Quadrilateral Surface Mesh Generation for Animation and Simulation

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von Diplom-Informatiker
David Bommes
aus Korschenbroich, Deutschland

Berichter: Prof. Dr. Leif Kobbelt
Prof. Dr. Pierre Alliez

Tag der mündlichen Prüfung: 11.10.2012

Acknowledgments

There are numerous people and institutions that deserve deep gratitude for supporting my work during the last years.

First of all, I would like to thank my advisor Leif Kobbelt, not only for his excellent scientific education and support, but also for letting me grow in a fruitful environment of motivated colleagues and friends.

Additionally for being my co-examiner I want to thank Pierre Alliez in particular for always willing to share his enormous expertise during interesting and insightful discussions about various geometry processing topics.

Furthermore, I would like to thank Mario Botsch for supporting me right from the beginning by giving me, a young 2nd semester student without experience, the opportunity to explore the fascinating area of computer graphics.

It was always a pleasure to work within the Computer Graphics group at RWTH Aachen and I want to express my gratitude to all current and former members. Special thanks go to Jan Möbius for outstanding technical administration and development of OpenFlipper, to Marcel Campen, Henrik Zimmer, Michael Kremer, Timm Lempfer, Tobias Vossemer and Christoph Vogel for being co-authors and to Dominik Sibbing, Ellen Dekkers, Darko Pavic, Martin Habbecke, Lars Krecklau Hans-Christian Ebke and many others for having inspiring discussions at our weekly "Geometry-Stammtisch".

Without my former teacher Heiner Platzbecker, who arouse my curiosity about the beauty of math and natural sciences by communicating his wonderful and extremely helpful intuition, I would have most likely never done this work.

Finally, I want to thank my family and close friends for their never-ending support and patience. In particular I'm thankful to my parents Manfred and Gertrud and my siblings Jarah, Micka, Christoph, Elisabeth, Marian and Martin and my goddaughter Siri for never letting me forget about what is important in life.

Contents

1. Introduction

Accurately describing the geometry of objects in a digital environment, i.e. computers, is an essential ingredient in many of nowadays applications. Often it is desired to forecast the behavior of real phenomena which depend on the geometry of objects by performing a simulation of, e.g. , a car crash, the flow around the wing of a plane, the stability of a building or the quality of the mobile phone network in a city to name just a few. Such simulations are indispensable in situations where an experiment cannot be performed as for instance the task of inspecting the stability of a building in case of an earthquake. However, even in cases where an experiment could potentially be performed, e.g. in the development of a new product, it often makes sense to run a simulation instead of the real-world experiment in order to reduce development cost and/or time.

Another ongoing trend is the virtualization of environments as can be seen for example in the area of navigation or internet shopping. A digital geometry representation enables the user to thoroughly explore a possibly faraway object not only from pre-chosen views but in its full variety. Moreover a digitalized environment offers the powerful possibility of interactively visualizing additional data which is designed to support the desired application as for instance overblended signs in a navigation software.

One step further, instead of replicating and enriching the real world in a digital environment, designers, artists or engineers are able to utilize the enormous potential of today's 3D modeling environments to create new complex objects or sometimes even completely artificial worlds as for example in animation movies.

Motivated by the huge amount of applications there is a long history of different digital geometry representations which were used in the past. Some applications require a solid (volumetric) representation of the object while for others it is sufficient to solely represent its boundary, i.e. the surface of the object. In this thesis we will focus on surface representations while an outlook on the analog volumetric problem will be given

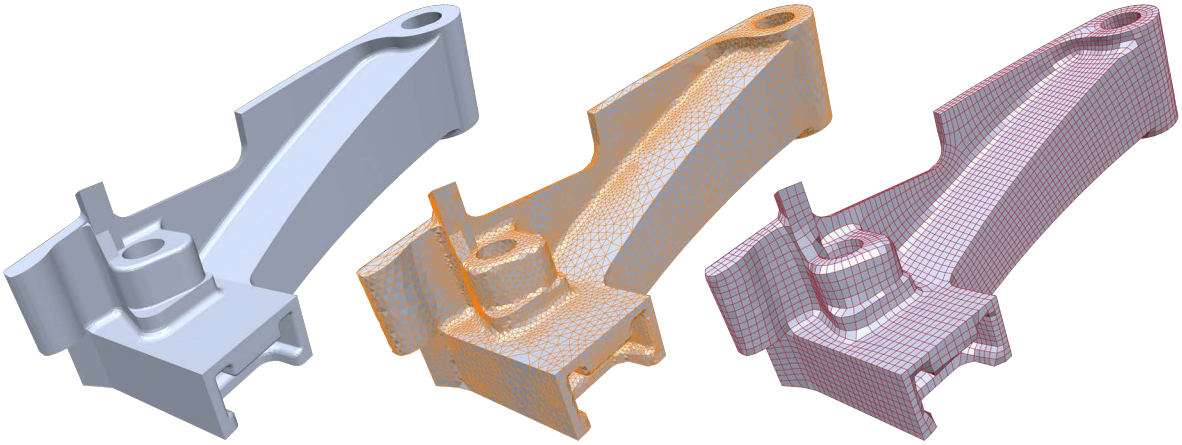


Figure 1.1.: A mechanical object (left) represented as a triangle mesh (middle) and a quadmesh (right). While the triangle mesh easily enables adaptive element sizes, the quadmesh exhibits a superior (mostly regular) structure.

in Chapter 11.

Surfaces in $\mathbb{R}^3$

Surface representations are often divided into three major classes, namely *implicit*, *explicit* and *parametric* representations. The main idea of implicit representations is based on the observation that a 2-manifold surface in $\mathbb{R}^3$ is of co-dimension 1 and consequently can be described as the kernel $K = \{\mathbf{p} \in \mathbb{R}^3 : f(\mathbf{p}) = 0\}$ of a single scalar function $f : \mathbb{R}^3 \mapsto \mathbb{R}$. Such an implicit representation is convenient in applications where topological changes occur like in *Constructive Solid Geometry* (CSG). However, explicitly evaluating points on the surface, for example to render the surface, is equivalent to a root finding process and thus typically very inefficient. In such situations explicit surface representations which are given as a (possibly infinite) set of geometric primitives like points or triangles are more advantageous. In Computer Graphics the probably most prominent explicit representation is the triangle mesh, i.e. the object surface is given as a set of triangles where each triangle is a three-tuple of points $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ (see Figure 1.1 for an example). One reason is that a triangle (or 2-simplex) is in some sense the simplest 2-dimensional entity. A triangle is uniquely defined to be equal to the planar point set of the convex hull of its three (not collinear) corner points and conversely can be efficiently represented by them. In the past strong results like the theory of Delaunay

triangulations in the plane have been developed for triangle meshes.

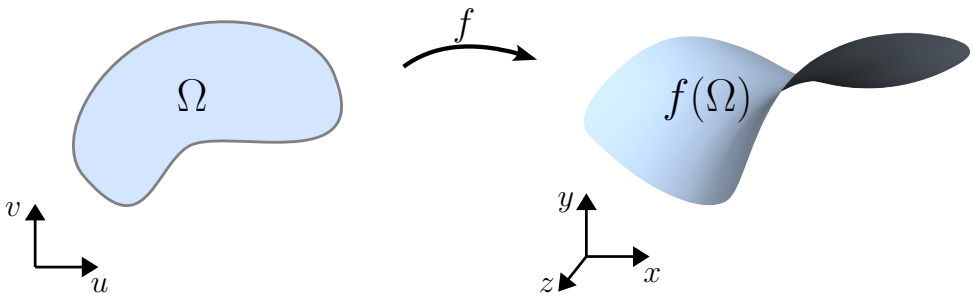


Figure 1.2.: Parametric Surface representation based on a continuous mapping between a 2D domain Ω and its 3D embedding $f(\Omega)$.

The main concept of parametric surface representations is to describe a surface through a mapping $\mathbf{f}(u, v) : \Omega \mapsto \mathbb{R}^3$ between a 2-dimensional base domain $\Omega \subset \mathbb{R}^2$ and the embedding space $\mathbb{R}^3$ (see Figure 1.2). In this setting the properties of the function (continuity, differentiability, etc.) are strongly connected to the shape of the surface and consequently the choice of an adequate function space is essential. Even more general is the concept of manifolds which enable the representation of topologically non-trivial objects by "stitching" several parametric representations with the help of transition functions which guarantee compatibility in overlapping areas. It is important to notice that a triangle mesh is not only an explicit surface representation but also possesses an intrinsic parametrization. Each triangle $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ can be parametrized by the barycentric linear mapping $\mathbf{f}(u, v) = u \cdot \mathbf{a} + v \cdot \mathbf{b} + (1 - u - v) \cdot \mathbf{c}$ with $u, v > 0$ and $u + v \leq 1$. In practice all these individual triangle mappings are often combined into one piecewise linear mapping, where maybe the most prominent example is texture mapping. In Computer Aided Design (CAD) parametric representations often appear in the form of tensor product NURBS (Non-Uniform Rational B-Spline) surfaces. The reason is that these piecewise polynomial surfaces are on the one hand equipped with a guaranteed smoothness but on the other hand still intuitively controllable by means of a net of control points.

Quadrilateral Surface Meshes

Besides triangle meshes, today quadrilateral meshes, also referred to as quadmeshes, enjoy a steadily increasing popularity. Especially in animation and simulation they are often preferred over triangle meshes. One reason is that their tensor-product nature easily generalizes to higher-order representations which are able to satisfy the C^2 continuity requirements that arise in many practical applications. Figure 1.1 depicts a mechanical object represented as a triangle as well as a quadrilateral mesh. A deeper exploration of the advantages of quadrilateral meshes over triangle meshes will be given in Section 2.2. However, it is important to notice from the beginning that a general quadmesh, i.e. a set of four-tuples of points, is not an explicit geometry representation comparable to a triangle mesh. In contrast to a triangle, a quadrangle might be non-planar and/or non-convex such that the specification of the intended surface is more delicate than just taking the convex hull of the corner points. Consequently apart from specialized applications which are restricted to the subset of convex and planar quadmeshes, a quadmesh is usually used as the control mesh of a parametric surface like tensor-product NURBS or generalizations to arbitrary topologies like Catmull-Clark subdivision surfaces.

The main topic of this thesis is the generation of quadrilateral surface meshes. To be useful in practice, a quadrilateral mesh typically has to fulfill strong quality requirements as we will see in Section 2.2. Besides local properties like regularity, element orientation and element shape, also global properties like the patch structure usually play an important role. Consequently instead of local optimization strategies, as typically applied in the generation and optimization of triangle meshes, here global optimization techniques are inevitable. This fact is reflected in the *parametrization based quadrilateral mesh generation* as presented in *Part II* as well as in the *quadmesh optimization* which is the topic of *Part III*. It turns out that the parametrization based quadmesh generation can be formulated as a *mixed-integer* problem (MIP) since it requires continuous optimization in order to compute a distortion minimizing parametrization as well as discrete optimization to determine the discrete connectivity in the quadmesh. Unfortunately traditional optimization approaches for mixed-integer problems are far too slow for the dimensional complexity that we encounter in quadmesh generation. Therefore in *Part I* we will develop an algorithm to rapidly approximate huge (quadratic) mixed-integer problems within a, in practice surprisingly, good tolerance.

In particular the contributions presented in this thesis are the following:

- *Efficient approximation of quadratic mixed-integer problems*, [BZK12]:
  We present a novel approximation algorithm for mixed-integer problems which is carefully designed for the requirements in geometry processing. It is applicable to the class of quadratic mixed-integer energy functionals which are subject to linear constraints. The efficiency, which in our examples is much higher than those of standard mixed-integer optimization methods, is achieved due to its adaptive solution strategy in combination with an elimination approach.

- *Domain optimization for user-guided quadmeshing*, [BVK10]:
  In layout based quadmeshing the user typically provides a coarse segmentation of the surface into quadrilateral patches. We present an algorithm which optimizes the parametrization domains as well as the quad-sampling for each of those patches. In contrast to other methods we allow a more general class of C^0 transition functions and T-junctions within the patch-layout in order to enable a higher mesh quality.

- *A general and flexible quadmeshing algorithm*, [BZK09]:
  Fully automatic quadmeshing algorithms are desired in order to achieve an efficient workflow. However, often not all design decisions are of geometric nature and thus a purely geometrical optimized mesh might lack some properties of the intended application. We present a parametrization based quadmeshing algorithm which is applicable in many scenarios, since on the one hand it is fully automatic but on the other hand the user still has the possibility to provide various high-level guidance constraints, if required.

- *Geodesic distance fields w.r.t. piecewise linear curves on surfaces*, [BK07]:
  Within our quadmeshing pipeline it is sometimes desirable to compute the geodesic distance w.r.t. feature curves or boundaries of the input geometry. We present a novel approach which generalizes the exact computation of the geodesic distance field from point sources to polygonal line sources.

- *Patch Coarsening of quadrilateral meshes*, [BLK11]:
  In many applications quadrilateral meshes with a coarse patch structure are preferred. We present a novel class of global operators, so called *GP-Operators*, which are able to influence the patch structure without introducing new irregular vertices.

Furthermore we propose a novel greedy algorithm which repairs helical structures within quadmeshes in order to optimize their patch structure.

2. Quadrilateral Surface Meshes

The introduction already gave a rough idea, why quadrilateral surface meshes are highly relevant in practice. This chapter is devoted to the task of extending this rough idea to a more complete picture. In Section 2.1 we will start with some basic notions and properties of quadrilateral surface meshes while in Sections 2.2 and 2.3 we will work out important quality criteria which stem from two practically relevant application areas, namely animation and simulation. Finally Section 2.4 discusses the related work, i.e. different classes of approaches which were used for quadrilateral mesh generation in the past.

2.1. Foundations

A *pure quadrilateral surface mesh* (or quadmesh) $Q = (V, E, F)$ is formally a tuple of three sets, namely the vertices V, the edges E and the faces F. An example is given in Figure 2.1. Each topological vertex $v_i \in V$ is equipped with the position of its embedding in space $\mathbf{p}(v_i) = \mathbf{p}_i \in \mathbb{R}^3$. Each edge $e_i \in E$ is a pair $e_i = (v_j, v_k)$ of two vertices. If an edge connects a vertex to itself it is called a *loop*. Each face $f_i \in F$ is a quadruple $f_i = (v_i, v_j, v_k, v_l)$ of diverse vertices which are cyclically connected to form a topological quadrangle. Note that in contrast to the here defined "pure quad" structure, the class of *quad-dominant* meshes allow for a small number of non-quadrangle faces like triangles or pentagons.

Surface Topology

Neighborhood relations between vertices, edges and faces are defined in the usual graph theoretical sense. Two elements are said to be *incident* iff the vertices of one are a subset of the vertices of the other, e.g. the edge $e_2 = (v_9, v_{10})$ is incident to the face $f_5 = (v_{10}, v_4, v_3, v_9)$ in Figure 2.1. While incidence describes the neighborhood relation between elements of different dimension, *adjacency* is a similar concept for entities of

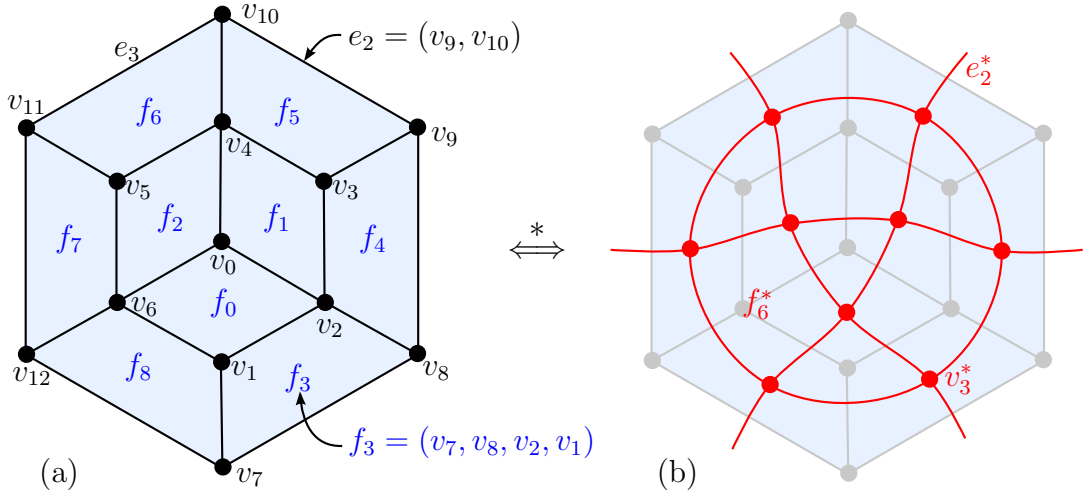


Figure 2.1.: (a) A quadmesh Q is described by vertices v_i, edges e_i and faces f_i. In a pure quadmesh each face is four-sided while irregular vertices like v_0 are allowed to deviate from the regular valence-4 case. (b) The dual Q^* of a quadmesh is an arrangement of curves where the face regularity translates into a vertex regularity.

equal dimension. Two vertices are *adjacent* iff they are incident to a common edge, two edges are adjacent iff they are incident to a common vertex and two faces are adjacent iff they overlap at a common edge. Examples for adjacency between vertices, edges and faces are $v_7 \sim_a v_8$, $e_2 \sim_a e_3$ and $f_1 \sim_a f_4$ in Figure 2.1 respectively. An edge is called *boundary* edge if it is incident to a single face only, otherwise it is called *interior* edge. Vertices inherit the boundary property from edges, i.e. a vertex is called boundary vertex if it is adjacent to at least one boundary edge, otherwise it is an interior vertex. Examples of a boundary edge and a boundary vertex are e_2 and v_8 in Figure 2.1 respectively. For a surface mesh we require the so called *unique-neighbor* property which means that a face has a unique neighboring face when traversing an incident non-boundary edge. Consequently the maximal number of faces which can be incident to a single edge is 2 and we deduce that the surface which is represented by such a quadmesh is essentially a 2-manifold with boundaries.

Irregular Vertices

The *valence* $val(v_i)$ of a vertex $v_i \in V$ is defined to be the number of edges incident to the vertex, with loops counted twice. If the valence is 4 an interior vertex is called *regular* and otherwise it is called *irregular* or *singular* or *extraordinary*. Analogously, on

the boundary a regular vertex is characterized by a valence of 3. As we will see later on, regular vertices are usually preferred over irregular ones. However, the following simple proof shows that for closed surfaces a quadmesh where all vertices are regular can be found only if the genus of the surface is 1.

Proof: In a regular quadmesh without boundary we know that the relation between the number of edges and faces is $|E| = 4/2|F| = 2|F|$ since each face is adjacent to exactly 4 edges and each edge is shared by exactly 2 faces due to the unique-neighbor property. Furthermore with an analog argument we know that $|E| = 2|V|$ because each vertex has valence 4 and each edge is adjacent to exactly 2 vertices and consequently $|F| = |V| = 1/2|E|$. The Euler characteristic χ relates the entities of a closed polyhedron in the following way to its genus g:

$$\begin{aligned} |V| - |E| + |F| &= 2(1-g) \\ \Leftrightarrow 0 &= 2(1-g) \\ \Leftrightarrow g &= 1 \end{aligned}$$

As a consequence of the above observation irregular vertices play an important role in the generation of quadmeshes. Even for genus 1 surfaces it is often desirable to introduce irregular vertices if the surface is more complex than a torus, like e.g. a coffee cup. The above statement can be further generalized to obtain a relation between the valences of the quadmesh vertices and the genus of a closed object:

$$\sum_{i=0}^{|V|-1} (4 - val(v_i)) = 8(1-g) \tag{2.1}$$

Proof: By observing that summation of all valences is equal to counting each edge twice, we deduce that $\sum_{i=0}^{|V|-1} val(v_i) = 2|E|$. Now using Euler's formula together with the above observation that $|F| = 1/2|E|$ results in Equation (2.1).

The above formula is a necessary condition on the sum of vertex valences in a quadmesh which represents a surface with genus g. For example a genus 0 object will require a

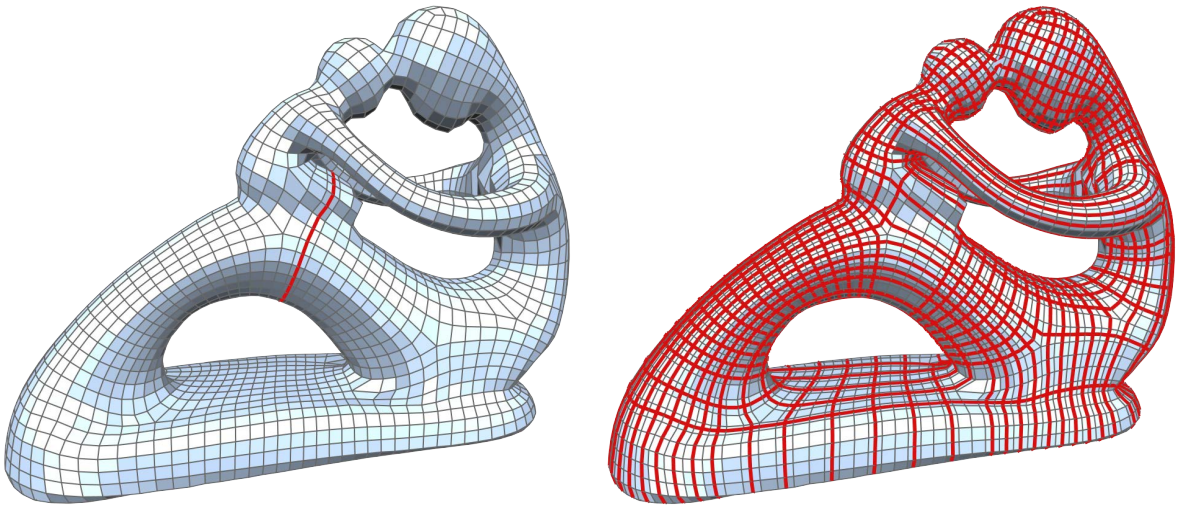
total *valence defect* of 8. Therefore a valid set of irregular vertices would be 8 irregular vertices with valence 3, i.e. each having a valence defect of 1. However, there are infinitely many different possibilities, since positive and negative valence defects cancel out like e.g. a valence 3 and a valence 5 irregular vertex together have a valence defect of 0. Similar to Euler's formula the above condition is necessary but not sufficient for the existence of a mesh.

It is well known that a simple polygon, i.e. planar and non-intersecting, can always be triangulated [FM84, Cha91]. In contrast to that, we will see in the next section that it is not always possible to quadrangulate a polygon. This observation indicates that the generation of quadrilateral meshes involves some global aspects which are not present in the generation of triangle meshes. In particular marching front or divide-and-conquer algorithms for the generation of quadmeshes always need to make sure that they do not run into a deadlock, i.e. a configuration for which no quadrangulation exists. In order to intuitively understand the intrinsic consistency constraint of quadrilateral meshes it is helpful to examine the problem from a dual point of view.

Dual Representation

The *dual* of a quadmesh $Q^* = (V^*, E^*, F^*)$ is given by an isomorphism ($\overset{*}{\Leftrightarrow}$) which uniquely maps k-dimensional entities of the primal to $2-k$ dimensional ones in the dual and vice versa. More precisely each vertex $v_i \in V$ is identified with a dual face $f_i^* \in F^*$, each edge $e_j \in E$ is identified with a dual edge $e_j^* \in E^*$ and each face $f_k \in F$ is identified with a dual vertex $v_k^* \in V^*$. For convenience we choose a mapping which preserves the indices, e.g. $v_6 \overset{*}{\Leftrightarrow} f_6^*$ in Figure 2.1. The connectivity of the dual is uniquely inherited by the primal. If for example two vertices are neighbored in the primal mesh, so will be the corresponding faces in the dual.

The 4-regularity of the faces in the primal translates into a valence-4 regularity of the dual. Consequently we can interpret each vertex of the dual mesh as the crossing of two dual curves (see Figure 2.1 (b) for an example). These dual curves, often called *poly-chords*, uniquely traverse bands of neighboring primal quads and induce the global connectivity of the quadmesh. While simple dual curves are usually preferred, even quadmeshes which seem to be well structured often exhibit long and complicated dual curves with many self-intersections (see Figure 2.2 (b)). However, since all vertices in the dual have a valence of 4 such a dual curve cannot end in the interior, i.e. each curve is



(a) (b)

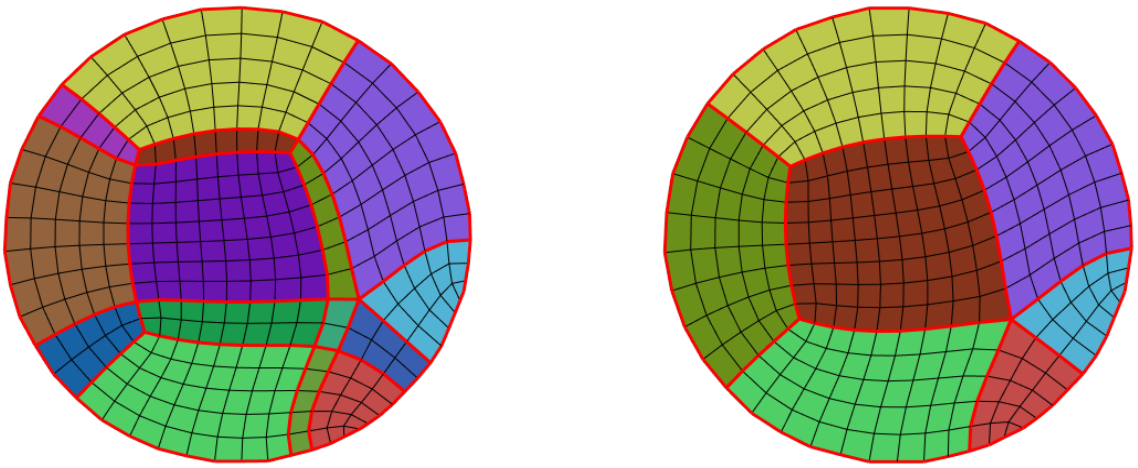
Figure 2.2.: Two different dual curves of the same quadmesh are marked in red. While the first is short and simple (a), the second takes a long path over the mesh, crossing itself several times.

either closed or crosses the boundary twice. Hence, from this point of view the following theorem is strikingly clear:

> **Quadrangulation Theorem:** A polygon in the plane can be quadrangulated if and only if the number of boundary edges is even.

