Computer Graphics
and Multimedia

RWTH AACHEN UNIVERSITY

Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehrstuhl für Informatik 8
Prof. Dr. Leif Kobbelt

Bachelor Arbeit

Towards Interactive Quadrangulation Using Local Remeshing

Patrick Schmidt
Matrikelnummer: 308691

September 2014

Erstgutachter: Prof. Dr. Leif Kobbelt
Zweitgutachter: Prof. Dr. David Bommes

Abstract

Quadrangulation of given triangle meshes receives lots of attention in ongoing research. High potential is expected in closing the gap between manual and automatic techniques.

This thesis analyzes the parameterization based *mixed-integer quadrangulation* method and proposes an extension allowing subsequent local updates. These form the core of new quad meshing tools enabling an interactive workflow based on an initial automatic solution.

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht zu haben.

Aachen, September 24, 2014

Patrick Schmidt

Contents

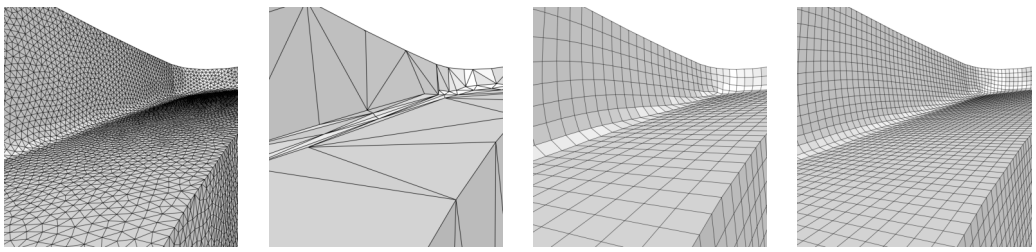
Chapter 1

Introduction

Polygonal meshes are the representation of choice for geometric objects in numerous applications. Most common in large fields of geometry processing is the use of triangular meshes, for which a vast number of algorithms has been created. In 3D rendering, a majority of techniques as well as dedicated hardware are based on triangles. Among other properties, triangles are popular due to their simplicity and the fact that any kind of polygon can easily be split into triangles.

Quadrilateral Meshes However, in some domains meshes consisting entirely of quadrilaterals (quads) are preferred. One central reason for this is that they allow for better alignment to the structure of the object they represent.

More specifically, the local shape of a surface naturally incorporates two main directions of curvature (the principal curvature directions) which are



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

Figure 1.1: Different representations of the same object. (a) shows an uniformly tesselated triangle mesh. (b) is a decimated version of (a). (c) shows an aligned quadrangulation and (d) is a subdivided version of (c)

always orthogonal to each other. A quad mesh can be aligned to these directions in such a way that following a straight path of edges means following the directions of e.g. maximal curvature. Doing the same with triangles only determines meaningful orientation for two edges per triangle, leaving an unnecessary degree of freedom for the third one (cf. [Bom+12] p. 3).

Quadrangulation Nevertheless, in practice most models exist in a triangular representation. This is either due to the use of triangle based sculpting tools or because they have been obtained by triangulating a point cloud scanned from a real object. As manually creating a corresponding quad mesh is a time-consuming and difficult task, automatic quadrangulation of input triangle meshes is a highly demanded feature. One of today's state-of-the-art techniques for this task is the mixed-integer quadrangulation pipeline introduced by Bommes et al. in [BZK09].

Depending on the intended application though, quality criteria of the resulting quad mesh are diverse. Some are even subjective to the respective artists and cannot be formalized. Thus, in general a fully automatic triangle to quad conversion will not be able to entirely satisfy all requirements and manual adjustments will always play a role (cf. [Bom12] p. 18).

Local Quad Remeshing Although the existing mixed-integer method allows some subsequent user interactions, these provide no immediate visual feedback since the entire algorithm has to be started again to see the effects. For high-resolution models this may take at least a few minutes and is due to the fact that each change has global impact on the final result.

This situation calls for new methods that allow more direct interaction. A promising approach is to (after computing an initial global solution) restrict the effect of such operations to a local region while keeping the rest of the mesh fixed. Since the local computation takes significantly less time, it is possible to provide fast feedback to the user. We call this approach *local remeshing*.

Contributions This thesis will give a first impression if and to which extent local quad remeshing is a way to establish an interactive workflow for quadrangulating given triangle meshes. For this purpose, the existing mixed-integer method is analyzed and then extended to support local updates. Based on that, an exemplary interactive tool is created and evaluated with respect to mesh quality, performance and integration into the overall process.

Thesis Structure The remaining part of chapter 1 gives some additional introduction into the general field of quad meshing by describing two common applications (Section 1.1) and summarizing the most important quality criteria (Section 1.2). In Section 1.3 a brief overview of different quadrangulation methods as well as other interactive approaches is given.

Chapter 2 is dedicated to the existing mixed-integer quadrangulation pipeline while Chapter 3 explains the modifications that are necessary to implement local remeshing. In its last section, a specific interactive UI metaphor exploiting local remeshing is introduced.

Finally, chapter 4 gives an overview over the achieved results and evaluates the merit of the new tool. Problems of the current approach are illuminated, which leads to multiple emerging research questions.

1.1 Applications

The advantages of high-quality quad meshes are exploited in various fields of application. Following [Bom12], character animation and physical simulation provide two popular examples. Despite their differences, both of them often use coarse quad meshes as control cages for either subdivision surfaces or continuous representations such as NURBS. Since the shape of these control cages directly defines the final result, the quality of the quad mesh is of great importance.

Character Animation When designing smooth objects like animatable characters, it is common practice to model a coarse quad mesh and then apply a subdivision scheme (e.g. Catmull-Clark [CC78]) to obtain a more densely tessellated representation (cf. [Bom12] p. 17f). On the one hand, this brings the advantage of being able to arbitrarily switch between different levels of subdivision during the process. On the other hand, the quad mesh can be aligned to the anticipated stretching directions in animation to reduce artifacts.

If using texture or displacement maps, it is possible to lead the inevitable distortion of these maps into a desired direction. When for example designing a human arm, the edge flow is chosen to form individual loops around it as well as uninterrupted lines from shoulder to wrist. In addition, a higher density of loops can be chosen around the elbow to still guarantee a sufficient mesh resolution if the arm is bent. For this purpose, explicit control over the resulting quad mesh is of great use to the designer.

Artists refer to (manually or automatically) converting a model from any other kind of representation into a quad mesh as the *retopology task*.

Physical Simulation In many cases, the use of quad meshes also benefits simulations like finite element analysis.

If used as discretization of a continuous surface, a badly tessellated mesh can have a large negative influence on the stability of a simulation. So quad meshes optimized for individual element quality (see Section 1.2) are a popular choice to represent the subject of simulation. In contrast to the animation example, capturing geometric details is not the only requirement for a quad structure. Here, the mesh resolution directly controls the accuracy of a simulation. Thus, even in flat regions without any geometric features, a fine tessellation is necessary. This however, can again be achieved by subdivision schemes as used for animation. Sometimes adaptive refinement is used in addition to obtain a more accurate result in some regions of the mesh.

In a similar way, quad meshes are used as control cages for B-Splines which provide a mathematically precise representation of continuous surfaces. As in subdivision, a high quality control mesh is crucial for satisfying results (cf. [Bom12] p. 19f).

1.2 Quality Criteria

Despite the fact that quality criteria of quad meshes differ from application to application while some are even completely subjective to the artist, it is still worth trying to formalize some of them. [BZK09], [Bom+12] and [Bom12] give the following list of criteria which we can roughly divide into the categories of *local* and *global criteria*.

Local Criteria

1. **Individual Element Quality:** Each quad is supposed to be as close as possible to a square (or a rectangle, if anisotropy is desired). The deviation from this ideal can be measured by the distance of the four vertices to a common plane, the difference of each interior angle from 90° and the length difference between opposing edges.

2. **Quad Orientation:** Edges should follow the principal curvature directions if these are well defined. Different confidence metrics are possible to determine how meaningful the curvature at a specific point is. Apart from that, and depending on the application, even completely different kinds of orientation fields can be used as a reference. A quality measurement can be obtained by locally computing the deviation from such a field.

Global Criteria

3. **Feature Alignment:** Sharp feature lines of the input object have to be consistently represented by a path of edges. Violating this criterion has extremely bad influence on the visual appearance and supports normal noise. A measurement is possible using the Hausdorff-distance, which makes this a global criterion. Problems arise if the input mesh itself is noisy.

4. **Placement of Irregular Vertices:** In most cases, vertices of valence other than four (called singularities) are necessary to capture the geometry of an object. Choosing their position, number and degree, introduces a trade-off between using many of them to enable better alignment and keeping the mesh as regular as possible.

5. **Additional Requirements:** As seen before, based on the particular application many other criteria are possible which can or cannot be rated objectively.

After all, even for criteria that can be formally measured, it is still a matter of preference how to weight them against each other.

1.3 Related Work

During the past decade, numerous works on the topic of automatic or semi-automatic quad mesh generation have been published. Since this section can only give a glimpse on a subset of them, the reader is referred to [Bom+12] for a comprehensive overview.

Generally, recent quadrangulation techniques can be split into the classes of *explicit* and *parameterization based* approaches:

Explicit Approaches Often, authors strive to directly alter the given topology of a mesh by turning adjacent triangles into a quads. In [Rem12] for example, a minimum-cost-perfect-matching problem is solved to find appropriate pairs of triangles. However, in general the result of this method is a pure but unstructured quad mesh.

To incorporate alignment to the geometric shape of an object, the authors of [MK04] trace lines on the surface of an object by following the two principal curvature directions. Instead of modifying the original topology, a quad-dominant mesh is obtained by creating vertices at intersections of these lines and inserting edges along line segments.

A combination of both concepts is used in [LKH08] where in a first step vertices of the existing triangle mesh are relocated such that a subset of edges follows the principal curvature directions. After that, the remaining diagonal edges are removed which again leads to a quad-dominant mesh.

Parameterization Based Approaches More attention has recently been paid to parameterization based approaches which all share the following pattern: The input mesh is mapped to a grid in a 2D parameter domain which, mapped back to the surface, determines the structure of a quad mesh. In [BZK09] these methods are again divided into two subclasses:

High-level methods like [Don+06], [Ton+06], [Hua+08] and [Zha+10] first start with a coarse layout of quadrangular patches which are then parameterized individually. By defining transition functions at their boundaries, compatibility between adjacent patches is assured resulting in a *globally smooth parameterization*. Since singularity positions are determined by the initial patch layout, this strongly influences the final result and a lot of effort is put into its computation (e.g. [CBK12]).

A different approach is taken by *low-level* methods such as [Ray+06] and [KNP07] where the parameterization is guided by a global orientation field that now dictates the placement of singularities. The recently most cited work in this class is [BZK09] which this thesis is mainly based on. In [Bom+13] this method is modified to provide reliable results even for extremely coarse quadrangulations. Another extension is made in [MZ13] which makes it possible to directly control the trade-off between element distortion and the number of singularities.

For all parameterization based techniques, the last step consists of extracting the final quad mesh from the mere images of grid lines. A robust way to do this is provided by [Ebk+13].

Interactive Approaches The field of interactive methods puts the user into the center of the process. It reaches from tools supporting manual mesh creation to semi-automatic techniques that are guided by the user in some steps.

In [Tak+13] and [TPSH14] a sketch-based framework for manual remeshing is proposed. It allows the user to draw a rough patch layout on the original object and assists by suggesting autocompletion candidates. The automatic quadrangulation of each patch can further be influenced by controlling the placement of interior singularities and the number of subdivisions.

A multilevel approach is taken in [Tie+12]. On the first level, a coarse mesh segmentation (by a so called *Reeb atlas*) is created semi-automatically

by specifying extraordinary vertex positions and manipulating proposed patch boundaries. After each resulting patch has been parameterized individually, the user can choose between mapping a uniform grid or a different 2D quadrangulation of the parameter domain (*connectivity texture*) back onto the surface. The proposed workflow allows to frequently switch between both levels. In a final automatic step, conflicts between adjacent patches are resolved and patch boundaries are stitched together.

Both interactive methods initially require a substantial amount of effort before any outcome can be obtained. In contrast to this, our approach aims to provide a complete *relaxed integer grid map* [Ebk+13] at any time along the process by starting with an automatically obtained solution which is subsequently altered by the user.

Chapter 2

Mixed-Integer Quadrangulation

This chapter describes the *mixed-integer quadrangulation* (MIQ) method introduced by Bommes et al. It first briefly summarizes the overall idea and then gives detailed explanations focusing on the parts that are relevant to this work. Where not stated otherwise, the contents of this chapter are based on the original publication [BZK09] as well as on the corresponding implementation.

Parameterizations The goal of this method is to obtain a 2D parameterization of a given 3D triangle mesh which can then be used to extract a quad mesh.

A *parameterization* F is a mapping from a parameter domain $\Omega \subseteq \mathbb{R}^2$ to a surface $\mathcal{S}$ embedded in $\mathbb{R}^3$. Often, not the function F itself but its inverse $f := F^{-1}$ is regarded. In the case of a discrete triangle mesh $\mathcal{M} = (V, E, T)$, f usually is a piecewise linear function that maps each vertex

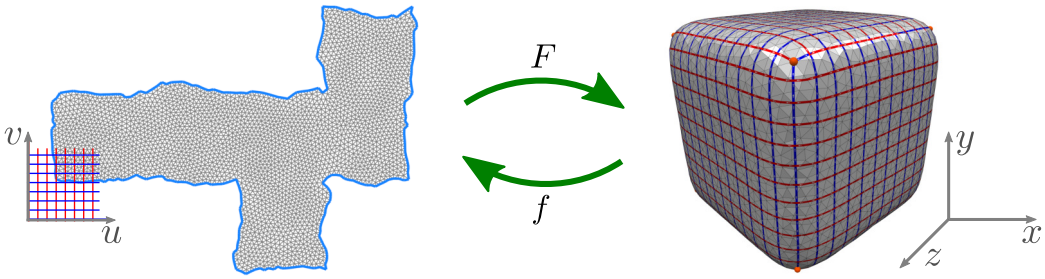


Figure 2.1: The function F maps from the 2D parameter domain (left) to the 3D mesh (right). The other way around, its inverse f maps each point $(x, y, z)^T \in \mathcal{S}$ on the surface to a parameter value $(u, v)^T \in \Omega$.

