and consists of three parts:

1. **Interface to OpenGL:** The loader library GLAD is written in C. To enrich it with the advantages of C++, there were implemented some wrapper classes. In addition, there are some classes to make the window toolkit GLFW more easy to use.

2. **Main Loop:** The implementation of the main loop and some interfaces to connect the discussed nets of the approach from chapter 4 to OpenGL objects.

3. **Shader:** Programs for the GPU to produce pictures from the raw calculations.

The interface to OpenGL in part 1 contains a window class that is connected to GLFW. It gives a type for the implementation to handle the program window and to centralize the event handling of GLFW. Additionally, there are types for OpenGL buffers and vertex arrays to make the observed objects visible.

The main loop in part 2 creates a scene or a stage and so there is a class implemented called Stage. The stage holds objects called Actors, which are then loaded by the main loop. Actors can be all kinds of visible objects, for instance a net from chapter 4 or other things like a Gauss map. More details on these classes are discussed in the sections 5.3 and 5.6. Furthermore, there are classes to implement a camera that can be moved freely in the space of the program.

The implemented shaders in part 3 were kept very simple. There are only one vertex shader and one fragment shader implemented in their simplest form. It was not a target of this thesis to implement optimized graphics programming.

5.3 Net classes

To make the implementation more intuitive there was the demand for a layer that connects the theoretical idea of discrete nets from definition 3.1.1 to the actors of the engine layer described in section 5.2.

The goal of the basic net class was to implement the idea of definition 3.1.1 based on the C++ library OpenMesh. OpenMesh is a very fast library and provides many tools to work directly with the elements of discrete nets. For instance, it calculates the edges of a discrete net automatically from its vertices and faces and provides functions to load the adjacent faces and vertices of an edge or vice versa.

The connection between the engine and the basic net class was implemented by a class named DrawNet, inherited from the basic net class and the actor class of the engine. This class provides functions to transform the nets of OpenMesh to vertex arrays for OpenGL. On top of this connection class there were implemented some handy net classes, for instance

- a control net class connected to the interface classes of the engine for GLFW to handle vertex drag events or
- a subdivision net class, to load an implemented subdivision algorithm from section 5.3.1 or
- a copy net class, which copies the combinatorics of another net and the location of its vertices, to conveniently implement visualization nets from section 5.6.

5.3.1 Subdivision classes

The subdivision net class of section 5.3 is a general class that handles any kind of subdivision algorithm. This makes it more flexible and would give the option to use it for different subdivision algorithms, although a comparison between different subdivision algorithms is out of scope of this thesis. To use a subdivision net, one has to implement a subdivision algorithm class. In the case of this thesis, this is the algorithm of Catmull-Clark.

The implementation works recursively and implements directly the algorithm described in section 3.1. Firstly, this reproduces the nature of a subdivision algorithm more closely, and secondly, it enables the algorithm to save the net of every iteration. These iterations are then used by the subdivision net class to make the implementation more flexible for using intermediate nets like described in section 4.1.3 and section 5.5.

5.4 Guided projection solver

Section 3.2 previously described the idea of the guided projection solver presented by Tang et al. in [11] and targeted also some variations of the implemented solver. This section will discuss some technical details of the implementation.

The guided projection solver is implemented as C++ template to make it more type flexible. This would allow to change for instance the floating point precision easily but leads to the need of some wrapper types, implemented in a specific class. Additionally, there need to be classes to implement types for target equations and fairness terms. The demands of these classes are to give the user interfaces for intuitive handling of the terms, but also to safe them efficiently with fast access times, to enable the solver to load all data quickly.

Quadratic target equations

$$\phi(x) = \frac{1}{2}x^T Ax + b^T x + c$$

are saved and handled by a matrix A, a vector b and a scalar c. Matrix and vector types are provided directly by the C++ library eigen. This library also provides sparse types, which would be of special interest for this implementation. The experiences with the implemented operators for sparse matrices of eigen (see section 7.2 for the used version) though were not fast enough, so an own class was implemented for target equations. This class uses a dense matrix that holds the non-zeros of an imagined sparse matrix and an index vector to link the indices of the saved dense matrix to the imagined sparse matrix.

Linear fairness terms

$$\|k^T x - s\|^2$$

are saved and handled by a vector k and a scalar c. Since there is not that much need for operators in the implementation for fairness terms like for target equations, the sparse types of eigen were used here directly.

Of course the core of the solver class is a solve function. This function takes a starting vector $x_0 \in \mathbb{R}^n$, a group of N target equations $\phi_i(x) : \mathbb{R}^n \to \mathbb{R}$ and a list of m fairness groups with fairness terms $(k_i^T x - s_i)^2$. The function then solves the linear system 3.4 of section 3.2.1 multiple times to obtain new solutions $x_{n+1} \in \mathbb{R}^n$ until a maximum iteration count is exceeded or

$$\|x_n - x_{n+1}\| < \epsilon$$

for a chosen threshold $\epsilon > 0$. The implementation allows to choose the maximum iteration count and threshold ϵ at run time. The needed calculations for the matrices H and K_j as well as the vectors r and s_j are performed more or less directly with the operators of eigen.

5.5 Optimization approach

This section targets the implementation of the approach covered in section 4. Basically all required tools were already described, so the optimizer class, that realizes this implementation, reads like a recipe. This is achieved by the net classes of section 5.3 and the guided projection solver of section 5.4. The helper classes for target equations and fairness terms allow to directly code the derived equations from section 4. The input of the optimization is the current state of the unknowns mentioned in section 4.1.4, the output is a new location of the vertices of the control net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$ and the intermediate net $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$. The new state of all other unknowns can be derived from the new control nets.

Besides the implementation of the target equations from section 4.1, the optimizer class fragments the discussed fairness terms of section 4.2 into fairness groups. Like mentioned in section 3.2, there also have to be weights related to the different groups. These weights are mutable during runtime and should be greater, if the input net of the optimization is not nearly developable. Since the fairness terms should stabilize the optimization, they are not needed anymore if the net is fairly developable. In practice this is achieved by the inputs of the user. If the user drags a vertex of the control net far away from its origin, the weights will be chosen greater. The fragmentation of the fairness terms is as follows:

1. One group for the developability equations from section 4.2.1.

2. One group for the distance to the old net, like described in section 4.2.2.
3. One group for the geodesic curvature of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ from section 4.2.3.2.
4. One group for the geodesic curvature of the control net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$.
5. One group for the geodesic curvature of the discrete unfolding $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ of $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ from section 4.2.3.1.
6. One group for the geodesic curvature of the unfolded intermediate net $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$.

Please refer to section 4.1.4 for the used symbols of the nets.

The design of the optimizer class allows it to operate on different levels of subdivision of the control net $(\mathcal{V}^0, \mathcal{E}^0, \mathcal{F}^0)$. Let's assume that $(\mathcal{V}, \mathcal{E}, \mathcal{F}) = (\mathcal{V}^i, \mathcal{E}^i, \mathcal{F}^i)$ like in section 4.1.4. Let further be $0 \leq j \leq i$, where $j \in \mathbb{N}$ is the iteration of the intermediate net $(\bar{\mathcal{V}}^j, \bar{\mathcal{E}}^j, \bar{\mathcal{F}}^j)$. The optimizer class can now operate "hierarchically": It can optimize the control net for the discrete net $(\mathcal{V}^k, \mathcal{E}^k, \mathcal{F}^k)$ where $0 \leq j \leq k \leq i$. For greater performance the optimization is performed for small k while the user drags a vertex to get direct feedback of the new vertex position. But if the user releases the vertex, it runs for greater or increasing k to achieve better optimization results.

5.6 Visualization

Like described in section 5.2 visualizations are handled as actors by the engine. So there are net visualizations derived from the DrawNet class of section 5.3, since DrawNet is an actor class, and non-net visualizations derived directly from the actor class.

5.6.1 Edge length

The basic idea of the approach is connected to the comparison of edge lengths via equation 4.2 of section 4.1.1. Hence the edge length visualization shows the relative length difference of edges of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ and its discrete unfolding $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$. This will give a feeling about the quality of the optimization.

Let $e \in \mathcal{E}$ be an edge of the discrete net and $\bar{e} \in \bar{\mathcal{E}}$ its counterpart of the unfolding. Then the visualization shows

$$\alpha := \frac{l(e)}{l(\bar{e})}$$

where $l(e)$ is the length of e, by coloring of the edges. If $\alpha = 1$, the edges have the same length, so the edge of the visualization will be drawn white. If $\alpha < 1$, e is shorter than $\bar{e}$ and the edge will be colored green. Otherwise if $\alpha > 1$, the edge e is too long, and it will be red. Furthermore, this visualization can display the quotient α for the diagonals of the quadrilateral faces of $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. See figure 5.2 for an example.