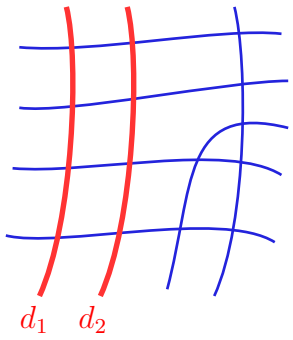
This theorem has important consequences for the design of quadmeshing algorithms, which we will illustrate by means of an example. Imagine that we want to perform a divide-and-conquer quadmeshing algorithm on a closed cylinder mesh. We would intuitively first segment the mesh into three parts, i.e. the curved body and the two flat caps. If we now generate a quadmesh for the curved body first, which is naturally induced by the cylindrical coordinates, we have essentially a probability of 50% that the boundary curves of the caps are formed by an odd number of edges which admit no quadrangulation at all. Obviously, achieving topological consistency becomes non-trivial for more complex objects.

To better understand the global structure of quadrilateral meshes, the complete set of



(a) (b)

Figure 2.3.: Visualization of the base complex. Different patches are depicted with different colors and the discrete separatrices are shown in red. With the same irregular vertices, the base complex can be finer (left) or coarser (right), depending on the variety of topologically different dual curves.



dual curves can be partitioned by a topological equivalence relation $\sim_p$ which clusters *topologically parallel* curves into equivalence classes. Two dual curves d_1 and d_2 are said to be *neighboring* if one is a transversal offset of the other, i.e. each curve segment of d_1 forms a topological quad with both transversally intersecting dual curves and a curve segment of d_2. Intuitively this corresponds to a "ladder" configuration (see embedded Figure). Now the transitive closure of the above (symmetric) neighborhood relation defines $\sim_p$ and classifies topologically parallel curves.

Interestingly, the number of equivalence classes, i.e. the number of topologically different dual curves, is invariant under regular refinement of the quadmesh and thus encodes important structural properties as shown next.

Base Complex

The *base complex* $\mathcal{BC}(Q)$ is a unique partitioning of a given quadmesh Q into rectangular patches P_i, each consisting of $n_i \times m_i$ quads of Q. Thus $\mathcal{BC}(Q) = (\mathcal{V}, \mathcal{E}, \mathcal{F})$ is itself a topological quadmesh where $|\mathcal{F}| = |\{P_i\}|$ denotes a measure of how many regular patches are required to generate Q by regular refinement. Figure 2.3 depicts two example base complexes. The base complex can be constructed in different ways, each uncovering interesting properties. In the primal setting, $\mathcal{BC}(Q)$ is constructed by connecting irregular vertices through straight chains of edges which can be seen as a discrete analogon of separatrices in vector fields. Accordingly, the connection between irregular vertices strongly influences how many patches are required, an observation which is very helpful when optimizing the topology of quadmeshes. From the primal point of view it is not directly clear why $\mathcal{BC}(Q)$ is guaranteed to be a quadmesh. This property becomes obvious when looking at the dual construction. Here the dual of the base complex $\mathcal{BC}(Q)^*$ is given by choosing exactly one representative of all topologically parallel curves classified through $\sim_p$ and each intersection of two such representatives generates one patch P_i in the primal. Since this reduced set of curves is still the dual of a quadmesh, we immediately see that $\mathcal{BC}(Q)$ is always a quadmesh as well.

Topological Quadmesh Classification

With the by now introduced notions we are ready to measure the topological quality of a quadmesh. In practical applications one often highly desired criterion is *regularity* as we will see in Section 2.2. The first measure of regularity is given as the number of irregular vertices s of the quadmesh. However, not all structural regularity is contained in this measure. The reason is, that among all quadmeshes with a constant number of irregular vertices (local regularity) there are arbitrarily large differences in the complexity of the base complex (global regularity). Figure 2.3 shows two example meshes with the same irregular vertices, but different base complexes. To overcome this ambiguity the second regularity measure consists in the size of the base complex, measured as the number of its faces $|\mathcal{F}|$. Note that the second measure is not independent of the first one due to the Euler characteristic. Remembering that the base complex is a quadmesh that contains all irregular vertices by construction, we immediately see that the number of base complex faces is bounded from below by

$$|\mathcal{F}| \geq= s - 2(1 - g) \tag{2.2}$$

On the contrary, since the base complex might contain an arbitrary large number of additional regular vertices, there is no upper bound for $\mathcal{F}$.

Which of both different measures is more important cannot be answered in general and strongly depends on the application in mind. Figure 2.4 depicts a linearized classification of quadmeshes, where the regularity is measured w.r.t. the lexicographical ordering of $\tau = (s, |\mathcal{F}|)$, i.e. the number of singularities is chosen as the dominant measure. It is important to mention that we choose an absolute regularity measure instead of a relative one in order to be independent w.r.t. regular refinement.

Within a practical application a typical task consists in finding "the best" quadmesh with a specified target complexity $|F|$. Of course the overall quality metric is not purely topological and a good compromise between topological and geometrical quality, as discussed in Section 2.3, has to be found. Although the regularity measure τ is continuous there is a loose classification of different mesh types which play an important role in practice. In contrast to the previously introduced absolute measures, this classification is based on the relative regularity measures $s/|V|$ and $|\mathcal{F}|/|F|$ which are both between 0 and 1.

- A *non-regular* or *unstructured* mesh exhibits a large fraction of irregular vertices and consequently the ratio $s/|V|$ is large. Since the number of base complex faces is bounded from below, $|\mathcal{F}|/|F|$ is also large for such meshes.

$$s/|V| \to 1, \quad |\mathcal{F}|/|F| \to 1$$

- A quadmesh is *valence semi-regular* if only a small number of vertices is irregular and thus $s/|V|$ is small. However, as discussed above the number of base complex faces and thus $|\mathcal{F}|/|F|$ can still be arbitrarily large.

$$s/|V| \to 0, \quad |\mathcal{F}|/|F| \to 1$$

- A *semi-regular* quadmesh is a regular refinement of a coarse base complex and consequently is small in both relative measures $s/|V|$ and $|\mathcal{F}|/|F|$.

$$s/|V| \to 0, \quad |\mathcal{F}|/|F| \to 0$$

- A *regular* quadmesh either contains no irregular vertices at all or 4 corner vertices at a single boundary. While the first configuration only exists on a genus 1 object,

the second one is applicable for objects with disk topology and simply consists in a geometrically distorted version of a rectangle of $n \times m$ quads.

$$s = 0 \quad \text{or} \quad s = 4, \quad |\mathcal{F}| = 1$$

The classification based on the introduced relative measures is reasonable only in case of a fixed target complexity. The reason is that without fixing the target complexity, on the one hand the reduction to the base complex would easily turn a semi-regular mesh into a non-regular one, while on the other hand each non-regular mesh can be regularly refined to a semi-regular mesh.

Examples for a typical mesh of each class are shown in Figure 2.4. There is a clear trade-off between regularity on the one hand and flexibility to represent different geometric configurations on the other hand. In practice it turns out that meshes that are valence semi-regular or semi-regular are mostly flexible enough and thus the best compromise. Interestingly, from an algorithmic point of view quite different techniques are necessary to generate semi-regular meshes instead of valence semi-regular ones as we will see in Chapters 8 and 10 respectively.

2.2. Applications

In many practical applications quadrilateral meshes are preferred over triangle meshes, two prominent areas being animation and simulation. However, the quality requirements are typically very strict such that fully automatic quadmesh generation remains a hard task. Therefore even today many quadmeshes, especially those in cutting edge applications, are designed manually by experts equipped with the indispensable knowledge and experience. Studies uncovered that in many of todays workflows the costs of meshing are extremely high, ranging up to 80% of the total costs [MTTT98].

In the following two sections we will review two of those workflows in order to identify their quality requirements. The overall goal is the design of either fully automatic or alternatively semi-automatic quadmesh generation algorithms which are on the one hand able to meet the strict quality requirements and on the other hand greatly speedup the design process and thus reduce the mesh generation costs.

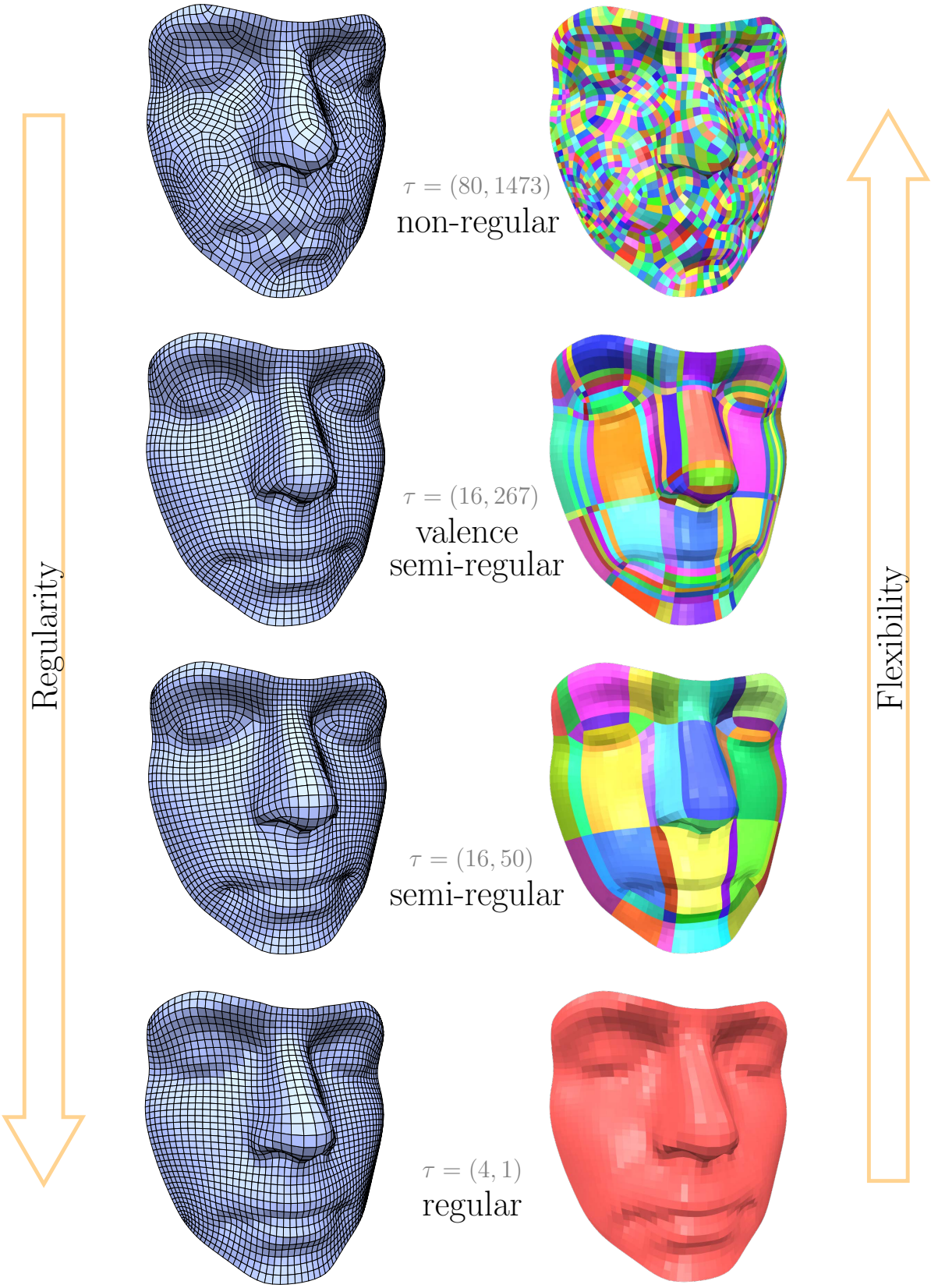


Figure 2.4.: Topological Quadmesh Classification

2.2.1. Animation

Today quadrilateral meshes are the state-of-the-art geometry representation in animation. One reason is that most animation and rendering systems are based on subdivision surfaces (see [ZS00] for details) where the quadmesh is used as the so called control mesh, which represents the discrete DOF's of the smooth surface in a geometrically reasonable way. The typically applied Catmull-Clark subdivision scheme generalizes bi-cubic uniform B-spline surfaces to arbitrary topology and is very well suited as a high-quality representation for smooth surfaces. By restricting the control mesh to regular quadmesh topology, bi-cubic uniform B-spline surfaces are automatically reproduced. In this special case of a regular control grid, the surface quality is optimal and consequently from a topological point of view, even though the handling of arbitrary topology is possible, the quadmesh should be as regular as possible.

The topological simplicity is often in conflict with geometric requirements. Maybe the most important one is that the quad elements have to be oriented carefully in order to well capture the local curvature. More precisely, in parabolic regions where the surface is bent in a single direction, a nice curvature distribution can be achieved only if the quad grid is aligned to the principal curvature directions (cf. [D'A00]) . A special case with infinite curvature in one direction are sharp creases which can be handled by an extended set of subdivision rules but again only if the edges of the quadmesh are well aligned, i.e. the sharp crease is explicitly represented by edges in the quadmesh.

In the animation community it is well known that irregular vertices (similarly non-quad faces) and badly oriented or heavily distorted quad elements are the main sources of artifacts that pop up in the rendering. Obviously, dynamic geometries like, e.g. , the face of a talking person are even more critical than their static counterparts since artifacts have to be prevented for many different configurations. Accordingly designers spend lots of work for puzzling out highly optimized so called "edge flows" that facilitate artifact-free animations.

Moreover, animation systems heavily exploit mapping techniques to decouple the rough geometric description from high frequency details. Most prominent examples are texture and displacement mapping techniques which are applied in order to represent high frequencies in material as well as in geometry. A well chosen quadmesh

is perfectly adapted to such mapping tasks, since a quad can be naturally mapped to a two-dimensional array which perfectly fits into todays memory architectures. Thus, from this point of view it is highly desirable to work with quadmeshes which are semi-regular, i.e. meshes that can be partitioned into a few regular patches of rectangular shape, acting as mapping domains.

Altogether, to be useful in animation, a quadmesh has to be a good compromise of topological simplicity (semi-regular) and geometric approximation (element distortion, curvature orientation, feature preservation, ...). Unfortunately the violation of a single criterion makes the mesh useless, which explains the manual design pipelines still often chosen in practice.

Today a typical mesh generation process consists of three steps. In the first step an artist designs a new object, e.g. a character, and builds a prototype out of clay. In the second step, a digital model is captured by a 3D laser scanner in form of an unstructured and noisy triangle mesh. Once such a digital representation is available, a designer manually works out a coarse quadrilateral patch layout which on the one hand has to be well aligned to the structure and on the other hand leads to a semi-regular mesh. In this step the designer incorporates her expert knowledge by placing irregular vertices away from qualitatively critical areas that will undergo large deformations in the planned animation sequences. Finally the coarse patch layout is refined regularly to a semi-regular mesh of adequate sampling density.

The task we are facing in this thesis consists in the simplification of the third step, i.e. the conversion from a low-quality triangle mesh to a high-quality quadrilateral mesh suitable for animation. It is questionable whether or not this step can be ever fully automatized, since a well chosen mesh contains many fuzzy aspects that are hard to formalize or to "correctly" weight against each other. Therefore the holy grail of mesh generation will be an algorithm which on the one hand is able to generate optimal results w.r.t. to a specifically chosen quality metric but on the other hand still allows for simple and time efficient high-level user control in cases where this quality metric does not lead to the structure that a human designer prefers due to whatever reason.

2.2.2. Simulation

As discussed in the introduction, in simulation the main goal is to accurately predict the physical behavior of digitally represented objects. In this context the task of meshing is two-fold, since on the one hand it is essential to accurately represent the geometry of the object itself, while on the other hand the result of the simulation has usually to be accurately represented by the same mesh. To illuminate the importance of the first aspect imagine a thin-shell simulation of e.g. a car body. The mathematical formulation of such a physical system contains second derivatives of the surface geometry itself and thus might be strongly influenced by a naive non-smooth discretization. The second aspect can be understood by investigating a simulation with a simple geometry like e.g. a cube. Here the geometry itself does not require a finely tessellated mesh, however, the simulation might necessitate a (locally) denser mesh to stay within an acceptable error tolerance. One example for such a setting would be heat propagation on a simple object but with complicated boundary conditions.

A successful trend in engineering, called *Isogeometric Analysis* [HCB05], consists in integrating *Computer-Aided Design* (CAD) and *Finite Element Analysis* (FEA) into a single process based on *Non-Uniform Rational B-Splines* (NURBS). One advantage of NURBS is that they can easily handle different kinds of refinements that are important in practice. The first option to achieve a more accurate function space consists in refining the control mesh itself, a so called *h-refinement*. Such a mesh refinement can be done either globally with B-Splines or locally by using a *T-Spline* generalization [SZBN03]. The second way to achieve a more expressive function space, which is called *p-refinement*, consists in increasing the degree of the polynomial basis functions. Due to its tensor-product nature such a p-refinement is an easy task within a structured quadrilateral mesh, while it is extremely complicated in an arbitrary unstructured mesh.

Indeed, in our case of surfaces, a single NURBS-patch is represented by a regular control quadmesh. Although the overall task in simulation is quite different from the one in animation, interestingly the quality requirements turn out to be quite similar. As before, a well chosen orientation of the quads is extremely important to on the one hand accurately capture the geometry itself and on the other hand capture the characteristics of the underlying partial differential equations (PDEs). The same holds for the preservation of sharp creases. In animation a “good shape” of the individual quads is

important to avoid visual artifacts, while in simulation a good shape is essential for the accuracy of the solution as well as for the conditioning of the discretized operators.

Today's most powerful solution approaches are adaptive in the sense that they start with a very coarse representation and then locally refine the mesh where it is required until a given target accuracy is reached. Such adaptive approaches strongly benefits from a semi-regular mesh that consists of a small number of well aligned patches and consequently a small number of initial NURBS patches.

2.3. Quality Criteria

The above discussion showed that measuring the quality of a quadrilateral mesh as a single number is not feasible in practice. Instead, a better way is to identify several, often conflicting, quality criteria whose individual importance strongly depends on the desired application. Nevertheless, a competitive and flexible quadmesh generation algorithm has to consider all of them and should offer the user a mechanism to influence the relative importance of each aspect.

As we have seen, many quality criteria are related to the approximation quality, either of the geometry itself or the solution of a physical simulation. Some of them are locally measurable while others require global considerations.

Element quality. Usually an individual quadrilateral element should be close to a square. This implies several aspects, namely 90 degree inner angles, equal edge length and planarity (cf. [Knu01]). On a curved surface it is typically not possible to find a quadrilateral mesh that consists of perfect squares only and the individual deviations are used as a (local) quality measure. Often histograms are plotted in order to compare the quality of different algorithms. In many applications the *worst element* limits the usability, e.g. in terms of accuracy, and therefore is chosen as quality measure instead of the average. Depending on the application sometimes rectangles are favored instead of squares due to their better approximation quality in case of anisotropic curvatures or anisotropic behavior in PDEs like the boundary layer in fluid simulation. It is worth to mention that apart from all their advantages quads are intrinsically more complicated than triangles. While a triangle is always planar and convex this is not true for a quad.

Element orientation. The orientation of elements is strongly connected to the approximation quality (cf. [D'A00]). For instance, a bad element orientation leads to noisy curvature distributions in the corresponding subdivision surface. Apart from this purely geometric aspect, many applications require the quadmesh to closely follow a prescribed orientation field. Examples are Langer's lines in animation, which correspond to the fiber directions of the human skin, or the characteristic lines of a PDE, which enable high accuracy in upwind schemes. To measure the orientation quality of a given quadmesh it is possible to compute the deviation from a prescribed reference orientation field, e.g. the principal curvature directions. However, finding a good metric to measure the overall orientation quality is non-trivial and therefore often a purely visual inspection is used instead.

Feature preservation. Sharp features in the input geometry should be preserved in the quadmesh by explicitly representing them by sequences of edges. Typically this is a binary quality criterion since a quadmesh generation algorithm either is or is not able to handle feature curves. One problem which arises in practice stems from the fact that feature recognition in noisy input data itself is a non-trivial task. Therefore, often a user has to specify or adjust the desired feature curves manually to achieve robust preservation.

Irregular vertices. A key property of a quadmesh is a well chosen and distributed set of irregular vertices. While a small number is favored to keep the mesh topology simple, a careful choice is inevitable in order to enable well behaved element quality and element orientation. Irregular vertices are required to (1) compensate the Gaussian curvature of the surface, (2) handle tangential curvature of the desired element orientations and (3) enable adaptive element sizing. Consequently, depending on the complexity of the input geometry, a number of irregular vertices are indispensable to achieve an overall good mesh quality. Intuitively an irregular vertex can be seen as an absorber of present non-regularity in its vicinity. However, since irregular vertices are only available in a discrete graduation, related to their integer-valued valence, they typically cannot absorb the locally optimal amount of non-regularity. As a consequence, the global positioning of irregular vertices is a very hard task, strongly related to the complexity of finding optimal solutions in discrete optimization.

Patch structure. While a globally good distribution of irregular vertices usually leads to valence semi-regular meshes, more effort has to be taken in order to construct a real semi-regular mesh. There are infinitely many different base complexes for the same set of irregular vertices and the chosen discrete separatrices, connecting irregular vertices, decide on the number of patches. Usually a small number of patches is preferred due to its increased regularity. However, there is again a trade-off between the number of patches and the element quality and orientation. In general, the mesh with the "best" element quality and orientation consists of a large number of patches and some distortions have to be made in order to reduce their number. Often the number of patches can indeed be greatly reduced by tolerating a small change in element-wise orientation and quality. However, a very aggressive reduction to the smallest possible number of patches typically results in heavy distortions.

In the subsequent parts of this thesis we will develop different approaches which are designed to explicitly address the above quality criteria. Obviously it would be desirable to optimize all criteria at once. However, it turns out that a simultaneous optimization is very hard and impractical in terms of runtime. One solution that we will investigate here is the splitting of the overall optimization in carefully chosen sub-steps that subsequently optimize different criteria, while staying reasonably close to previously optimized criteria.

Before presenting the main contributions of this thesis, we provide next a short overview of related work in the area of quadmesh generation and quadmesh processing.

2.4. Related Work

In the last years quadmesh generation attracted a lot of attention and it is out of the scope of this dissertation to exhaustively review all of the developed techniques. Here, only a short summary and classification of the most important approaches is given, based on our state-of-the-art report [BLP*12]. Additionally we will relate our main contributions to these other work. Most of the approaches that are presented in this thesis are based on articles that we already published individually. To avoid confusion, we mention all these articles and their relation to this thesis within this section. For more details on the individual topics we refer the interested reader to excellent surveys about mesh

generation and processing [AUGA05, BLP*12].

The developed techniques can be classified into *quadmesh generation* and *quadmesh processing* algorithms. While the former class consists of algorithms that convert geometric data given in a different representation like e.g. a point cloud, triangle mesh or polynomial surface into a quadmesh, the latter class of algorithms starts with an input quadmesh and outputs a somehow improved quadmesh. Of course algorithms of both classes can be combined into quadmeshing systems like for instance a simple conversion algorithm followed by a quality optimization algorithm.

Quadmesh generation

Tri-to-quad conversion. The most simple quadmesh generation algorithms perform a tri-to-quad conversion [VZ01, LKH08, TPC*10, GLLR11, RLS*11]. The main idea of these algorithms is to pair neighboring triangles into quads by gluing them along their common edge. In general these algorithms strongly depend on the sampling of the input triangle mesh. Consequently it is inevitable to either pre-process the triangle mesh [LKH08] or to further optimize the resulting quadmesh [TPC*10]. However, even then the resulting quadmeshes are typically unstructured, i.e. of non-regular type, and thus not well suited for applications like animation or simulation.