$v \in V$ to a position $(u, v)^T \in \Omega$ in the parameter domain. An intuitive way of understanding this is to see it as an unfolding of the mesh to a flat surface (see fig. 2.1).

The Cartesian grid, i.e. the set of points in $\mathbb{R}^2$ for which at least one of the parameters u, v is an integer, is called the *parameter grid*. Its individual lines are called *integer-iso-parameter lines*.

We are looking for a function f such that these parameter lines mapped back to the surface form a network with quad mesh topology. Points in which multiple iso lines meet, i.e. u and v are both integers, are intended to be vertices of this quad mesh. The process of obtaining such a parameterization and extracting a quad mesh is split into multiple steps forming a pipeline.

As pipeline input, we regard a triangle mesh that is already equipped with some directional information in important feature regions. Such information consists of directional vectors assigned to a subset of triangles. These directions will be used as guidance for the edge flow of the eventual quad mesh. It is of great importance for the success of this method to only use a sparse set of constraints. This way, there are enough degrees of freedom left to adjust the solution in non-feature regions.

The MIQ pipeline consists of the following steps:

1. Computing a Smooth Cross Field In the first step, these directional guidances have to be propagated over the entire mesh in a reasonable way. Consequently, a direction is now assigned to each face. In constrained triangles, these directions exactly match the already given ones, whereas in free faces they are chosen to interpolate the constraints as smooth as possible.

Since the intended result is a quad mesh, directions should be indifferent to rotations by 90°. Thus, the directional vectors are extended to crosses living in the tangent plane of the respective triangle. Each cross has a single

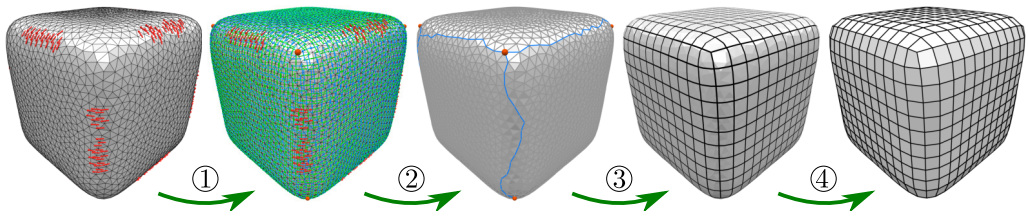


Figure 2.2: Steps of the MIQ Pipeline: (1) Computing a smooth cross field. (2) Cutting the mesh open. (3) Computing a global parameterization. (4) Extracting the quad mesh.

degree of freedom (its rotation in the tangent plane) and is invariant under rotations by a multiple of 90°.

The resulting smooth cross field is used as input for the next steps. An important property is that the singularities (see Section 2.1) of this cross field already determine the position and degree the irregular vertices of the quad mesh.

2. Cutting the Mesh To be able to unfold the mesh into a 2D parameter domain, it has to be topologically equivalent to a disk. This is achieved by cutting the mesh open along paths of edges. Depending on the genus of the object, a certain number of cuts is necessary. In addition, cuts to singularities have to be added.

3. Computing a Global Parameterization In this step, the eventual parameterization is computed by assigning a pair of (u, v) coordinates to each vertex. This way, the piecewise linear function $f : \mathbb{R}^3 \to \mathbb{R}^2, (x, y, z)^T \mapsto (u, v)^T$ is defined.

The (u, v) coordinates are chosen in such a way, that the directions of the iso-parameter lines mapped back onto the mesh match the given cross field directions from step 1 as well as possible.

4. Quad Mesh Extraction The fourth step of extracting the final quad mesh is not covered by this work. The problem of generating a valid quad mesh even from degenerated parameterizations including numerical inaccuracies has been successfully solved in [Ebk+13].

In the following sections, steps 1-3 are explained in more detail with respect to their relevance for our changes in Chapter 3.

2.1 Computing a Smooth Cross Field

This section explains in detail, how a smooth cross field is computed based on some sparse directional constraints.

Problem Setting For each triangle t_i we want to obtain the rotation of its cross which is expressed by a single angle θ_i. This angle is measured between a reference vector and the main axis of the cross. The reference vector can be chosen arbitrarily but fixed for each triangle, e.g. as the direction of one of its edges. The other three axis of the cross are obtained by iteratively rotating the main axis by 90°. (See figure 2.4)

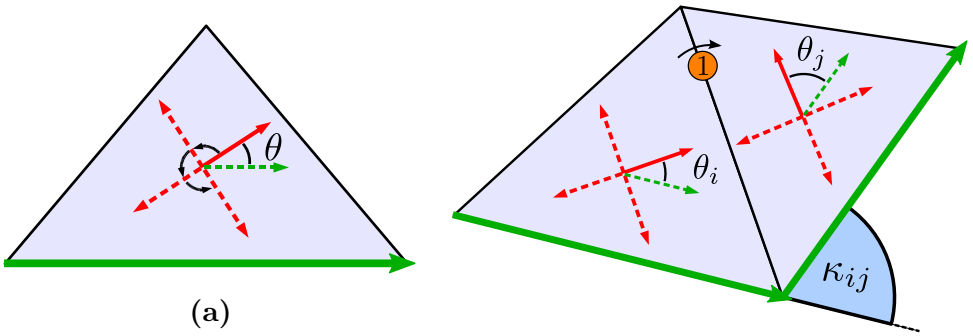


Figure 2.3

Figure 2.4: (a) Reference direction (green), main axis (red) and rotations by 90° (red, dashed) of a cross. (b) Perfectly aligned crosses in neighboring triangles with period jump $p_{ij} = 1$. (Figure adapted from [BZK09])

Smoothness Energy Now, an energy is formulated which measures the deviation between two crosses in neighboring triangles t_i and t_j. Because their angles θ_i and θ_j are measured with respect to different reference edges and the crosses are defined in different planes, they first have to be transformed to the same coordinate system. This is done by adding an angle κ_{ij} which is obtained by unfolding the pair of triangles into a common plane and then taking the angle difference between their respective reference edges. Thus, $\theta_i + \kappa_{ij} - \theta_j$ measures the deviation between the two main axis.

Furthermore, to compensate rotations by multiples of 90°, an integer variable p_{ij}, called *period jump*, is introduced. It tells how many of these rotations have to be applied to the cross in t_i to match the one in t_j as well as possible.

Altogether, the energy between two neighboring crosses can be expressed as

$$\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j.$$

By summing up the squared energies over the entire mesh, a global smoothness energy is obtained:

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

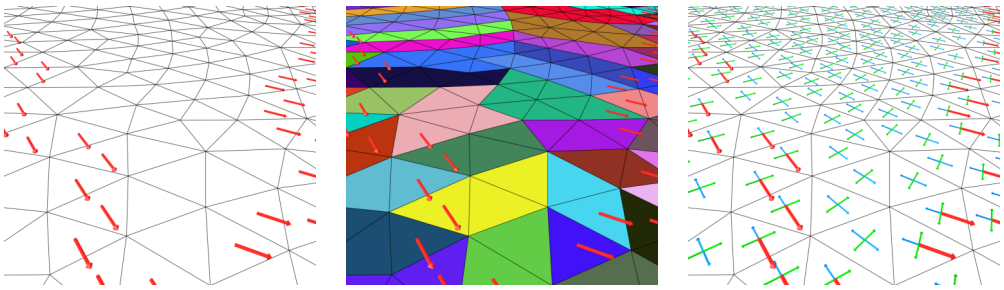
When setting up the equation system, κ_{ij} will be fixed real values per edge, p_{ij} will be free integer variables per edge and θ_i, θ_j free real variables per face.

Minimizing the Energy Since we want to make the resulting cross field as smooth as possible, this energy has to be minimized. Because E_{smooth} is a positive quadratic function, its minimizer can be found by setting its gradient to zero:

$$\nabla E_{\text{smooth}} = \begin{pmatrix} \vdots \\ \frac{\delta E_{\text{smooth}}}{\delta \theta_k} \\ \vdots \\ \frac{\delta E_{\text{smooth}}}{\delta p_{ij}} \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \sum_{e_{kj} \in N(t_k)} 2(\theta_k + \kappa_{kj} + \frac{\pi}{2} p_{kj} - \theta_j) \\ \vdots \\ \pi(\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j) \\ \vdots \end{pmatrix} \stackrel{!}{=} 0$$

The adaptive greedy solver described in section 2.4 can be used to get a good approximation of the solution. To do so, $\nabla E_{\text{smooth}} \stackrel{!}{=} 0$ is written as the linear system $Qx \stackrel{!}{=} 0$ with x being the result vector $(\dots \theta_k \dots p_{ij} \dots 1)^T$. Since the coefficient matrix Q is symmetric positive definite, we can instead regard its factorization $Q = B^T B$ and set up the more intuitive matrix B. Each column of B corresponds to an element of x and each row to an edge e_{ij}. Filling each row with the coefficients of $\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j$ gives us $B^T B = Q$. So we can compute Q and pass it to the solver.

Furthermore we set up a matrix C which allows us to encode a set of linear constraints to the variables of the system. The solver guarantees that $Cx = 0$ is always true. So for all cross field angles which are already given, and thus have to stay fixed, we add a row to C setting the k-th column to 1 and the last column to $-\theta_k$.



(a) **(b)** **(c)**

Figure 2.5: (a) A subset of faces is equipped with directional constraints. (b) The discrete Voronoi cells of constrained triangles are used to eliminate redundant degrees of freedom. (c) The final cross field interpolates the given directions smoothly.

Removing Redundant Degrees of Freedom Up to now, infinitely many solutions are possible. To demonstrate this, we pick an arbitrary triangle t_k for which an optimal θ_k has been computed. We can now rotate θ_k by an arbitrary multiple of 90° and compensate the rotation by adjusting the period jumps of the three incident edges without changing the total energy.

The solution can be made unique by the following strategy. Each fixed triangle is chosen to be the root of a dual spanning tree. We let these trees grow such that each triangle is contained in exactly one tree and is connected to its root node on a shortest path.

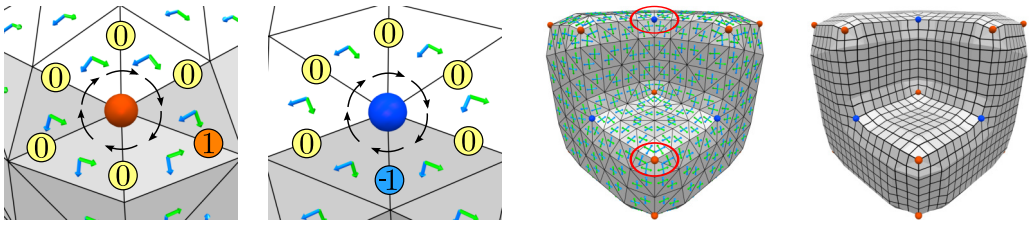
The resulting forest of spanning trees corresponds to a discrete Voronoi diagram (fig. 2.6b) of the constrained faces. According to [BZK09] we can now fix the period jump of each primal edge of the Dijkstra forest to an arbitrary value, which leaves us with exactly one minimal solution without changing its energy. The respective constraints are also added to the matrix C.

After the solver finished its job, the optimal rotation is applied to all crosses.

Singularities An important property of this approach is that it already provides us with the position and degree of singularities. For this reason, the cross field *index* of a vertex v is defined as

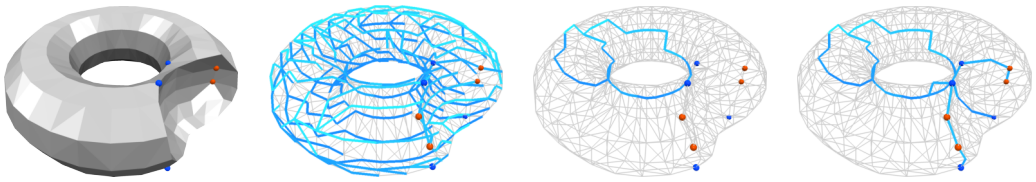
$$I(v) = I_0(v) + \sum_{e_{ij} \in N(v)} \frac{p_{ij}}{4}. \tag{2.1}$$

Here, $I_0(v)$ is the *base index* which is a constant integer determined by the angles κ_{ij} around v, and is further explained in [Ray+08]. It is important to consistently sum up the period jumps in a clockwise manner, since they are antisymmetric, i.e. $p_{ij} = -p_{ji}$.



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

Figure 2.6: Singularities of index $\frac{1}{4}$ (a) and $-\frac{1}{4}$ (b) which are placed in geometrically meaningful locations (c). They later correspond to vertices of degree 3 and 5 (d).



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

Figure 2.7: (a) A genus one object with some singularities. (b) The original cut graph used to achieve disk topology. (c) The reduced cut graph after removing open paths. (d) The final cut graph connecting all singularities.

Vertices with an index other than zero are called *singularities*. At a singularity it is not possible to align a regular quad mesh, therefore these will later correspond to irregular vertices. More specifically, a singularity of index I will result in a quad mesh vertex of valence $-4I + 4$.

2.2 Cutting the Mesh Open

In this chapter the computation of a valid cut graph is explained briefly. More details are given in section 3.3.

The cut graph $G = (V, E^{\text{cut}})$ contains a subset of the mesh's vertices and edges $V^{\text{cut}} \subseteq V$, $E^{\text{cut}} \subseteq E$. The mesh will be cut open along this graph when it is being unfolded into the parameter domain. To allow a valid parameterization, the cut graph has to fulfill two requirements:

1. After cutting, the mesh has to be topologically equivalent to a disk, which is required to allow a planar unfolding at all.

2. All singularities have to be connected to the cut graph. This allows a parameterization such that a number of iso-parameter lines other than four can meet in such a point.

1. Disk Topology This property is established by picking a random face as root and growing a dual spanning tree. Now all primal edges whose dual is not contained in the tree already induce a valid cut graph. Growing the spanning tree can be seen as iteratively gluing faces together while all others remain cut.