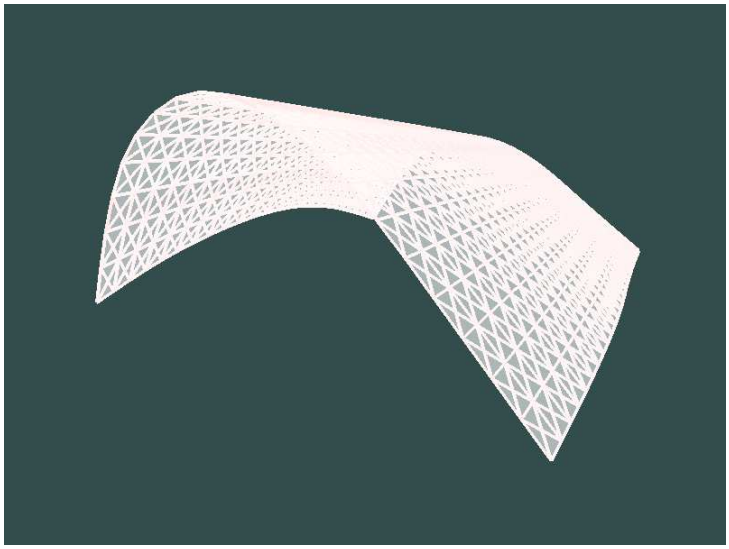


Figure 5.2: Edge length visualization

5.6.2 Distortion

If the implemented optimization would output developable surfaces, there would be no distortion in every point of the surface compared to its unfolding. However, the goal of this approach is to generate nearly developable surfaces. To get a better control of the distortion of the generated surface the target equations 2 of section 4.1.4 were implemented. These target equations were motivated in section 4.1.1, where the squared principal distortions of a surface compared to its unfolding were identified as

$$\begin{aligned} \sigma_{1,2} &= B \pm \sqrt{B^2 - A} \\ A &= \frac{EG - F^2}{\bar{E}\bar{G} - \bar{F}^2} \\ B &= \frac{\bar{E}G + E\bar{G} - 2F\bar{F}}{2\left(\bar{E}\bar{G} - \bar{F}^2\right)}. \end{aligned}$$

$\sigma_{1,2}$ can be calculated for every face of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ and its discrete unfolding $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$.

The implementation uses two colored visualizations for distortions. The first one shows the greater value of $\sqrt{\sigma_{1,2}}$ to present the maximum distortion of every face, the second one shows the smaller value to present the minimum distortion. Let γ_1 and γ_2 be again the chosen distortion bounds by the user like mentioned in section 4.1.1. Then the faces will be colored green if $\gamma_1 < \sqrt{\sigma} < \gamma_2$ and will become brighter the nearer $\sqrt{\sigma}$ gets to the bounds γ_1 and γ_2. The faces will be drawn white if $\sqrt{\sigma} = \gamma_1$ or $\sqrt{\sigma} = \gamma_2$ and will become red, if the distortions are out of the specified bounds. An example can be seen in figure 5.3.

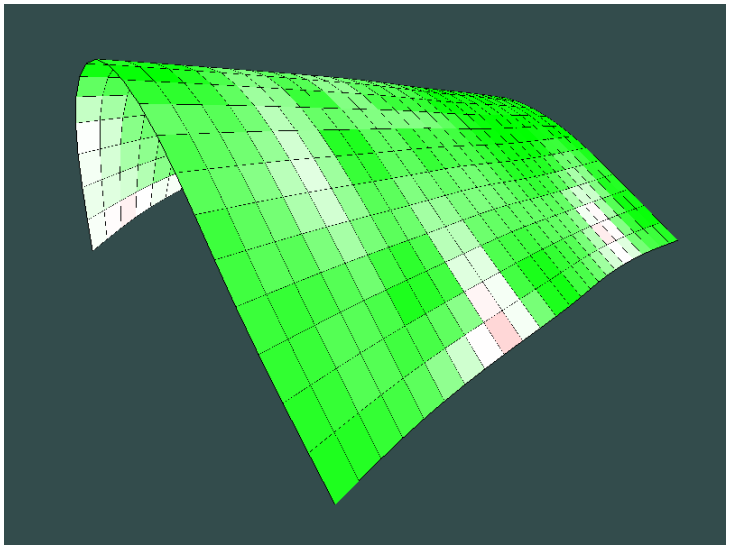


Figure 5.3: Distortion visualization

5.6.3 Gauss map

Since the tangent plane along every ruling of a developable surface is constant, its Gauss map is a finite union of curves on the unit sphere. This means that the surface covered by the Gauss map on the unit sphere gives a feeling about the "developability" of a surface. To display the Gauss map of a surface, one has to calculate approximated surface normals. Since this approach works with discrete surfaces, it has to estimate the normals in each vertex of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$.

Let $v \in \mathcal{V}$ be an inner vertex and let $e_1, e_2, \ldots, e_m \in \mathcal{E}$ be the adjacent edges of v. Let there further be for every $i \in \{1, \ldots, m\}$ a face $f_i \in \mathcal{F}$ such that e_i and $e_{i\oplus 1}$ are adjacent to f_i, where $i \oplus 1 := i + 1 \mod m$. Since the edges e_i are adjacent to v, this can be achieved by relabeling of the edges. Let's define

$$\forall i \in \{1, \ldots, m\} : n_{v,i} := \frac{e_i \times e_{i\oplus 1}}{\|e_i \times e_{i\oplus 1}\|}$$

then $n_{v,i}$ can be assumed as the unit normal of the face f_i. Note that the edges $e_i = \{v, v_i\}$ of the discrete net are here identified with their edge vector $v_i - v$. The estimated unit normal of the vertex v of the implemented Gauss map is then defined as

$$n_v := \frac{\sum_{i=1}^{m} n_{v,i}}{\| \sum_{i=1}^{m} n_{v,i} \|}.$$

To present the Gauss map to the user, the vectors n_v are drawn as red net on the unit sphere. The combinatorics of the net is the same as of the underlying discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. An example is shown by figure 5.4.

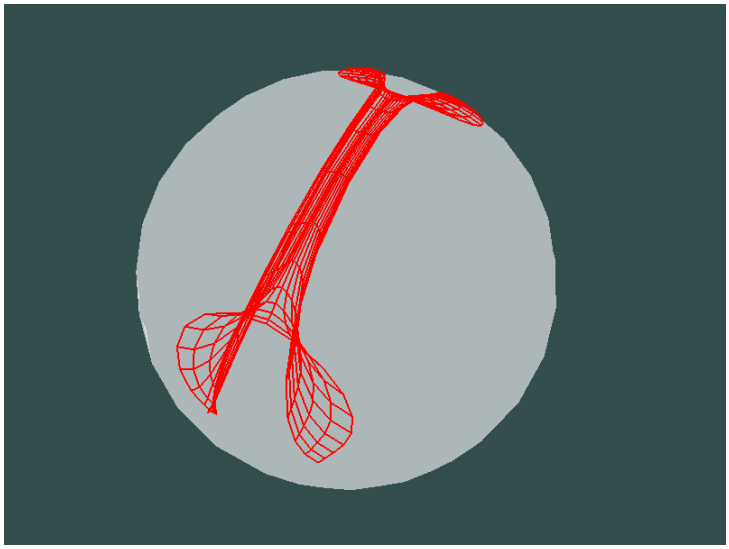


Figure 5.4: Gauss map

5.6.4 Registration

Another visualization to get a better understanding of the quality of the produced surface can be gained by registration. Hence the faces of the discrete unfolding $(\bar{\mathcal{V}}, \bar{\mathcal{E}}, \bar{\mathcal{F}})$ will be registered to their counterparts of the discrete net $(\mathcal{V}, \mathcal{E}, \mathcal{F})$. By drawing the registered quads of the unfolding instead of the quads of the discrete net, one gets a "quad soup" that shows how good the undistorted quads of the unfolding fit together in 3D space. This "soup" is an example for a visualization that is directly derived from the Actor class, since this is easier to handle in this case. One example of this visualization is shown in figure 5.5.

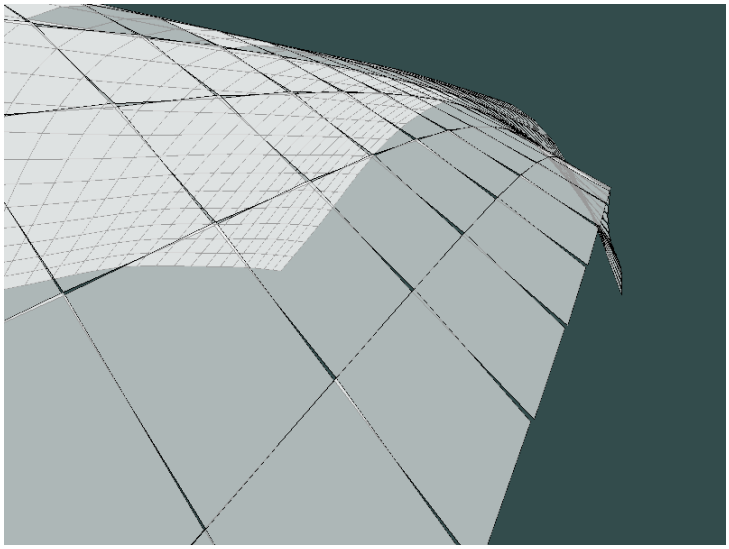


Figure 5.5: Registration soup

Chapter 6

Results

This chapter shows some results of the presented approach. Since the key features of the approach are the provision of an unfolding at any time and the option to determine distortion bounds, the shown examples should focus on these points. Additionally, since the approach works on Catmull-Clark surfaces, they will prove, that the approach can handle irregular vertices.

Except for the visualizations presented in section 5.6 the results will also be presented in renderings. All data for these renderings were exported by the implemented design tool from chapter 5 and then processed by the ray-tracer POV-Ray. In particular, the shown Gauss maps are not screenshots of the implementation, instead they were exported and externally rendered. So it can be ensured, that all views on the different Gauss maps are the same and different results stay comparable.