Explicit curve tracing. A second approach to quadmesh generation is the explicit tracing of group-wise orthogonal curves which induce a curvilinear grid on the surface. From an approximation point of view (c.p. Section 2.3) it is beneficial to choose curves along the principal curvature directions [ACSD*03, MK04]. The difficult part in these algorithms is the achievement of well-behaved curve distributions. Although powerful techniques known from streamline placement in flow visualization were adapted, a globally smooth distribution of distortions cannot be expected from those methods. Additionally to the afore-mentioned drawback, in general the traced curves form a quad-dominant mesh which requires one step of Catmull-Clark subdivision in order to generate a pure quadmesh.

Parametrization based. In order to achieve a smoother distribution of distortions, the largest class of quadmesh generation algorithms is based on global parametrization. Here the main idea consists in the construction of a mapping from the surface embedded

in 3D to a 2D domain such that the quadrangulation in the domain becomes trivial. Usually the domain is tessellated by a regular tiling like the canonical quadmesh formed by the Cartesian grid of integer isolines, as explained in more detail in Chapter 6. The tricky part in this setting is the design of consistency conditions that ensure a correct stitching of isolines along the cutgraph which is essential to cut open surfaces of non-disk topologies. However, after fixing stitching as well as other desired boundary conditions, the "optimal" mapping function is typically found implicitly by minimizing an energy functional and thus the distortion is distributed smoothly. Depending on the function space, which is chosen for finding the optimal mapping, there are two different categories of approaches, namely *layout based* and *triangle-chart based*.

Algorithms from the first category decompose the overall problem into a segmentation phase, where a layout is generated by partitioning the surface into quadrilateral patches, and a parametrization phase, where the best mapping w.r.t. the previously fixed layout is found. Note that such a layout is identical to the base complex of the resulting mesh. While such approaches usually lead to nice semi-regular meshes, unfortunately, automatically finding a high-quality patch layout is still an unsolved problem. Therefore some approaches rely on a manually designed patch layouts [THCM04, TACSD06, BVK10]. As an alternative some algorithms construct the layout by means of the Morse-Smale complex of a scalar function [DBG*06, TACSD06, ZHLB10]. Chapter 7, which is based on our work [BVK10], describes layout based approaches in more depth in order to develop a powerful user-guided reverse engineering algorithm.

Triangle-chart based approaches are more flexible in the sense that the base complex is not fixed from the beginning. The only constraint is that the individual mappings of neighboring triangles have to be compatible to each other, typically enabling many different layouts within the optimization. Often such approaches are guided by 4-symmetric orientation fields [HZ00, PZ07, RLL*06, RVLL08, RVAL09, BZK09, LJX*10] which stem from the extrapolation of confident principal curvature directions as estimated by a discrete shape operator [CSM03] or jet fitting [CP05]. While in contrast to the layout based approaches triangle-chart based algorithms like [RLL*06, KNP07, BZK09] are able to generate high-quality quadmeshes in a fully automatic manner, a clear drawback consists in the fact that the resulting meshes usually lack a nice patch structure and thus are only valence semi-regular. Chapter 8, which is based on our work [BZK09], is devoted to the task of developing the mathematical details of a flexible orientation field guided

quadmesh generation approach.

Due to their good performance in the case of triangle meshes, parametrization based approaches were also generalized to inputs consisting of point clouds [LLZ*11] or range image sets [PTSZ11] in order to circumvent the process of generating a triangle mesh first.

Voronoi based. A well known class of triangular remeshing algorithms is based on the Voronoi diagram [AGK], i.e. the partitioning of surface points into regions containing the closest surface points w.r.t. a set of seed points. Starting with a set of random seeds and performing Lloyd's relaxation [Llo82], which iteratively generates the Voronoi diagram and moves each seed point into the barycenter of its corresponding Voronoi cell, results in the so called *Centroidal Voronoi Tessellation* algorithm [DFG99] which optimizes the distribution of seeds. In [LL10] this concept is generalized to Voronoi diagrams of L_p norms, well suited to generate samplings that after tri-to-quad conversion lead to high-quality quad-dominant meshes.

Quadmesh processing

Algorithms from this class work completely in the space of quadmeshes. In practice desired optimizations can be grouped into *geometry optimization*, *simplification* and *connectivity optimization* methods as discussed below in more detail. Approaches that solely optimize the geometry, i.e. the 3D position of vertices, are very similar to their counterparts known from triangle mesh processing as e.g. described in [BKP*10]. However, due to the global topological restrictions within a quadmesh, approaches which alter the connectivity usually require fundamentally different methodologies.

Geometry optimization. A building block of many of today's algorithms consists in tangential mesh smoothing. Such a smoothing operation is typically applied to distribute large local distortions within the neighborhood. In the case of triangle meshes, often Laplacian smoothing [DMSB99] is performed which can be easily adapted to quadrilateral meshes [ZBX05] or even polygonal meshes [AW11] as well. Typically after a smoothing step all vertices are projected onto the input surface. This can lead to instabilities or bad approximation quality of the resulting mesh, especially if it is coarse. To overcome these problems, parametrization based smoothing algorithms were developed

[DBG*06, PTC10] which perform quite good if the input mesh exhibits a coarse base-complex.

One very important step within a subdivision surface framework consists in fitting the surface generated by subdivision to a reference geometry mostly given as a dense triangular mesh. A general approach to subdivision surface fitting was developed in [LLS01]. In case of Catmull-Clark subdivision this step updates the 3D vertex positions of a given control quadmesh until the distance between subdivision- and reference surface cannot be reduced further by local movements. Obviously this process is highly non-linear and requires a good initial configuration.

For several applications planar quad elements are desirable, e.g. the design of buildings covered by glass panels in architecture. Starting from an initial configuration, planarization algorithms like [PW08, GSC*04] typically optimize non-linear energies composed of planarity, smoothness and fitting terms. This concept was extended to more general constraints in [YYPM11]. Another trend in architecture consists in finding meshes where the number of different tiles is minimized [EKS*10, FLHCO10, ZCBK12] in order to reduce the production cost. To do so, typically clustering techniques are interleaved with geometric optimization.

Within *Finite-Element Methods* (FEM), a single element of bad quality can possibly screw up the whole simulation. Therefore clean-up methods were developed in order to locally fix such configurations [Kin97]. In contrast to the previously discussed optimizations here also connectivity changes are performed.

Simplification. The aim of mesh simplification, also called *decimation*, consists in reducing the number of mesh elements. This topic is well understood for triangle meshes, see e.g. surveys [CMS97, GGK98, Lue01]. A typical simplification algorithm is build from a set of operators together with a quality metric, such that greedily the "best" operation can be performed until a desired target mesh complexity is reached. While in the case of a triangle mesh very simple local operators are sufficient, for a quadmesh the situation is more difficult. Recently, different simplification operators which preserve the quadmesh topology were proposed in [DISSC08, DSC09a, SDW*10, TPC*10]. Unfortunately all local operators in a quadmesh require the change of local valencies and thus often generate additional irregular vertices in each step. Global operators like the

poly-chord collapse (cf. Section 2.1) do not require additional irregularities, however, they are strongly dependent on the dual structure of the input mesh and thus typically not flexible enough. As a consequence, the performed greedy algorithms tend to get stuck in local minima with sub-optimally distributed irregular vertices.

One strength of quadmesh simplification algorithms is their robustness. Accordingly it is possible to aggressively decimate the mesh in order to generate a coarse base complex for a semi-regular mesh as done in [DSC09b]. But again the greedy strategy is usually not able to generate structure aligned patches with well distributed irregular vertices. As a result the quality of the base complex is typically far away from manually designed ones.

Connectivity optimization. As discussed above, parametrization based quadmesh generation algorithms are able to automatically generate high-quality valence semi-regular meshes, i.e. meshes with a small number of globally well distributed irregular vertices but without a coarse patch structure. Since many practical applications require a semi-regular mesh instead, recently algorithms were developed that convert a valence semi-regular mesh into a semi-regular one [BLK11, TPP*11]. The main idea of these algorithms consists in changing the global connectivity, i.e. the discrete separatrices which connect irregular vertices, without altering the set of irregular vertices itself or deviating too much from the initial structure alignment. It turns out that such a connectivity change requires global operators that can be generated by a graph search [BLK11] or explored by a backtracking algorithm [TPP*11]. In Chapter 10, based on our work [BLK11], we explain the concept of so called *grid-preserving operators* that can be applied to optimize the base complex by iteratively removing helical structures within the quadmesh.

Another possibility to optimize the connectivity consists in the manual movement of pairs of irregular vertices [PZKW11]. Note that due to the topological restrictions it is not possible to move a single irregular vertex alone.

Most recently, we proposed an approach which, instead of optimizing the global connectivity, is able to directly construct a coarse base-complex [CBK12]. It exploits the dual point of view, where the topological restrictions are easier to handle. The main idea consists in finding a minimal number of dual loops, that roughly follow a given

orientation field and reproduce its singularities.

All these (topological) connectivity optimization algorithms are usually combined with a geometric optimization step as explained above.

Part I.

Mixed-Integer Optimization in Geometry Processing

The first part of this thesis is devoted to the topic of mixed-integer optimization within geometry processing applications. This combination is a delicate issue due to the following reason: On the one hand mixed-integer programming belongs to the most difficult problems in optimization and can be very hard even for small instances. On the other hand geometry processing applications are very likely to deal with large instances like thousands or even millions of points or triangles and the optimization criteria are often nonlinear. Therefore, although powerful general mixed-integer optimization strategies exist, in geometry processing it is inevitable to specifically adapt them to a given class of problems to be able to find adequate solutions in reasonable time. The main idea for doing so is to exploit the geometric intuition of the underlying problem in order to find more efficient formulations of the given problem. It turns out that many of the general optimization strategies, developed in the past, are as well based on geometric considerations stemming from polyhedral theory. Nevertheless, for many problems there is still no hope of finding the optimal solution and often the best one can hope for is a good approximation algorithm that is able to find feasible solutions that are at least close to the optimal one.

Part I is structured in the following way. Chapter 3 starts with the definition and description of general mixed-integer problems. Then in Chapter 4 some widely applied general solution strategies are described on an intuitive level. These approaches, especially the *branch-and-cut* method, represent the state-of-the-art in the area of mixed-integer optimization and will be our reference for comparison. Finally, based on [BZK09, BZK12], Chapter 5 presents our method to rapidly approximate large (quadratic) mixed-integer problems, which is specially designed for the requirements of quadmesh generation algorithms as presented in Part II.

3. Mixed-Integer Nonlinear Programming

The class of *mixed-integer nonlinear programs* (MINLP) consists of all optimization problems, where some of the unknowns are *continuous* while others are *discrete*. Such problems naturally arise whenever discrete decisions are optimized simultaneously with continuous one. For example, when designing a gearbox, the radius of individual gears can be varied continuously while the number of gears is clearly a discrete quantity. For clarity, in the following description the continuous variables are always chosen to be $\mathbf{x} \in \mathbb{R}^n$ and analogously the discrete ones are $\mathbf{y} \in \mathbb{Z}^m$. The objective function of the optimization problem is an arbitrary, scalar valued, nonlinear function $E(\mathbf{x}, \mathbf{y})$. Additional constraints of the form $C(\mathbf{x}, \mathbf{y}) \leq 0$ can be specified in order to restrict the set of valid solutions to $\mathcal{F} \subset \mathbb{R}^n \times \mathbb{Z}^m$ which is called *feasible region*. These scalar valued constraint functions might be arbitrary nonlinear functions as well. In summary a MINLP can be written shortly as

$$\begin{aligned} \text{minimize} \quad & E(\mathbf{x}, \mathbf{y}) \\ \\ \text{subject to} \quad & C_i(\mathbf{x}, \mathbf{y}) \leq 0 \qquad i = 1 \dots l \\ & \mathbf{x} \in \mathbb{R}^n \\ & \mathbf{y} \in \mathbb{Z}^m \end{aligned} \tag{MINLP}$$

In such a MINLP there are several aspects, which potentially induce a very hard optimization problem. First of all, finding the optimal assignment for the discrete variables is a very complicated task. In contrast to continuous optimization it is not possible to improve the solution values of discrete variables by performing small steps into the negative gradient until the solution is locally optimal. Such an approach would violate the integrability constraint of discrete variables and in fact the continuous optimum can potentially be far away from the best discrete one. An illustration of a mixed-integer

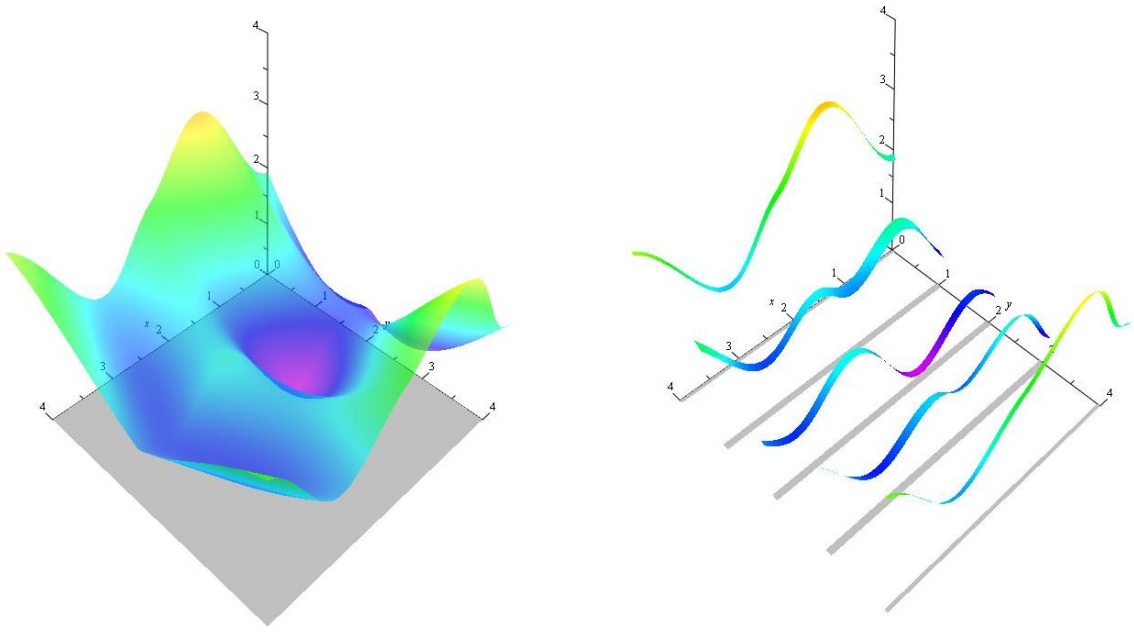
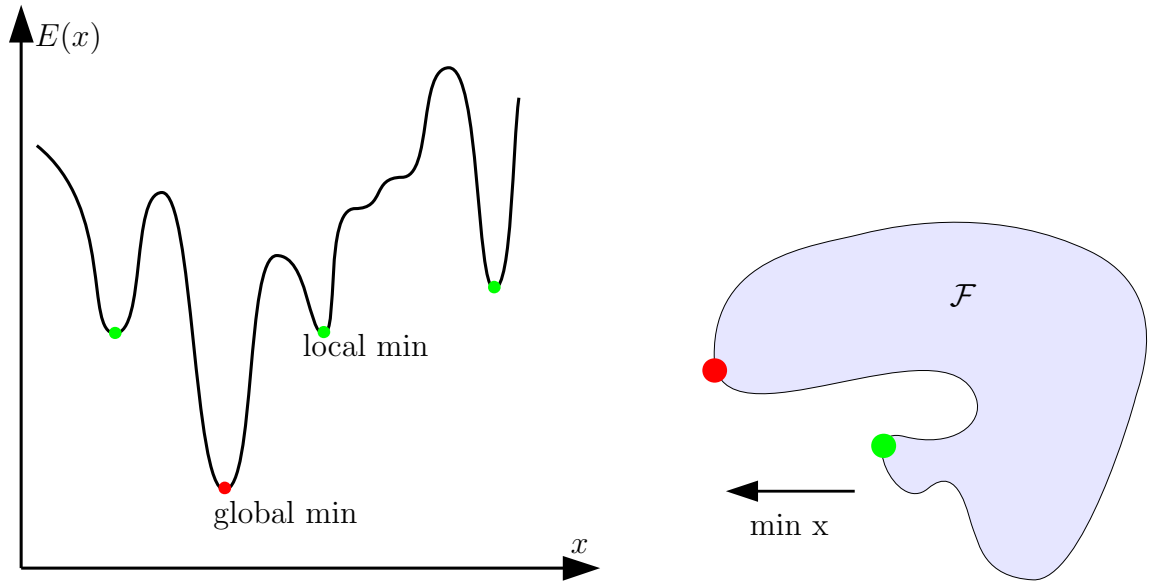


Figure 3.1.: (left) a continuous optimization problem where each point in the plane $\mathbb{R} \times \mathbb{R}$ is a feasible solution, i.e., a point which fulfills all constraints of the problem. (right) a mixed-integer problem where the set of feasible solutions is $\mathbb{R} \times \mathbb{Z}$. For minimizing such problems typically all discrete possibilities have to be tested explicitly.

objective with one continuous and one discrete variable is depicted in Figure 3.1. It turns out that for discrete optimization methods completely different techniques are required compared to their continuous counterparts. The main principles of discrete optimization will be discussed in more detail in Chapter 4 while here only one intuitive example is given in order to illustrate the enormous complexity which is connected to discrete optimization. For simplicity assume that all variables are discrete and restricted to $\{0, 1\}$, such a problem is called a *binary problem*. The most straightforward way to optimize such a problem consists in enumerating all 2^n many different assignments, computing their objective and selecting the best one. The time complexity for doing so is exponential and even a small problem with 100 binary variables will consist of $2^{100} \approx 1.2 \cdot 10^{30}$ many sub-problems. In practice *branch-and-bound* algorithms, as explained in Section 4.1, are often applied. These algorithms aim at reducing the number of examined sub-problems in the complete tree of all discrete variable assignments by pruning away sub-trees where it can be proven that they do not contain the optimal solution. But still the worst case complexity of discrete optimization is exponential and it can be shown to be NP-hard.



(a) (b)

Figure 3.2.: The behavior of nonconvex objective functions and nonconvex constraints is very similar. (a) Several local minima of a nonconvex function are shown as green dots while the global minimum is colored in red. (b) The feasible region $\mathcal{F}$ is nonconvex. When searching for the point with minimal x-coordinate, the green point is locally optimal and cannot easily be optimized continuously to the global optimum shown in red.

Accordingly, the first issue which makes MINLP hard consists in the discrete variables. However, there are some other pitfalls which are not related to the discrete variables. If the integer constraints are neglected, the resulting problem is typically called the *relaxed problem* which is a sub-problem of several optimization strategies and given by the following description

$$\begin{aligned} \text{minimize} \quad & E(\mathbf{x}, \mathbf{y}) \\ \text{subject to} \quad & C_i(\mathbf{x}, \mathbf{y}) \leq 0 \qquad i = 1 \dots l \\ & \mathbf{x} \in \mathbb{R}^n \\ & \mathbf{y} \in \mathbb{R}^m \end{aligned} \tag{R-MINLP}$$

In this relaxed problem, which is an instance of *Nonlinear Programming (NLP)*, there

are two aspects that might give rise to intractability. The first one is due to the nonlinearity of the objective function. It is well known that finding the global optimum of an arbitrary nonlinear function can be very hard since there might exist arbitrarily many local minima (cf. Figure 3.2a) where an optimization strategy could potentially get stuck in. Although there are methods like *simulated annealing* that are able to escape local minima, such algorithms are typically too expensive in the context of high-dimensional geometry processing problems we are targeting at. Depending on the problem there are different ways to overcome the issue of poor local minima. The maybe most elegant one consists in designing a *convex* objective function that possesses the same, or at least a similar, global optimum as the original one. For convex objective functions there are many approaches, as for instance trust-region or interior-point methods [NW06] that can easily find the global optimum. If no convex formulation can be found, the second option consists in designing a heuristic that provides a good initial solution such that the closest local minimum is at least of adequate quality. The hardness of complicated nonlinear objective functions is a well understood topic, however, in the context of mixed-integer optimization it is important to notice that the time complexity of one instance is even more critical since typically a very large number of such instances with different assignments of discrete variables have to be solved.

The third aspect which has to be taken care of is related to the constraints C_i. Similarly to the objective function it is strongly desirable that all constraint functions are convex. Otherwise, optimization strategies often result in locally optimal solutions that are far away from the global optimum, as illustrated in Figure 3.2b. Moreover, in the case of complicated constraint functions it can be hard to even find any feasible solution, i.e. a point that satisfies all constraints simultaneously. Imagine e.g. the case of several complicated equality constraints. Finding a feasible solution then is equivalent to the non-trivial task of finding a solution to the induced nonlinear equation system. For some powerful optimization approaches it is required to start with a feasible initial point. Such a point is often generated by solving the following feasibility problem:

$$
\begin{aligned}
\text{minimize} \quad & z \in \mathbb{R} \\
\\
\text{subject to} \quad & C_i(\mathbf{x}, \mathbf{y}) \leq z \qquad i = 1 \ldots l \qquad \text{(FEASIBILITY)} \\
& \mathbf{x} \in \mathbb{R}^n \\
& \mathbf{y} \in \mathbb{R}^m
\end{aligned}
$$

If a solution with $z \leq 0$ can be found, clearly the corresponding point is feasible and thus can be used as a starting point for the optimization w.r.t. the original objective function. If no such point can be found, the problem seems to be infeasible.

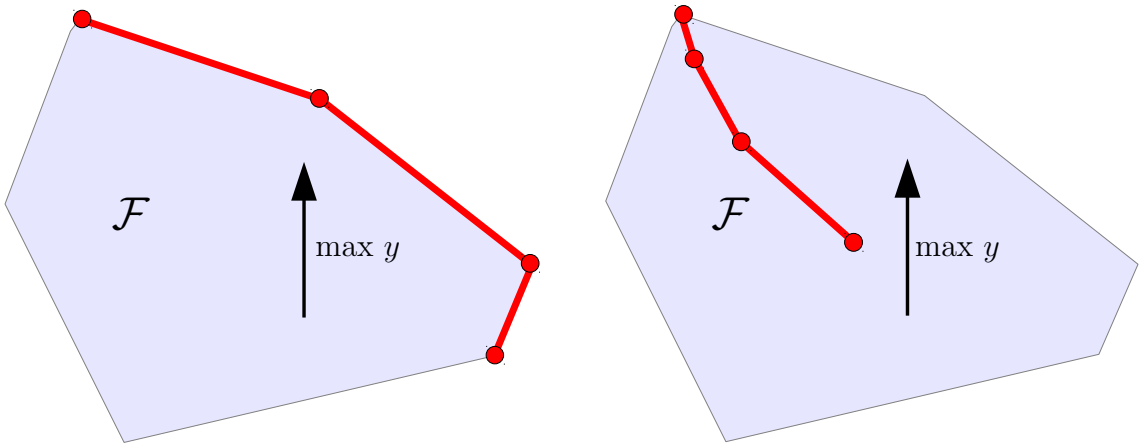
Altogether, it turns out that finding optimal solutions to MINLP is feasible only if the objective function as well as the constraint functions are convex [Gro02]. In this case, the optimal solution of the relaxed problem can always be found efficiently by performing local improvements. We will call such instances *convex Mixed-Integer Nonlinear Programming (cMINLP)*. Highly optimized algorithms like those of [GRB11] or [IBM12] exist for special cases of this convex setting, namely *Mixed-Integer Linear Programming (MILP)* and *Mixed-Integer Quadratic Programming (MIQP)*. The first class is characterized by linear objective functions in combination with linear constraints, while problems from the second class can be quadratic, but solely in the objective function.

4. General Optimization Approaches

The topic of this chapter consists in general optimization strategies for MINLP. The presented methods, namely *branch-and-bound*, *cutting-plane approach* and *branch-and-cut*, are able to find the global optimum of cMINLPs, i.e. convex MINLPs. They are also applicable to nonconvex problems, however, in such cases without any quality guarantees. All three approaches are based on the successive refinement of continuously relaxed problems. The main idea is to replace the discrete integrability constraint, which cannot be tackled with continuous optimization, by well-chosen sets of (continuous) inequality constraints. In this way, all powerful techniques known from continuous optimization can indeed be applied. Essentially there are two aspects that are important when designing a solution strategy for MINLPs based on the improvement of continuous relaxations with inequality constraints.

The first requirement can be stated as *convergence to an integral solution*, i.e. the series of continuous relaxations should converge to a solution in which all discrete variables receive valid (integer) values. This requirement is very natural since, obviously, if the series of continuous relaxations does not converge to a feasible point (if one exists), the overall approach would be incapable of solving the original MINLP. The second requirement ensures that the continuous relaxations do not overlook the best feasible solution and can be stated as *preservation of feasible solutions*. More precisely this requirement restricts the candidate set of addable inequality constraints to those that do not cut away feasible solutions of the original MINLP. Again this is a very intuitive requirement, since naively adding arbitrary inequality constraints will clearly change the feasible region in an arbitrary manner and thus in the end generate a solution which is not optimal w.r.t. the original MINLP. We will see that all general solution approaches, described below, are designed to respect these fundamental requirements.