Since the cut graph is now quite large (figure 2.7b), we can start stitching faces back together along unnecessary paths. To do so, we iteratively remove

each vertex from V^{cut} that has only one incident edge in E^{cut}. The remaining graph will be much smaller but is still sufficient to achieve disk topology.

In case of a genus zero object like a cube, the result is a single vertex. On a torus (genus one) this cut graph consists of two circles along its main axis. In general, an object of genus g requires $2g$ cuts.

2. Connecting Singularities In a second step, all singularities are iteratively connected to the existing graph. For each one, a simple Dijkstra search is started and the shortest path to any vertex of the graph is added.

After cutting, vertices contained in V^{cut} will have an *instance* on each side of a cut. More specific, a vertex with n incident cut edges ($n \geq 1$) has n instances which may be mapped to different locations in the parameter domain.

2.3 Computing a Global Parameterization

In this last step, we aim to find the eventual parameterization function $f : \mathcal{S} \subseteq \mathbb{R}^3 \to \Omega \subseteq \mathbb{R}^2$ which is aligned to the already computed cross field as well as possible. In order to imply an actual quad mesh, some additional requirements have to be fulfilled as explained later.

Orientation Energy The piecewise linear function f is defined by assigning a pair of (u, v) values to each vertex. Along a cut, each instance receives its own (u, v) pair.

Similarly to section 2.1, we formulate an energy that describes how well the alignment of a parameterization matches the given cross field. For this purpose we split f into two scalar functions u and v with $f = (u, v)^T$ and regard their gradients ∇u and ∇v. The gradient $\nabla u : \mathbb{R}^3 \to \mathbb{R}^3$ maps a point on the surface to the direction in which the parameter u increases maximally. Locally, this gradient is perpendicular to the direction of a parameter line mapped back onto the surface. The same holds for the other parameter v.

Because the functions u, and v are both linear within a triangle, we can in practice define them individually for each face using a local coordinate system. This reduces the dimensions to $u, v : \mathbb{R}^2 \to \mathbb{R}$ and $\nabla u, \nabla v : \mathbb{R}^2 \to \mathbb{R}^2$ respectively.

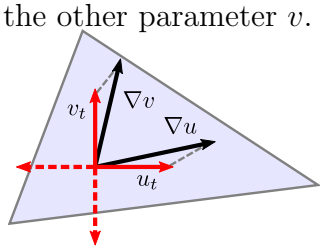


Figure 2.8: Local deviation of the gradients ∇u, ∇v from the cross field directions u_t, v_t.

Given normalized cross field directions u_t and v_t for a single triangle, we can express the de-

viation of the gradients from the cross field by $||\nabla u - u_t||$ and $||\nabla v - v_t||$. (See figure 2.8.)

To control the edge length of the resulting cross field, an additional scaling parameter h is introduced. Again, we are interested in the squared deviation which leads to the energy per triangle defined as

$$E_t = ||h\nabla u - u_t||^2 + ||h\nabla v - v_t||^2.$$

To obtain a global energy for the entire mesh, these values are summed up while each one is weighted with its corresponding triangle area A_t:

$$\begin{aligned} E_{\text{orient}} &= \sum_{t \in T} E_t A_t \\ &= \sum_{t \in T} \left(||h\nabla u - u_t||^2 + ||h\nabla v - v_t||^2 \right) A_t \\ &= \sum_{t \in T} \left((h\nabla u_u - u_{tu})^2 + (h\nabla u_v - u_{tv})^2 + (h\nabla v_u - v_{tu})^2 + (h\nabla v_v - v_{tv})^2 \right) A_t \end{aligned}$$

Minimizing the Energy We are again dealing with a positive quadratic function that can be minimized by setting its gradient to zero:

$$\nabla E_{\text{orient}} = 2h \begin{pmatrix} \vdots \\ hA_t\nabla u_u - A_t u_{tu} \\ hA_t\nabla u_v - A_t u_{tv} \\ hA_t\nabla v_u - A_t v_{tu} \\ hA_t\nabla v_v - A_t v_{tv} \\ \vdots \end{pmatrix} \stackrel{!}{=} 0$$

Unfortunately, the unknowns in this system are the gradients ∇u and ∇v, but we want to obtain the functions u and v itself. Luckily, we can write ∇u, which is constant within a triangle, in dependency of the u values of its three vertices u_0, u_1, u_2:

$$\nabla u = \frac{1}{2A_t} \sum_{i=0}^{2} e_i^{\perp} u_i$$

Here, $e_i^{\perp}$ is the edge vector opposite to vertex i rotated by 90° so that it points into the triangle. Of course, the same works for ∇v. Inserting this

into the above equation gives us:

$$\nabla E_{\text{orient}} = 2h \begin{pmatrix} \vdots \\ \frac{1}{2}h\left(e_{0u}^{\perp}u_0 + e_{1u}^{\perp}u_1 + e_{2u}^{\perp}u_2\right) - A_t u_{tu} \\ \frac{1}{2}h\left(e_{0v}^{\perp}u_0 + e_{1v}^{\perp}u_1 + e_{2v}^{\perp}u_2\right) - A_t u_{tv} \\ \frac{1}{2}h\left(e_{0u}^{\perp}v_0 + e_{1u}^{\perp}v_1 + e_{2u}^{\perp}v_2\right) - A_t v_{tu} \\ \frac{1}{2}h\left(e_{0v}^{\perp}v_0 + e_{1v}^{\perp}v_1 + e_{2v}^{\perp}v_2\right) - A_t v_{tv} \\ \vdots \end{pmatrix} \stackrel{!}{=} 0$$

This linear system can again be written in matrix notation as $Bx \stackrel{!}{=} 0$ with real variables u_i, v_i.

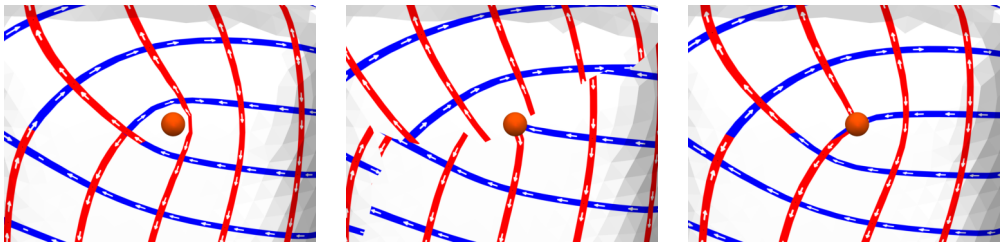
Singularities at Integer Locations If the system is solved with all unknowns being real numbers, no vertex is required to lie on an iso-parameter line. For singularities this means the parameter grid mapped back to the surface can not only form irregular vertices but also irregular faces (See figure 2.9a). To prevent this and create a quad-only mesh, singularities are snapped to junctions of the parameter grid by requiring both its u and its v coordinate to be integral. This way, a quad mesh with irregular vertices instead of irregular faces is created.

Cut Compatibility As shown in figure 2.9b the parameterization on both sides of a cut is not yet expected to match. In order to remove visible seams, the relation between parameter values on both sides has to be constrained. Formally, the mapping between the (u, v) coordinates on either sides is called the *transition function*. Because the desired result is a quad structure, rotations by a multiple of 90° as well as translations by full integers are allowed. Thus, for each cut edge $e = \overline{pq}$ we set up the conditions

$$\begin{aligned} (u'_p, v'_p) &= Rot_{90}^{i_e}(u_p, v_p) + (j_e, k_e) \\ (u'_q, v'_q) &= Rot_{90}^{i_e}(u_q, v_q) + (j_e, k_e). \end{aligned} \tag{2.2}$$

A transition function limited to these degrees of freedom is called a *grid automorphism* [KNP07]. As described later, the number of rotations i_e can be precomputed. This leaves us with two additional integer variables j_e, k_e per cut edge that describe the translational part of the transition function.

Thus, we expand the matrix B by the appropriate number of columns, filling all of them with zeros, since the actual values have no impact on the quality of our solution. The result vector x is now of the form $(\dots u_p, v_p \dots j_e, k_e \dots 1)^T$ for vertices p and cut edges e.



(a) (b) (c)

Figure 2.9: (a) A singularity not constrained to an integer location. (b) Incompatible parameterizations on either sides of a cut. (c) Parameterization with singularities on integer positions and cut compatibility.

As done in the cross field computation, we also set up a constraint matrix C for which the solver guarantees the equation $Cx = 0$ to hold. Per cut edge, we add four rows representing equation 2.2.

Still, there are infinitely many solutions to the problem, as the image of the mesh inside the parameter domain can be arbitrarily translated by integers and one of four possible rotations can be chosen.

The translational issue can be solved by fixing a single vertex to an arbitrary parameter value e.g. the origin. To introduce no change to the parameterization energy, it is important to choose a singularity if possible.

Consistent Orientation As an additional pre-processing step before setting up the problem matrices, the existing cross field is altered in order to achieve *consistent orientation*. This means, the main axis of all crosses roughly point into the same direction. More specific: the period jump between two neighboring triangles is zero. Only on cut edges, a period jump other than zero is permitted, which is necessary to allow for rotations in the transition function.

The algorithm establishing this property works as follows:

1. Pick an arbitrary triangle.
2. Set the period jump of all its edges to zero. Compensate this change by rotating the neighboring crosses by multiples of 90°.
3. Propagate this pattern in a breadth first manner. Stop at cut edges.

This method iteratively drags period jumps to the cut edges which solves multiple problems:

- In figure 2.8 it was not clear which axis of the cross had to be chosen as u_t and v_t. Since crosses are now oriented consistently, we can choose e.g. the main axis as u_T and the one rotated by 90° to the left as v_T.
- This also fixes the rotation of the entire mesh in the parameter domain to the orientation of the first triangle we picked.
- The rotational part of the transition function i.e. the i_e in equation 2.2 can now be computed by simply comparing the crosses main axis on both sides of the cut.

We are now ready to finally assemble the matrices B and C and pass them to the solver, which gives us the eventual parameterization function f defined by a (u, v) coordinate for each vertex instance. We are also provided with values for j_e, k_e which can be ignored in our case.

2.4 The Mixed-Integer Solver

In this section, the mixed-integer solver used in 2.1 and 2.3 will be described briefly with respect to the information that is used in the following chapter.

The solver, which was specifically designed to perform well in the context of quadrangulation, was first presented in [BZK09]. For a full description and some improvements see [BZK12]. Even more details and extensive background information is provided in [Bom12].

Problem Definition The task of the solver is to minimize a quadratic energy function $E : \mathbb{R}^n \times \mathbb{Z}^m \to \mathbb{R}$ that depends on some real-valued as well as some integer variables. In addition, some linear equality constraints of the form $C_i^T x - d_i = 0$ with $C_i \in \mathbb{R}^{n+m}$ and $d_i \in \mathbb{R}$ are given. As seen before, E can be minimized by setting its gradient ∇E to zero and solving a linear system.

Since, due to the integer variables, finding an optimal solution to this problem is $\mathcal{NP}$-hard [Flo95], it is approximated using an adaptive greedy approach that gives good results in practice.

Greedy Rounding The main idea of the algorithm is to first treat all unknowns as real variables and then iteratively round some of them to the closest integer value.

Therefore, initially a continuous solution $x^0 \in \mathbb{R}^{n+m}$ for the so called *relaxed problem* is computed. After that, from all variables that should become integers, the one with the smallest rounding error $|round(x_j) - x_j|$ is chosen

and fixed to $round(x_j)$. Then, the remaining continuous system is updated locally, i.e. all real variables that now introduce a residual above some error bound ε are adapted via a Gauss-Seidel step. This is repeated until all integer constraints are fulfilled.

In [BZK12], even better performance was achieved by rounding a set of variables, which only have low influence on each other, simultaneously.

Linear Constraints For each given linear equality constraint, one real variable of the system can be eliminated in a pre-processing step. However, it is possible that some constraints are linear dependent. In this case no additional information is given by the dependent constraint and it will be ignored.

Although the solver produces very good solutions in most cases, in the next chapter we can observe some issues that are based on its greedy strategy.

The existing quadrangulation pipeline described here as well as the adaptions explained in chapter 3 use the open source C++ implementation CoMISo available at [Com10].

Chapter 3

Local Remeshing

This chapter is dedicated to a new method called local remeshing. After giving some motivation and introducing the general idea of this approach, the necessary modifications to the mixed-integer quadrangulation pipeline are discussed in detail. Finally, an exemplary interactive tool which makes use of the new technique is presented.

By construction of the MIQ pipeline, the quality of the resulting quad mesh strongly relies on the initial set of directional constraints. In the current state of the project, these are created by a simple heuristic. This heuristic uses principal curvature directions, in regions where they stable enough, as cross field constraints (cf. [BZK09]). The user can influence the resulting number of constraints by setting a confidence threshold. Apart from this, many other global approaches for feature detection are possible. More sophisticated ones are discussed in [Ibi14].

After running the MIQ pipeline up to the parameterization step, the resulting quad structure can be inspected by the user. As already seen in the introduction, quality criteria are diverse which makes it likely that the user is not satisfied and wishes to choose a different set of constraints. Now, one option is to tweak some parameters and use the same mechanism a second time. As an alternative to using automatically obtained constraints again, certain manual user interactions are possible, all of which share the same pattern: First, the existing set of constraints is manipulated and then the entire parameterization pipeline is executed again to obtain an updated result. As an example, we focus on two main kinds of operations:

Manual Feature Lines By drawing a line on the mesh, the cross field angles of all affected triangles are set to a fixed value. Since the eventual parameterization is only aligned to these directions in a least squares sense,

they are called *soft constraints*.

If in addition, constraints are added to the parameterization system such that the specified path is mapped exactly to an iso-parameter line, we speak of *hard constraints*.