To make the results of this chapter reproducible, table 6.1 shows the implemented weights of the fairness groups mentioned in section 5.5. Since the weights can vary during runtime depending on the input net for the optimization, the table gives a range for every fairness weight. The weight ϵ^2 mentioned in section 3.2 was chosen statically

$$\epsilon^2 := 10^{-9}$$

	$\min\left(\epsilon_j^1\right)$	$\max\left(\epsilon_j^1\right)$
Developability equations (section 4.2.1)	$5 \cdot 10^{-7}$	$5 \cdot 10^{-3}$
Distance to old net (section 4.2.2)	10^{-10}	10^{-6}
Geodesic curvature of discrete net (section 4.2.3.2)	$5 \cdot 10^{-7}$	$5 \cdot 10^{-3}$
Geodesic curvature of control net (section 4.2.3.2)	$5 \cdot 10^{-7}$	$5 \cdot 10^{-3}$
Geodesic curvature of discrete unfolding (section 4.2.3.1)	$5 \cdot 10^{-8}$	$5 \cdot 10^{-4}$
Geodesic curvature of unfolded intermediate net (section 4.2.3.1)	$5 \cdot 10^{-8}$	$5 \cdot 10^{-4}$

Table 6.1: Chosen fairness weights

6.1 Design of nearly developable surfaces

The described optimization approach from section 4 should enable a user to design nearly developable surfaces, so this section features 2 examples of such surfaces. Both examples have in common, that the allowed mean distortion chosen by the distortion bounds γ_1 and γ_2

is 1 or in other words

$$\frac{\gamma_1 + \gamma_2}{2} = 1.$$

One example allows distortions in the interval $[0.9, 1.1]$, whereas the other example only allows distortions in $[0.99, 1.01]$.

Figure 6.1 shows renderings of the examples together with their Gauss map and unfolding. Figure 6.2 shows the user interactions to achieve the examples. Please note that the thick drawn net in figure 6.2 is the control net of the thin drawn Catmull-Clark surface. Red vertices are chosen by the user to remain fixed (see target equations 4 of section 4.1.4) and yellow vertices were dragged by the user to model the desired surface.

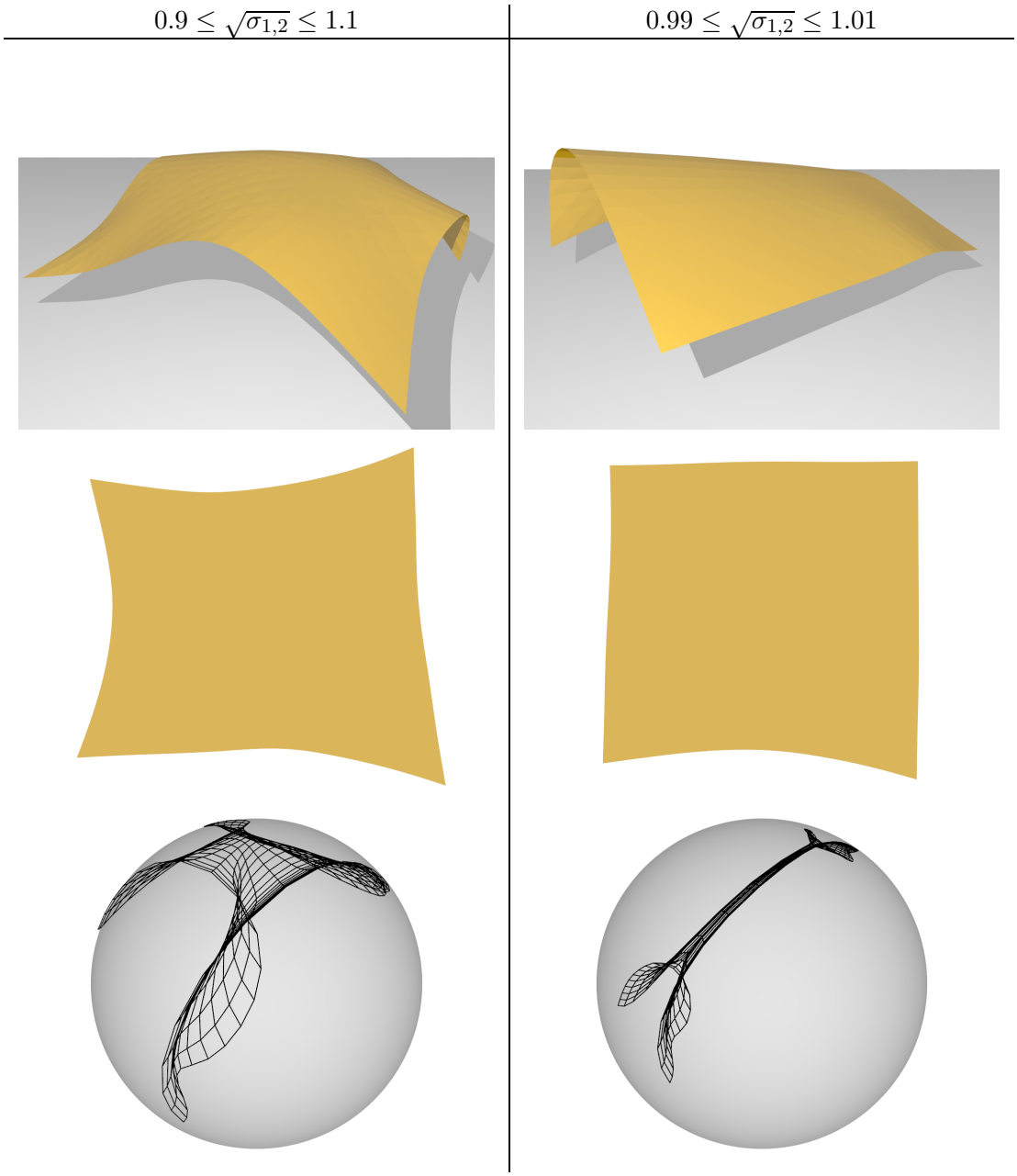


Figure 6.1: Nearly developable surfaces

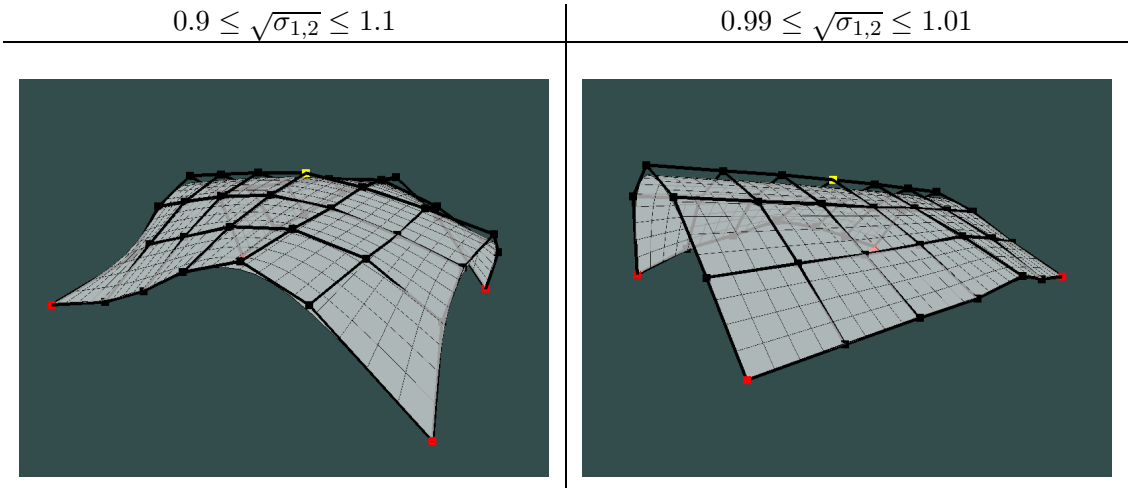


Figure 6.2: User interactions to generate nearly developable surfaces

6.1.1 Allowed distortions between 0.9 and 1.1

Let's first take a closer look on the first example which allows distortions between $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ before the other example is observed in more detail. The edge lengths shown in figure 6.3 may be a bit longer than the corresponding ones of the unfolding in the center of the surface (they are drawn a little bit reddish). However, they give the first impression, that the distortions of the designed surface are much nearer to 1 than allowed by the chosen bounds. Also the distortions of the surface shown in figure 6.4 give this feeling, although the surface looks not even nearly developable. See also the Gauss map on the left side of figure 6.1.

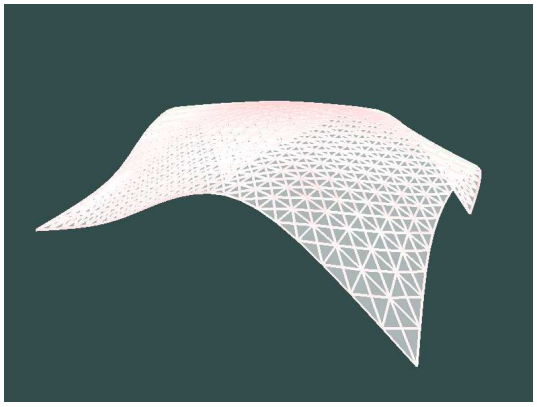
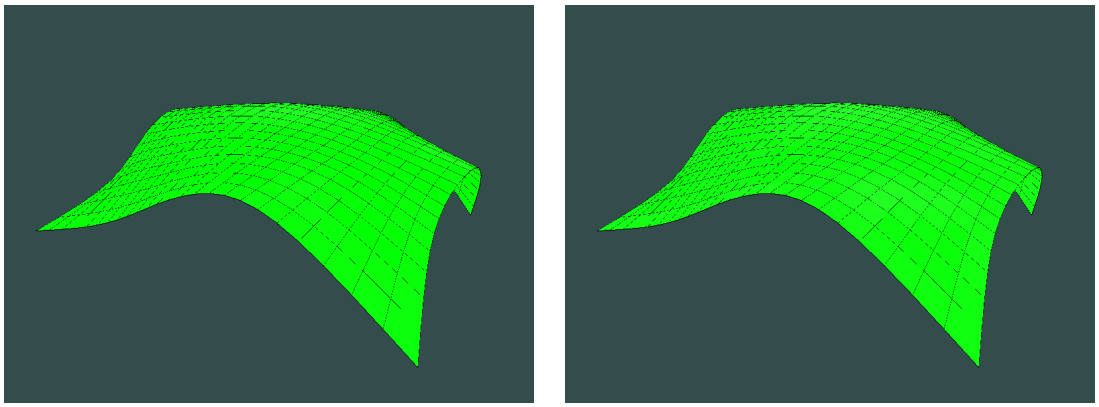


Figure 6.3: Edge lengths of a nearly developable surface, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.4: Distortion of a nearly developable surface, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$

Furthermore, figure 6.5 and 6.6 compare the distortions of both examples directly. In figure 6.4 the distortion visualizations color the distortion bounds γ_1 and γ_2 white, which are 0.9 and 1.1. Instead, figure 6.5 and 6.6 color the distortions 0.99 and 1.01 white for both surfaces to make the distortions better comparable. The faces of the left example are just slightly red, or in other words the distortions of the surface are approximately between 0.98 and 1.02. This corresponds to the first impression that the distortions of the example are much nearer to 1 than the allowed bounds $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$.