In order to handle the continuous relaxations, an algorithm to solve NLPs is required. Since a detailed discussion of continuous optimization is out of the scope of this thesis,



(a) (b)

Figure 4.1.: Conceptual comparison of the optimization trajectories (shown in red) of active set and interior point methods for maximizing the y coordinate within the feasible region $\mathcal{F}$. While active set methods, like e.g. the simplex method, traverse the boundary until a locally optimal point is found (a), interior point methods traverse the inner part of $\mathcal{F}$ until they eventually converge to a locally optimal point at the boundary (b).

here we will introduce only some basic ideas while referring the interested reader to the excellent book [NW06]. In general there are two different classes of algorithms which were developed to solve NLPs, namely *active set* and *interior point* (also know as *barrier*) methods.

Active set methods: Roughly speaking, the main idea of active set approaches consists in iteratively refining the set of active constraints, i.e. the set of constraints that describe the boundary part of the feasible region where the optimal solution lies on. Typically the process is iterated until a locally optimal point is found. Notice that the well known *simplex algorithm* for solving *Linear Programs (LP)* belongs to the class of active set methods. The simplex algorithm traverses the corners of a polyhedron until the optimal solution is found. In this approach, the set of active constraints is given by those hyperplanes that define the current corner. Moving along an edge of the polyhedron from one corner to another one consequently is equivalent to the exchange of one hyperplane in the active set. A typical optimization trajectory of this approach is visualized in Figure 4.1a. Apart from the simplex algorithm several generalizations of this basic idea to handle arbitrary NLPs were developed.

Interior point methods: The main idea of interior point methods is completely different compared to that of active set methods. While active set methods traverse the boundary of the feasible region, interior point methods search a path through the interior of the feasible region (cf. Figure 4.1b). This is done by constructing an artificial, strictly convex objective function with an additional parameter, usually called μ, which controls a penalty function related to the boundary of the feasible region $\mathcal{F}$. This penalty function is constructed in such a way that for large μ the optimum will be close to the centroid of $\mathcal{F}$ while for decreasing μ it will converge to a point on the boundary that is locally optimal w.r.t. the original objective function. This simple basic idea is typically complemented with many sophisticated heuristics for determining good individual update steps in order to achieve fast convergence. A detailed description of a very efficient general interior point method is given in [WB06].

4.1. Branch-and-Bound

Branch-and-bound is a very general concept to tackle many different problems. The main idea of these methods is twofold. On the one hand a *divide-and-conquer* strategy is performed in order to iteratively partition the feasible region until the solution of the induced sub-problems can be found easily. This part is called *branching*. On the other hand, simultaneously to branching, conservative upper and lower bounds for the optimal solution are estimated such that hopefully many sub-problems can be neglected. This *bounding* part tries to reduce the exponential complexity of the branching part as much as possible by pruning away complete sub-trees. Clearly, the success of a branch-and-bound method strongly depends on the characteristics of a given problem and can be heavily influenced by adapting the branching as well as the bounding strategy. For different problem classes it turns out that completely different strategies lead to the most efficient technique. Here we will focus on branch-and-bound strategies for MINLPs only.

Given an MINLP P, a lower bound of the objective function at the optimum can be estimated by neglecting the integer conditions, i.e. by solving the corresponding relaxed R-MINLP Q. Now there are two different cases that might arise. Either the optimal solution of Q is integral such that $\tilde{\mathbf{y}} \in \mathbb{Z}^m$, or it is not which implies that at least one integer constraint $\tilde{y}_i \notin \mathbb{Z}$. In the first case, the algorithm trivially terminates, since there cannot be a better solution. Otherwise, the feasible region of the integer-constrained

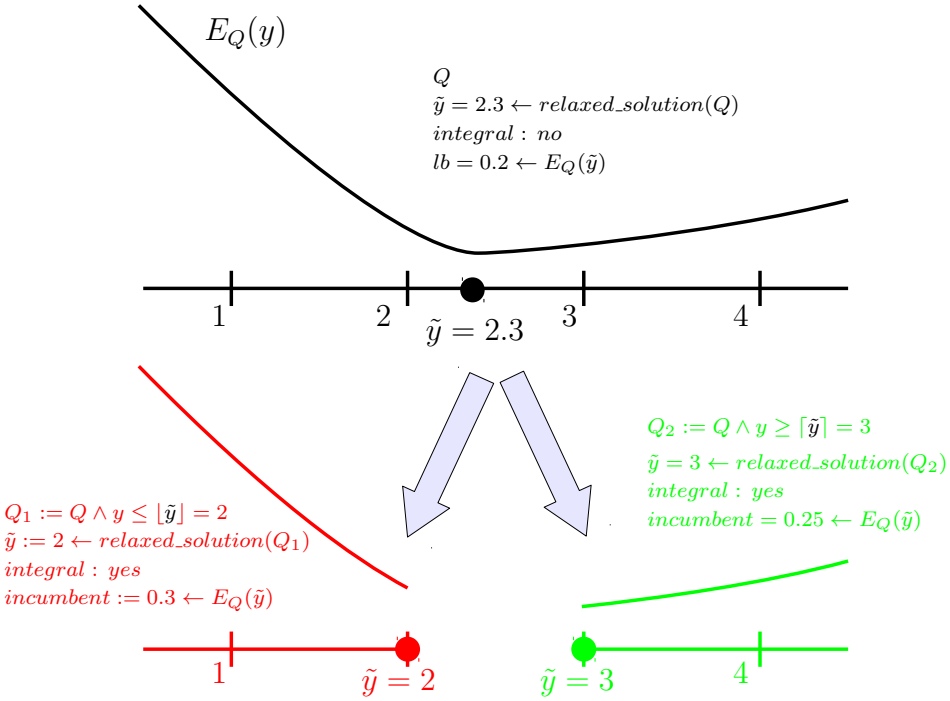


Figure 4.2.: Branch-and-bound on a one-dimensional integer program Q: The optimum of the relaxed problem is non-integral and thus triggers a branching step, which induces sub-problems Q_1 (red) and Q_2 (green). The relaxed solutions of both sub-problems are integral where Q_2 leads to the optimal solution $(3, 0.25)$ due to its lower objective value.

problem P is partitioned in such a way that the current continuous $\tilde{y}_i$ is no longer a valid solution. This can be achieved by replacing Q through two new problems Q_1 and Q_2, where in Q_1 we search for a solution with $y_i \leq \lfloor \tilde{y}_i \rfloor$ and analogously in Q_2 add the constraint $y_i \geq \lceil \tilde{y}_i \rceil$. Clearly no valid integer solution is lost, while the relaxed optimum of Q is excluded as desired. A simple example of this branching procedure is shown in Figure 4.2. Similarly to other divide-and-conquer algorithms this process is iterated for Q_1 and Q_2 until the best solution is found. All generated sub-problems are added into the set of yet unsolved problems $\mathcal{P}$. Every time an integral solution with $\mathbf{y} \in \mathbb{Z}^m$ is observed, the branching stops, which is called *fathoming* and the current solution is a candidate for the optimal solution $(\mathbf{x}^*, \mathbf{y}^*)$. The minimal objective value of all so far found valid solutions is called *incumbent*. All sub-problems that exhibit a lower bound (objective of the relaxed solution) that is larger than the incumbent cannot lead to an optimal solution and can thus be safely neglected. This bounding operation is sometimes

called *pruning*.

A high-level description of the above branch-and-bound algorithm for the optimization of convex MINLPs is given below:

Algorithm: *Branch-and-Bound*
Input: $P \in MINLP$
Output: optimal feasible solution $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{Z}^m$
or INFEASIBLE

```
01: incumbent ← ∞
02: P ← {relaxed(P)}
03: while P ≠ ∅    // unsolved sub-problems left?
04:     select Q ∈ P
05:     P ← P \ {Q}
06:     if is_feasible(Q) then
07:         (x̃, ỹ) ← relaxed_solution(Q)
08:         if E_Q(x̃, ỹ) ≤ incumbent then    // improvement w.r.t. current best solution?
09:             if ỹ ∈ ℤ^m then
10:                 incumbent ← E_Q(x̃, ỹ)    // tighten upper bound
11:                 (x*, y*) ← (x̃, ỹ)    // update current best solution
12:             else
13:                 select i with y_i ∉ ℤ    // select branching variable
14:                 P ← P ∪ {{Q ∧ y_i ≤ ⌊ỹ_i⌋}, {Q ∧ y_i ≥ ⌈ỹ_i⌉}}    // partition feasible region
15:             end if
16:         end if
17:     end if
18: end while
19: if incumbent ≤ ∞ then
20:     return (x*, y*)    // optimal solution found
21: else
22:     return INFEASIBLE    // no solution found
23: end if
```

This algorithm still exhibits some degrees of freedom that will be discussed in more detail below. More precisely, there are typically several candidates for the sub-problem

selection in line 04 as well as the branching-variable selection in line 13. However, before presenting the details of these decision steps, we first discuss an extension of the above algorithm which is designed to find approximate solutions in reduced time.

Approximate solutions: Instead of proceeding until the optimal solution is found, the algorithm could be stopped as soon as any feasible solution is found. One advantage of branch-and-bound methods consists in being able to provide a bound on the maximal deviation from the optimal solution. The minimum of lower bounds of all remaining sub-problems defines the current lower bound lb of the given instance. Note that the optimal objective function can never be below this value, since all lower bounds are estimated conservatively. Accordingly, the relative deviation is typically measured as

$$d_r = |incumbent - lb|/(|incumbent| + \epsilon) \ \in [0, 1]$$

Typically it is desirable to stop the computation if d_r falls below a small relative tolerance of, let's say, 5%. Even if the current solution is already optimal, it often takes many additional iterations and thus a very long time to "prove" that there exists no better solution. To get an idea of the impact of this aspect, note that it frequently happens that a very good solution can be found within seconds, while actually proving the optimality takes several hours.

Sub-problem selection strategies: The sub-problem selection strategy usually has a strong influence on the overall runtime. A straightforward greedy strategy consists in always selecting the sub-problem with the smallest lower bound which is called *best bound* strategy. While the overall runtime for proving optimality is typically quite good, the required time to find any feasible solution is often very high. The reason is that sub-problems with a less restricted feasible region usually enable a small objective function by violating many integrability constraints. Therefore, especially when searching for an approximate solution, a *depth-first* strategy might be favorable. In this strategy always the sub-problem which is deepest in the branching-tree, and thus typically is closest to a feasible integer solution, is selected. Sometimes it is worth spending even more time in the sub-problem selection to determine an estimate of the best feasible solution in a sub-problem. This strategy, called *best estimate*, can be very powerful if the given problem class offers a precise estimate.

Branching strategies: In each branching step one non-integral variable $y_i \notin \mathbb{Z}$ is selected for partitioning of the feasible region. Similarly to sub-problem selection, variable selection has a strong influence on the characteristics of the solution process. Here typical choices include *minimum infeasibility*, *maximum infeasibility* and *pseudo-costs* strategies. Minimum infeasibility selects the variable which is already closest to an integer, i.e. , it minimizes $min(|y_i - \lfloor y_I \rfloor|, |y_i - \lceil y_I \rceil|)$. With this strategy typically good feasible solutions are found in an early stage of the algorithm. On the contrary it turns out that choosing the variable which is farthest from an integer, and thus maximizes the former expression, often reduces the runtime to find solutions close to the global optimum. The reason is that the most critical decisions are unrolled first, before getting lost in a sub-optimal sub-tree. Both former strategies can also be combined by first performing some steps with maximum infeasibility strategy, followed by decisions based on minimum infeasibility. Finally, the branching strategy can be based on pseudo-costs, i.e. , on estimating the cost (change in objective function) for rounding the variable to a feasible solution. Again all three possibilities, namely minimal costs, maximal costs or a mix of both, can be advantageous, depending on the problem characteristics.

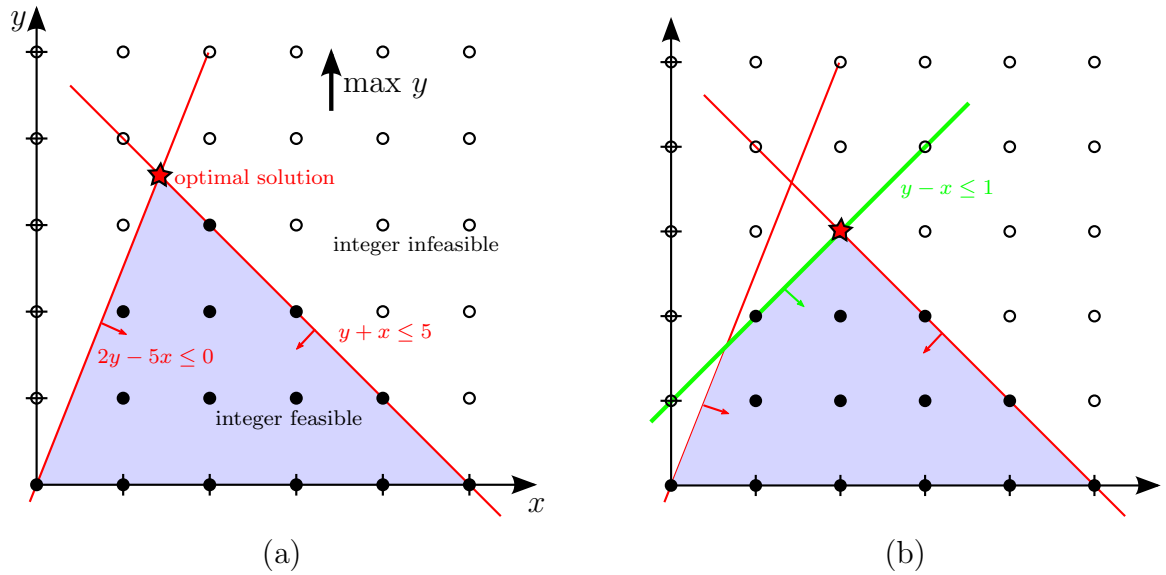


Figure 4.3.: Cutting-plane approach: (a) The optimum of the relaxed linear program (red star) is typically not integer feasible. (b) By adding a new hyperplane (green line), that cuts off the continuous optimum while maintaining all integer feasible points (black dots) the solution of the relaxed problem converges to the integer feasible optimum.

4.2. Cutting-Plane method

The second very prominent solution strategy for mixed-integer problems is the so called *cutting-plane approach*. It shares the idea of solving a series of continuously relaxed problems with the previously discussed branch-and-bound approach. However, instead of generating a tree of sub-problems which partition the feasible region, the cutting-plane approach refines a single so called *master problem* until an integral feasible solution is found. The solution process starts with the relaxation of the original problem, i.e. $Q_0 \leftarrow \text{relaxed}(P)$, and iteratively adds inequality constraints, $Q_{i+1} \leftarrow Q_i \wedge Cut_{new} \leq 0$, until an integral feasible solution is found as shown below:

Algorithm: *Cutting-Plane approach*
Input: $P \in MILP$
Output: optimal feasible solution $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{Z}^m$
or INFEASIBLE

01: $i \leftarrow 0$ *// iteration counter*
02: $Q_0 \leftarrow \text{relaxed}(P)$ *// initialization*
03: **repeat**
04: **if** NOT is_feasible(Q_i) **then**
05: return INFEASIBLE
06: **else**
07: $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \leftarrow \text{relaxed\_solution}(Q_i)$
08: **if** $\tilde{\mathbf{y}} \notin \mathbb{Z}^m$ **then**
09: $Q_{i+1} \leftarrow Q_i \wedge Cut_{new} \leq 0$ *// add cutting plane*
10: $i \leftarrow i + 1$ *// increase iteration counter*
11: **end if**
12: **end if**
13: **until** $\tilde{\mathbf{y}} \in \mathbb{Z}^m$ *// optimal solution found*

The name of the method stems from the fact that the inequality constraints are called *cuts*. For the sake of simplicity in this chapter we will restrict to the case of linear integer and mixed-integer programs and postpone the discussion of extensions to the nonlinear case to later on.

Originally this approach was indeed developed for *Integer-Linear Programs* (ILPs), where the feasible region of the relaxed problem is a polyhedron. From this linear point of view, adding a new valid cut is equivalent to cutting away a part of the feasible region which does not contain any integer-feasible point (cf. Figure 4.3). If this process converges to an integer-feasible point this is obviously the global optimum, since valid cuts always preserve the feasible region of the ILP. The cuts are typically constructed in such a way that especially the current non-integral optimal solution of the relaxed problem Q_i is cut off in Q_{i+1}. Consequently, the optimal solution of the relaxed problem changes in each step and progress of the algorithm is guaranteed. Since the relaxed problem is more and more restricted, the objective function, i.e. the current lower bound, monotonically increases. However, it should be mentioned that the number of required iterations till convergence can get enormous.

The most important ingredient of a cutting-plane approach is the generation of "strong" valid cuts, i.e. inequalities that push the solution of the relaxed problem as much as possible towards the optimal integer-feasible solution without removing it from the feasible set. Again there is no general approach which is optimal for all problem instances. Accordingly, many different cut generation heuristics, leading to different classes of cuts, were developed in the past. State-of-the-art algorithms often apply a greedy strategy to select from all available cuts. Additionally, the cut selection strategy can typically be tuned by means of several parameters, in order to optimize the performance w.r.t. the intended class of problems. In the following, we will discuss the very general class of *Gomory cuts* (GC), which is the basis for many other classes of cuts.

Gomory cuts: The main idea of Gomory cuts is straightforward and builds on the idea of integer rounding. We start with the following ILP, given in normal form:

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{y} \\ \text{subject to} \quad & A\mathbf{y} \leq \mathbf{b} \\ & \mathbf{y} \in \mathbb{Z}_+^m \\ & \mathbf{c} \in \mathbb{R}^m \\ & A \in \mathbb{R}^{n \times m} \\ & \mathbf{b} \in \mathbb{R}^n \end{aligned} \tag{ILP}$$

First notice that each linear combination of the above inequalities with positive coefficients, i.e. $\alpha^T A\mathbf{y} \leq \alpha^T\mathbf{b}$ with $\alpha \in \mathbb{R}_+^n$, is a valid inequality for the ILP as well. For

example, if $y_1 - y_2 \leq 2$ and $y_3 \leq 4$, then of course $y_1 - y_2 + y_3 \leq 6$. Now, since $y_i \geq 0$, the value of the left-hand side of the equation can be further reduced by rounding down the coefficients leading to $\sum_{i=1}^{m} \lfloor \alpha^T A_i \rfloor y_i \leq \alpha^T \mathbf{b}$. After this operation the left-hand side is always an integer such that the fractional part of the right-hand side is meaningless and can be rounded off as well. Altogether we end up with the following Gomory cut: $\sum_{i=1}^{m} \lfloor \alpha^T A_i \rfloor y_i \leq \lfloor \alpha^T \mathbf{b} \rfloor$. It can be shown that this simple procedure is powerful enough to generate **all** valid inequalities for an ILP [Chv73]. Since this class of Gomory cuts is huge - every linear combination with positive coefficients generates a Gomory cut - the next question consists in, how can a "good" cut from this class be found efficiently? In practice, Gomory cuts are mostly applied in combination with the Simplex algorithm. It turns out that in this setting the generation of well behaved Gomory cuts is very cheap, since adequate linear combinations of the constraints can be taken from the usual Simplex tableau without any modification. These linear combinations are guaranteed to cut off the current non-integral solution.

The presented form of Gomory cuts is only applicable in case of pure integer problems, however, the described concept can be generalized to *mixed-integer Gomory cuts* (MIGC). More details on Gomory cuts, as well as mixed-integer Gomory cuts and many other cuts can be found in [MMWW02, Rus06].

Extension to nonlinear problems: The above cutting-plane approach is applicable in the case of linear programs only. However, exploiting the insight that every **convex** function can be approximated arbitrarily accurate by local linearizations, i.e. , a set of tangential hyperplanes, the above algorithm can be easily generalized to nonlinear convex programming. The idea of iteratively refined linearizations works for the nonlinear convex objective as well as for the convex feasible region described by the nonlinear convex constraints as illustrated in Figure 4.4. The resulting algorithms building on top of this idea, known as *Extended Cutting-Plane Method* [WP94] and *Outer-Approximation* [DG86], alternatingly solve the linearized mixed-integer problem (the approximation resulting from the current set of cuts) and refine the linearization (add further cuts to improve the approximation) until an approximate solution within a given tolerance to the optimal one is found. Solutions to the linearized problem could e.g. be found by the above cutting-plane method for mixed-integer linear programs.

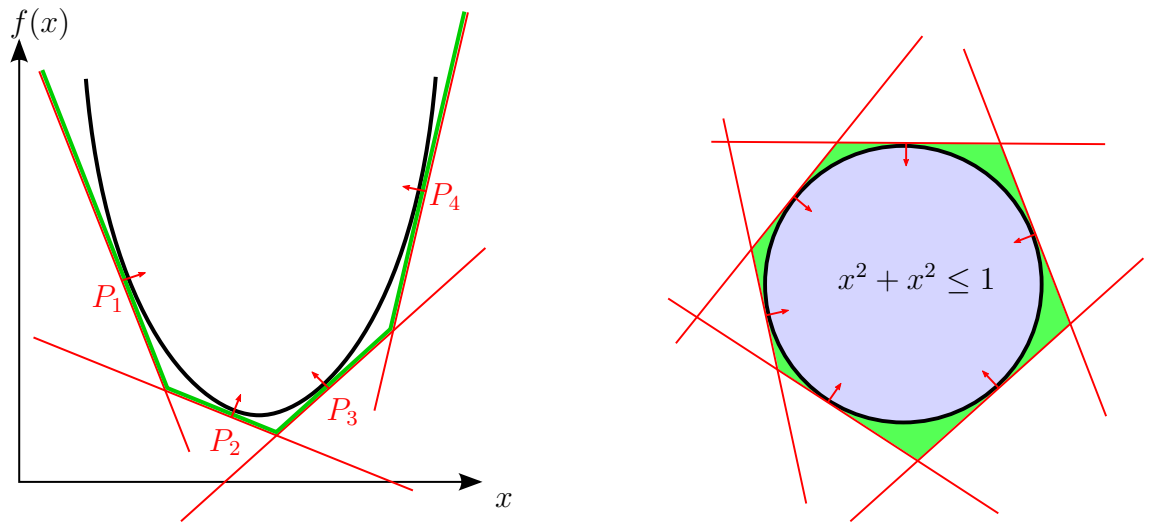


Figure 4.4.: Linear approximation of convex nonlinear objective functions and feasible regions: (a) A convex nonlinear objective function can be arbitrarily accurate approximated by a set of tangential hyperplanes P_i. Thus the minimization of $f(x)$ can be replaced by the minimization of the linear function y subject to linear constraints $dist(P_i, (x, y)) \geq 0 \ \forall i$ which induce the green contour. (b) The same is possible for nonlinear convex feasible regions. The difference between the unit circle and its linearization is shown in green. This deviation can be arbitrarily reduced by adding more hyperplanes.

4.3. Branch-and-Cut

As we have seen, both approaches, namely branch-and-bound and cutting-plane, can be used to solve mixed-integer problems. However, practical experiments have shown that the runtime of both approaches can be unsatisfactorily high. The reason seems to be that both algorithms are driven by heuristics that might fail in their assumptions. On the one hand, the branch-and-bound method suffers from an exponential growth in sub-problems and especially in cases where the branching and sub-problem selection heuristics do not perform well, many of these sub-problems have to be explored until an adequate solution is found. On the other hand, the performance of the cutting-plane approach is very dependent on the heuristic that generates new cuts. If in each step only a small piece of the polyhedron is cut off, then the overall approach might take very

long until convergence to an integer-feasible point.

The branch-and-cut approach is based on the idea of combining branch-and-bound and the cutting-plane method into one algorithm that dynamically switches between both optimization strategies. This is done by embedding a slightly modified cutting-plane method into the branch-and-bound algorithm. More precisely, after each sub-problem selection of the branch-and-bound algorithm, cutting-planes are added as long as the relaxed solution can be improved significantly. Due to this stop criterion, the cutting-plane phase switches back to branching as soon as adding more cutting planes becomes inefficient. The resulting branch-and-cut approach is given below, where the modifications (lines 06-08) are highlighted in red.

Algorithm: *Branch-and-Cut*
Input: $P \in MINLP$
Output: optimal feasible solution $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^n \times \mathbb{Z}^m$
or INFEASIBLE

01: $incumbent \leftarrow \infty$
02: $\mathcal{P} \leftarrow \{\text{relaxed}(P)\}$
03: **while** $\mathcal{P} \neq \emptyset$ *// unsolved sub-problems left?*
04: select $Q \in \mathcal{P}$
05: $\mathcal{P} \leftarrow \mathcal{P} \setminus \{Q\}$
06: **while** relaxed_solution(Q) can be significantly improved
07: $Q \leftarrow Q \wedge Cut_{new} \leq 0$ *// add cutting plane*
08: **end while**
09: **if** is_feasible(Q) **then**
10: $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \leftarrow$ relaxed_solution(Q)
11: **if** $E_Q(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \leq incumbent$ **then** *// improvement w.r.t. current best solution?*
12: **if** $\tilde{\mathbf{y}} \in \mathbb{Z}^m$ **then**
13: $incumbent \leftarrow E_Q(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ *// tighten upper bound*
14: $(\mathbf{x}^*, \mathbf{y}^*) \leftarrow (\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ *// update current best solution*
15: **else**
16: select i with $y_i \notin \mathbb{Z}$ *// select branching variable*
17: $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{Q \wedge y_i \leq \lfloor \tilde{y}_i \rfloor\}, \{Q \wedge y_i \geq \lceil \tilde{y}_i \rceil\}\}$ *// partition feasible region*
18: **end if**

```
    end if
  end if
end while
if incumbent ≤ ∞ then
  return (x*, y*)   // optimal solution found
else
  return INFEASIBLE   // no solution found
end if
```

Today branch-and-cut algorithms belong to the most powerful general MI optimization strategies. Especially for the case of MILP and MIQP very powerful commercial solvers, as for instance [GRB11, IBM12], are available that complement the above base-algorithm by several other heuristics which increase the efficiency in practice. However, it is very important to notice that the solution of mixed-integer problems is inherently more complicated then its continuous counterparts. As a consequence, it cannot be expected that these libraries, although highly optimized, always perform good when applied with default parameters. It is indispensable to keep in mind that the efficient solution of mixed-integer problems usually requires some knowledge of the user in order to tune the parameters for the desired class of problems. A behavior, like in the case of sparse linear system solvers, where default parameters always perform well and the solution time is roughly a function of the number of non-zeros, can never be expected for mixed-integer solvers.