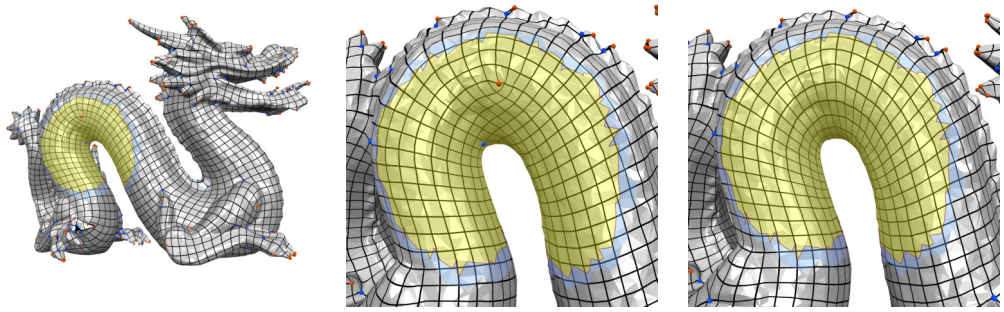
Index Constraints By constraining the sum of period jumps around a certain vertex, it is possible to directly control its cross field index. In other words, we can arbitrarily determine the position and degree of singular vertices. This can be used to let the user drag existing singularities to new locations or even merge multiple singularities of low degree to a single one of higher degree and vice versa.

These and other operations proved to be quite powerful to manually improve the resulting quad structure to the specific needs of an application. Due to the fact that each change has global impact on the eventual parameterization, providing any visual feedback to the user means running the entire algorithm again from the very beginning. Unfortunately, this can take a significant amount of time (e.g. more than a minute for a mesh with $300,000$ triangles), which makes it almost impossible to establish a reasonable workflow in practice.

This situation forms the basis for this thesis. A promising approach to make the process of modifying a quadrangulation more interactive is provided by *local remeshing*.

Local Remeshing As before, we assume that an initial global parameterization has already been computed. Based on that, the central idea of local remeshing is to select a region on the mesh and compute an updated parameterization only within this region. On the inside, arbitrary adjustments to the constraints can be made using the operations described above. However, for those parts of the mesh that are not contained within the region, the existing parameterization stays fixed. Care has to be taken, that the local result blends into the outside solution in a seamless way. See figure 3.1 for an example.

Due to the smaller size of the problem, the local solution can be computed in significantly less time, allowing for more direct visual feedback. In addition, the local influence of an operation can help to make the result of an user interaction more predictable.



(a) (b) (c)

Figure 3.1: A local remeshing operation. (a) Full quadrangulation before the operation. (b) Badly aligned edge flow below the dragon's spine. (c) The edge flow has been corrected solely within the selected region.

3.1 Terminology

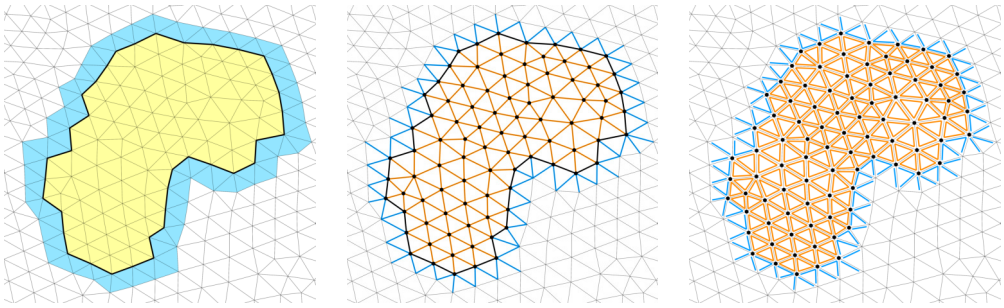
This section introduces some basic definitions and notations which will be used throughout the rest of the thesis.

3.1.1 Regions

In the following we want to restrict the impact of algorithms to a local region of a mesh. Thus, we need a precise definition of which parts of the mesh are affected. Since we want to achieve a smooth transition to the rest of the mesh, we also have to take a boundary ring around the region into account.

A region is chosen based on the input triangle mesh, not on the resulting quad mesh. There are multiple reasons for this decision: 1) The mixed-integer quadrangulation has to be applied again, which of course needs a triangle mesh as input. 2) This work only interferes with the parameterization of the triangle mesh and is independent of the final quad mesh extraction step. 3) It is way more straightforward to implement.

We are given a 2-manifold triangle mesh $\mathcal{M} = (V, E, T, HE)$ that lets us directly access its sets of vertices V, edges E and triangles T. To make some formulations easier, we also assume a set HE containing two directed half-edges per edge, which are oriented in counter-clockwise direction for each triangle. More details about half-edge representations can be seen in [Ket99] and [Bot+].



(a) (b) (c)

Figure 3.2: The individual elements of a region. (a) Inner faces are colored yellow, outer faces blue. (b) Edges are categorized into inner edges (orange), boundary edges (black) and outer edges (blue). (c) Region vertices are depicted as black dots, inner half-edges are shown in orange, outer half-edges in blue.

Definition A *region* $\mathcal{R} = (V_\mathcal{R}, E_\mathcal{R}, T_\mathcal{R}, HE_\mathcal{R})$ with $V_\mathcal{R} \subseteq V, E_\mathcal{R} \subseteq E, T_\mathcal{R} \subseteq T$ and $HE_\mathcal{R} \subseteq HE$, is uniquely determined by an arbitrary, non-empty subset of triangles that we want to modify. We call it the set of *inner faces* $T_\mathcal{R}^{\text{inner}} \subseteq T_\mathcal{R}$.

Based on this, we define the set of *region vertices*:

$$V_\mathcal{R} := \{v \in V \mid v \text{ incident to } t \in T_\mathcal{R}^{\text{inner}}\}$$

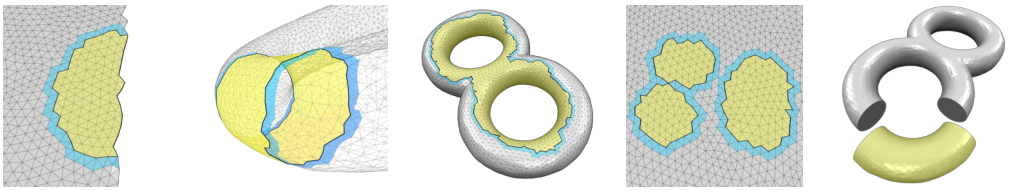
Outer faces form a ring around the inner region. This ring will be part of our computation but is kept fixed in order to make the local solution blend smoothly into the outside.

$$T_\mathcal{R}^{\text{outer}} := \{t \in T \mid t \text{ incident to } v \in V_\mathcal{R} \text{ and } t \notin T_\mathcal{R}^{\text{inner}}\}$$

Edges are split into three sets. Those that lie in the *inner* part of the region, those on its *boundary* and some *outer edges* that will also stay fixed.

$$\begin{aligned} E_\mathcal{R}^{\text{inner}} &:= \{e \in E \mid e \text{ incident to } t_1, t_2 \in T_\mathcal{R}^{\text{inner}},\ t_1 \neq t_2\} \\ E_\mathcal{R}^{\text{outer}} &:= \{e \in E \mid e \text{ incident to } t_1, t_2 \in T_\mathcal{R}^{\text{outer}},\ t_1 \neq t_2\} \\ E_\mathcal{R}^{\text{boundary}} &:= \{e \in E \mid e \text{ incident to } t \in T_\mathcal{R}^{\text{inner}}, e \notin E_\mathcal{R}^{\text{inner}}, e \notin E_\mathcal{R}^{\text{outer}}\} \end{aligned}$$

We choose all half-edges that point to a region vertex to also be part of the region. Since each half-edge is incident to exactly one face, we can split them



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.3: A region that (a) touches a mesh boundary, (b) forms a loop, (c) is of higher genus, (d) consists of multiple components (note that the left half of the region is connected and only the right part forms an individual component) and (d) covers an entire mesh component.

into sets of *inner* and *outer half-edges* as we did with the faces.

$$HE_{\mathcal{R}}^{\text{inner}} := \{he \in HE \mid he \text{ points to } v \in V_{\mathcal{R}} \text{ and } he \text{ incident to } t \in T_{\mathcal{R}}^{\text{inner}}\}$$

$$HE_{\mathcal{R}}^{\text{outer}} := \{he \in HE \mid he \text{ points to } v \in V_{\mathcal{R}} \text{ and } he \text{ incident to } t \in T_{\mathcal{R}}^{\text{outer}}\}$$

Note that all these sets are pairwise disjoint. Finally, we combine them to:

$$\begin{aligned} E_{\mathcal{R}} &:= E_{\mathcal{R}}^{\text{inner}} \quad \dot\cup \quad E_{\mathcal{R}}^{\text{boundary}} \quad \dot\cup \quad E_{\mathcal{R}}^{\text{outer}} \\ T_{\mathcal{R}} &:= T_{\mathcal{R}}^{\text{inner}} \quad \dot\cup \quad T_{\mathcal{R}}^{\text{outer}} \\ HE_{\mathcal{R}} &:= HE_{\mathcal{R}}^{\text{inner}} \quad \dot\cup \quad HE_{\mathcal{R}}^{\text{outer}} \end{aligned}$$

Figure 3.2 shows an example region and highlights its different elements. Since the initial set of inner faces can be chosen quite liberally, there are some special cases worth mentioning:

- The region can touch a mesh boundary. In this case, the ring of outer faces is interrupted.
- Extending $\mathcal{R}$ to a a closed surface can give us an object of arbitrary genus. An upper bound is given by the genus of $\mathcal{M}$.
- A region that e.g. forms a loop around a cylindrical object has multiple boundary rings. On arbitrary surfaces, regions can have an arbitrary number of holes.
- The set of inner faces is not required to be connected. In fact, a region can consist of multiple components. We define a *region component* by the property that all of its vertices can be pairwise connected by a path of edges in $E_{\mathcal{R}}^{\text{inner}} \cup E_{\mathcal{R}}^{\text{boundary}}$.

- If the region equals a complete mesh component, there are no outer faces for this component. The region may even contain the entire mesh, which gives us $T_{\mathcal{R}}^{\text{inner}} = T$. In this case we want our local algorithm to behave exactly like the original global version.

Note that algorithms described in the following chapters have to handle all cases which are allowed by the above definition. For the methods presented here, nothing of the remaining part of $\mathcal{M}$ has to be regarded, which makes them strictly local.

Disambiguation Edges can be called *boundary* in two different contexts. We speak of a *mesh boundary* if the edge has only one incident face in $\mathcal{M}$. In contrast to this, a *region boundary* separates inner from outer faces in $\mathcal{R}$. Also, one must not confuse a *mesh component*, which is connected in $\mathcal{M}$, and a *region component*, which is connected (as described above) in $\mathcal{R}$.

3.1.2 Index Mapping

In the original mixed-integer quadrangulation, as explained in the previous chapter, we assumed that each face, edge and half-edge has a unique index. We directly used these indices to address rows and columns of the matrices we set up. Since now we want to set up smaller matrices for the local region only, we cannot use these global indices anymore.

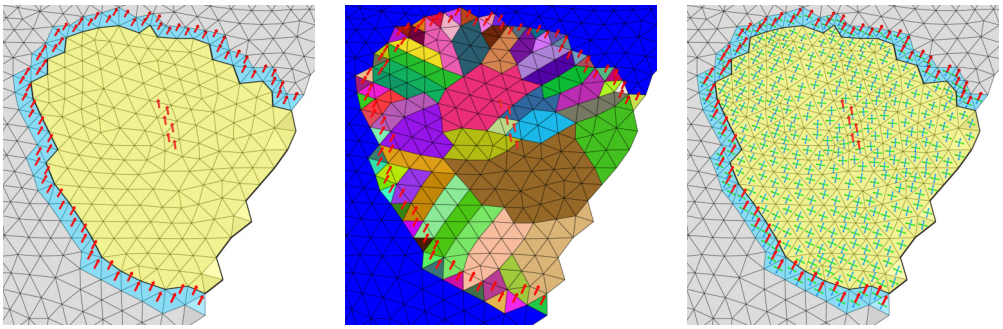
Hence, we introduce new local indices for the elements of $\mathcal{R}$. This is done by the following bijective maps:

$$\begin{aligned} idx_E : &\quad E_{\mathcal{R}} \quad \rightarrow \quad \{0, 1, ..., |E_{\mathcal{R}}| - 1\} \\ idx_T : &\quad T_{\mathcal{R}} \quad \rightarrow \quad \{0, 1, ..., |T_{\mathcal{R}}| - 1\} \\ idx_{HE} : &\quad HE_{\mathcal{R}} \quad \rightarrow \quad \{0, 1, ..., |HE_{\mathcal{R}}| - 1\} \end{aligned}$$

An index mapping for vertices is omitted because it will not be used in the following.

3.2 Recomputing a Local Cross Field

To locally update the existing cross field inside a region $\mathcal{R}$, we set up an equation system as in section 2.1 which solely contains elements of $\mathcal{R}$. To minimize the energy E_{smooth}, we again assemble the system matrix B and the constraint matrix C. This time, we use one real variable per face in $F_{\mathcal{R}}$ to determine cross field angles and one integer variable per edge in $E_{\mathcal{R}}$ for period jumps.



(a) (b) (c)

Figure 3.4: Local Cross Field Computation. (a) Existing directions of outer edges are fixed while some new directions have been added to the inner region. (b) The discrete Voronoi diagram of constrained faces is created. (c) The new cross field smoothly interpolates the fixed region boundary as well as the inner constraints.

Smooth Transition In contrast to before, we now have to establish a smooth transition from the new cross field to the old one outside the region. For this reason, we keep the outer ring of faces around the region fixed, i.e. we constrain all angles of faces $t_i \in T_{\mathcal{R}}^{\text{outer}}$ to their existing value θ_i. To do so, for each one, we add a row to C setting the $idx_T(t_i)$-th column to 1 and the last column to $-\theta_i$.

Since the energy inside the region is minimized, the interior crosses will smoothly approach this fixed outer ring. If in some places the region touches a mesh boundary, no outer faces exist and thus no additional constraints have to be fulfilled.

Period jumps are treated in a similar way. On outer edges, the period jump will always stay fixed since the angle of both incident faces is also fixed. Thus, we can constrain them to their current value as done for the outer faces. Boundary edges $e_{ij} \in E_{\mathcal{R}}^{\text{boundary}}$ however have to stay free because the angle of the incident inner face can change, which might have to be compensated by p_{ij}.