$0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$	$0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

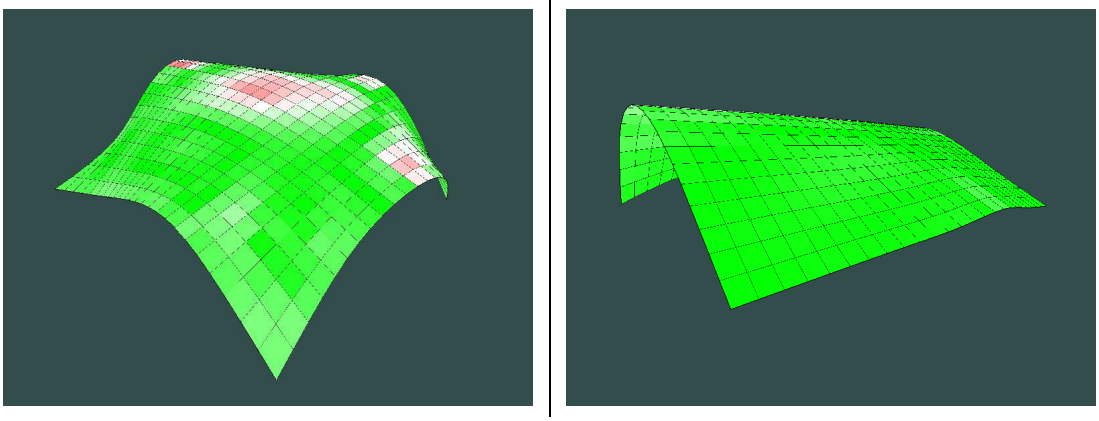


Figure 6.5: Minimum distortions $\sqrt{\sigma_1}$ of both nearly developable surfaces

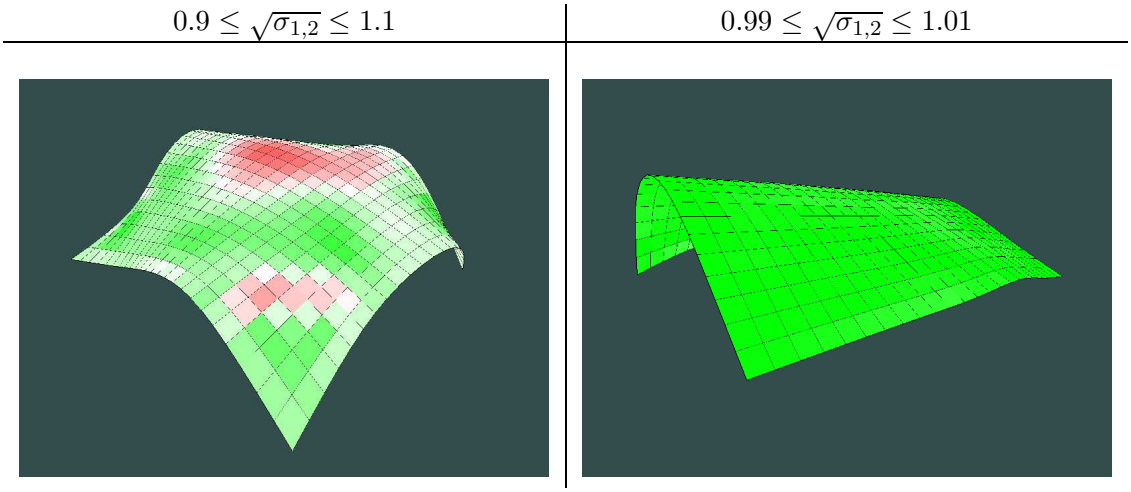


Figure 6.6: Maximum distortions $\sqrt{\sigma_2}$ of both nearly developable surfaces

In fact, industrial applications can show that just small cuts in a sheet allow huge deformations to get non-developable surfaces. Figure 6.7 shows a quad soup of the registered faces of the unfolding on the faces of the discrete net. It shows many such small cuts and may explain the bad developability of the generated surface, although the calculated distortions are small.

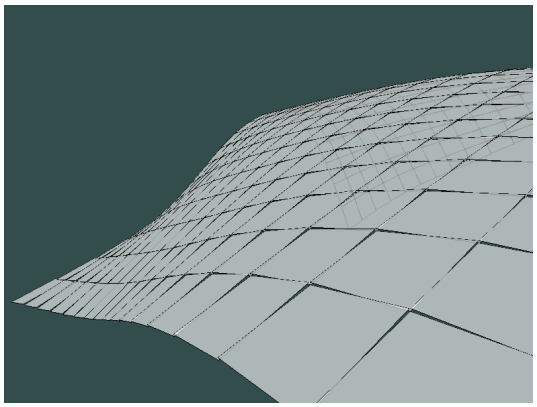


Figure 6.7: Face registration on nearly developable surfaces, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$

6.1.2 Allowed distortions between 0.99 and 1.01

The second presented example was created by allowing only distortions between 0.99 and 1.01. Like seen on the right side of figure 6.1 the optimization has created a surface with good developability and a slim Gauss map. Also the edge lengths of the generated surface shown in figure 6.8 and the distortions presented in figure 6.5 and 6.6 correspond to these results.

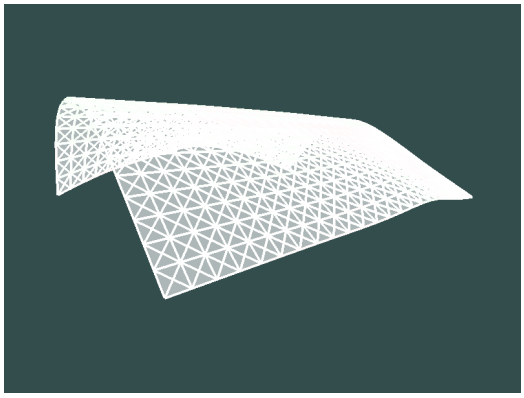
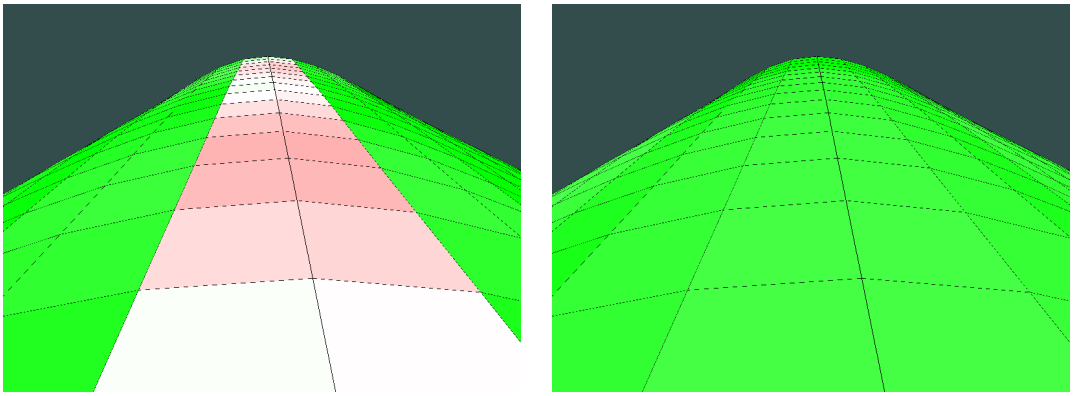


Figure 6.8: Edge lengths of a nearly developable surface, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

But figure 6.9 takes a closer look on the bend of the surface on its top and shows, that the optimization doesn't satisfy the distortion bounds there. The slight red coloring of the faces of the minimum distortions means, that the distortions there are in the interval $[0.98, 1.01]$. Also figure 6.10 features a slight overlap of the registered faces of the unfolding on the discrete net there.



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.9: Distortion violation, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

The investigation of this effect led to an evaluation of the target equations for the distortion bounds (equations 2 of section 4.1.4). All target equations of this type are in the interval $[-2 \cdot 10^{-6}, 0]$ and sum up to $-2 \cdot 10^{-4}$ in the presented example, so the guided projection solver delivers a very good result. It seems that the designed optimization hits numeric limits at this point. Furthermore, the approach to convert inequalities to target equations used in equation 4.7 in section 4.1.2 is numerically controversial.