5. Efficient Approximation of Quadratic MI-Problems

In the previous chapters about mixed-integer problems, the general problem setting and some general solution strategies were discussed. It turns out, that although highly optimized, these solution strategies are still too time consuming for our overall goal of quadmesh generation due to problem sizes typically consisting of thousands or even millions of (unbounded) discrete as well as continuous variables. The general parametrization based quadmesh generation problem, as it will be introduced in Chapter 6, induces a non-convex MINLP. Direct solution of this formulation seems to be intractable, even for very small problems. Therefore, in Chapters 7 and 8 we will develop two simplified formulations, which under some reasonable assumptions convert the non-convex MINLP into a convex MIQP. Furthermore, the derived formulations are free of inequality constraints and solely contain linear equality constraints. However, in case of typical problem sizes, these strongly simplified formulations still cannot be solved efficiently with available commercial solvers. Experiments showing the performance of commercial solvers in comparison with our method can be found in Section 5.3. The reason of the bad performance of general solution approaches is that these solvers do not fully exploit the specific characteristics inherent to our class of problems. Furthermore, instead of spending much time in searching for the optimal solution, we design an algorithm which greedily searches for a good approximate solution and thus is often orders of magnitude faster.

In this chapter, which is based on [BZK12], we present an algorithm for efficiently and accurately approximating *quadratic* MIPs represented by quadratic energy functions

$$E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^t A\mathbf{x} - \mathbf{x}^t\mathbf{b} \to min, \quad \mathbf{x} \in \mathbb{R}^n \tag{5.1}$$

with A symmetric and positive definite, subject to n_I integer constraints

$$x_{i \in I} \in \mathbb{Z}, \quad I \subseteq \{1, \ldots, n\}. \tag{5.2}$$

Notice that in order to achieve a simpler presentation, the notation convention is slightly changed compared to the previous chapters. Additionally the feasibility of the solution $\mathbf{x}$ is restricted by n_C linear *equality* constraints of the form

$$\mathbf{C}_i \cdot \mathbf{x} = d_i \qquad \text{with} \qquad \mathbf{C}_i \in \mathbb{R}^n,\ d_i \in \mathbb{R} \tag{5.3}$$

which can be assembled into a single matrix $C\mathbf{x} = \mathbf{d}$ with dimension $C \in \mathbb{R}^{n_C \times n}$. Here n, n_I and n_C denote the number of variables, integer constraints and linear constraints respectively. Note also that the above formulation differs slightly from the most general setting of mixed-integer problems where additionally *in-equality* constraints are given (cf. Chapters 3 and 4).

Our algorithm successively determines the values of the discrete variables $x_{i \in I} \in \mathbb{Z}$ in a greedy fashion. Fixing the value of a discrete variable is equivalent to adding one explicit linear constraint $x_i = k$ with $k \in \mathbb{Z}$ to our optimization problem. Therefore our algorithm successively transforms integer constraints into explicit linear constraints until all of them are fulfilled. More precisely we start by neglecting the n_I integer constraints and compute the minimizer of the relaxed problem by setting the partial derivatives $\frac{\partial E}{\partial x_i} = 0$ and solving the resulting linear system

$$A\mathbf{x}^0 = \mathbf{b}. \tag{5.4}$$

In this simple example we assume $n_C = 0$ to increase clarity. The values of the solution vector $\mathbf{x}^0$ can be seen as continuous estimates of the desired discrete integer variables. However, we found that estimating all integer constraints at once, i.e., requiring $\forall i \in I :\ x_i^1 = round(x_i^0)$, leads to poor results since the individual estimates cannot influence each other. Motivated by this observation we instead successively determine single integer constraints $x_j^{k+1} = round(x_j^k)$ which are henceforth used to solve subsequent relaxed problems until a feasible solution of our initial optimization problem is found, meaning that all $x_{i \in I}$ are integers.

By greedily choosing the continuous estimate which has the smallest deviation from an integer, i.e. $|round(x_j^k) - x_j^k|$, in each step, these subsequent relaxed problems can be

solved very efficiently by a carefully designed three-level solver as presented in Section 5.2.2. The performance can further be improved by identifying sets of relaxed variables which do not interfere too much and hence can be safely estimated simultaneously within the same iteration.

In order to facilitate an efficient handling of arbitrary linear constraints C_i we propose to eliminate one variable for each constraint (Section 5.1) and support this approach by a fill-in reducing constraint reordering (Section 5.1.2) which in practice significantly reduces the runtime.

In Section 5.3 the capabilities of the presented solver are illustrated exemplary by applying it to the surface quadmesh generation problem. We illustrate the immense performance benefit due to the novel extensions of the previously applied variant from [BZK09], i.e., the rounding of sets of variables and the fill-in reducing reordering of constraints.

Previous Work To the best of our knowledge the idea of approximating MIPs by a series of real-valued problems started with [Rin88], set in the field of Structural Engineering. Ringertz' idea of rounding variables iteratively and re-solving the problem has been cited several times and depending on the problem setting small variations appear in the proposed solutions. While some researchers argue for the use of post-processing methods as, depending on the problem and the type of variables, always rounding up (or down) might not be meaningful [YZJ03], others suggest rounding both up and down and keeping the solution with lower cost [GSS96].

Regardless of the rounding strategy, what these approaches all have in common is that the full-sized system of linear equations needs to be re-solved in each iteration, making the iterative strategy infeasible for many practical applications.

In the field of Geometry Processing the idea of approximating quadratic MIPs by rounding variables of a real-valued linear system has been successfully adapted by several authors (see e.g.,[KNP07, RVLL08]). Here direct-rounding strategies were used, where the system had to be solved only twice, once initially and once after all integer constraints have been estimated (all at once). This approach is usually only applicable for MIPs with a small number of integer variables that do not interfere too much and

otherwise leads to a poor approximation of the optimal solution.

This chapter is structured as follows: Section 5.1 describes the proper handling of linear constraints within the optimization of a quadratic energy, which is central also for the integer constraints discussed in Section 5.2. In Section 5.2.2 we describe an efficient update strategy which enables the iterative addition of integer constraints. Finally in Section 5.3 we discuss some experiments performed within the context of quadrilateral surface remeshing.

5.1. Linear Constraints

The ability to properly handle constraints is vital for the wide applicability of an optimization method. For a problem to be solvable, usually some boundary constraints are needed to restrict the solution space, or often additional user-defined design constraints might be incorporated to shape the resulting solution. In our setting we also have to deal with integer constraints which translate into simple linear conditions as soon as the specific integer is known. A common way to handle linear constraints are *Lagrangian Multipliers* as discussed next.

5.1.1. Lagrangian Multipliers

The method of Lagrangian Multipliers turns a constrained problem into an unconstrained one by adding one additional term per constraint to the energy. Updating energy (5.1) with the constraints (5.3) we end up with the following new energy:

$$E_L(\mathbf{x}) = E(\mathbf{x}) + \sum_{i=1}^{n_C} \lambda_i (C_i \cdot \mathbf{x} - d_i) \tag{5.5}$$

where the optimum is characterized by the following system of linear equations

$$\frac{\partial E_L}{\partial \mathbf{x}} = 0 \quad \rightsquigarrow \quad \begin{bmatrix} A & C^T \\ C & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{d} \end{bmatrix} \tag{5.6}$$

describing the stationary point of the adapted energy. Note that the approach of Lagrangian Multipliers is not restricted to quadratic energies nor linear constraints but can be applied to non-linear problems as well, for more details see e.g., [NW99]. Unfortunately the approach of Lagrangian Multipliers comes with certain disadvantages making

them impractical for our purpose. Instead of decreasing the number of degrees of freedom, as more constraints are added, the opposite is the case since for each constraint a Lagrangian Multiplier λ_i is introduced. Furthermore the symmetric positive definiteness (s.p.d.), inherent in linear systems arising from convex quadratic energies is destroyed by the diagonal block of zeros $\mathbf{0}$, effectively disabling the use of highly efficient solvers such as CHOLMOD [CDHR06] and necessitating the use of slower more general solvers such as SuperLU [DEG*99]. Moreover, as will be seen in Section 5.2 the s.p.d. property is crucial for the efficient local updates of the adaptive three-level solver in Section 5.2.2. Therefore next we describe a proper handling of linear constraints which maintains the s.p.d. property.

5.1.2. Elimination Approach

Assume we want to minimize a quadratic energy $E(\mathbf{x})$ with $\mathbf{x} \in \mathbb{R}^n$ subject to a single linear constraint $\mathbf{D}^T\mathbf{x} - d = 0$. Geometrically this means restricting the solution space to a $n-1$ dimensional hyperplane. Consequently, it is possible to convert the above problem into a new unconstrained one with $n-1$ degrees of freedom. Assume w.l.o.g. that $D_n \neq 0$ such that we can solve the linear constraint for x_n expressing it as a function of $(x_1, \ldots, x_{n-1})$

$$x_n(\underbrace{x_1, \ldots, x_{n-1}}_{\tilde{\mathbf{x}}}) = (d/D_n) - \sum_{j=1}^{n-1}(D_j/D_n)x_j =: f - \mathbf{F}^T\mathbf{x} \tag{5.7}$$

and transforming the above constrained problem into the desired unconstrained form

$$\tilde{E}(\tilde{\mathbf{x}}) := E\underbrace{\begin{pmatrix} \tilde{\mathbf{x}} \\ x_n(\tilde{\mathbf{x}}) \end{pmatrix}}_{\mathbf{y}} \tag{5.8}$$

with equivalent minima where x_n can be computed from $\tilde{\mathbf{x}}$ by equation (5.7).

To compute the minimizer $\tilde{\mathbf{x}}$ we now have to solve a $(n-1)$ dimensional system of linear equations $\tilde{A}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$ which can be derived by partitioning the matrix A of equation

(5.1) into four blocks (with $\overline{A} \in \mathbb{R}^{(n-1)\times(n-1)}$, $\mathbf{v} \in \mathbb{R}^{n-1}$ and $w \in \mathbb{R}$) and re-factorizing the result:

$$\begin{aligned} \tilde{E}(\tilde{\mathbf{x}}) &= \frac{1}{2}\mathbf{y}^T A\mathbf{y} - \mathbf{y}^T\mathbf{b} = \frac{1}{2}\mathbf{y}^T \begin{pmatrix} \overline{A} & \mathbf{v} \\ \mathbf{v}^T & w \end{pmatrix} \mathbf{y} - \mathbf{y}^T \begin{pmatrix} \overline{\mathbf{b}} \\ b_n \end{pmatrix} \\ &= \frac{1}{2}\tilde{\mathbf{x}}^T \underbrace{\left(\overline{A} - \mathbf{v}\mathbf{F}^T - \mathbf{F}\mathbf{v}^T + w\mathbf{F}\mathbf{F}^T\right)}_{\tilde{A} \in \mathbb{R}^{(n-1)\times(n-1)}} \tilde{\mathbf{x}} - \tilde{\mathbf{x}}^T \underbrace{\left(\overline{\mathbf{b}} + \mathbf{F}(fw - b_n) - \mathbf{v}f\right)}_{\tilde{\mathbf{b}} \in \mathbb{R}^{n-1}} + \text{const} \end{aligned} \tag{5.9}$$

Note that since A is s.p.d. $\tilde{A}$ is also s.p.d. enabling highly efficient solution methods used in our three-level solver described in Section 5.2.2. Of course instead of eliminating the last variable each other variable can be chosen by re-indexing.

Multiple Constraints: In general we want to handle an arbitrary number of linear constraints which can be achieved by iteratively eliminating one variable for each constraint from $\{C_1, \ldots, C_{n_C}\}$. One very important aspect of multiple constraints is that in each step it is necessary to eliminate the chosen variable from all subsequent constraints since obviously once a variable is constrained and eliminated from the optimization problem it should not be reintroduced by a subsequent constraint. More precisely, after constraining a variable x_k through a constraint C_j we have to do Gaussian elimination in the constraint matrix C in order to bring all $C_{i,k}$ with $i > j$ to zero. Clearly the constraints in the updated matrix are equivalent to the original problem.

Choosing elimination variables: For each linear constraint we have to pick a variable which is subsequently constrained by the induced linear function and eliminated from the problem. All non-zero coefficients of the linear constraint induce a valid possibility. To increase numerical stability we select the variable whose coefficient has the largest absolute value. However, there is one important aspect to consider whenever a variable x_k with $k \in I$, i.e., which has to satisfy an integer constraint, is selected for elimination. In general it can get very problematic to guarantee that the values of the induced linear function are chosen in such a way that x_k becomes an integer. Consequently for the elimination we always prefer non-integer variables with $k \notin I$. For constraints where all non-zero coefficients belong to integer variables we currently support only those cases where all coefficients are integer and their greatest common divisor (gdc) is one of these coefficients. In such a case we can safely divide all coefficients by their gdc and eliminate a variable with coefficient ± 1 since a linear combination of integers multiplied

by integers is always an integer as well and consequently the integer constraint is fulfilled by construction. For many practical problems (like quadrilateral surface remeshing) the above assumptions are always fulfilled and therefore we leave the more complicated general case for future work. Whenever the above assumption is violated it may happen that in the result some of the integer constraints will not be satisfied in the end.

Linear dependent or conflicting constraints: Since we iteratively process the individual constraints it is easy to identify linear dependent or conflicting constraints. This is a big advantage compared to the method of Lagrangian Multipliers which would construct an underdetermined system of linear equations not suitable for efficient standard solvers. In our implementation linear dependent or conflicting constraints are simply neglected. This behavior is very convenient since the user does not have to spend additional effort identifying the subset of linear independent constraints e.g., in the case of user provided side conditions. Due to numerical inaccuracies of floating point numbers linear dependency is checked against a tolerance with a default value of 10^{-6}.

Fill-in reducing constraint reordering: Although mathematically equivalent, the linear system belonging to the unconstrained optimization problem after processing all constraints can take many different patterns, strongly depending on the processing sequence of the constraints. In spirit of sparse Cholesky methods like [CDHR06] we are interested in finding an ordering of the constraints which minimizes the fill-in (nonzero elements) and hence increases performance. Unfortunately there is no known algorithm to achieve the best ordering apart from the naive one which explicitly checks all orderings. Obviously such an approach is far too slow such that a good compromise in form of a cheap heuristic is more desirable. Our experiments show that processing the constraints sorted by their number of non-zero coefficients leads to much higher performance than just using a random ordering (see Section 5.3 for timings). Please note that this ordering is dynamic since while processing the constraints, their number of non-zero coefficients is altered by the Gaussian elimination steps.

After eliminating one variable per linear constraint we obtain a new (unconstrained) equivalent optimization problem, i.e., a quadratic energy minimization subject to a set of integer constraints. Next, we describe how a good approximate solution can be found efficiently.

5.2. Integer Constraints

The integer constraints of our initial problem dictate that for each feasible solution a subset of the variables have to be integers, i.e., $x_{i\in I} \in \mathbb{Z}$. Finding a feasible solution is simple in this formulation, since there are no dependencies between the individual variables. Therefore just setting up a set of additional linear constraints which fix the $x_{i\in I}$ variables to arbitrary integers like e.g., $x_{i\in I} = 0$ and enforcing them with the method from the previous section would indeed result in a feasible solution. However the problem of finding the best one of all these possible assignments, i.e., the one which minimizes the energy, is very hard. In contrast to continuous convex optimization it is not sufficient to simply walk into the direction of the negative gradient (see Figure 3.1). As we have seen in Chapters 3 and 4, typically much more expensive optimization strategies like branch-and-cut have to be applied in order to find the optimal solution.

5.2.1. Direct Rounding

Instead of achieving optimality, for practical problems we aim at finding an approximate solution which is close enough to the optimum but can be computed in a fraction of time. The most efficient way to determine adequate assignments for the integer variables is to estimate them from a relaxed solution, i.e., computing the minimizer $\mathbf{x}^c$ where all variables are allowed to be continuous leading to the estimates $x_{i\in I} = round(x_i^c)$. Following (5.9) the elimination approach results in a very simple update for such explicit constraints:

$$\tilde{A} = \overline{A} \quad \text{and} \quad \tilde{\mathbf{b}} = \overline{\mathbf{b}} - \mathbf{v} \cdot round(x_i^c) \tag{5.10}$$

Estimating all integer assignments at once which is called *direct rounding* is very efficient since it requires the solution of only two linear systems. However the drawback is that the interrelation between the discrete variables is completely ignored which often leads to poor results (cf. Figure 5.2). This suggests to successively add one integer constraint at a time and immediately compute the altered relaxed problem to update the estimates of the yet unconstrained discrete variables. This strategy is denoted *iterative rounding* and is discussed in more detail next.

5.2.2. Iterative Greedy Rounding

The key to an efficient implementation of the iterative rounding is the observation that, for problems with sparse variable dependencies (few non-zeros per row), changing the value of one variable usually has little influence on "far-away" variables. This is a property inherent in many Geometry Processing problems formulated over, e.g., simplicial complexes or spline bases with local support.

The problem inherent to iterative rounding is that it requires the solution of $|I| + 1$ many linear systems which can get very slow when implemented in a naïve way. Fortunately in many steps of this iterative process the solution changes only slightly which can be exploited by carefully designed iterative solvers.

Suppose that we have computed the solution of the relaxed problem $A\mathbf{x} = \mathbf{b}$ and that we want to add a single integer constraint. Following (5.10) the residual $\mathbf{e} = \tilde{A}\tilde{\mathbf{x}} - \tilde{\mathbf{b}}$ after adding the new constraint has the same nonzero pattern as $\mathbf{v}$. Consequently, for a sparse $\mathbf{v}$ the relaxed solution from the previous step $\tilde{\mathbf{x}}$ violates only a few equations of the linear system. Due to this observation we first try to iteratively update the solution only where it is necessary, i.e., for all variables $\tilde{x}_i$ with $|e_i| > \epsilon$. This so called *Local Gauss-Seidel* method executes single Gauss-Seidel updates for variables with a local residual above the allowed tolerance. All these candidates are stored in a queue and convergence is reached when the queue gets empty meaning that all residuals are below the prescribed tolerance. Notice that due to the elimination approach the system matrix remains s.p.d. guaranteeing convergence of the Gauss-Seidel method. The complete algorithm is depicted below:

Algorithm: *Local Gauss-Seidel*

Input: Linear system $A\mathbf{x} = \mathbf{b}$ (which is not fulfilled)
Index set of variables with non-zero residual N,
End conditions ϵ and $maxiters_{GS}$

Output: Updated $\mathbf{x}$ with residuals $|e_k| < \epsilon$ or NOT converged.

01: push N onto *queue*
02: $iter = 0$
03: **while** *queue* not empty **and** $iter < maxiters_{GS}$
04: iter = iter +1

05: $x_k = \text{pop}(\ queue\ )$
06: $e_k = b_k - \sum_{j=1}^{n} A_{kj} x_j$
07: **if** $|e_k| > \epsilon$ **then**
08: $x_k \leftarrow x_k + e_k / A_{kk}$
09: push $nonzero(A_{k*})$ onto $queue$
07: **end if**
10: **end while**

The parameters ϵ and $maxiters_{GS}$ are specified by the user. In cases where the above method does not converge within the prescribed number of iterations, a more global conjugate gradient method is used and in rare cases where this is still not sufficient after a few iterations a sparse Cholesky method is executed. This adaptive solution strategy is very fast if the previous solution is close to the new one and only spends more time if a novel integer constraint has global impact. In our implementation the conjugate gradient solver is taken from the GMM++ library [Ren03] and the Sparse Cholesky solver is the CHOLMOD solver [CDHR06].

In this iterative rounding strategy we can choose $|I|!$ many different orders in which the integers are estimated. A natural greedy choice is the yet unconstrained integer variable whose estimate has the smallest deviation $|x_i - round(x_i)|$ from an integer since it is most likely to be correct. A nice side effect of this strategy is that it increases the efficiency of the above hierarchical solution strategy. The reason is that for small deviations from an integer also the non-zero residuals usually get small. The complete iterative greedy rounding algorithm is shown below:

Algorithm: *Iterative Greedy Rounding*
Input: Linear system of relaxed problem $A\mathbf{x}^c = \mathbf{b}$ with $\mathbf{x}^c, \mathbf{b} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$
index set of integer variables $I \subset \{1, \ldots, n\}$
Output: Approximation of mixed-integer solution $\mathbf{x} \in \mathbb{R}^n$ satisfying $x_{i \in I} \in \mathbb{Z}$

01: $\mathbf{x} = \mathbf{x}^c$
02: **while** $I \neq \emptyset$
03: *// greedy selection*
04: $j = \arg\min_{i \in I}(|x_i - round(x_i)|)$

```
05:     I ← I \ j
06:     // add new constraint and get nonzero residuals N
07:     N = eliminateConstraint( x_j = round(x_j) , A , x , b )
08:     // update solution
09:     converged = localGaussSeidel( A, x, b, N) // level 1
10:     if not converged then
11:       converged = conjugateGradient( A, x, b) // level 2
14:       if not converged then
15:         sparseCholesky( A, x, b) // level 3
16:       end if
17:     end if
18: end while
```

To avoid the necessary re-indexing of the variables in the above algorithm the update rule (5.10) was slightly modified by keeping an identity row and column for each eliminated variable x_k, i.e., $\tilde{A}_{kj} = \tilde{A}_{jk} = \delta_{kj} \quad \forall j$.

In our implementation the user is able to control the behavior of the adaptive three level solver with several parameters. First of all the tolerance ϵ for checking convergence of the iterative methods (level 1 and 2) and a maximum number of iterations $maxiters_{GS}$ and $maxiters_{CG}$ can be adjusted. Furthermore it is possible to disable complete levels. The reason is that for mixed-integer problems where it is known that the rounding of a discrete variable always has global impact it is, e.g., not reasonable to execute the Local Gauss-Seidel step since it would almost never converge. Therefore it is very important to carefully adapt these parameters in order to optimize the performance for a specific class of problems. In Section 5.3 we will provide two different useful settings for the quadrangulation problem.

Simultaneous Rounding: The motivation for the iterative rounding strategy was mainly the observation that the estimates of individual integer variables should influence each other to achieve satisfactory accuracy. It would be possible to achieve the same accuracy in fewer computation steps if some prior knowledge about the rate of influence between variables is available. Clearly variables which do not influence each other could be rounded simultaneously in one step without introducing an error. Unfortunately

computing the influence between variables corresponds to the solution of a full-sized linear system which would be too expensive. What we need instead is a fast-to-compute a priori estimate which never underestimates the interdependency. A very simple a priori estimate which holds for many problems is the following one: If one variable is changed by a value of Δx due to a constraint, all other variables are changed by a value smaller or equal than Δx. Consequently in each step several variables can be rounded as long as their estimated maximal deviation $\sum_i \Delta x_i$ does not influence any of the rounding decisions. Obviously this a priori estimate does not hold for all problems. However we included the possibility to use it into our implementation since it is useful for many practical applications and can speed up the computation significantly. Finding a cheap way for estimating sharper bounds for the interdependency between discrete variables is an interesting question for future work.