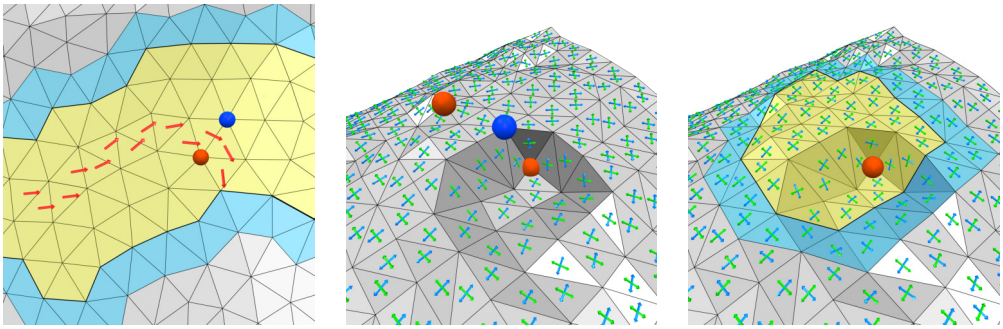
The reason why period jumps of outer edges are part of the system in the first place is that we want to allow the user to apply index constraints to vertices on the region boundary. In this case, the sum of all incident edges has to be constrained, which includes outer edges.

While the outer elements of the region are always fixed exactly as explained here, inner elements can be constrained arbitrarily for the purpose of the particular operation.

Ensuring One Constraint Per Mesh Component In case that the region consists of an entire mesh component, there are no outer faces to be constrained. If in addition no fixed angles in the interior are given, we have to add an arbitrary constraint to prevent infinitely many equivalent solutions.

Reducing the Number of Variables As in the original algorithm, we can get rid of unnecessary degrees of freedom by computing the discrete Voronoi cells of constrained faces and fix period jumps along the individual spanning trees to zero.

Now the local system will be given to the solver and the cross field inside the region is updated according to the new solution.



(a) **(b)** **(c)**

Figure 3.5: (a) Adding some constraints to the inner region creates a pair of singularities. (b), (c) Locally updating the cross field without changing any constraints leads to a different solution due to greedy rounding.

Observations Depending on the constraints that we add to the inner region, the amount and placement of singularities changes. As expected, singularities appear and vanish pairwise such that the sum of cross field indices within the region stays constant (figure 3.5a). Yet, this does not guarantee that the resulting singularity configuration allows for a valid parameterization. More details on this problem are given in section 4.2.

A rather surprising situation is shown in figure 3.5b and c. Here, the initial global solution provides us with two singularities of degree three (orange) and one of degree five (blue) within the visible image section. These are due to the local geometry since no constraints were involved. Now, we choose a small region and locally recompute the cross field without adding any constraints. The expected result is the exact same cross field as before. But instead

it does change and two singularities cancel out each other. However, this behavior is plausible when regarding the greedy nature of the mixed-integer solver (cf. section 2.4).

Period jumps are represented by integer variables which are successively rounded during the solving process. When we keep the existing solution outside the region fixed but set up a new relaxed problem for the inside, we might have created a situation that did not exist during the global solving process. Thus, rounding integers inside the region is now based other continuous solutions than before. Because of this, it is possible that a variable is rounded to a different integer. This way the new local cross field can change completely. In practice, this issue is observed quite frequently but did not turn out to be a big problem for interactive situations.

3.3 Updating the Local Cut Graph

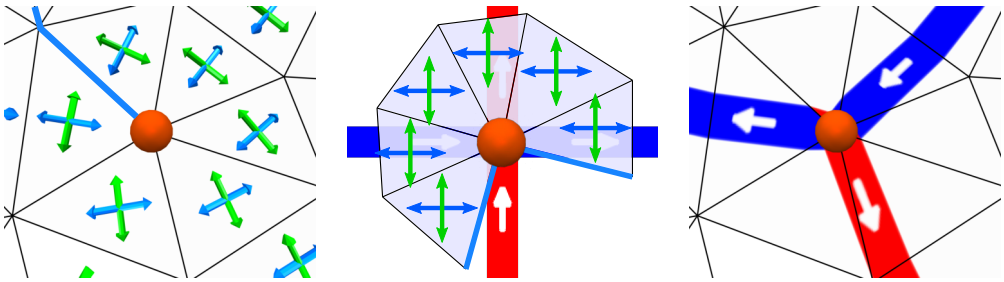
In this section, the existing cut graph is updated in order to allow a reparameterization of the region $\mathcal{R}$. Again, all of our actions are strictly local and only effect elements of $\mathcal{R}$.

Because initially the original parameterization pipeline was executed, we are already provided with a global cut graph. As stated in section 2.2 it fulfills two important requirements:

1. The mesh is cut open so that it is topologically equivalent to a disk.
2. All singularities lie on the cut graph.

While the point of property 1 is fairly intuitive, property 2 deserves some further explanation. By definition, the cross field around a singularity p cannot be aligned to a regular grid when mapped to the parameter domain. Hence, a low-distortion parameterization does not seem possible. Adding a cut to p solves this issue as the patch around p is torn open such that each cross can be aligned to the directions of the parameter grid. An example for a valence three singularity can be seen in fig 3.6. After adding a cut to each singularity, all of them lie on the boundary of the parameter domain. Because singularities will later be positioned at intersections of the grid, it is now possible that, mapped back onto the surface, other than four parameter lines meet at vertex p.

Before we start manipulating the cut graph, another important property has to be investigated. As explained in section 2.3, consistent orientation is established by setting period jumps to zero everywhere except on cut edges.



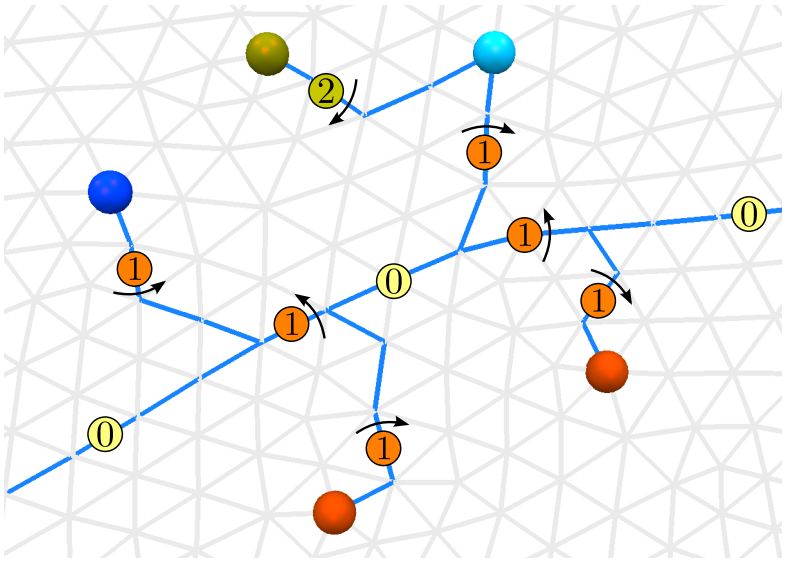
(a) **(b)** **(c)**

Figure 3.6: (a) A singularity of degree three is connected to the cut graph. (b) Opening the surrounding triangle patch by an angle of 90° inside the parameter domain compensates the valence defect of one. (c) Projected back to the surface, three parameter lines meet at the vertex.

After that, each path segment of the cut graph is associated with a distinct period jump which is equivalent to the rotational part of the transition function. This period jump does only change at singularities or branching points of the graph and is otherwise constant along a path. This can be expressed by the following formula which has to hold at every vertex v in order to allow a valid parameterization:

$$\underbrace{\left(\sum_{e_{ij} \in N(v)} p_{ij}\right)}_{\text{Sum of period jumps}} - \underbrace{4\, I(v)}_{\text{"Valence defect"}} + 4 \underbrace{I_0(v)}_{\text{Base index}} = 0 \qquad (3.1)$$

Note, that this is just a rearranged version of the cross field index definition (2.1). For singularities, $4I(v)$ equals the *valence defect* in the resulting quad mesh. Following [Bom12], the valence defect of a quad mesh vertex q is defined as the deviation from regular valence, i.e. $4 - val(q)$. $I_0(v)$ again represents the base index. Since period jumps are antisymmetric, it is important to consistently sum them up in clockwise direction. For an intuitive understanding, it is sufficient to only regard the sum of period jumps and the vertex index: At a regular vertex, $4I(v)$ is zero and the equation is fulfilled if the incoming and outgoing period jumps sum up to zero. If e.g. the vertex is a singularity of degree three, the sum of incoming has to exceed the outgoing period jumps by one.



(a)

Figure 3.7: Singularities of degree two (dark yellow), three (orange), five (dark blue), and seven (light blue) are connected by a cut graph. Equation 3.1 holds at every vertex. Black arrows show the orientation of period jumps, which are anti-symmetric and have to be summed up clockwise.

Problem Setting In the first step of our local recomputation the cross field has been changed. This means, that singularities have changed their location and degree, have been deleted or new ones have been created. Now, new singularities have to be connected to the cut graph, while paths to old singularities can be removed. All along, we have to make sure that our changed cut graph still satisfies all of the above requirements. In addition, another global aspect has to be taken care of. Original cut paths that intersect the region must not be disconnected since they could be necessary to supply other singularities outside the region with the correct period jump.

Influence on Parameterization The result of the final parameterization step is designed to be independent of the shape of the cut graph as long as the above properties are fulfilled[1]. However, a larger number of cuts introduces more integer variables to the system, which is why the size of the graph should be kept relatively small. Nevertheless, we can stick to greedy heuristics that find a valid but not inevitably minimal cut graph.

[1]In fact, due to the greedy strategy of the mixed-integer solver, a different number of variables can yield a slightly different solution as explained in the previous chapter.

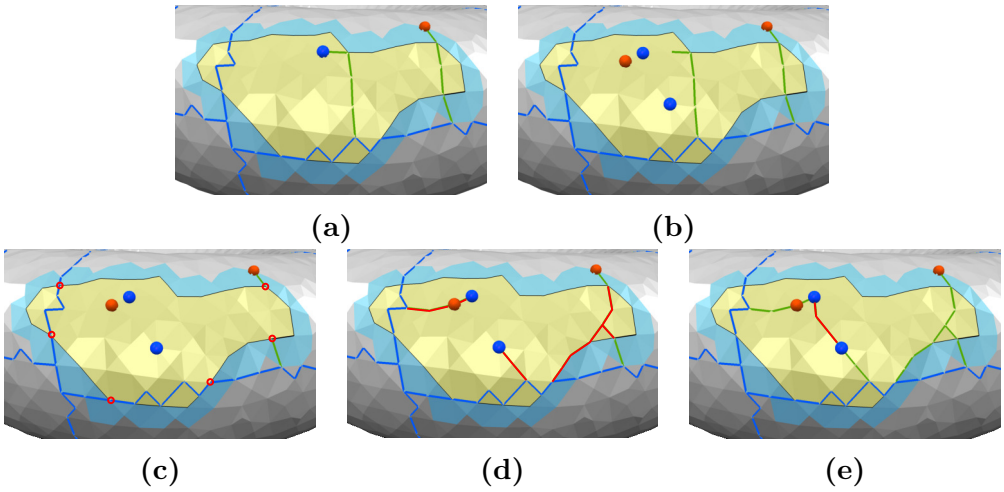


Figure 3.8: (a) The original setting before local remeshing. Disk cuts are painted in blue, singularity cuts in green. (b) Singularities have changed due to an updated cross field. (c) Inner singularity cuts have been cleared. Entry points are marked in red. (d) All entry points and singularities have been connected to the existing graph (red). (e) Both resulting graph components have been connected (red).

Different Types of Cuts For the existing graph within the region, $G_{\mathcal{R}} = (V_{\mathcal{R}}^{\text{cut}}, E_{\mathcal{R}}^{\text{cut}})$, we now distinguish between two kinds of cuts. We call those cuts that were initially created to achieve disk topology *disk cuts* and name the induced graph $G_{\mathcal{R}}^{\text{disk}}$. All remaining cuts in $G_{\mathcal{R}} \setminus G_{\mathcal{R}}^{\text{disk}}$ are called *singularity cuts*.

Entry Points Points at which a cut path enters or leaves a region are called *entry points*. More specific, an entry point is a vertex $v \in V_{\mathcal{R}}$ which is incident to an outer edge $e \in E_{\mathcal{R}}^{\text{outer}}$ that is part of the cut graph, i.e. $e \in E_{\mathcal{R}}^{\text{cut}}$.

It turns out that all requirements can be fulfilled using the following strategy:

1. Keep all disk cuts and delete singularity cuts in the inner region.
2. Connect singularities and entry points to existing cuts.
3. Connect all graph components inside the region.

1. Keep Disk Cuts As a first step, we clear all singularity cuts in the inner part of $\mathcal{R}$ but keep disk cuts. Thus, we initialize the new local cut

graph $G'_{\mathcal{R}}$ with $G^{\text{disk}}_{\mathcal{R}}$ plus all outer edges that were also part of the old cut graph.

2. Connect Singularities and Entry Points In case $G'_{\mathcal{R}}$ is still empty, i.e. the region does not contain any disk cuts, we create a root node, by adding an arbitrary singularity or entry point to $G'_{\mathcal{R}}$. If there is none, we can stop here and the local cut graph remains empty. Otherwise, we now successively connect all singularities and entry points to $G'_{\mathcal{R}}$ on a shortest path using a single-source multiple-target version of Dijkstra's algorithm.

3. Connect Graph Components Its is still possible, that the graph inside the region does not form a single connected component. This occurs if $G^{\text{disk}}_{\mathcal{R}}$ consisted of multiple components because none of the above shortest paths will connect them. In this case, an arbitrary graph component is marked as discovered and a multiple-source multiple-target Dijkstra search is used to find a shortest path to any undiscovered component. On success, the path is added to $G'_{\mathcal{R}}$ and both the path and the new component are marked as discovered.

This finally provides the new local cut graph $G'_{\mathcal{R}}$ which can be used in the following parameterization step. Apart from satisfying all properties listed above, some additional observations can be made.