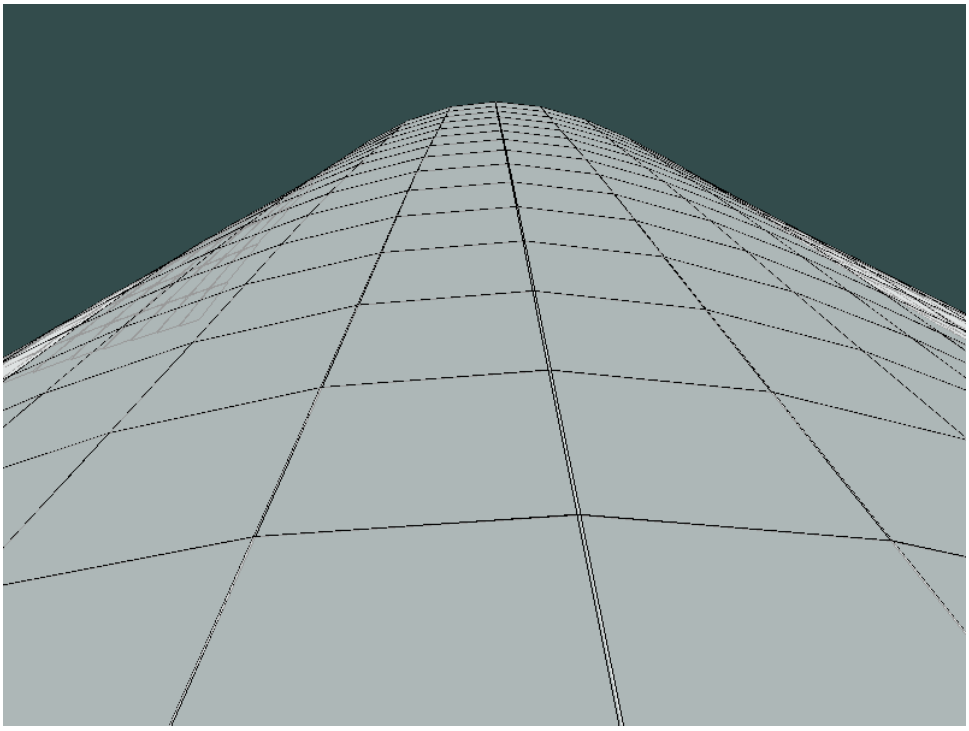


Figure 6.10: Face registration: Distortion violation, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

6.2 Variation of allowed mean distortion

This section should feature results to show how the optimization reacts on chosen distortion bounds γ_1 and γ_2 with

$$\frac{\gamma_1 + \gamma_2}{2} \neq 1.$$

Renderings of the presented examples and their Gauss map can be seen in figure 6.11.

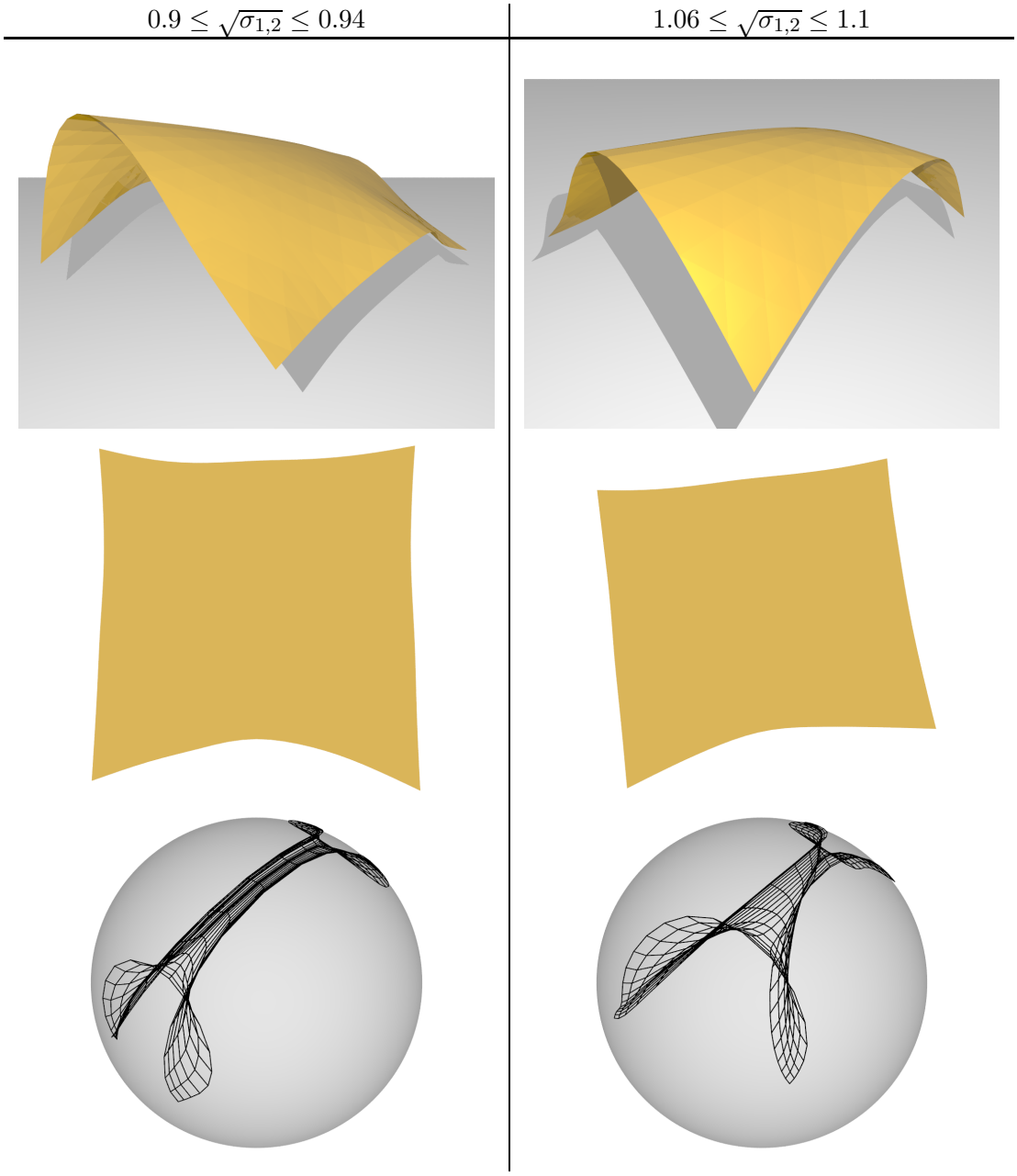


Figure 6.11: Variation of allowed mean distortion

The presented examples in figure 6.11 have slighter Gauss maps and are more developable than expected. Since the distortions used in the optimization are relative distortions of the

discrete nets compared to their unfoldings, figure 6.12 and 6.13 show what happened. The implementation just stretches or compresses the whole net to satisfy the given distortion bounds. The quad soups in figure 6.13 show that the registered quads of the unfolding overlap in case of $\frac{\gamma_1+\gamma_2}{2} < 1$ and are separated in case of $\frac{\gamma_1+\gamma_2}{2} > 1$. Like one can see in figure 6.14 and 6.15 the generated nets indeed satisfy the distortion bounds. This behavior was not really covered by the presented approach; the implementation reacts correct but unexpected.

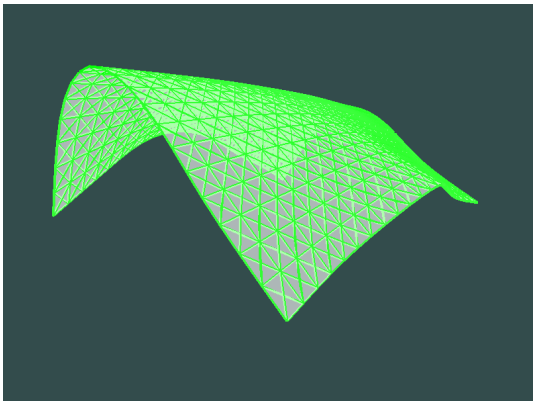
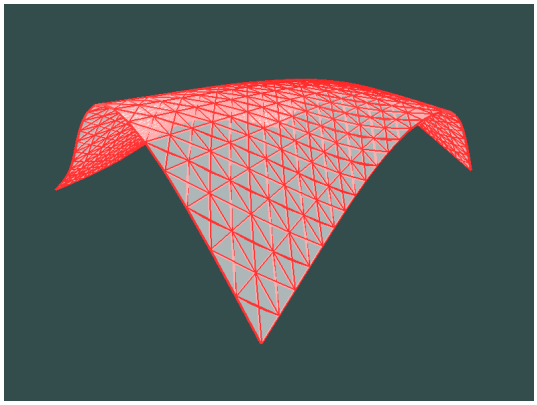
$0.9 \leq \sqrt{\sigma_{1,2}} \leq 0.94$	$1.06 \leq \sqrt{\sigma_{1,2}} \leq 1.1$
	