5.3. Evaluation

We evaluate our algorithm by applying it to the surface quadrangulation problem as formulated in Chapter 8. In this method two mixed-integer problems have to be solved where the first one is the computation of a smooth orientation field while the second one is a seamless parametrization. Here, to compare different solvers, it is sufficient to think of these quadratic mixed-integer problems in an abstract mathematical way by specifying the number of discrete and continuous variables. For more details about the problem formulation see Chapter 8. With the help of several experiments, we derived two different parameter sets for the two diverse problems. For the computation of the orientation field we used $\epsilon = 10^{-3}$, $maxiters_{GS} = 100000$ and $maxiters_{CG} = 50$ while for the parametrization we chose $maxiters_{GS} = 0$, $maxiters_{CG} = 20$ and the use of sparse Cholesky was disabled completely. The reason for two parameter settings is that both problems have quite different characteristics. While the orientation field exhibits a large number of integers with local influence, the parametrization problem induces only few integers but with rather global influence. With the above settings we were able to compute visually equivalent results compared to the original algorithm proposed in [BZK09] within a fraction of time. The performance benefit is a result of the tuned parameters as well as the novel extension which are the fill-in reducing reordering, the simultaneously rounding and some changes within the internal data structures. All examples were computed on a single CPU of an intel i7 quadcore 2.80GHz with 32GB of

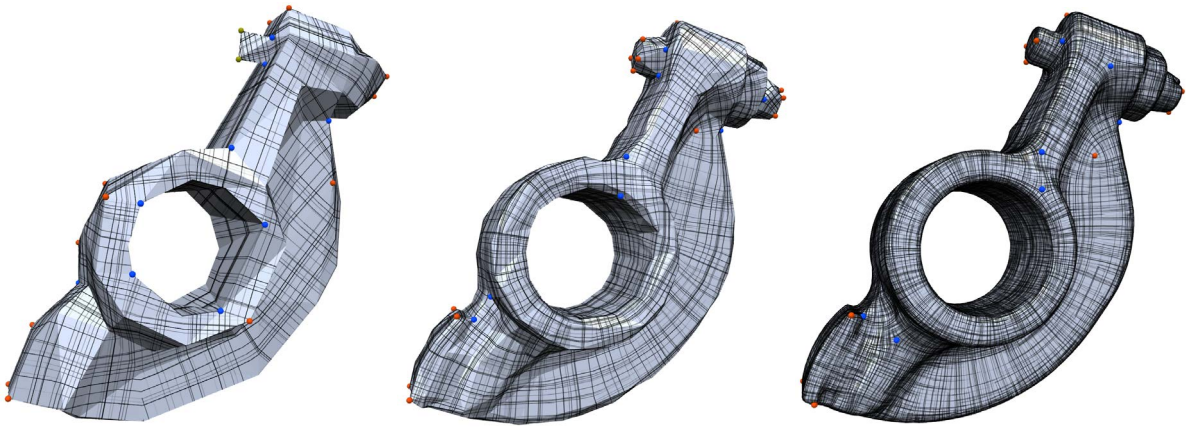


Figure 5.1.: Testbed for the solver comparison. For the ROCKERAM model with 200, 1k and 20k triangles (from left to right), first an orientation field is generated, followed by a global parametrization. Both steps induce quadratic mixed-integer problems.

RAM. The only exception is the commercial solver CPLEX [IBM12], used for comparison and exploiting all four cores.

Comparison to other solvers: The first experiment evaluates the performance and solution quality of our proposed solver, named constrained mixed-integer solver (CoMISo), in comparison to (1) a direct rounding approach, where all integers are estimated from the first continuous relaxation, (2) a naive greedy rounding, where in each step a sparse Cholesky update is performed and (3) the commercial solver CPLEX [IBM12], based on a branch-and-cut approach. The CPLEX solver was tried with different settings and the empirically best ones were chosen for our experiments. In case of the CPLEX solver, the tables show two different results. The first one gives the runtime which is required to find an approximate solution which is comparable to the one found by CoMISo, while the second one shows the best solution which was found after 900 seconds. It is worth mentioning that even for the smallest problems, CPLEX could not prove within these 900s that the found solution is the globally optimal one.

The testbed of our solver comparison consists in four different resolutions of the ROCKERARM model, three of them are shown in Figure 5.1. Table 5.1 shows the results of the first problem, namely the computation of an orientation field. In this problem, the number of discrete and continuous variables both grow linearly with the input size. Each

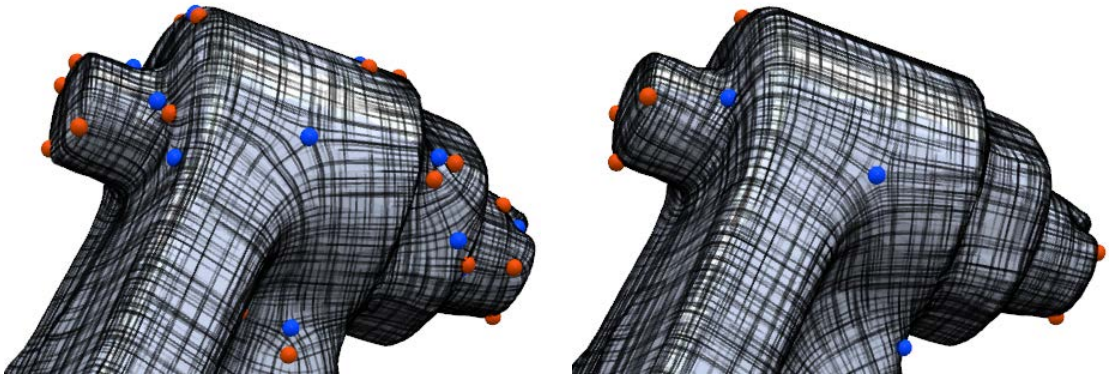


Figure 5.2.: Comparison of the orientation field quality of direct rounding (left) and CoMISo (right). The direct rounding approach leads to a poor smoothness induced by many undesired irregular vertices. In contrast, the CoMISo solver respects the interdependencies between discrete variables and thus achieves a much smoother result.

entry of the table lists runtime/energy, where a smaller energy value is better.

Table 5.1.: Solver Comparison w.r.t. Orientation Field Time/Energy

	#discr.	#cont.	CoMISo	direct rnd	naive rnd	CPLEX	CPLEX 900s
ROCKER200	111	153	6ms/8.6	2ms/10.4	10ms/8.6	500ms/8.6	900s/8.6
ROCKER1k	515	841	26ms/14.1	7ms/21.5	238ms/13.8	28s/14.0	900s/13.7
ROCKER2k	910	1414	36ms/21.9	14ms/30.9	640ms/21.6	57s/21.9	900s/21.3
ROCKER20k	10119	17759	480ms/33.9	160ms/58.7	124s/33.7	-	900s/40.5

While the fastest solver, i.e. , direct rounding, leads to solutions with an inadmissible energy value, the CoMISo solver is able to get close to the CPLEX reference solution in approximately three times the runtime of direct rounding. This is orders of magnitude faster than the naive rounding approach as well as CPLEX. Furthermore, in case of CoMISo, the runtime seems to increase much slower with growing input complexity compared to naive rounding and CPLEX. As a result, the runtime difference for the largest test case with 20k triangles consisting in 0.5s, 124s and >900s for CoMISo, naive rounding and CPLEX respectively, is enormous. It is worth mentioning that in this example the CPLEX solver could not find a solution comparable to that of CoMISo within 900s. Even worse, the gap between lower and upper bound is typically - also for the smaller examples - so big that it would be difficult to specify a good termination criterion for the CPLEX solver. However, in practice, typical input complexities are

often much higher, such that naive rounding as well as CPLEX become infeasible anyway.

Obviously, the runtime of direct rounding is much faster (approximately three times) and it seems to be tempting to use this algorithm. However, as can be seen in Figure 5.2, the resulting orientation field quality is too far away from the optimum and exhibits many additional counterproductive irregular vertices. The overall number of irregular vertices on the 20k ROCKERARM model is 28 for the CoMISo solution compared to 88 for the direct rounding approach.

The second mixed-integer problem, arising in quadmesh generation, computes a globally smooth seamless parametrization. The integer conditions are due to stitching constraints along a cutgraph that are indispensable for quadmesh generation. Compared to the mixed-integer problem arising from orientation field computation, the parametrization problem shows different characteristics. The number of discrete variables is typically smaller and does not increase with growing resolution. It is related to the geometric complexity, i.e. , the number of irregular vertices of the orientation field. However, again the CoMISo solver offers the best compromise of runtime and quality, as can be seen in Table 5.2. Due to the comparatively small number of discrete variables, the runtime differences are not as dramatic as in the orientation field computation. The runtime factor between CoMISo and CPLEX is approximately 45 for the 20k example. Even more important, the runtime complexity grows slower for the CoMISo solver, enabling much larger input sizes.

Table 5.2.: Solver Comparison w.r.t. Parametrization Time/Energy

	#discr.	#cont.	CoMISo	direct rnd	naive rnd	CPLEX	CPLEX 900s
ROCKER200	62	138	14ms/251	7ms/262	32ms/252	300ms/248	900s/245
ROCKER1k	60	940	41ms/164	13ms/173	127ms/163	500ms/163	900s/161
ROCKER2k	60	1940	68ms/232	21ms/254	258ms/232	4s/231	900s/230
ROCKER20k	56	19944	653ms/258	221ms/323	4.3s/256	30s/258	900s/255

Performance evaluation: We now want to investigate the general runtime behavior w.r.t. different input characteristics. In contrast to linear equation systems, where the runtime typically scales with the number of non-zero coefficients, the runtime of our mixed-integer solver typically behaves non-linear, strongly influenced by the interdependence between the continuous and discrete variables. First, we will compare the

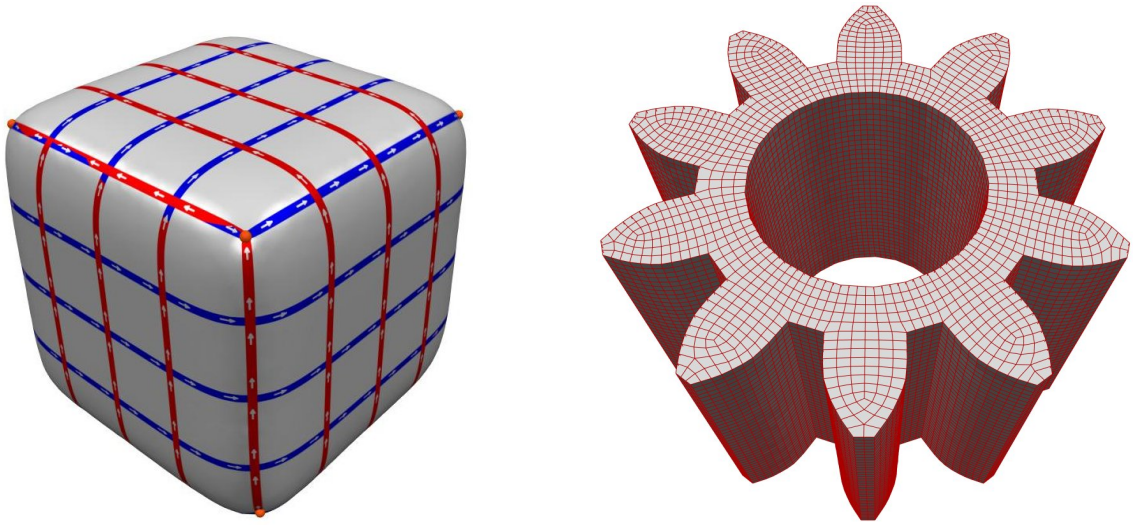


Figure 5.3.: (left) the SMOOTHED CUBE model with low geometric complexity. (right) the PINION model with many sharp features.

Table 5.3.: Orientation Field Timings in s

MODEL	10k	50k	200k	800k
ARMADILLO	0.3	1.2	6.3	33.9
CUBE	0.11	0.5	2.8	18.5

Table 5.4.: Parametrization Field Timings

MODEL	10k	50k	200k	800k
ARMADILLO	1.3	5.3	21.4	100.3
CUBE	0.15	0.9	6.7	55.1

runtime of the here presented optimized solver with a non-optimized variant used in [BZK09]. To give one representative example the, orientation field computation on the LEVER model of [BZK09] took $3.3s$ compared to $0.22s$ while the parametrization timing decreases from $19.9s$ to $2.8s$. However, further experiments showed that the runtime strongly depends on the geometric complexity of the object. In Table 5.3 we compare the timing of the orientation field computation of the ARMADILLO model (Figure 5.4) and a simple SMOOTHED CUBE (Figure 5.3). For the same number of triangles the geometric more complex ARMADILLO (121 singularities) model needs more computation time than the smoothed cube (8 singularities). In the case of constant geometric complexity the runtime depends almost linearly on the number of triangles, enabling very large inputs. A similar behavior can be observed for the parametrization problem in Table 5.4. The algorithm behaves sensitive to the geometric input complexity and nicely adapts to situations of different difficulty which is due to the simultaneous rounding approach.

Figure 5.4.: A quadrangulation of the ARMADILLO model, used in our benchmarking. Its organic structure leads to more complicated optimization problems compared to designed mechanical objects.

To underline the importance of the fill-in reducing reordering we did a separate experiment where the PINION model (Figure 5.3) with many sharp features was parametrized, leading to a huge set of dependent integer constraints. By applying the reordering the computation took $1.3s$ and the system matrix had $418k$ nonzero entries compared to a much slower runtime of $7.4s$ and $581k$ nonzero entries without the reordering.

Part II.

Parametrization based Quadrilateral Mesh Generation

The second part of this thesis is devoted to parametrization based quadmesh generation. The main concept of this class of approaches consists in mapping the canonical quadmesh, induced by the Cartesian grid of integer-isolines, i.e. $\{(u, v) \in \mathbb{Z} \times \mathbb{R} \cup \mathbb{R} \times \mathbb{Z}\}$, onto the surface. If this mapping fulfills some special conditions, that will be developed and explained in detail in Chapter 6, the mapped grid induces a quadmesh on the given surface. An illustration of this methodology can be found in Figure 5.5. Some of the special conditions arise at the artificial boundary of the parametrization domain Ω, where compatibility between the mapped grid-lines has to be ensured. Especially these compatibility conditions introduce integer degrees of freedom which necessitate the application of mixed-integer solvers as introduced in the first part.

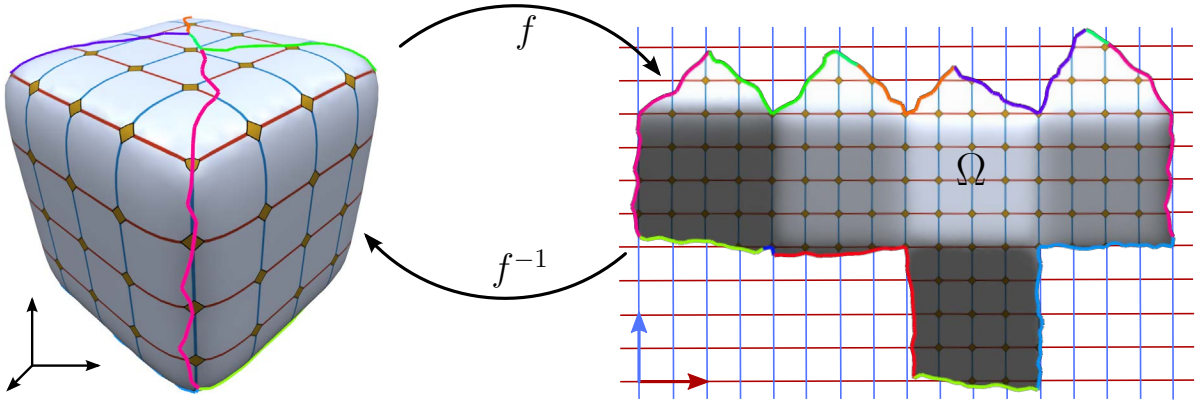


Figure 5.5.: The main idea of parametrization based quadmesh generation methods consists in mapping the canonical quadmesh formed by the 2D Cartesian grid onto the surface. First the mesh is parametrized, i.e. cut and flattened by a function f onto a 2D domain Ω. Then the intersection between the Cartesian Grid and the domain $\Omega \cap G$ is inversely mapped onto the surface, where a quadmesh is achieved by contouring. Special compatibility conditions are required along the colored cut-curves.

This part is structured as follows: First the mathematical framework of valid mappings, so called *Integer-Grid Mappings* (IGM), is specified and discussed in Chapter 6. Due to the enormous complexity of the induced mixed-integer problems, searching directly for high-quality IGMs is infeasible. The reasons are a huge number of discrete degrees of freedom in combination with complicated non-linearities. Therefore in Chapters 7 and 8 we develop two indirect approaches that are build on simplifying assumptions in order to enable efficient algorithms for the generation of high-quality IGMs

and thus high-quality quadmeshes.

The first approach, called *layout guided approach*, exploits a user-specified base-layout, which dramatically reduces the dimension of the discrete search space and additionally eliminates the non-linearities. This approach is well suited for reverse-engineering applications, since the user specifies the base structure of the resulting quadmesh. However, a drawback of the layout guided approach consists in the experience that specification of the complete base-layout can be very time consuming and necessitates expert knowledge. To overcome these limitations, the second approach, which is called *orientation-field guided approach* and presented in Chapter 8, is designed to perform fully automatic.

This orientation field guided approach is based on a splitting of the overall problem into two sub-problems, namely orientation field computation and orientation preserving parametrization, described in Sections 8.2 and 8.4 respectively. Intuitively the construction of the overall IGM is split into first estimating the rotational part of the mapping followed by generating the best mapping under the assumption of the estimated rotations. Due to the splitting, the non-linearities can be effectively removed. Additionally, the huge number of integer degrees of freedom, which are estimated in the rotational part, loose their global impact such that a good solution of the induced mixed-integer problem can be found in reasonable time. Moreover, the second approach not only enables a fully automatic method but in addition to it allows for a very flexible set of optional user-provided guiding constraints as presented in Section 8.6. It turns out that the layout based approach can be seen as a special case of the orientation field guided approach, where the user provides all irregular vertices and a complete base-layout. However, in the orientation field approach there is no need to specify everything from scratch and a user can iteratively improve the automatic solution by additional guidance constraints until she is satisfied. Often, even the first fully automatic solution leads to a pleasing result and otherwise typically a few additional high-level constraints are sufficient to achieve a quadmesh with the desired characteristics.

The main limitation of the orientation-field guided approach consists in the separation of rotation and metric estimation, which is necessary to achieve practical runtime. Therefore it turns out that often exploiting information of geodesic distance relations in the first rotation estimation step is beneficial. One example consists in preventing the generation of irregular vertices that are closer to a feature line than the desired edge

length of the quadmesh. Therefore the final Chapter of this part, i.e. Chapter 9, is not directly related to parametrization based quadmesh generation but instead develops an algorithm to efficiently compute exact geodesic distance fields on triangle meshes. These distance fields can be computed not only w.r.t. point sources but moreover w.r.t. polygonal line sources.

6. Integer-Grid Mappings

The main principle of parametrization based quadmeshing algorithms is the mapping of the canonical quadmesh formed by the 2D Cartesian grid of integer iso-lines onto a surface embedded in 3D, see Figure 5.5 for an illustration. However, this mapping has to fulfill several requirements such that the image of the 2D integer-grid stitches to a valid quadmesh on the surface.

In the following we will restrict to piecewise linear mappings given per triangle. More precisely given a triangle mesh $\mathcal{M} = (V, E, T)$ composed of vertices, edges and triangles, a mapping f is given as the union of all individual triangle mappings specified by the images of their corresponding three vertices:

$$f_i : (p_i, q_i, r_i) \in \mathbb{R}^{3\times 3} \mapsto (u_i, v_i, w_i) \in \mathbb{R}^{2\times 3}$$

Note that following [KNP07] each triangle is an individual chart and consequently a single vertex might have multiple different images.

The class of Integer-Grid Mappings is defined to be the subset of all these mappings, which additionally correctly stitch the grid of integer iso-lines to a valid quadmesh. The necessary and sufficient conditions are the following:

- *Transition Functions:* The transition function $g_{i\to j}$ from the chart of triangle t_i into the chart of a neighboring triangle t_j and identifying their common edge has to be an integer grid automorphism [KNP07, BZK09] of the form

$$g_{i\to j}(\mathbf{a}) = R_{90}^{r_{ij}}\,\mathbf{a} + \mathbf{t_{ij}} \tag{6.1}$$

  consisting of a $r_{ij} \in \{0, 1, 2, 3\}$ times $\pi/2$ rotation and an integer translation $\mathbf{t_{ij}} \in \mathbb{Z}^2$.

- *Singular Points:* With the above transition functions it is possible to represent cone singularities with quarter-indices which are characterized by a nonzero angle

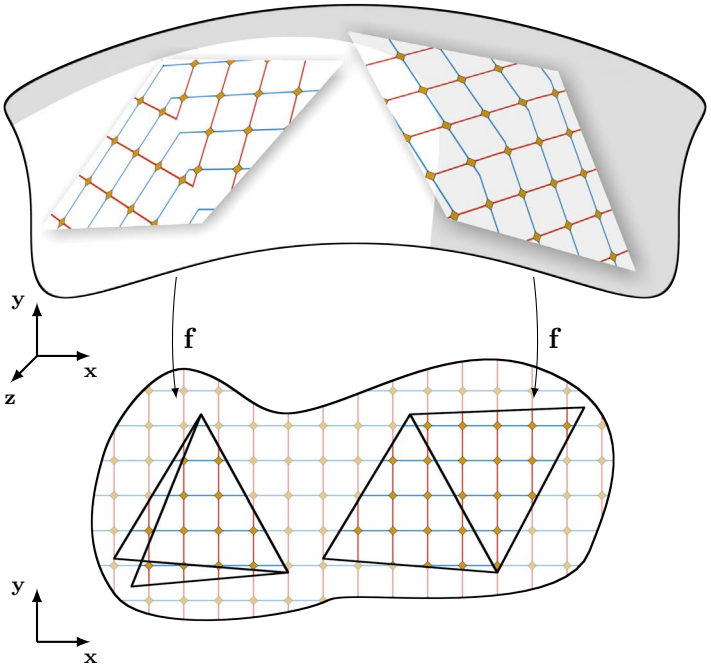


Figure 6.1.: Integer-Grid Mapping: A local foldover where two adjacent triangles have different orientations in the domain induces non quad elements like 2-gons and 6-gons in the mapping (left), while a consistent orientation correctly maps the integer grid to quad elements only (right).

defect[1] in the domain. Let S be the set of all singular points, then in order to guarantee a pure quadmesh all singular vertices have to be mapped to integer locations in the domain, i.e.

$$\mathbf{f}(s_i) \in \mathbb{Z}^2 \ \forall s_i \in S \tag{6.2}$$

- *Consistent Orientation:* All mapped triangles $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ with $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^2$ should have a positive orientation, meaning that

$$det\,[\mathbf{v} - \mathbf{u}, \mathbf{w} - \mathbf{u}] > 0 \tag{6.3}$$

The consistent orientation condition is illustrated in Figure 6.1.

[1]The angle defect is defined in the usual way to be $2\pi - \sum_i \alpha_i$ for interior and $\pi - \sum_i \alpha_i$ for boundary vertices

6.1. MINLP Formulation

In parametrization based quad-remeshing typically a variational quality metric $E_q(f)$ is chosen that penalizes undesired distortions of the resulting quad elements on the basis of f. Often $E_q(f)$ is designed to be a (convex) quadratic functional that on the one hand prefers the alignment of quad elements along dominant principal curvature directions and on the other hand tries to achieve a user specified element density. Usually the unconstrained minimizer of $E_q(f)$ is not an Integer-Grid Mapping and therefore in order to achieve a quadmesh we would like to solve the following instance of MINLP:

$$\text{minimize} \quad E_q(f) \quad \text{s. t.} \quad (6.1),\ (6.2),\ (6.3) \tag{6.4}$$

The above naive problem formulation (6.4) consists of $6|V| + 3|E|$ unknowns with at least $3|E|$ discrete variables. Due to (6.1) there are $2|E|$ many equality constraints that are nonlinear in r_{ij} and linear in t_{ij}. Furthermore (6.3) generates $|T|$ many non-convex quadratic inequality constraints. Although these degrees of freedom can be reduced along a spanning tree without loosing the optimal solution (see [BZK09]) the resulting number of unknowns is still in the order of $O(|V|)$ in the continuous as well as in the discrete variables.

Unfortunately, problems of the class MINLP are very hard to optimize since they imply all difficulties from continuous as well as discrete optimization. Even by neglecting all integer constraints there is little hope of finding good solutions since due to (6.3) the continuous relaxation is within the very difficult class of non-convex Nonlinear Programs. Figure 6.2 gives an idea on how complicated the situation is. Optimizing the unfolding of the one-ring neighborhood of a vertex w.r.t. a convex energy functional and subject to triangle orientation constraints of Equation (6.3), the optimization typically gets trapped in a poor local minimum when started from a random initial point. Consequently, in case of a triangle mesh with thousands of vertices finding a global optimum is extremely difficult.

Accordingly, in the next chapters we will investigate different simplification assumptions in order to find high-quality integer-grid mappings in reasonable time.