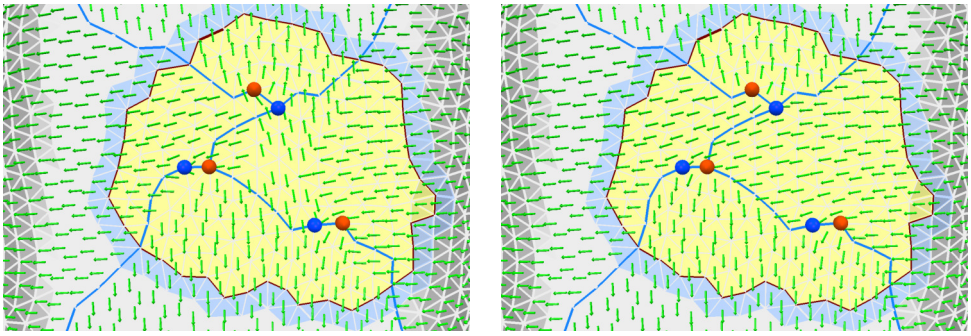
Global Cycles If $G^{\text{disk}}_{\mathcal{R}}$ did not contain any cycles, the resulting cut graph within the region is a tree. If there were cycles, it is guaranteed that no new ones are added within $\mathcal{R}$. Still, it is possible that the local changes close global cycles (figure 3.8e). These cycles are not justified by the genus of the object and thus will create isolated parts of the mesh. If we subsequently only perform a local parameterization of $\mathcal{R}$ as intended, this will not be a problem. If however we should decide to execute the global parameterization step instead, these isolated cut components can be arbitrarily translated within the parameter domain. To solve this issue, the translational part of the transition function at one cut vertex has to be constrained to the identity. Since rotations by 90° are still possible, care has to be taken to choose a singularity for this job if possible. Rotating the cut component around a non-integer position while keeping the rest of the parameterization fixed would result in a transition function which is no longer a grid-automorphism.

Still, by subsequently computing local cut graphs in different regions, it is possible to create an arbitrary topology of isolated components. In this case the transition functions between components are fixed along a spanning forest in which each cut component is represented by a node.

Since this heuristic connects all entry points with each other, successively choosing certain badly shaped regions can provoke a very large cut graph. This does not affect the quality of the resulting parameterization but has an influence on performance due to an increased number of integer variables.

3.4 Recomputing a Local Parameterization

In the last step, the parameterization function f has to be updated locally with respect to the new cross field and cut graph. As for the local cross field computation, we keep the current parameterization fixed in an outer ring around the region and set up an equation system providing us with a new solution for the inside.



(a) **(b)**

Figure 3.9: The main directions of the cross field are shown as green arrows. (a) Some directions in the region are not oriented consistently. (b) This has been corrected by concentrating all period jump on cut edges (blue).

Consistent Orientation As a pre-processing step we have to establish the property of consistent orientation (cf. section 2.3), i.e. period jumps turn zero everywhere but on cut edges.

In the global algorithm, the orientation of an arbitrary cross has been taken as reference and was propagated over the entire mesh. Now, we are already provided with orientations in all outer triangles which we must not change in order to preserve global consistency. Thus, we start propagating these from the outer ring into the inner region and stop at cut edges. Since the cut graph cannot isolate any faces within $\mathcal{R}$, all inner faces are affected.

This task can be performed by the below breadth-first search algorithm. We use a half-edge based search because each half-edge uniquely defines an edge (whose period jump will be set to zero) and an incident face (whose

cross will be rotated). The breadth-first approach guarantees that if the period jump of an edge has been set to zero, it will not be changed again.

foreach *edge* $e_{ij} \in E_{\mathcal{R}}^{\text{boundary}}$ **do**
  Push incident inner half-edge $he \in HE_{\mathcal{R}}^{\text{inner}}$ to queue Q.
  Flag incident outer face as visited.
end
while Q *not empty* **do**
  Pop he from Q.
  Let e_{ij} be the corresponding half-edge and t_i the incident triangle.
  if e_{ij} *is no cut edge and* t_i *unvisited* **then**
    Flag t_i as visited.
    $\theta_i \leftarrow \theta_i + \frac{\pi}{2} p_{ij}$
    $p_{ij} \leftarrow 0$
    foreach *half-edge of* t_i **do**
      Push opposite half-edge to Q.
    end
  end
end

Setting Up the System As a next step, we can set up the local problem matrices B and C to find a solution minimizing the energy E_{orient}. This time, only the u and v coordinates of vertices in $V_{\mathcal{R}}$ are variables of the system. Again, vertices that are part of the cut graph have multiple images under f. The matrix B is filled exactly as in the global version, containing two columns for each vertex instance, one column for each integer variable used to constrain the transition function and one column for the right-hand side of the system. Each vertex instance produces four rows due to the partial derivatives of the energy function. C still contains four rows per cut edge, to preserve cross-cut compatibility. This time, we only need to add these constraints for edges in $E_{\mathcal{R}}^{\text{inner}}$ and $E_{\mathcal{R}}^{\text{boundary}}$. Outer edges do not have to be constrained, because the parameterization on both sides will be fixed anyway. As before, to imply a quad-only mesh, it is important to make sure that singularities are mapped to integer positions.

Boundary Compatibility In order to retain compatibility to the parameterization outside the region, we fix all (u, v) values of vertices on the boundary of $\mathcal{R}$ to their current value. If such a boundary vertex has multiple instances, we fix all instances belonging to the outer ring of the region while those of the inner region remain free. If the region covers an entire mesh component such that there are no outer faces, just as in the global algorithm

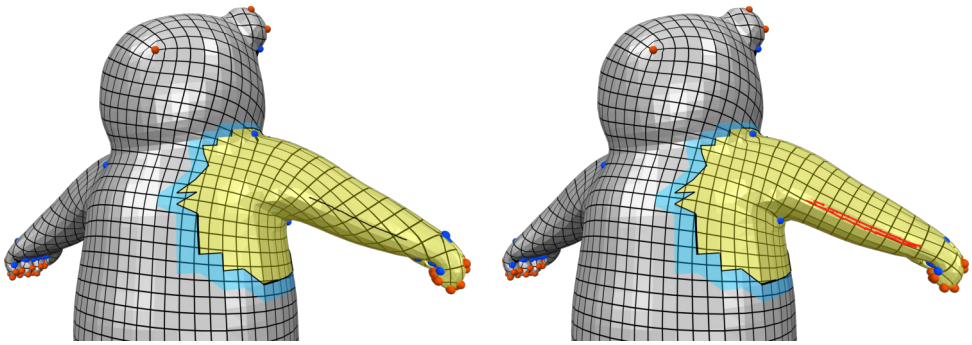
a single vertex of the region will be fixed to the origin of the parameter domain.

Solving the resulting system finally provides us with an updated parameterization for our region and the local remeshing operation is completed.

3.5 An Interactive Comb Tool

The local parameterization approach presented so far paves the way for an entire new set of interactive quad meshing tools. Designing such a tool involves three major degrees of freedom: 1) Defining a user interaction, 2) determining a set of constraints, and 3) finding an appropriate way to choose a region. In this section, an exemplary tool is created to demonstrate one of many possible combinations.

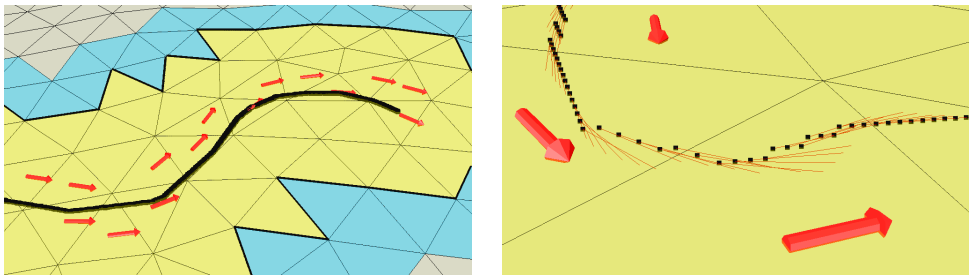
The general idea of this tool is to let the user manually brush the alignment of an existing parameterization into a direction of his choice.



(a) **(b)**

Figure 3.10: A stroke of the comb tool is used to correct the edge flow around the arm of the model. (a) Displays the original situation and an input line drawn by the user. (b) Shows the extracted directional constraints as well the updated parameterization. The highlighted region contains all triangles within a user-specified distance to the new constraints.

User Interaction Using the mouse, brush strokes can be performed on the surface of an object. After each stroke, the existing parameterization is updated locally to match the direction of the stroke as well as possible. A single interaction consists of pressing a mouse button, drawing a path on the mesh surface and finally releasing the button.



(a) (b)

Figure 3.11: (a) Shows the end of a curved input line and the resulting constraints. In a closeup (b) the projected samples p_i as well as their smoothed direction vectors d_i are visible. Some path segments have been rasterized by Bresenham's algorithm due to slow input processing.

Obtaining Directional Constraints The task of turning the user's mouse movement into a set of directional constraints is split into multiple steps.

First, the different mouse positions between pressing and releasing the button are captured over time and saved as a list of screen-space samples $s_i \in \mathbb{Z}^2$. Care is taken that all samples are stored in their order of appearance and all pairs of consecutive samples s_i, s_{i+1} are distinct. If due to the sampling rate of the particular UI environment samples s_i and s_{i+1} are no direct neighbors (i.e. the x or y coordinate differs by more than one), a straight line is rasterized in between using Bresenham's line algorithm [Bre65].

After that, all samples are projected to the mesh. The projected points, each located within a triangle of the mesh, are called $p_i \in \mathbb{R}^3$. Now, for each p_i a direction vector $d_i \in \mathbb{R}^2$ is computed within the local coordinate system of the respective triangle. To do so, we take the k preceding and k following samples into account and compute a regression line minimizing the squared distances to these points $p_{i-k}, \dots p_{i+k}$ similarly to [Hop+92]. Since this is done in 2D and not all affected samples lie within the same triangle, they are first projected onto the tangent plane of the triangle p_i lives in. At the beginning and the end of the path not all $2k$ neighboring samples are present. So only those that are available are taken into account. Obviously, k works as a smoothing parameter here which is desired in order to avoid jagged directions. In most examples $k = 5$ has proven to be a reasonable choice.

Now, for all samples that lie in the same triangle, the corresponding directional vectors are summed up and converted into an angle which is directly used as a constraint for the cross field computation.

Choosing a Region The affected region is simply chosen based on the geodesic distance to the drawn comb path. The user can control the size of the region by specifying a radius r.

We call the set of triangles that contain at least one sample p_i in their interior T_{path}. Now, starting with all faces in T_{path} as root, we grow a dual spanning tree consisting solely of shortest paths by using Dijkstra's Algorithm. To roughly approximate the geodesic distance we use the euclidean distance between the midpoints of two adjacent triangles as edge costs. We stop extending a dual path if its length reached the given radius r. At last, the region $\mathcal{R}$ is initialized using all triangles covered by the spanning tree as the set of inner faces.

Before applying the directional constraints from above, we have to choose between two options. In the first one, all existing constraints from previous operations are removed from the interior of $\mathcal{R}$. This is useful if the affected area was already constrained quite densely or the new and old directions disagree too much. The second option involves keeping the original constraints in addition to the new ones and only replacing them if both affect the same triangle. This comes in handy when few initial constraints are given and the mesh is brushed iteratively, since this way two subsequent strokes with overlapping regions cannot sweep off each others constraints.

Finally, the local remeshing operation can be started to update the parameterization within the region. Results are analyzed in the next chapter.

Chapter 4

Conclusion

This chapter evaluates how local remeshing, as described so far, behaves in a set of test cases. It shows that the new technique is indeed a big step towards interactive quad meshing but there is still some challenging work to be done until it reaches its full potential in practice.

For this purpose section 4.1 analyzes the achievements of this work with respect to performance, usability and resulting quality. In contrast to that, section 4.2 illuminates shortcomings of the presented approach and strives to formulate emerging research questions, solutions to which promise an even better outcome.

4.1 Results

While performance and usability fully meet our expectations, the achieved quality jumps between surprisingly good results in some cases but also completely degenerated ones in other cases.

Performance Table 4.1 and 4.2 each show time measurements of local remeshing operations using varying region sizes on two models. In both cases the inner region does not contain any constraints.

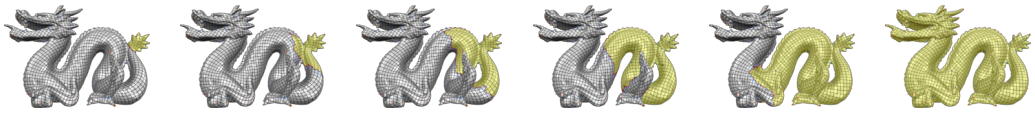
Although we regard the number of triangles in the region as input size, the actual performance does also depend on other properties like the number of singularities and cut edges. Still, the following measurements give an indication for the order of magnitude we are dealing with.

In test sessions, typical region sizes varied between (a) and (c) for both examples. Most operations were completed in less than one second. Even with a maximum response time of 2.74 seconds for very large regions, this seems to be sufficient to establish an interactive workflow. However, smooth

Model: Stanford Dragon

	Local					Global
Figure	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**	**(f)**
Triangles	**2786**	**5530**	**11053**	**22086**	**44178**	**88178**
Cross Field	0.15s	0.17s	0.31s	0.65s	1.49s	3.90s
Cut Graph	0.003s	0.005s	0.01s	0.04s	0.06s	0.14s
Parameterization	0.19s	0.19s	0.40s	1.48s	10.73s	40.11s
Total	**0.34s**	**0.36s**	**0.72s**	**2.18s**	**12.28s**	**44.15s**

Table 4.1: Runtime measurements of local remeshing operations on the Stanford Dragon using different region sizes are compared to the global algorithm.



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 4.1: Regions used in table 4.1.

Model: Armadillo

	Local					Global
Figure	**(a)**	**(b)**	**(c)**	**(d)**	**(e)**	**(f)**
Triangles	**10798**	**21525**	**43260**	**86407**	**172567**	**345944**
Cross Field	0.38s	0.68s	1.42s	3.12s	7.80s	21.85s
Cut Graph	0.01s	0.02s	0.04s	0.10s	0.22s	0.78s
Parameterization	0.27s	0.50s	1.28s	2.98s	11.85s	56.25s
Total	**0.66s**	**1.20s**	**2.74s**	**6.20s**	**19.87s**	**78.88s**

Table 4.2: Runtime measurements of local remeshing operations on the Armadillo model using different region sizes are compared to the global algorithm.