Figure 6.12: Edge lengths for $\frac{\gamma_1+\gamma_2}{2} \neq 1$

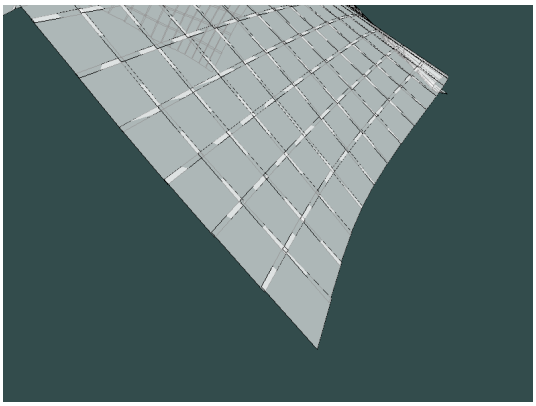
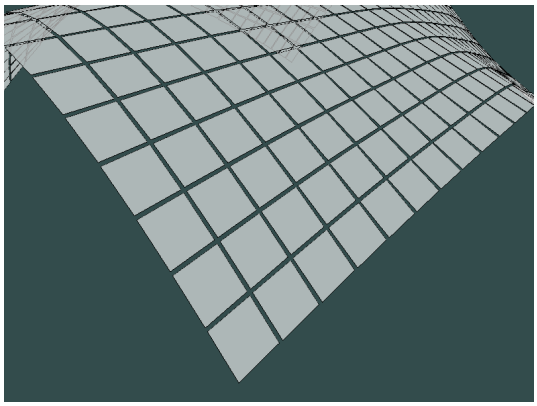
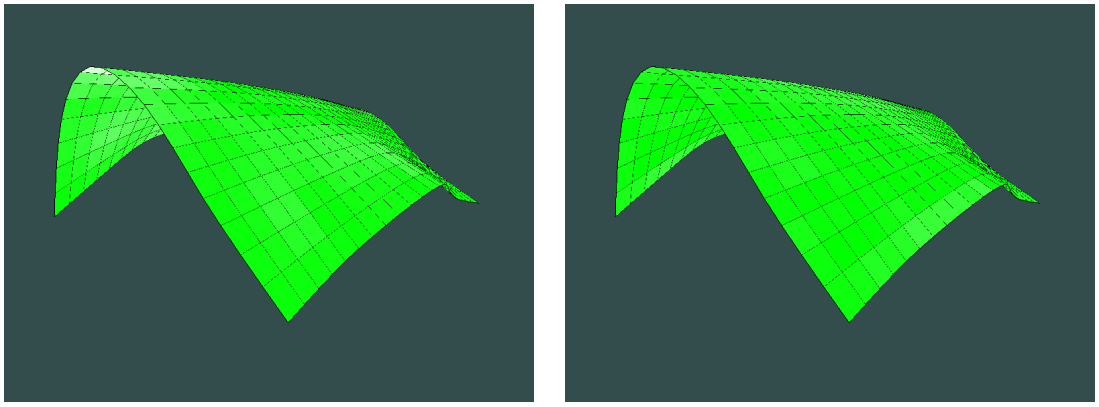
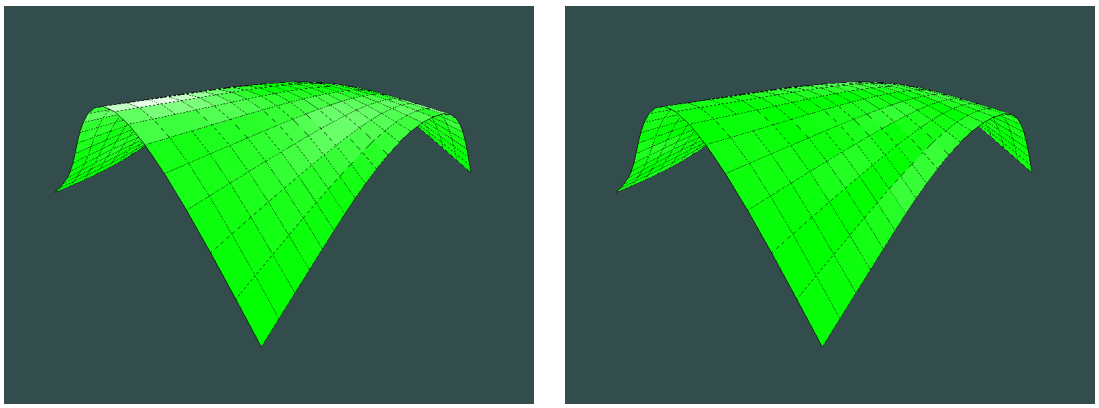
$0.9 \leq \sqrt{\sigma_{1,2}} \leq 0.94$	$1.06 \leq \sqrt{\sigma_{1,2}} \leq 1.1$
	

Figure 6.13: Face registration for $\frac{\gamma_1+\gamma_2}{2} \neq 1$



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.14: Distortion of a surface with $0.9 \leq \sqrt{\sigma_{1,2}} \leq 0.94$



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.15: Distortion of a surface with $1.06 \leq \sqrt{\sigma_{1,2}} \leq 1.1$

6.3 Handling of irregular vertices

An example of a control net with irregular vertex and allowed distortions in the interval $[0.99, 1.01]$ can be seen in figure 6.16. The net contains a single irregular vertex in the middle of the net with valence 6 and some special vertices along the edges starting from the irregular vertex. Like discussed in section 4.2.3.3, the fairness terms for the avoidance of geodesic curvature cannot be applied there. Figure 6.17 shows the irregular vertex of the control net of the example in yellow and the marked special vertices in red.

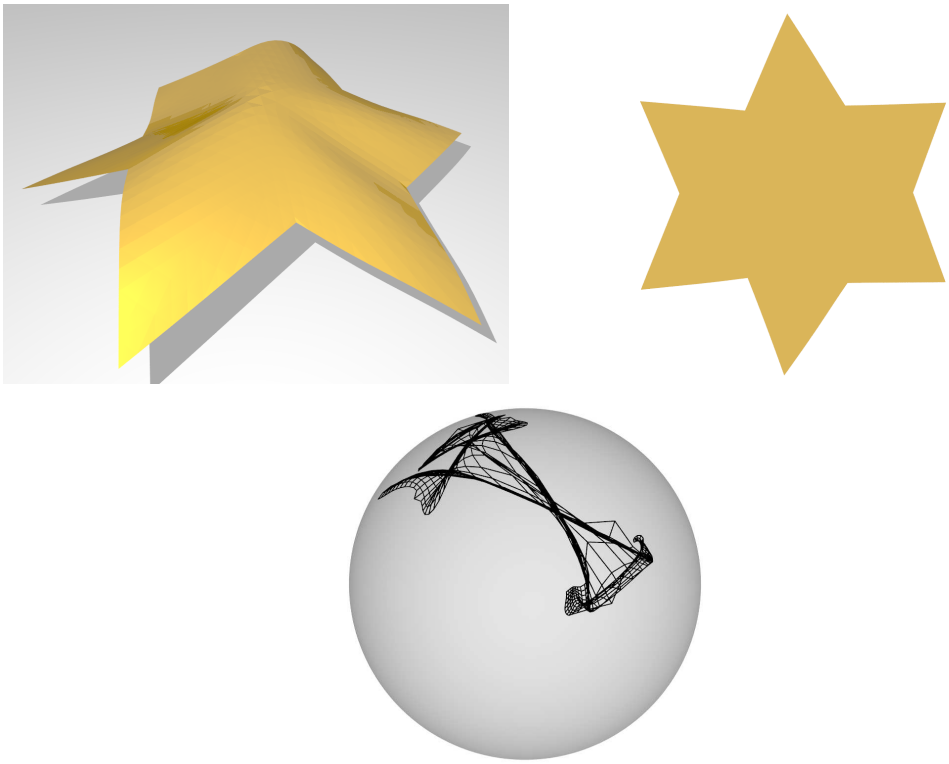


Figure 6.16: Irregular vertex example, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

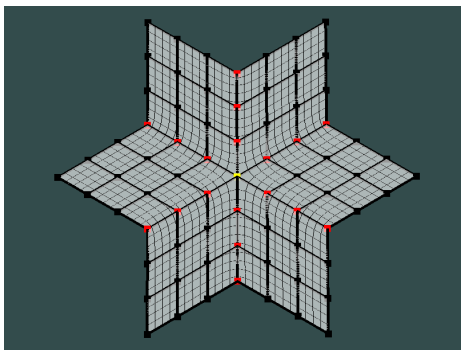
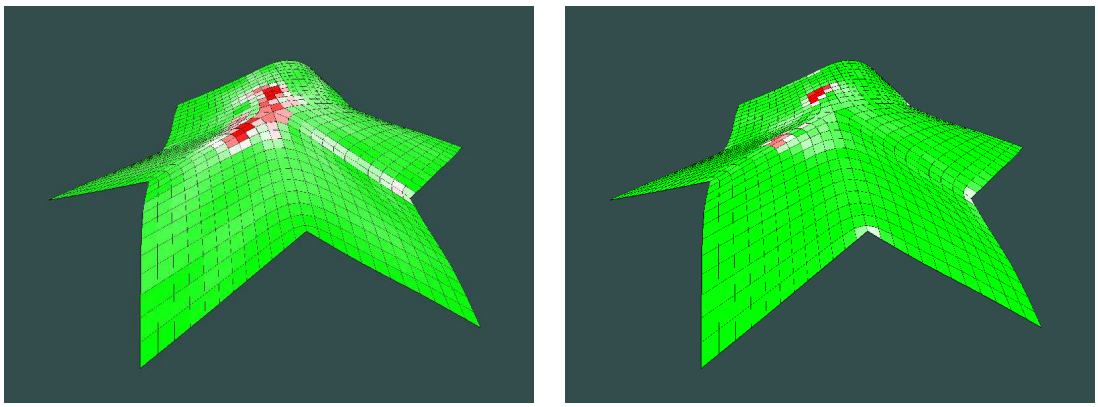


Figure 6.17: Irregular vertex and marked special vertices

The distortion of the example is depicted by figure 6.18. It seems to be hard for the optimization to satisfy the distortion bounds at the irregular vertex and the marked special vertices. This also fits to some experiments which showed, that the optimization doesn't deliver good results if the fairness terms about geodesic curvature are disabled. One has to define some assumptions about the regularity of the discrete net to get good developability over the whole surface. In case of skipping these assumptions along some edges, like the implementation skips the fairness terms about geodesic curvature, the desired quality of developability is only satisfied piecewise.