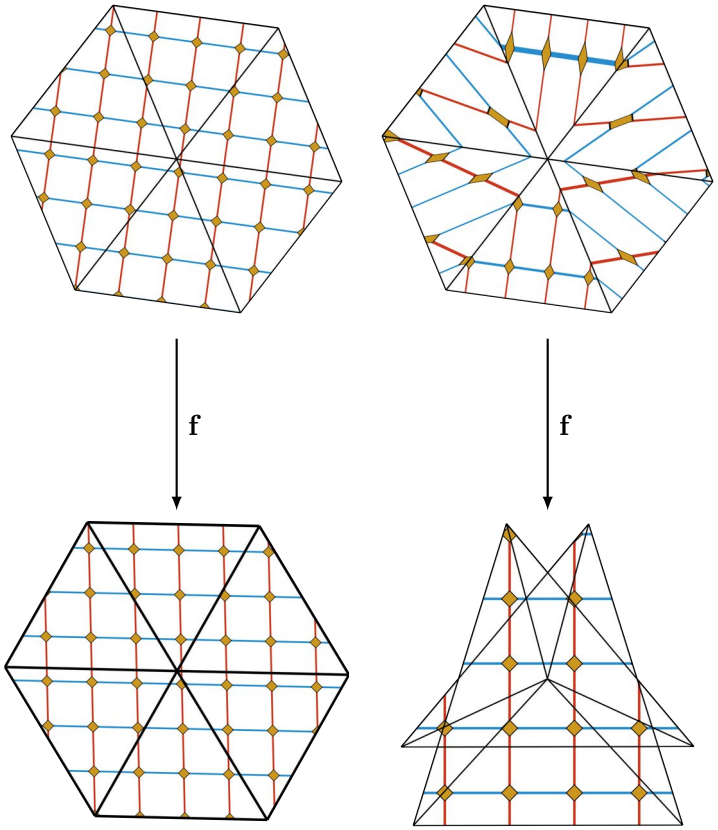


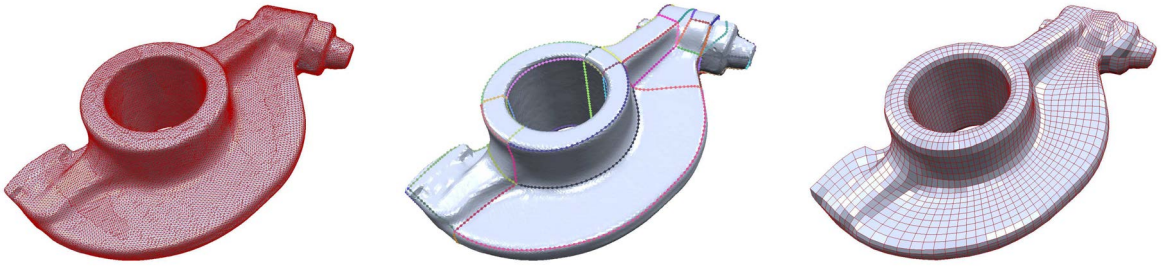
Figure 6.2.: Given six planar triangles the global optimum of a convex parametrization energy is the identity function (left). Optimizing the same energy with non-convex triangle orientation constraints (6.3) from a random starting point, the solver gets trapped in a poor local minimum (right). The result is indeed a valid solution since all triangles have a positive orientation. However, from this local minimum there is no continuous deformation which reaches the global minimum without violating at least one triangle orientation constraint in between.

7. Layout guided Approach

The main idea of layout guided approaches consist in exploiting a known partitioning of the surface into rectangular patches. Roughly speaking, each rectangular patch is mapped to an integer-sized rectangle in the 2D domain which induces a quadrangulation of the surface patch (cf. Chapter 6). There is no reason that the quadmeshes of neighboring patches stitch together seamlessly. Accordingly, the mappings of neighboring patches have to be constructed subject to some transition functions which equate the number of generated quads along the patch boundary. In this chapter, based on [BVK10], we develop a consistent layout based quad meshing approach.

In the view of general integer-grid mappings, as introduced in Chapter 6, the space of possible mappings is strongly reduced by respecting the given patch layout. It turns out that all rotational degrees of freedom of the transition functions are removed, which directly implies that all irregular vertices are determined by the given layout. As shown next, the resulting mixed-integer problem, related to (1) the regular subdivision of every rectangular patch into a quadmesh and (2) to a geometrically optimal embedding of the induced quadmesh, is much simpler compared to the unguided general approach.

The only remaining difficulty consists in the consistent orientation of mapped triangles. If the surface patches are far away from being rectangularly developable, large distortions in the mapping might arise and sometimes induce some flipped triangles. However, since the consistent orientation constraint is non-convex, including it directly into the optimization is not an option. As a solution, we propose a richer set of transition functions, which act between the mappings of neighboring patches, that allow for arbitrarily shaped quadrilateral domains. As a result, the generated mapping is typically more isometric and free of foldovers in the vicinity of the constrained patch corners (see for example Figure 7.2).



(a) (b) (c)

Figure 7.1.: Reverse Engineering Pipeline: (a) The input is a dense, unstructured triangle mesh. (b) The user provides a coarse layout controlling the quadrangulation. Singularities can only occur at nodes of this layout. (c) A distortion minimizing parameterization is computed to extract a pure quadmesh.

Layout guided approaches are often applied in the context of *reverse engineering*, i.e. the procedure of converting a given unstructured triangle mesh into such a structured quadrangulation. Figure 7.1 depicts a typical reverse engineering pipeline.

Although, even in this setting a fully automatic algorithm would be preferable, sometimes design decisions depend on the intended usage and cannot be forecast by pure geometric considerations. Therefore in reverse engineering full user-control, where the user can easily provide the topology, i.e. the number and position of singularities, and some alignment constraints for the resulting mesh, is typically preferred over time efficiency. This can be achieved in a simple way by using coarse layouts which partition the surface in quadrilateral patches as illustrated in Figure 7.1b. From this layout a globally smooth parameterization can be computed by assigning a two-dimensional chart to each patch and connecting the parameterizations of neighboring charts with so called transition functions (see Figure 7.1c).
The resulting mesh quality strongly depends on the metric distortion of the parameterization and on the alignment to sharp features. Consequently we present a new method to handle both tasks in a robust way, enabling the usage of global parameterization techniques for reverse engineering. Our main contribution in this chapter consists in a chart optimization technique which minimizes the distortion of the resulting global parameterization. In contrast to other methods each chart is allowed to be an arbitrary 5 degree-of-freedom (DOF) quadrilateral with interior angles possibly differing from 90 degrees. As a result we need to specify generalized transition functions between these

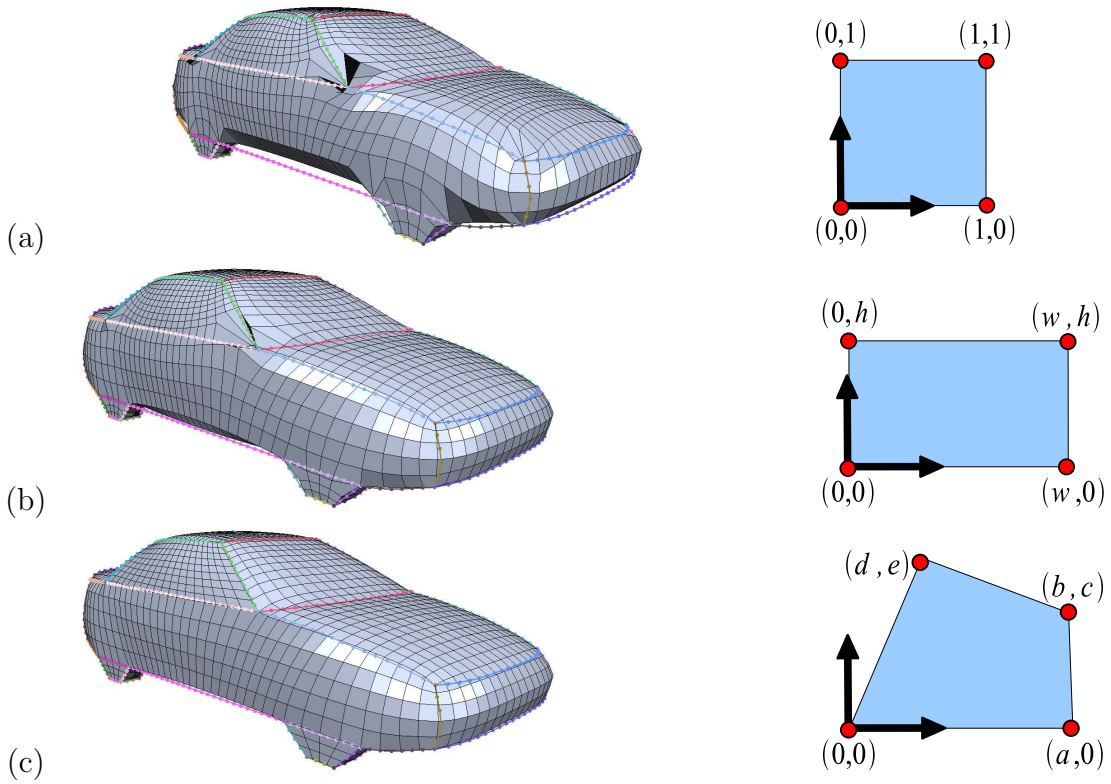


Figure 7.2.: (a) A global parameterization using unit charts leads to large distortions and foldovers for a simple car model. (b) Even with optimized rectangular domains the distortions get large where the patches have a trapezoidal shape. (c) Our generalized, quadrangular parameterization leads to low distortion. Notice that in this example no alignment constraints were applied.

charts. Other important ingredients of our practical reverse engineering method are alignment constraints and T-Vertices, enabling simplified layout design. Figure 7.2 illustrates the gain in quality due to our chart optimization where chart corners are chosen to form a unit square (a), an optimized rectangle (b) and an optimized general quadrilateral (c).

Comparison to previous work: A user designed coarse layout, here called singularity graph, was also used in [TACSD06] to compute globally smooth harmonic parameterizations. These layout-based techniques are closely related to our method. Therefore we

will discuss them in more detail.

The method of Dong et al. [DBG*06] uses simple unit squares as charts for a globally smooth parameterization. This is justified because in their layout, neighboring surface patches, originating from the Morse-Smale complex of the Laplacian eigenfunction, have similar size. Furthermore the layout vertices, representing the mesh singularities, are relaxed on the surface to prevent foldovers and large distortions. In reverse engineering such a relaxation technique is not reasonable since it interferes with the desired user-control. Figure 7.2a shows the result of a globally smooth parameterization onto unit square shaped charts with a fixed user provided layout. The result contains large distortions and foldovers reflecting the fact that neighboring surface patches are far from being equally sized. Thus, obviously unit square charts are not sufficient for our setting.

If one would restrict the layout to quadrilaterals and choose all free coefficients to one the globally smooth parameterization technique of Tong et al. [TACSD06] is exactly the same as the one discussed in the last paragraph. Notice that this equality is non-trivial since both papers use a different formalism to derive the final global linear system. Besides, the parameterization of Tong et al. is more general because it allows a larger class of charts. Each chart is a polygon where the vertices lie on integer positions and all edges are aligned to the coordinate axes, accordingly all interior angles of a chart are multiples of 90 degree. In our car example this means moving from unit squares to rectangular charts with two DOF's, namely the two independent edge lengths. In the original method this new DOF's are chosen manually or by using a heuristic which simply rounds the length of the corresponding layout edges to integer. Figure 7.2b shows the result of the car example using rectangular charts. Here we already used our chart optimization technique presented in Section 7.2, instead of their heuristic, to minimize the length distortion. However, we still observe large distortions, for example near the corner of the front window.

The problem is that the surface patches are far from being rectangular. Consequently, we consider an even more general class of charts, i.e. we allow charts to be arbitrary quadrilaterals with five DOF's. We exploit these DOF's to minimize the distortion of the parameterization and the result can be seen in 7.2c. Even without alignment constraints, the quadmesh edges follow the user prescribed layout and the length distortions are much lower. This introductory example motivates our design choices for a practical

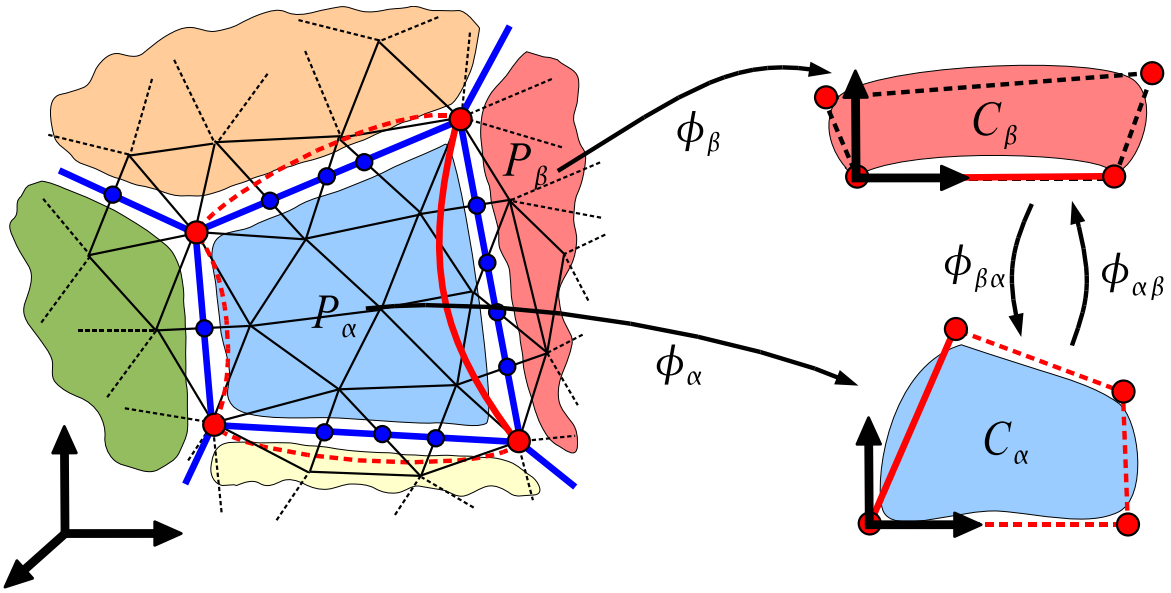


Figure 7.3.: The left part shows the layout of a multi-chart parameterization. Vertices of the layout graph (red) lie at mesh vertices and edges of the layout graph (blue) cut several mesh edges. Each inner vertex of a patch P_α stores its parameter coordinates w.r.t the local frame of chart C_α. For all pairs of neighboring patches transition functions $\phi_{\alpha\beta}$ exist which translate between their charts. Notice that the red quadrilateral, connecting the four corners of Chart C_α, mapped to the surface is generally not identical to the blue layout.

reverse engineering method.

In the subsequent paragraphs our method is explained in more detail. We start with a mathematical description of chart based global parametrization in Section 7.1, where our main contribution, i.e. , the domain optimization, is given in Section 7.2. Finally, we conclude this chapter with an evaluation in form of some exemplaric meshing tasks in Section 7.3.

7.1. Layout Parametrization

The input to our quadrangular multi-chart parameterization method is a triangle mesh $M = (V, E, F)$ of arbitrary genus, which is a set of vertices, edges and faces, and a layout graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{F})$. For each edge of the layout graph the user can additionally

set a tag which enforces the alignment of the parameterization onto this layout edge, as described in Section 7.2. The scenario of a multi-chart parameterization is depicted in Figure 7.3. The vertices of the layout graph (red points) lie at triangle mesh vertices and each edge of the layout graph intersects several mesh edges (blue points). In this way all mesh vertices are partitioned into several surface patches, which are disjoint except for the layout vertices that belong to all neighboring patches. Each such patch P_α is equipped with a two-dimensional chart C_α. Assume for simplicity that each layout graph face has exactly four vertices, we will discuss in Section 7.2 how to incorporate more general settings. The task is now to compute a piecewise linear multi-chart parameterization, i.e. each vertex $v_i \in \mathbb{R}^3$ belonging to P_α is mapped by the function ϕ_α to the parameter coordinates $u_i^\alpha \in \mathbb{R}^2$ expressed w.r.t the frame of chart C_α. Additionally for triangles with vertices in different patches, for instance P_α and P_β, we need a transition function $\phi_{\alpha\beta}$ to translate between their charts in order to parameterize them. Obviously both directions are possible and inverse to each other $\phi_{\beta\alpha} = \phi_{\alpha\beta}^{-1}$ and the transition from one chart into itself is simply the identity $\phi_{\alpha\alpha} = Id_2$.

A discrete harmonic parameterization of a surface with disc topology mapping to a single chart is a well studied topic where typically the boundary of the surface is mapped to the boundary of a disc and each interior vertex has to fulfill the discrete harmonic equations

$$\sum_{j \in N_i} \bar{w}_{ij}(u_j - u_i) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{7.1}$$

where N_i are the one-ring neighbors of vertex v_i and $\bar{w}_{ij}$ are normalized edge weights which sum to one $\sum_{j \in N_i} \bar{w}_{ij} = 1$. In all our examples we used the normalized discrete harmonic weights

$$w_{ij} = \frac{1}{2}(cot\alpha_{ij} + cot\beta_{ij}) \quad with \quad \bar{w_{ij}} = w_{ij} / \sum_{j \in N_i} \bar{w}_{ij} \tag{7.2}$$

where α_{ij} and β_{ij} are the two angles opposite to edge e_{ij}. There are many other good choices like Floater's Mean Value Coordinates, see [HLS07] for more details. The key observation is that in our multi-chart parameterization setting we can compute a harmonic parameterization in the same way. The only difference is that instead of fixing a whole boundary we now only fix the corner vertices of the layout graph in each chart and use the transition functions to compute the harmonic conditions in a common frame:

$$\sum_{(j,\beta)\in N_i} \bar{w}_{ij}(\phi_{\beta\alpha}(u_j^\beta) - u_i^\alpha) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{7.3}$$

.

In this formulation a global relaxation is achieved. If the transition functions are affine the combination of the above equations for all free vertices form a global linear system of dimension $2(n-k) \times 2(n-k)$ where n is the number of triangle mesh vertices and k is the number of layout vertices. The translational part of the affine transition function as well as known values of constrained layout corners are moved to the right-hand-side. Remember that the coordinates of layout corners cannot be unique because they belong to different charts with different frames. So we need to specify $4|\mathcal{F}|$ many corner positions.

These parameter coordinates of the four patch corners can be in general position (keeping the same orientation as on the surface). However, we choose the first one to be the origin and the second one to lie on the first coordinate axis which makes the representation unique. So we end up with five DOF's (a, b, c, d, e) for an arbitrary quadrilateral (see Figure 7.2c). The transition function between neighboring charts, which share a common edge (red), are simple affine functions, combinations of translations, rotations and a scaling as depicted in Figure 7.4.

$$\phi_{\alpha\beta} = T_\beta^{-1} R_\beta^{-1} S R_\alpha T_\alpha \tag{7.4}$$

They can be precomputed as 3×3 matrices in extended coordinate representation before accumulating the resulting values into the global system matrix.

The only question left is how to choose adequate corner parameter coordinates (a, b, c, d, e) for a given patch. In [TACSD06] the average length of two opposing layout edges rounded to an integer was used to fix width and height of the corresponding rectangle. In the case of a five DOF chart we could do something similar by using all lengths of the patch's boundary. However, as explained in the next section the available DOF's can be used to optimize the resulting parameterization in a more founded but still efficient way, which in general leads to better results.

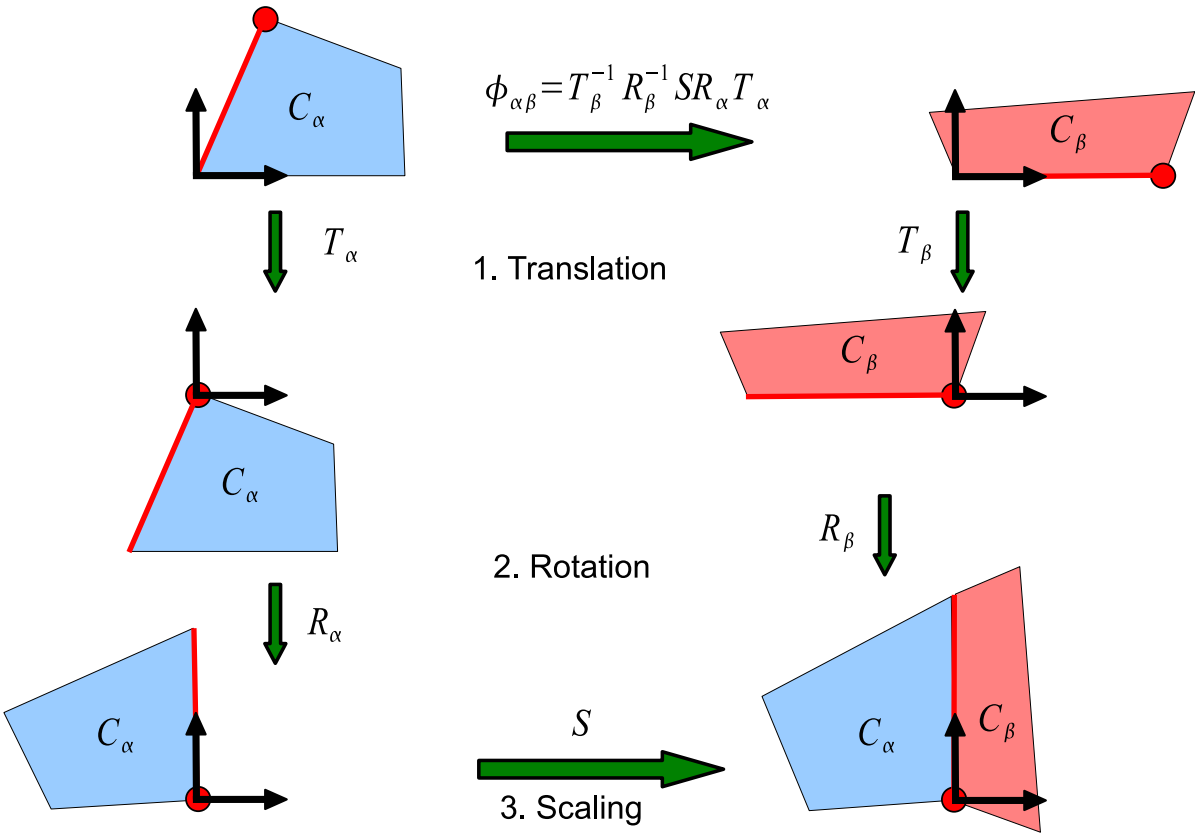


Figure 7.4.: A common coordinate frame of two charts C_α and C_β can be constructed by first translating a common point into the origin, then rotating the common edge to a coordinate axis and finally scaling along this axis to end up with the same edge length. The transition functions between the charts are constructed in the same way by using the inverse of either α or β functions.

7.2. Domain Optimization

The idea of our chart optimization algorithm is to minimize the metric distortion of the parameterization ϕ. The local distortion near a surface point p_0 in direction v (in local coordinates of the tangent plane) is described by the first order Taylor expansion

$$\phi(p_0 + v) \approx \phi(p_0) + J_\phi(p_0)v \quad \Rightarrow \quad \phi(p_0 + v) - \phi(p_0) \approx J_\phi(p_0)v \tag{7.5}$$

where J_ϕ is the Jacobi matrix which can be written as two rotations and a scaling by applying the singular value decomposition

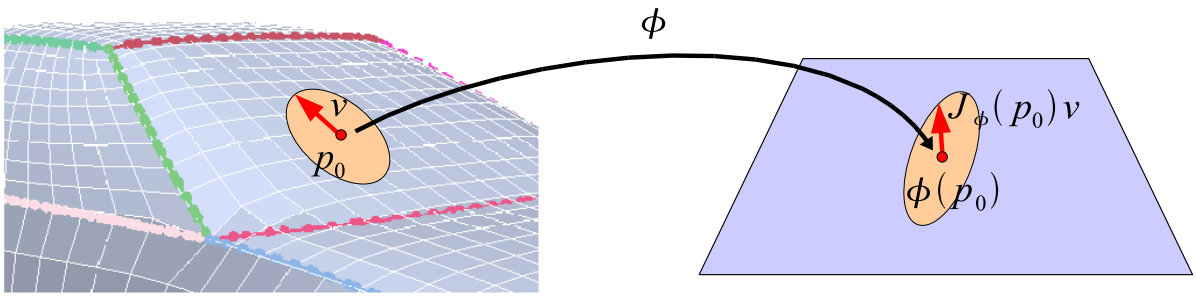


Figure 7.5.: Mapping a small disc from the tangent plane around a point p_0 the transformation can be approximated by the Jacobi matrix J_ϕ of the mapping ϕ. This means mapping circles into ellipses where the length of the principal axes are related to the singular values of J_ϕ.

$$J_\phi = U \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V^T \tag{7.6}$$

Mapping a unit length vector $\|v\| = 1$, lying in the tangent plane of p_0, into its chart the resulting vector has length $\|J_\phi v\| \in [\sigma_1, \sigma_2]$. Consequently a circle on the surface is mapped to an ellipse in the chart as illustrated in Figure 7.5. There are some well known special cases [HLS07]:

1. $\sigma_1 = \sigma_2$ is a conformal mapping which maps circles to scaled circles
2. $\sigma_1 \cdot \sigma_2 = 1$ is an equiareal mapping
3. $\sigma_1 = \sigma_2 = 1$ is an isometric mapping with no distortion

Clearly an isometric mapping is the best we can hope for. So we try to choose our chart corners to get as isometric as possible. The desired isometry measure is $E = |\sigma_1 - 1| + |\sigma_2 - 1|$. To approximate this measure we take the quadratic Frobenius norm of the 2D strain tensor

$$E = \|J_\phi^T J_\phi - I\|_2^2 \tag{7.7}$$

which is 0 in the case of isometry and $(\sigma_1^2 - 1)^2 + (\sigma_2^2 - 1)^2$ when the mapping is conformal.