(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 4.2: Regions used in table 4.2.

realtime updates (e.g. while performing a stroke with the comb tool) do not seem to be possible yet.

All performance tests have been executed on the same workstation with an intel i7-2600 3.4 GHz processor and 16 GB main memory. Due to the sequential nature of the algorithm, CPU usage was at 100% on a single core with a maximum memory consumption of 2 GB.

Usability Rating the user experience of new tools can be very subjective. For this reason, a user study is suggested as future work. For now, some simple test sessions will suffice.

When experimenting with the comb tool, the following strategy proved to be most effective: The initial global solution is computed with a very sparse set of automatically created constraints to achieve a rough alignment to the global structure of the object. After that, multiple comb strokes are used to achieve a more desirable alignment to local features. Starting at one "end" of the object (e.g. an extremity) and iteratively working ones way towards another end turned out to be more effective than randomly applying operations to different parts of the object. This way, even undesirable quad structures covering a much larger part of the object than the individual region sizes can be corrected. When the desired alignment contradicts existing constraints, it is advised to first clear a larger region from constraints before using the comb tool. If an operation does not yield the expected effect, increasing the region size often improves the result. Depending on the resolution of the input mesh, the trade-off between region size and computation time has to be handled with care. Choosing the region too small will give bad results as due to limited degrees of freedom, while choosing it too large will lead to bad performance. Experience showed that using a geodesic radius of 10 to 15 times the target edge length is a good starting point.

All in all the comb tool turns out to be very intuitive to use. Since the region radius is the only additional setting that has to be controlled by the user, no complicated parameter tweaking is involved. Even knowledge about topological issues like singularity placement is not required.

Unfortunately, some local operations fail on a regular basis, as explained in the next chapter. This heavily effects the workflow in its current state. Implementing undo/redo functionality eases the problem until an actual solution to this issue is available.

Quality In many cases the visual quality of an initial automatic solution can be improved significantly. Figure 4.3 shows an example where the forearm of the armadillo model has been reparameterized using the comb tool.

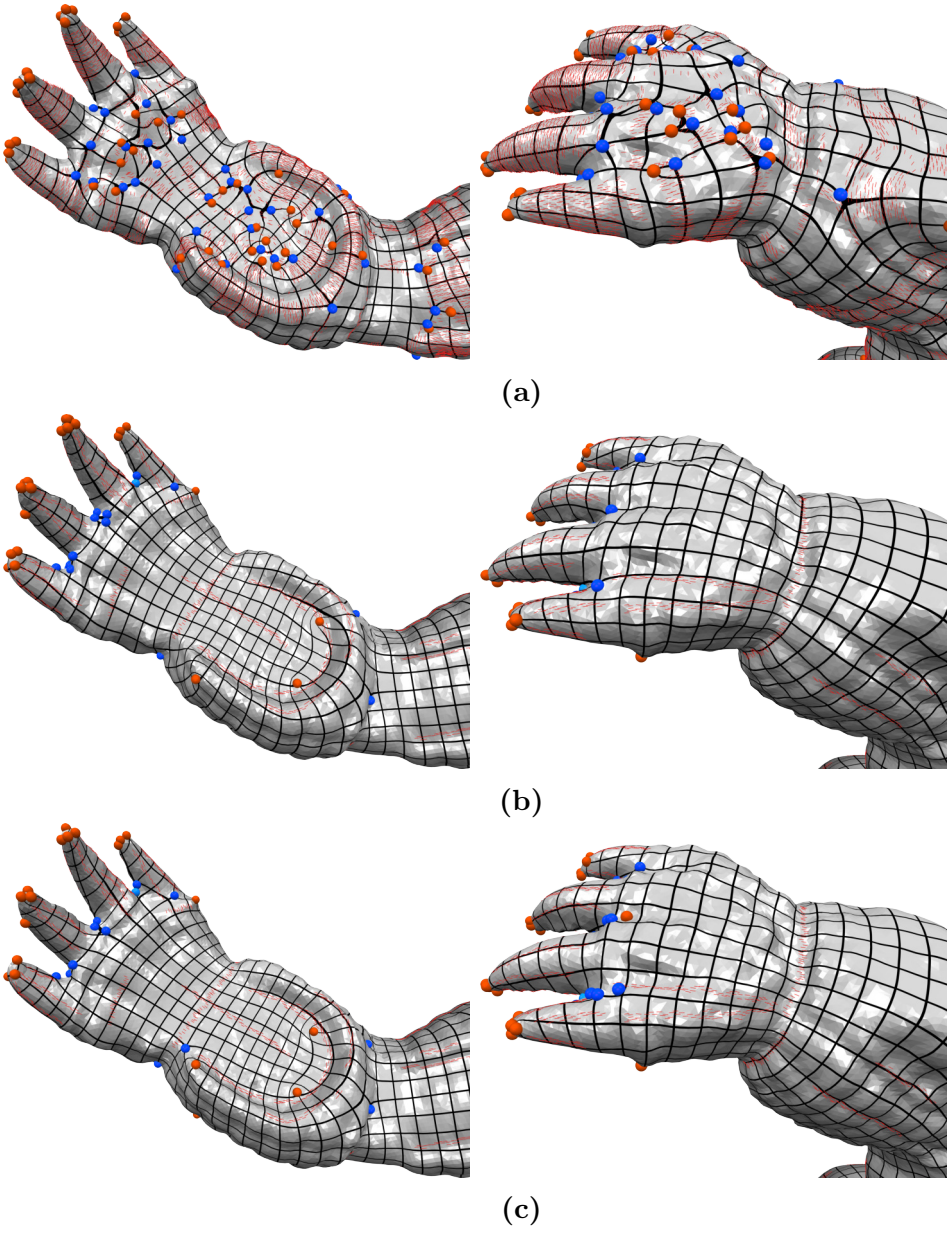
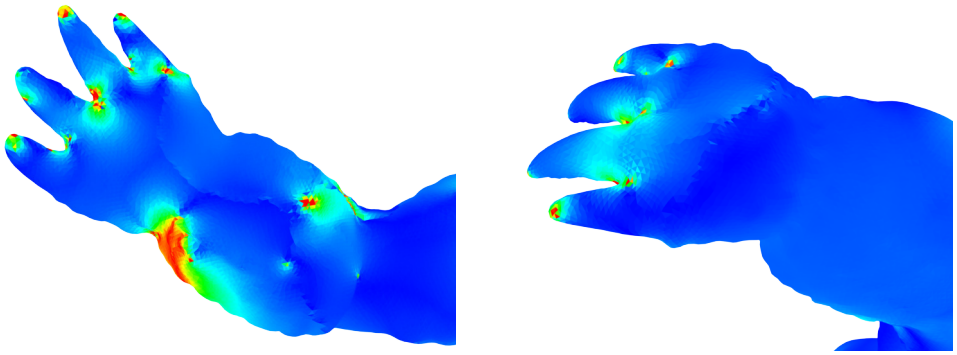


Figure 4.3: (a) Hand and forearm of the armadillo model have been parameterized with noisy constraints. (b) After removing these, the parameterization has been brushed using the comb tool. (c) Finally running the global algorithm again yields slight improvements.



(a) **(b)**

Figure 4.4: Shows the angular deviation between the parameterizations in fig. 4.3 (b) and (c).

Initially it contained an undesirable amount of singularities due to the high frequency geometry. After a sequence of comb strokes, all singularities are located at geometrically meaningful locations although the user did not place any of them explicitly. The new edge flow is more regular but still aligned to possible bending directions in animation.

The energies E_{smooth} and E_{orient} on the manipulated part of the model decreased from 355.42 and 78.28 to 133.66 and 66.48 respectively. This is obviously due to the more moderate set of directional constraints.

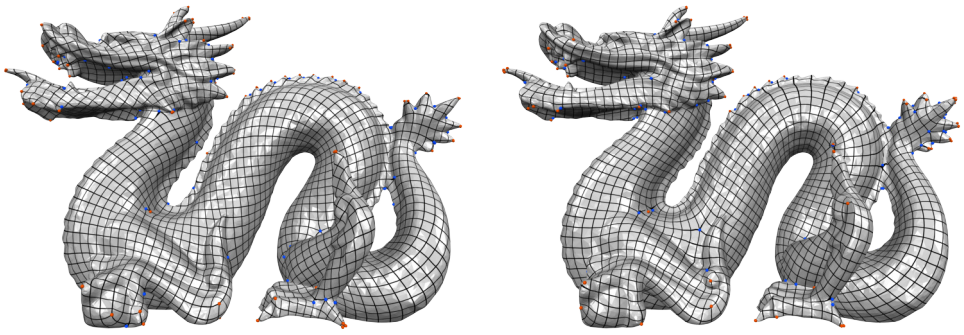
Still, the solution might not be optimal because it has been iteratively computed within multiple closed regions. By running the original MIQ algorithm again as a final step, the solution approximates a global minimum based on the new set of constraints. Applied to the armadillo example, the cross field energy improves only slightly to 133.35 while the alignment energy drops by almost 30% to 46.72.

Anyhow, this post-processing step is only reasonable if the final edge flow does not diverge from the interactive result too much, i.e. it still represents the user's prospect. Fortunately, in most examples this seems to be the case. Figure 4.4 shows the results of both steps as well as a metric describing the angular deviation between the directions of the parameter lines[1].

[1]Expressed per triangle as $\min_{i\in\{0..3\}} \left[\cos^{-1}\left(\frac{\nabla u'^T \,\text{ref}_u^i}{||\nabla u'||||\,\text{ref}_u^i\,||}\right) + \cos^{-1}\left(\frac{\nabla v'^T \,\text{ref}_v^i}{||\nabla v'||||\,\text{ref}_v^i\,||}\right)\right]$ where the gradients $\nabla u'$, $\nabla v'$ after the global optimization are compared to the gradients ∇u, ∇v before. To be invariant to rotations of 90° in the parameter domain, the minimum of all four possible rotations is taken:

$$\begin{array}{llll} \text{ref}_u^0 = \nabla u, & \text{ref}_u^1 = \nabla v, & \text{ref}_u^2 = -\nabla u, & \text{ref}_u^3 = -\nabla v, \\ \text{ref}_v^0 = \nabla v, & \text{ref}_v^1 = -\nabla u, & \text{ref}_v^2 = -\nabla v, & \text{ref}_v^3 = \nabla u. \end{array}$$

To further test the effectiveness of the comb tool, a completely unconstrained parameterization of the Stanford dragon has been improved in an interactive session (figure 4.5). Care was taken to align the edge flow to possible bending directions.

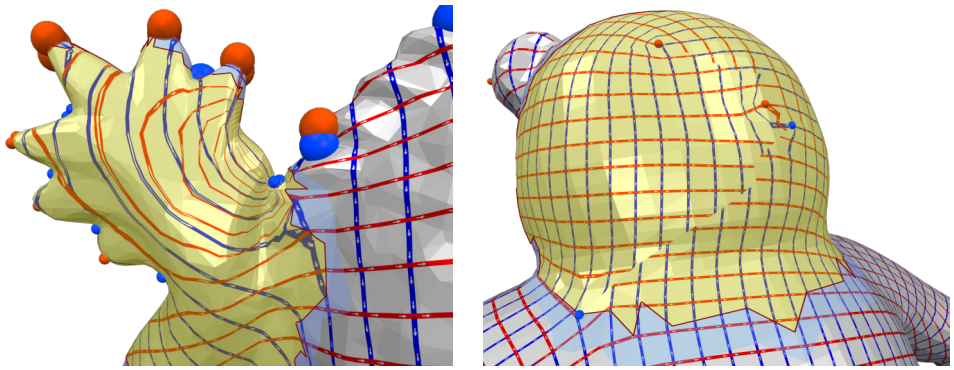


(a) **(b)**

Figure 4.5: An initial randomly oriented parameterization (a) of the Stanford dragon has been edited using the interactive comb tool (b).

4.2 Limitations and Future Work

In this last section, problems in the current state of the project are addressed. The most unpleasant issue right now is that a substantial amount of remeshing operations fails because these lead to topologically impossible situations.



(a) (b)

Figure 4.6: Local operations leading to (a) non-quad configurations and extreme distortion or (b) non-quad configurations and a broken transition function.

Topological Problems In most practical situations it is possible to find a local cross field, satisfying all constraints. In contrast to this, the resulting singularity configuration does often not allow a valid quad mesh that still matches the region boundary. Consequently, the parameterization step fails. In our implementation this leads to any combination of non-quad configurations, extreme distortion (figure 4.6a) and violated constraints such as illegal transition functions (figure 4.6b).

Known Restrictions In [Pen+11], Peng et al. investigate which topological modifications of given quad meshes are feasible within a local region and which are not. They focus on changing singularity configurations while keeping the boundary of an otherwise regular region fixed. For certain possible cases, they provide operations working explicitly on the existing quad topology. In addition, they proof that some particular other modifications are never possible. Their results can be directly applied to our method in order to show that some changes within a region do not allow a valid (or high-quality) parameterization. Two examples are presented in the following:

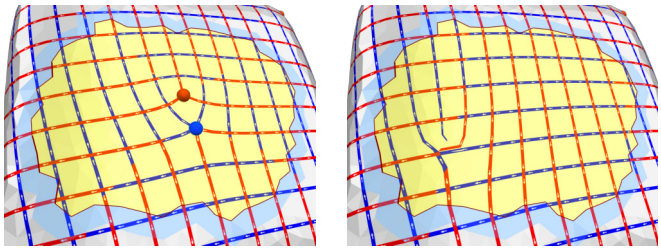


Figure 4.8: In a region with only two singularities (here of degree three and five) it is not possible to cancel these. Our implementation yields a parameterization with a non-quad configuration.

> "A region containing only one irregular vertex cannot be edited." [Pen+11]

In figure 4.7, we choose a region around a valence three singularity and add constraints that force it to move. Since there is only one valid quad mesh topology for this region, the only way to change the singularity position is to alter the geometry of the implied quad mesh. Consequently the parameterization is stretched extremely.