Like mentioned in section 2.1 and 4.3 other approaches in literature enable the user to define creases for the design of developable surfaces. The marking of special vertices seems to be an equivalent of the presented approach to such crease tools. Figure 6.19 finally gives another look at the behavior of the optimization at the irregular vertex.



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.18: Distortion of the irregular vertex example, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$

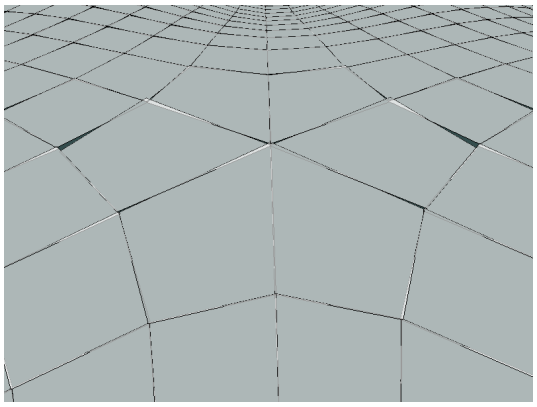


Figure 6.19: Face registration at the irregular vertex

6.4 Influence of developability fairness

Section 4.1.4 states that the identity of the first fundamental forms

$$I = \bar{I}$$

overrules the distortion bound equations 2 in the same section. Hence they were implemented as fairness like mentioned in section 4.2.1. But a higher weight of these fairness terms (or even using them as target equations) leads to results with great developability, what should be shown by this section. The presented example allows distortions between 0.9 and 1.1 and weights the developability fairness term of section 4.2.1 between 10^{-5} and 10^{-1}. All other fairness terms are weighted like in the other examples of this chapter and summarized by table 6.1. The result is shown in figure 6.20 and presents a very slight Gauss map.

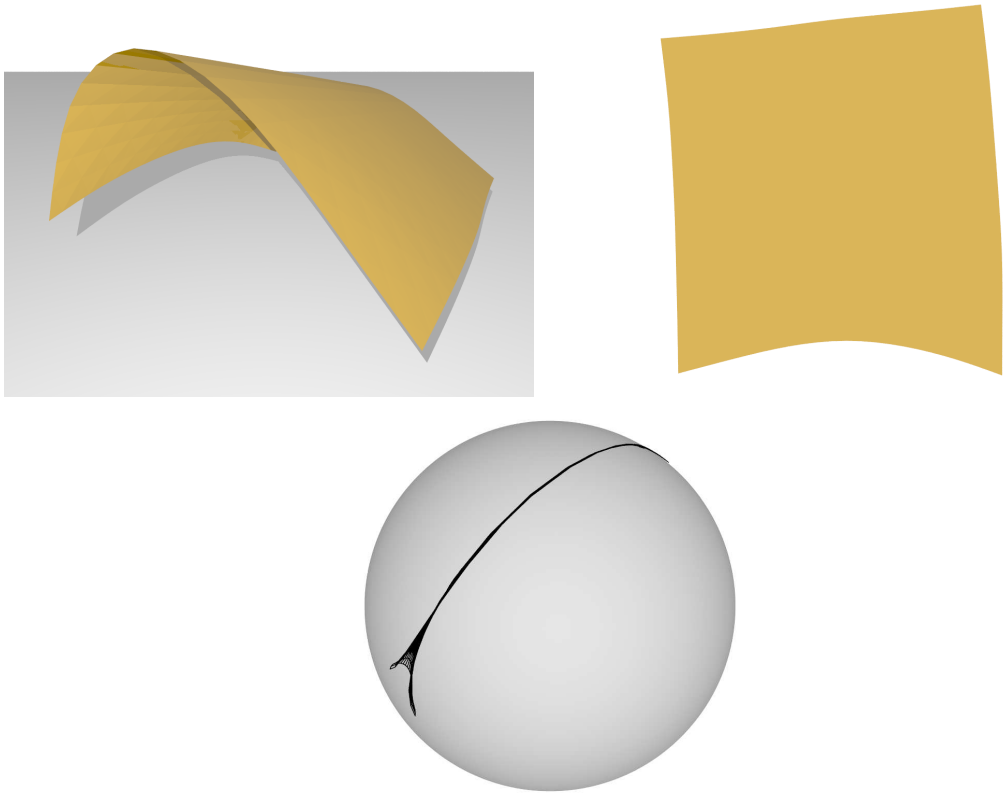
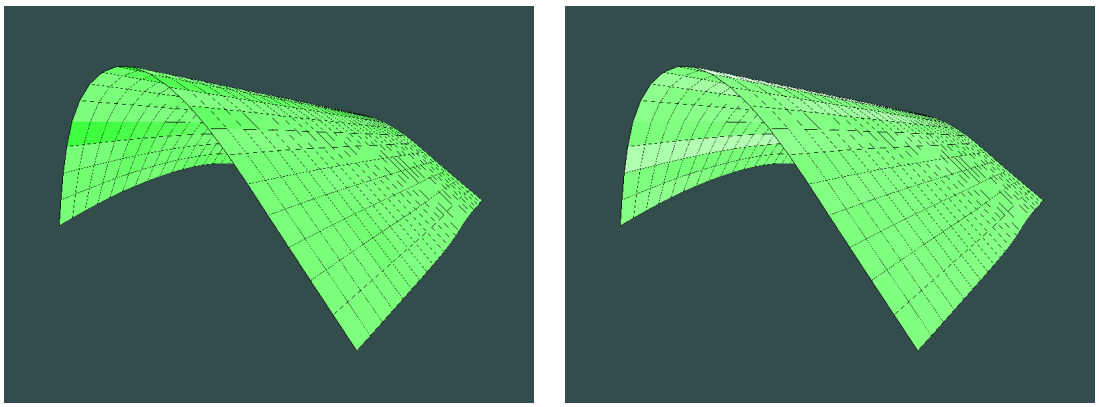


Figure 6.20: Influence of developability fairness, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$

Figure 6.21 shows the distortions of the generated surface. But instead of the allowed distortion interval $[0.9, 1.1]$, the figure colors the interval $[0.99, 1.01]$ green. This means that the optimization generates a much more developable surface than allowed. The implementation doesn't generate a wrong result in this case, but maybe a not desired one by a designer choosing these distortion bounds. In other words, the chosen bounds were overruled.



a. minimum distortions $\sqrt{\sigma_1}$ b. maximum distortions $\sqrt{\sigma_2}$

Figure 6.21: Influence of developability fairness: Distortion, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$

Chapter 7

Appendix

7.1 Examples

This section features some examples modeled with the implemented design tool.