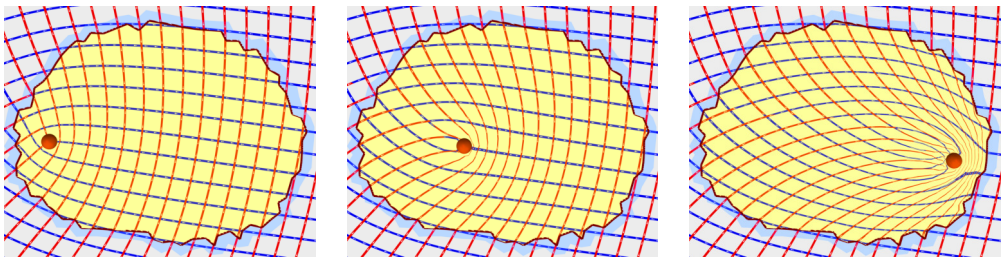


Figure 4.7: The quad topology inside a region with exactly one singularity is fixed. Moving it introduces high distortion to the parameterization.

> "[In] a region containing two irregular vertices [...these] cannot be canceled." [Pen+11]

Figure 4.8 shows how the current implementation reacts if we force the singularities to cancel out each other.

Detecting Impossible Operations Despite these results, there still is no rule describing which topological changes of a quad mesh region are possible in general. If such a rule is found in future, impossible operations can be

detected and aborted before executing the local remeshing algorithm. Since this feature is essential to make our method applicable in practice, it forms the most important research question for future work.

Finding a Feasible Region Based on such a rule, further improvements are conceivable. It could be investigated if a region can be extended in such a way that a previously infeasible operation becomes possible.

Optimizing the Region Shape Apart from the above considerations, regions are currently chosen in a very naive fashion. Simply using a geodesic region around user constraints does not exploit any geometric properties of the input surface. Since the effect of changing constraints might not be distributed uniformly, regions could be optimized to only cover areas of high influence. For example the effect of an operation may decrease when passing already constrained features. Thus the region could be reduced there and extended in another direction. Therefore, finding regions that compensate user operations more efficiently constitutes another research question.

Controlling the Amount of Singularities In some cases, a set of successive strokes with the comb tool does not yield the expected alignment. Often the necessary degrees of freedom can be achieved by inserting additional singularities. Since manual placement of singularities can be a very difficult task, it is desirable to introduce a parameter controlling the trade-off between element distortion and the number of singularities as in [MZ13]. It will be worthwhile to investigate if their technique can be integrated into our interactive approach.

Improvements to the Comb Tool The comfortable use of the comb tool can be further improved by enabling more precise operations. E.g. snapping manually drawn brush strokes to geometric features is very likely to improve the achieved accuracy. Using the geometric snakes algorithm [LL02] can be a way to achieve this property.

Additional Tools In various test sessions a demand for additional tools, complementing the comb, has been identified. The most prominent are: 1) An eraser brush which clears constraints in a certain radius around the mouse cursor, and 2) a reparameterize brush that can be used to quickly select a region which will be updated without changing any constraints. This way, a suboptimal edge flow resulting from multiple operations with small region sizes can be improved. Both concepts proved useful in some manual tests.

4.3 Summary

In this thesis we gave an introduction into the field of quad meshing by explaining possible applications in animation and physical simulation as well as listing common quality criteria. The need for new interactive tools was motivated by troublesome manual creation on the one hand and automatic methods lacking fast user response on the other hand.

Next, the existing mixed-integer quadrangulation pipeline was described. This includes generating a smooth cross field, finding a valid cut graph and computing a global parameterization. Using this knowledge, we went through all the adjustments that have been made to the individual steps in order to enable local remeshing. Equipped with the ability to update an existing parameterization in closed regions, we described what it takes to build interactive tools on that basis. As a first example we developed a comb tool and analyzed its results in terms of performance, usability and resulting mesh quality.

Although impressing results could be achieved in some cases, others failed completely. Based on existing research we confirmed that certain situations occurring during local operations are indeed impossible to solve. As a consequence, we formulated multiple research questions. These include: 1) Finding a formal description of which operations are feasible or infeasible in general. 2) Providing ways to extend regions such that all necessary degrees of freedom are obtained. 3) Choosing regions to precisely cover areas which are highly influenced by a particular operation.

Finally it can be said that local remeshing does have the potential of becoming a part of practical workflows but a number of related problems has yet to be solved.

Acknowledgments

I would like to thank Prof. Dr. Leif Kobbelt and Prof. Dr. David Bommes for making this thesis possible.

Special thanks go to my supervisor Hans-Christian Ebke for the idea of this work, his extensive help and the opportunity of taking part in his research. I also wish to thank Marcel Campen for his patient advice.

Finally, I thank Prof. Dr. David Bommes for providing the MIQ pipeline, Hans-Christian Ebke and Moritz Ibing for reviewing and numerous people in the student lab for their help with technical problems.

Bibliography

[Bom12] David Bommes. "Quadrilateral Surface Mesh Generation for Animation and Simulation". Dissertation. RWTH Aachen, 2012. URL: http://darwin.bth.rwth-aachen.de/opus3/volltexte/2013/4373/.

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. "Mixed-integer Quadrangulation". In: *ACM SIGGRAPH 2009 Papers*. SIGGRAPH '09. New Orleans, Louisiana: ACM, 2009, 77:1–77:10. ISBN: 978-1-60558-726-4. DOI: 10.1145/1576246.1531383. URL: http://doi.acm.org/10.1145/1576246.1531383.

[BZK12] David Bommes, Henrik Zimmer, and Leif Kobbelt. "Practical Mixed-integer Optimization for Geometry Processing". In: *Proceedings of the 7th International Conference on Curves and Surfaces*. Avignon, France: Springer-Verlag, 2012, pp. 193–206. ISBN: 978-3-642-27412-1. DOI: 10.1007/978-3-642-27413-8_12. URL: http://dx.doi.org/10.1007/978-3-642-27413-8_12.

[Bom+13] David Bommes et al. "Integer-grid Maps for Reliable Quad Meshing". In: *ACM Trans. Graph.* 32.4 (July 2013), 98:1–98:12. ISSN: 0730-0301. DOI: 10.1145/2461912.2462014. URL: http://doi.acm.org/10.1145/2461912.2462014.

[Bom+12] David Bommes et al. "State of the Art in Quad Meshing". In: *Eurographics STARS*. 2012.

[Bot+] Mario Botsch et al. "OpenMesh: A Generic and Efficient Polygon Mesh Data Structure". In: OpenSG Symposium 2002.

[Bre65] J. E. Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems Journal* 4.1 (1965), pp. 25–30. ISSN: 0018-8670. DOI: 10.1147/sj.41.0025.

[CBK12] Marcel Campen, David Bommes, and Leif Kobbelt. "Dual Loops Meshing: Quality Quad Layouts on Manifolds". In: *ACM Trans. Graph.* 31.4 (July 2012), 110:1–110:11. ISSN: 0730-0301. DOI: 10.1145/2185520.2185606. URL: http://doi.acm.org/10.1145/2185520.2185606.

[CC78] E. Catmull and J. Clark. "Recursively generated b-spline surfaces on arbitrary topological meshes". In: *Computer-Aided Design* 10.6 (1978), pp. 350 –355. ISSN: 0010-4485. DOI: http://dx.doi.org/10.1016/0010-4485(78)90110-0. URL: http://www.sciencedirect.com/science/article/pii/0010448578901100.

[Com10] Computer Graphics Group at RWTH Aachen University. *CoMISo Constrained Mixed-Integer Solver*. 2010. URL: http://www.graphics.rwth-aachen.de/comiso (visited on 08/06/2014).

[Don+06] Shen Dong et al. "Spectral Surface Quadrangulation". In: *ACM SIGGRAPH 2006 Papers*. SIGGRAPH '06. Boston, Massachusetts: ACM, 2006, pp. 1057–1066. ISBN: 1-59593-364-6. DOI: 10.1145/1179352.1141993. URL: http://doi.acm.org/10.1145/1179352.1141993.

[Ebk+13] Hans-Christian Ebke et al. "QEx: Robust Quad Mesh Extraction". In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 168:1–168:10. ISSN: 0730-0301. DOI: 10.1145/2508363.2508372. URL: http://doi.acm.org/10.1145/2508363.2508372.

[Flo95] C.A. Floudas. "Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications". In: Topics in Chemical Engineering. Oxford University Press, USA, 1995, p. 112. ISBN: 9780195100563. URL: http://books.google.de/books?id=GJgiUZCMCSEC.

[Hop+92] Hugues Hoppe et al. "Surface Reconstruction from Unorganized Points". In: *SIGGRAPH Comput. Graph.* 26.2 (July 1992), pp. 71–78. ISSN: 0097-8930. DOI: 10.1145/142920.134011. URL: http://doi.acm.org/10.1145/142920.134011.

[Hua+08] Jin Huang et al. "Spectral quadrangulation with orientation and alignment control." In: *ACM Trans. Graph.* 27.5 (2008), p. 147. URL: http://dblp.uni-trier.de/db/journals/tog/tog27.html#HuangZMLKB08.

[Ibi14] Moritz Ibing. "Soft Feature Detection For Quad Meshing". Bachelor's Thesis. RWTH Aachen, 2014.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. *QuadCover - Surface Parameterization using Branched Coverings.* 2007.

[Ket99] Lutz Kettner. "Using generic programming for designing a data structure for polyhedral surfaces". In: *Computational Geometry* 13.1 (1999), pp. 65 –90. ISSN: 0925-7721. DOI: http://dx.doi.org/10.1016/S0925-7721(99)00007-3. URL: http://www.sciencedirect.com/science/article/pii/S0925772199000073.

[Kru+99] S.O. Krumke et al. "Flow Improvement and Network Flows with Fixed Costs". In: *Operations Research Proceedings 1998*. Ed. by Peter Kall and Hans-Jakob Luethi. Vol. 1998. Operations Research Proceedings 1998. Springer Berlin Heidelberg, 1999, pp. 158–167. ISBN: 978-3-540-65381-3. DOI: 10.1007/978-3-642-58409-1_15. URL: http://dx.doi.org/10.1007/978-3-642-58409-1_15.

[LKH08] Yu-Kun Lai, Leif Kobbelt, and Shi-Min Hu. "An Incremental Approach to Feature Aligned Quad Dominant Remeshing". In: *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling*. SPM '08. Stony Brook, New York: ACM, 2008, pp. 137–145. ISBN: 978-1-60558-106-4. DOI: 10.1145/1364901.1364921. URL: http://doi.acm.org/10.1145/1364901.1364921.

[LL02] Yunjin Lee and Seungyong Lee. "Geometric snakes for triangular meshes". In: *Computer Graphics Forum* 21.3 (2002), pp. 229–238.

[Li+06] Wan-Chiu Li et al. "Representing Higher-Order Singularities in Vector Fields on Piecewise Linear Surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 1315–1322. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.173. URL: http://dx.doi.org/10.1109/TVCG.2006.173.

[MK04] M. Marinov and L. Kobbelt. "Direct anisotropic quad-dominant remeshing". In: *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*. 2004, pp. 207–216. DOI: 10.1109/PCCGA.2004.1348351.

[MZ13] Ashish Myles and Denis Zorin. "Controlled-distortion Constrained Global Parametrization". In: *ACM Trans. Graph.* 32.4 (July 2013), 105:1–105:14. ISSN: 0730-0301. DOI: 10.1145/2461912.2461970. URL: http://doi.acm.org/10.1145/2461912.2461970.

[Pen+11] Chi-Han Peng et al. “Connectivity Editing for Quadrilateral Meshes”. In: *Proceedings of the 2011 SIGGRAPH Asia Conference*. SA ’11. Hong Kong, China: ACM, 2011, 141:1–141:12. ISBN: 978-1-4503-0807-6. DOI: 10.1145/2024156.2024175. URL: http://doi.acm.org/10.1145/2024156.2024175.

[Ray+08] Nicolas Ray et al. “N-symmetry Direction Field Design”. In: *ACM Trans. Graph.* 27.2 (May 2008), 10:1–10:13. ISSN: 0730-0301. DOI: 10.1145/1356682.1356683. URL: http://doi.acm.org/10.1145/1356682.1356683.

[Ray+06] Nicolas Ray et al. “Periodic Global Parameterization”. In: *ACM Trans. Graph.* 25.4 (Oct. 2006), pp. 1460–1485. ISSN: 0730-0301. DOI: 10.1145/1183287.1183297. URL: http://doi.acm.org/10.1145/1183287.1183297.

[Rem12] J. and Seny Remacle J.-F. and Lambrechts. “Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm”. In: *International Journal for Numerical Methods in Engineering* 89.9 (2012), pp. 1102–1119. ISSN: 1097-0207. DOI: 10.1002/nme.3279. URL: http://dx.doi.org/10.1002/nme.3279.

[TPSH14] Kenshi Takayama, Daniele Panozzo, and Olga Sorkine-Hornung. “Pattern-Based Quadrangulation for N-Sided Patches”. In: *Computer Graphics Forum (proceedings of EUROGRAPHICS Symposium on Geometry Processing)* 33.5 (2014), pp. 177–184.

[Tak+13] Kenshi Takayama et al. “Sketch-based Generation and Editing of Quad Meshes”. In: *ACM Trans. Graph.* 32.4 (July 2013), 97:1–97:8. ISSN: 0730-0301. DOI: 10.1145/2461912.2461955. URL: http://doi.acm.org/10.1145/2461912.2461955.

[Tie+12] Julien Tierny et al. “Interactive Quadrangulation with Reeb Atlases and Connectivity Textures”. In: *Visualization and Computer Graphics, IEEE Transactions on* 18.10 (2012), pp. 1650–1663. ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.270.

[Ton+06] Y. Tong et al. “Designing Quadrangulations with Discrete Harmonic Forms”. In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. SGP ’06. Cagliari, Sardinia, Italy: Eurographics Association, 2006, pp. 201–210. ISBN: 3-905673-36-3. URL: http://dl.acm.org/citation.cfm?id=1281957.1281983.

[Zha+10] Muyang Zhang et al. “A Wave-based Anisotropic Quadrangulation Method”. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH ’10. Los Angeles, California: ACM, 2010, 118:1–118:8. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778855. URL: http://doi.acm.org/10.1145/1833349.1778855.