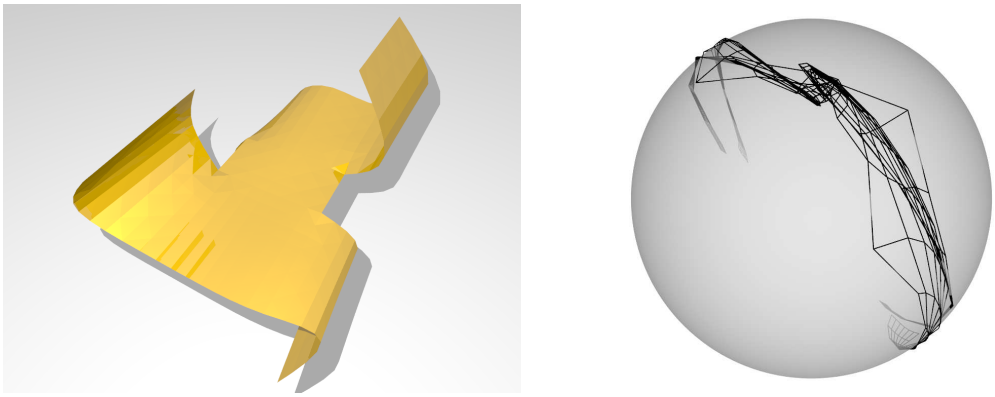


Figure 7.1: Appendix example 1

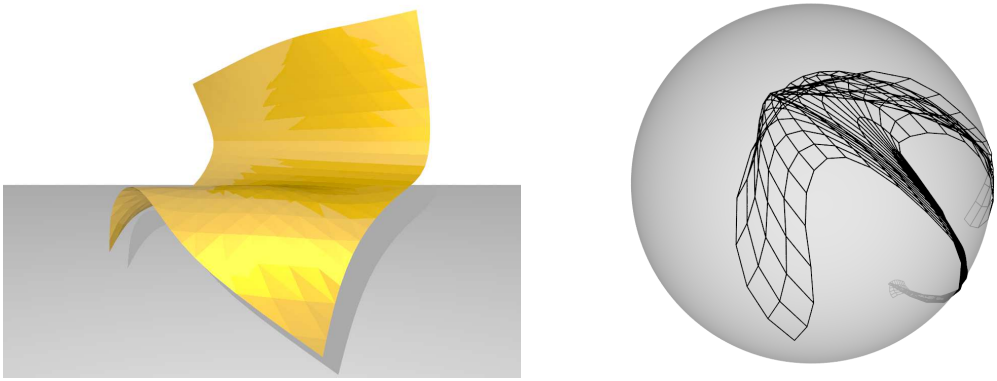


Figure 7.2: Appendix example 2

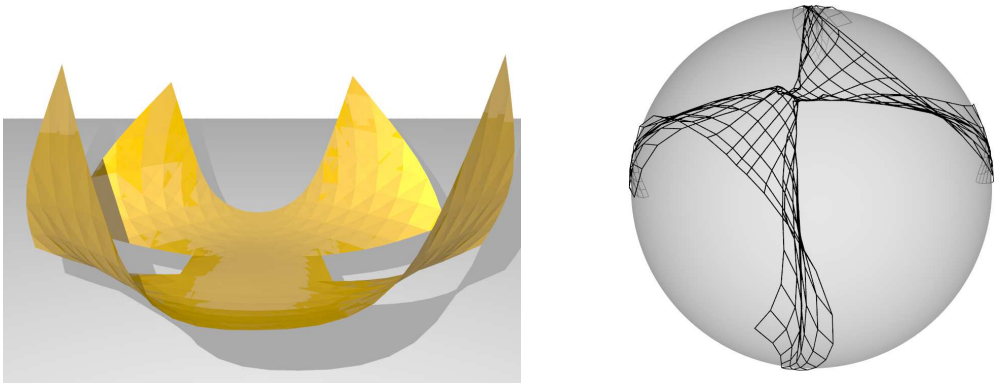


Figure 7.3: Appendix example 3

7.2 Used system

Computer components	
CPU	Intel Core i5 4570, 4x3.2GHz
Memory	Corsair CMZ8GX3M2A1600C9, 2x4GB, dual channel, DDR3, 1600MHz
Mainboard	Asus H87-Pro
BIOS version	American Megatrends 0806
Used Software	
Windows	10 Pro N, 1709, Build 16299.192, 64 bit
C++-Compiler	gcc version 6.2.0 (x86_64-win32-seh-rev1, Built by MinGW-W64 project)
OpenMesh	version 6.3
Eigen	version 3.3.3
GLFW	version 3.2.1
GLAD	version 0.1.12a0, API gl 4.0
GLM	version 0.9.8.1
Eclipse	IDE for C/C++ Developers, V Neon.1 Release (4.6.1)
POV-Ray	V 3.7.0.msvc10.win64
GIMP	V 2.8.22
Inkscape	V 0.92.1 r15371

Table 7.1: Used system

List of Figures

1.1 Walt Disney Concert Hall and origami model . . . 1
1.2 Blender modeling tool . . . 2

2.1 Ruled surface . . . 3
2.2 Developable surface . . . 4
2.3 Envelop of a tangent plane family . . . 5
2.4 Types of developable surfaces . . . 6
2.5 Offset of a surface . . . 6
2.6 Offset of discrete nets . . . 7
2.7 Angles in discrete geodesic nets . . . 8

3.1 Algorithm of Catmull-Clark . . . 11

4.1 Quadrilateral net indices . . . 16
4.2 Discrete geodesic curvature . . . 23
4.3 Special regular vertex . . . 24
4.4 Mark special vertices . . . 24
4.5 Type change of developable surface . . . 25

5.1 Implemented design tool . . . 26
5.2 Edge length visualization . . . 31
5.3 Distortion visualization . . . 32
5.4 Gauss map . . . 33
5.5 Registration soup . . . 33

6.1 Nearly developable surfaces . . . 35
6.2 User interactions to generate nearly developable surfaces . . . 36
6.3 Edge lengths of a nearly developable surface, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ . . . 36
6.4 Distortion of a nearly developable surface, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ . . . 37
6.5 Minimum distortions $\sqrt{\sigma_1}$ of both nearly developable surfaces . . . 37
6.6 Maximum distortions $\sqrt{\sigma_2}$ of both nearly developable surfaces . . . 38
6.7 Face registration on nearly developable surfaces, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ . . . 38
6.8 Edge lengths of a nearly developable surface, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$ . . . 39
6.9 Distortion violation, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$ . . . 39
6.10 Face registration: Distortion violation, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$ . . . 40
6.11 Variation of allowed mean distortion . . . 41
6.12 Edge lengths for $\frac{\gamma_1+\gamma_2}{2} \neq 1$ . . . 42
6.13 Face registration for $\frac{\gamma_1+\gamma_2}{2} \neq 1$ . . . 42
6.14 Distortion of a surface with $0.9 \leq \sqrt{\sigma_{1,2}} \leq 0.94$ . . . 43
6.15 Distortion of a surface with $1.06 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ . . . 43

6.16 Irregular vertex example, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$ 44
6.17 Irregular vertex and marked special vertices 44
6.18 Distortion of the irregular vertex example, $0.99 \leq \sqrt{\sigma_{1,2}} \leq 1.01$ 45
6.19 Face registration at the irregular vertex 45
6.20 Influence of developability fairness, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ 46
6.21 Influence of developability fairness: Distortion, $0.9 \leq \sqrt{\sigma_{1,2}} \leq 1.1$ 47

7.1 Appendix example 1 48
7.2 Appendix example 2 48
7.3 Appendix example 3 49

List of Tables

6.1 Chosen fairness weights 34

7.1 Used system 49

Bibliography

[1] C. Tang et al. “Interactive design of developable surfaces”. In: *ACM Transactions on Graphics* 35.2 (2016).

[2] E. Catmull and J. Clark. “Recursively generated B-Spline surfaces on arbitrary generated meshes”. In: *Computer-Aided Design* 10.6 (1978), pp. 350–355.

[3] A. Ball and D. J. T. Storry. “Conditions for tangent plane continuity over recursively generated B-spline surfaces”. In: *ACM Transactions on Graphics* 7.2 (1988), pp. 83–102.

[4] J. Peters and U. Reif. “Conditions for tangent plane continuity over recursively generated B-spline surfaces”. In: *SIAM Journal on Numerical Analysis* 35.2 (1998), pp. 728–748.

[5] G. Aumann. “A simple algorithm for designing developable Bézier surfaces”. In: *Computer Aided Geometric Design* 20.8-9 (2003), pp. 601–619.

[6] G. Aumann. “Degree elevation and developable Bézier surfaces”. In: *Computer Aided Geometric Design* 21.7 (2004), pp. 661–670.

[7] C.-H. Chu and C. H. Séquin. “Developable Bézier patches: properties and design”. In: *Computer-Aided Design* 34 (2002), pp. 511–527.

[8] P. Bo and W. Wang. “Geodesic-Controlled Developable Surfaces for Modeling Paper Bending”. In: *Computer Graphics Forum* 26.3 (2007), pp. 365–374.

[9] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976. ISBN: 0-13-212589-7.

[10] H. Pottmann and J. Wallner. “Approximation algorithms for developable surfaces”. In: *Computer Aided Geometric Design* 16.6 (1999), pp. 539–556.

[11] C. Tang et al. “Form-finding with Polyhedral Meshes Made Simple”. In: *ACM Transactions on Graphics* 33.4 (2014).

[12] W. Frey. “Modeling buckled developable surfaces by triangulation”. In: *Computer-Aided Design* 36.4 (2004), pp. 299–313.

[13] J. Solomon et al. “Flexible Developable Surfaces”. In: *Computer Graphics Forum* 31.5 (2012).

[14] F. Pérez and J. Suárez. “Quasi-developable B-spline surfaces in ship hull design”. In: *Computer-Aided Design* 39.10 (2007), pp. 853–862.

[15] M. Sun and E. Fiume. “A Technique for Constructing Developable Surfaces”. In: *Proceedings of the Conference on Graphics Interface ’96*. GI ’96. `http://dl.acm.org/citation.cfm?id=241020.241071`. Toronto, Canada: Canadian Information Processing Society, 1996, pp. 176–185.

[16] D. Cohen-Steiner and J.-M. Morvan. “Restricted Delaunay Triangulations and Normal Cycle”. In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*. SCG ’03. http://doi.acm.org/10.1145/777792.777839. New York, USA: ACM, 2003, pp. 312–321.

[17] C. Wang and K. Tang. “Achieving developability of a polygonal surface by minimum deformation: a study of global and local optimization approaches”. In: *The Visual Computer* 20.8-9 (2004), pp. 521–539.

[18] K. Tang and M. Chen. “Quasi-Developable Mesh Surface Interpolation via Mesh Deformation”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.3 (2009), pp. 518–528.

[19] M. Rabinovich, T. Hoffmann, and O. Sorkine-Hornung. “Discrete Geodesic Nets for Modeling Developable Surfaces”. In: *CoRR* abs/1707.08360 (2017). http://arxiv.org/abs/1707.08360.

[20] J. Mitani and H. Suzuki. “Making Papercraft Toys from Meshes using Strip-based Approximate Unfolding”. In: *ACM Transactions on Graphics* 23.3 (2004), pp. 259–263.

[21] Y. Liu et al. “Geometric Modeling with Conical Meshes and Developable Surfaces”. In: *ACM Transactions on Graphics* 25.3 (2006), pp. 681–689.

[22] D. Rohmer et al. “Folded Paper Geometry from 2D Pattern and 3D Contour”. In: *Eurographics 2011 (short paper)*. Ed. by S. L. Nick Avis. https://hal.inria.fr/inria-00567408. European Association for Computer Graphics. Llandudno, United Kingdom, 2011, pp. 21–24.

[23] H. Pottmann. *Industrial Geometry*. Springer, 2014.

[24] T. DeRose, M. Kass, and T. Truong. “Subdivision Surfaces in Character Animation”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’98. http://doi.acm.org/10.1145/280814.280826. New York, USA: ACM, 1998, pp. 85–94.

[25] Y.-J. Liu, K. Tang, and A. Joneja. “Modeling dynamic developable meshes by the Hamilton principle”. In: *Computer-Aided Design* 39.9 (2007), pp. 719–731.

[26] E. English and R. Bridson. “Animating Developable Surfaces using Nonconforming Elements”. In: *ACM Transactions on Graphics* 27.3 (2008).