Using a triangle mesh where the mapping is piecewise-linear, the Jacobi-matrix of a triangle is constant and depends linearly on the parameter values u_0, u_1 and u_2 of the triangle,

$$J_\phi = [u_0 u_1 u_2] \begin{bmatrix} p_0 & p_1 & p_2 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{7.8}$$

In the above equation p_0, p_1 and p_2 are the 3D triangle vertices in local 2D coordinates and u_0, u_1 and u_2 are the corresponding parameter values. Therefore $J_\phi^T J_\phi$ is quadratic and the isometry measure of a triangle t is a quartic polynomial in the parameter values, $E_t(u_0, u_1, u_2) = \|J_\phi^T J_\phi - I\|_2^2$

The aim of this section is to optimize the isometry of the harmonic parameterization by finding adequate parameter coordinates for the four corners of a chart. Consequently we need to express the isometry measure of a triangle w.r.t. these values (a, b, c, d, e). To approximate the relation between the global parameterization and the change of chart corner positions we assume that the dependency is bilinear, which is a good approximation for all interior vertices of a chart:

$$u_i = u_i(a, b, c, d, e) = s_i(1 - t_i) \begin{pmatrix} a \\ 0 \end{pmatrix} + (1 - s_i)t_i \begin{pmatrix} b \\ c \end{pmatrix} + s_i t_i \begin{pmatrix} d \\ e \end{pmatrix} \tag{7.9}$$

.

Since we use these bilinear coordinates s_i and t_i in the sense of freeform deformation, the parameter coordinates u_i are linear in the corner positions (a, b, c, d, e) and so the measure $E_t(a, b, c, d, e)$, now expressed in dependency of the four chart corners, is still a quartic polynomial. Finally we sum up the measures of all triangles lying completely inside the polygon formed by the chart corners that we want to optimize and weight them by the area of the corresponding surface triangle.

$$E_\alpha = \sum_{t \in C_\alpha} E_t(a, b, c, d, e) \cdot A_{\phi^{-1}(t)} \tag{7.10}$$

In this optimization phase all layout edges are always tagged for alignment which ensures that all vertices of patch P_α are mapped into C_α. This energy only depends on five variables and is very well conditioned because of its geometric nature. Therefore we can use a simple and efficient Newton method to find a local minimum. Since the bilinear

dependency is only an approximation we have to recompute the parameterization after each chart optimization. To initialize the charts we can simply use unit charts or the heuristic of [TACSD06]. The complete algorithm works as follows:

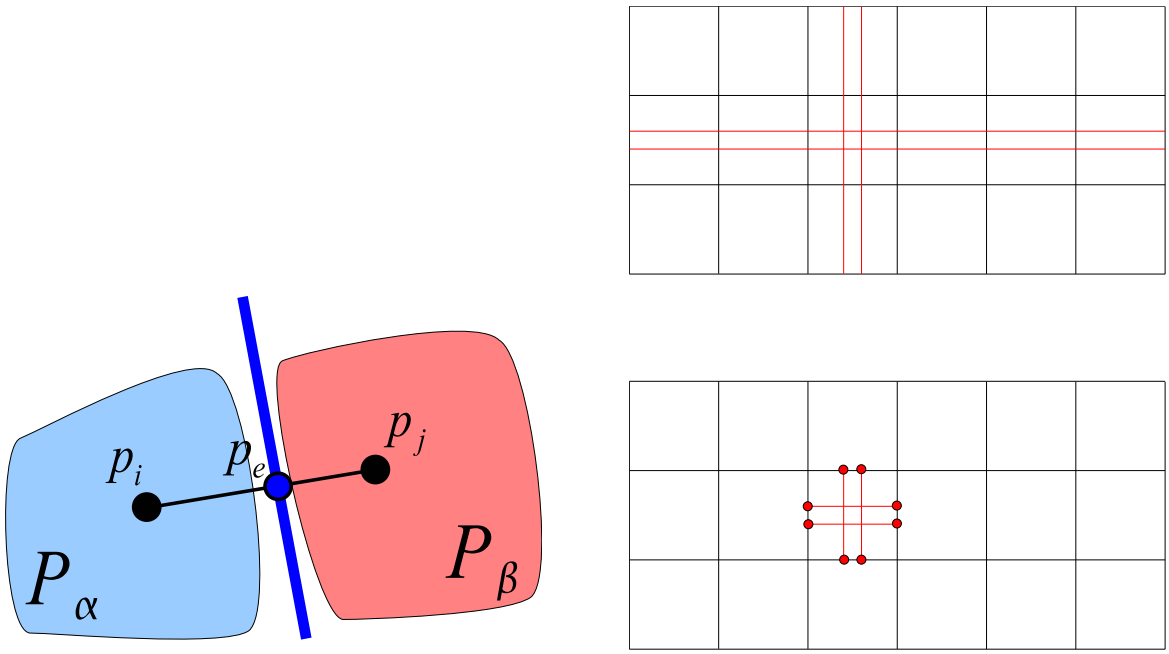
1. tag all layout edges for alignment
2. compute an initial parameterization with unit charts
3. iterate k times
   a) optimize all charts individually
   b) update transition functions
   c) recompute parameterization
4. restore user-provided alignment tags and compute final parameterization

The bilinear relation is close to the exact dependency, therefore in all our experiments three iterations were sufficient to converge. Notice that our method is related to [DBG*06]. However, instead of relaxing the layout vertices on the surface, we relax them within the charts. This is more suitable for reverse engineering where the user provided layout is in general not allowed to be changed. In the next section we will discuss how to incorporate layout alignment constraints into the computation.

Alignment Constraints The user can tag a subset of layout edges for alignment which ensures that it will be explicitly represented in the meshing. For the parameterization this means that the mapping of a tagged layout edge should be the straight line connecting both corresponding corners in the chart. Or in other words the parameter coordinates along the layout edge are not independent. The parameter coordinates at a point $p_e = (1-\lambda)p_i + \lambda p_j$ on the layout edge cannot be computed directly in the form $u_e = (1-\lambda)u_i + \lambda u_j$ because u_i and u_j are represented w.r.t different charts (see 7.6a). However by employing the transition function, we can express the alignment constraint in a simple form where the image of the layout edge is constrained to have the first coordinate equal to zero. This is exactly the lower right setting in Figure 7.4:

$$u_e^\gamma = (1-\lambda) S R_\alpha T_\alpha u_i^\alpha + \lambda R_\beta T_\beta u_j^\beta \stackrel{!}{=} \begin{pmatrix} 0 \\ * \end{pmatrix} \tag{7.11}$$

.



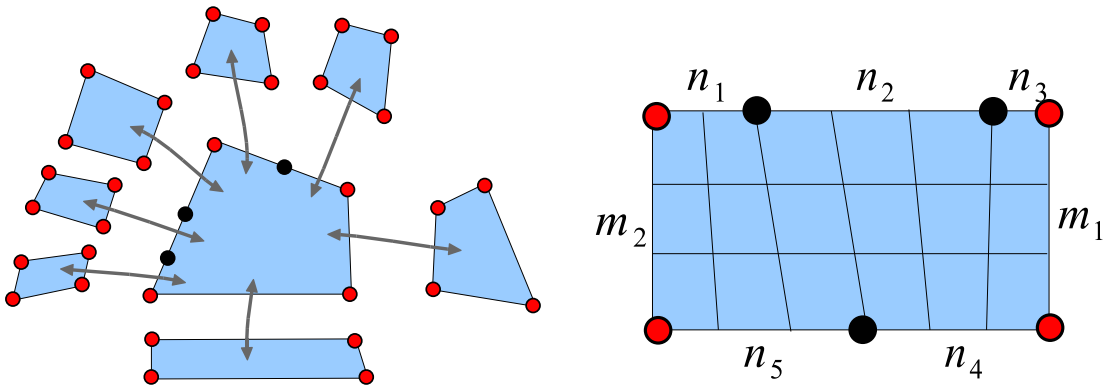
(a) (b)

Figure 7.6.: (a) The parameter values at a layout edge can be computed at the intersection points of triangle mesh edges p_e by incorporating the transition function between both charts. (b) A local refinement of a quadrangular layout has global support (top). By allowing T-Vertices, the refinement of the layout remains local (bottom).

The global linear system already has full rank, therefore after adding the alignment constraints we have to relax some other equations to be optimized only in least squares sense. A good choice are the harmonic constraints of all vertices which are involved in alignment constraints. This means pulling the parameterization onto the layout edge by allowing slight non-harmonicity near the constraint. Notice that our alignment constraints restrict only one coordinate of the parameterization and there is still a global relaxation in orthogonal direction. Finally the parameterization is formulated as a mixed least squares system of the form

$$\begin{bmatrix} A^TA & B^T \\ B & 0 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A^Tb \\ c \end{pmatrix} \tag{7.12}$$

where the equations $Bx = c$ are fulfilled exactly and the equations $Ax = b$ are satisfied in a least squares sense. In the next section we will describe how to simplify the layout generation by allowing T-vertices.



(a) (b)

Figure 7.7.: (a) Allowing T-Vertices in the layout is possible in a simple way by computing a transition function per layout edge. (b) To get a closed quadrangulation the number of samples on opposite edges of a chart must be equal.

T-Vertices Restricting the layout graph to consist only of four sided polygons, as done before, is too restrictive in practice. Performing a local refinement to keep more features, a global refinement would result as illustrated in Figure 7.6b. Remember that in many reverse engineering scenarios this layout is directly designed by a user and the effort should be as low as possible. Therefore we allow an arbitrary number of T-vertices per layout edge. This can be easily achieved by computing a separate transition function for each part of a layout edge. The parameter coordinates of T-vertices in a chart are defined by linear interpolation of the corners to keep the number of variables of a chart constant and allow to extract a mesh consisting only of quadrilaterals as explained in the next section.

Meshing The meshing proceeds as follows, first a consistent quadmesh is constructed in the 2D charts of the parameterization which is then mapped to the surface. The four corners of a chart form a four sided polygon in the plane whereas each edge can be partitioned by T-vertices into several subintervals as depicted in Figure 7.7. By backmapping the chart polygon edge it is possible to compute the desired number of samples n_d which is the quotient of the length of the backmapped curve and the target edge-length for the meshing provided by the user. This value may be chosen differently for each layout edge. However in the case of a consistent quadmesh the number of samples cannot be chosen arbitrarily. There are the following consistency constraints:

1. The number of segments (quadmesh edges) on opposite edges of the chart polygon must be equal. In the example of Figure 7.7a this means $n_1 + n_2 + n_3 = n_4 + n_5$ and $m_1 = m_2$.

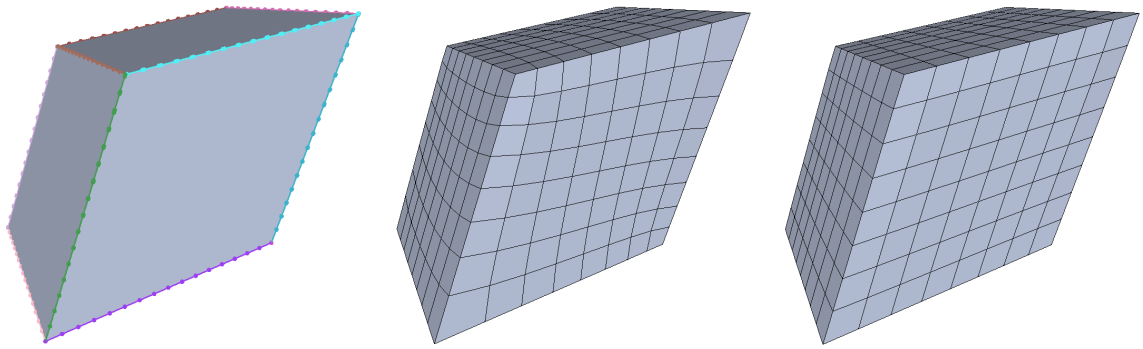
2. Each T-vertex lies on a sampling position.

3. In each subinterval the samples are distributed linearly which guarantees that neighboring charts stitch together compatibly.

With these restrictions a consistent quadmesh can be constructed by connecting opposite sample pairs. This is always possible since condition 1 states that the number of samples is equal on opposing sides. We assemble the two equations per layout face in a common linear system $Bn = 0$ and compute free variables via Gauss elimination as done in [TACSD06]. Simply fixing the free variables by rounding the corresponding entries from the local desired number of samples n_d leads to poor results since the free variables computed by the Gauss elimination strongly depend on the numbering of the variables and can lead to strong deviations from the expected number of samples on other edges. Therefore we first compute the best continuous solution n_c which meets the constraint $Bn_c = 0$ and thus minimizes the deviation from the desired values n_d in a least squares sense. As a result we solve the linear system from equation 7.12 with $A = I_d$, $B = B$, $b = n_d$ and $c = 0$. Then rounding the free variables to the integer closest to the value of the continuous solution n_c leads to appealing results because the continuous solution captures the global necessary edge-length distribution.

7.3. Evaluation

In this section we will discuss the properties of the presented method by exploring some results. The first example is a sheared cube with unit edge length, depicted in Figure 7.8a. This simple model illustrates the difference between the parameterization of [TACSD06] and our method which are displayed in (b) and (c) respectively. In (b) edge length distortions and S-shaped isolines are unavoidable because of the inherent tangential continuity of this method. This can be seen by unfolding neighboring faces of the cube where the isolines in the case of 7.8b are smooth since the necessary curvature of the cone singularities is distributed over the whole geometry. In contrast to this result our method 7.8c concentrates the tangential curvature at feature lines, i.e. regions of high geometric curvature, where tangential continuity is not important. This example



(a) (b) (c)

Figure 7.8.: (a) A sheared cube with unit edge length is segmented along the geometric edges. (b) Restricting to charts with 2 DOF's, length distortions and S-shaped isolines are unavoidable since the necessary curvature of the cone singularities is distributed over the whole geometry. (c) Our method concentrates the tangential curvature at geometric features where they don't influence the mesh quality. This approach leads to the expected result of uniformly shaped quadrilaterals.

is indeed a hint on how to use the presented method. Charts with five DOF's are advantageous for patches with a layout, lying on geometric features while charts with two DOF's are better suited within smooth or flat regions. Typical objects consist of both types of regions, such that the user should select for each patch which optimization is performed. This is possible in a straightforward way due to the fact that the optimization of individual charts is independent.

The second example is the car model depicted in Figure 7.2 and already discussed in the introduction. Figure 7.9 shows all chart polygons after 3 steps of optimization with 2 DOF's and 5 DOF's in (a) and (b) respectively. The presented optimization algorithm finds well shaped chart polygons robustly and produces almost symmetric configurations since the user-provided layout is almost symmetric. Compared to the time which is necessary for the solution of the global linear system, the optimization of charts is neglectable. Altogether the computation timings are comparable to [DBG*06] while in practice we need fewer iterations to converge. In all our examples we used the sparse direct solver SuperLU as proposed in [BBK05].

In Figure 7.10 we demonstrate the usage of alignment constraints. Between the front window and the hood of the car there is a sharp edge which should be preserved in the final mesh in order to prevent sampling artifacts. Therefore, the lower red layout

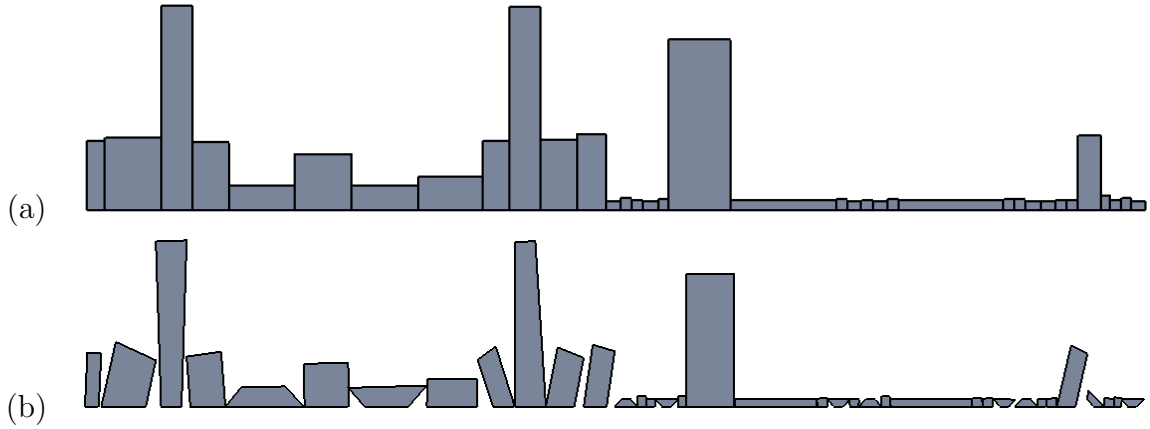
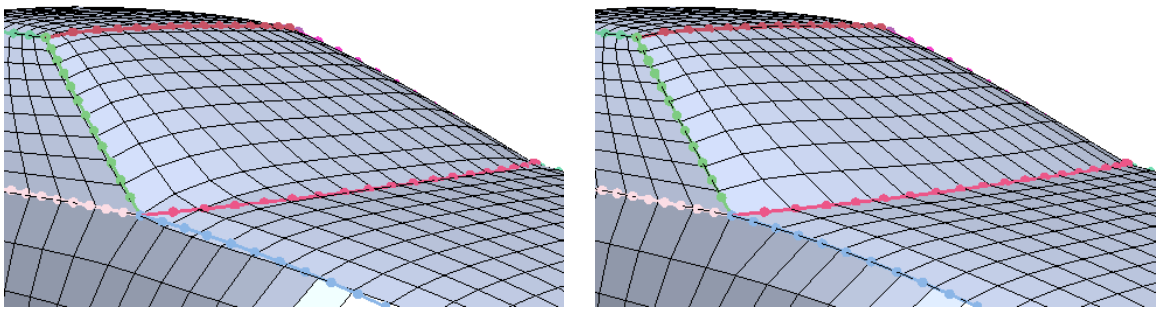


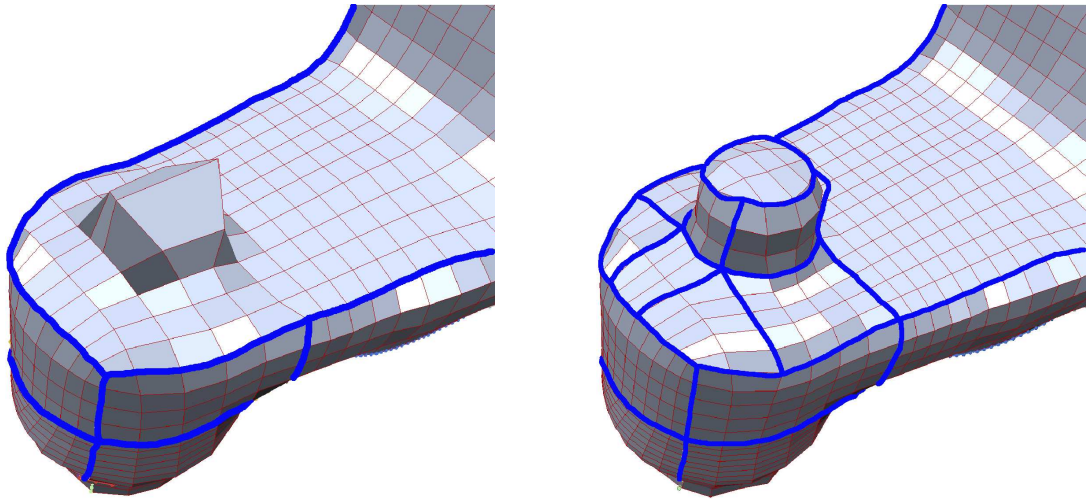
Figure 7.9.: (a) The corner positions of the car model's charts are optimized to lie on a rectangle. The resulting parameterization maximizes the isometry. (b) In this optimization the corners were allowed to lie in general position. Thus the resulting polygons are planar approximations of the surface patches.



(a) (b)

Figure 7.10.: (a) The sharp corner between front window and hood is not represented in the globally smooth parameterization without alignment constrains. This leads to sampling artifacts, i.e. triangles that cut away the sharp corner. (b) The layout curve is tagged for alignment and consequently the mesh edges are pulled onto it, leading to a better approximation of the input geometry.

curve is tagged for alignment. As one can see the isoline of the quadmesh connecting both endpoints of this layout curve in Figure 7.10a is pulled onto the layout curve in Figure 7.10b without introducing unnecessary distortion. However by using alignment constraints the computation time for solving the resulting mixed least-squares linear



(a) (b)

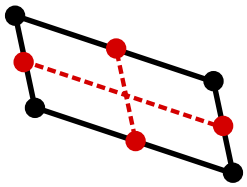
Figure 7.11.: (a) A rough layout (highlighted in blue) leads to large length distortions near a geometric feature. (b) T-vertices can be used to locally refine the layout with minimal effort. The new layout captures the geometric feature much better and avoids the length distortions.

system is higher because of the doubled dimension.

The third example is the rockerarm model from Figure 7.1. For this mechanical part first a coarse layout was designed to guide the meshing. Figure 7.11a shows a close up from the backside where large distortions appear near a geometric feature, not represented in the layout. In 7.11b the layout was locally refined by using T-vertices. In this way the designer can control the reverse engineering procedure hierarchically by starting with a rough layout which is refined until all features are captured up to the desired accuracy. The overall effort to design a layout is strongly reduced by using T-vertices.

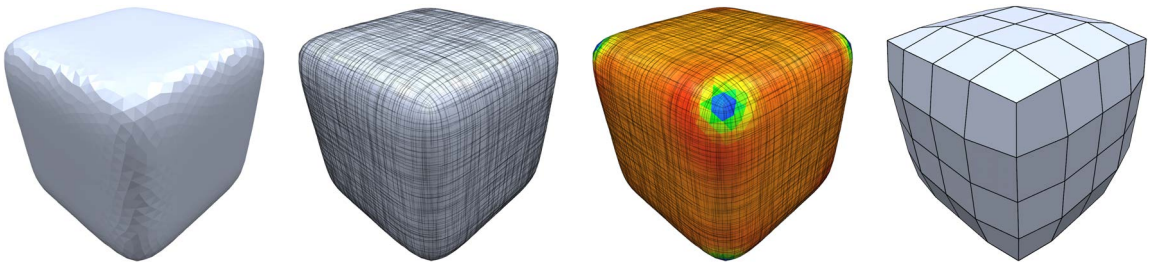
8. Orientation-field guided Approach

This class of methods is characterized by explicit control over local properties of quad elements in the mesh by means of the guiding fields [BLP*12]. Typically, the most interesting local properties are the orientation and the size of quad elements which can be specified by a *cross field*, also called *frame field*, which smoothly varies over the entire surface. A single cross can be seen as the representative of a parallelogram which is formed by parallel translation of both intersecting lines, as illustrated on the right. For each cross there are essentially four degrees of freedom that can be encoded in different ways. Often a cross field is given in a polar representation where we split the cross into its angular and length components which are then stored in two individual fields, namely an *orientation field* and a *sizing field*. Important subclasses with a reduced number of degrees of freedom (DOF's) are *4-symmetric direction fields* [RVLL08, LJX*10] which represent orthogonal crosses where both orientations are rigidly coupled and *isotropic sizing fields* where both lengths are equal.



A cross field exhibits the same types of singularities that can be observed in quad meshes and consequently the generation of a highly regular quad mesh is strongly related to the generation of a cross field with few singular points. Depending on the application, a cross field can be either designed manually or generated automatically. Automatic methods are typically driven by principal curvature information which can be shown to optimize the approximation quality [D'A00].

Apart from the pure guidance point of view, note that field guided methods decompose the difficult quad mesh generation problem into several simpler subproblems. This advantage alone motivates their usage since in each sub-step different aspects of the quad mesh can be optimized individually which turns out to be much more tractable than optimizing all aspects simultaneously. A prototypical field guided method is depicted in Figure 8.1 which consists of three steps:



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

Figure 8.1.: Prototype of a field guided method: Given an input triangle mesh (a) in the first step an orientation field (b) is computed which represents the local rotation of quad elements. In the second step a sizing field (c) is determined which specifies the sample density, which in this example is isotropic and close to uniform, with slight deviations color coded from blue to red. In the third step, a consistent quadmesh (d) is generated that closely reproduces both guiding fields.

1. Orientation field generation
2. Sizing field generation
3. Quad mesh synthesis exploiting the results of 1 and 2.

One advantage of field guided methods is that in each step the most suitable data representation can be chosen independently of the other steps. For example, a polar representation is often more powerful for steps 1 and 2 while a vector based representation may be preferred in step 3. The downside of this decomposition is that it is more difficult to integrate direct optimization of quadrangulation quality measures into the choice of cone locations which are determined at step 1. An iteration repeating the steps, and using information from step 3 in step 1 and 2 offers one possible solution.

In the following we will present our orientation-field guided approach, which is based on [BZK09]. In the first part we restrict to a constant sizing field, leading to almost uniform quadmeshes. Other possibilities for the generation of reasonable sizing fields are given in Section 8.3. Figure 8.2 illustrates the four steps of our orientation-field guided quadmesh generation algorithm. First the salient orientations, where the orientation of the quad elements is important to achieve a good approximation quality, are identified Figure 8.2a. In the second step, shown in Figure 8.2b, the salient orientations are smoothly extrapolated over the surface to achieve a dense orientation field. This orien-

tation field is used to guide a parametrization, which induces an integer-grid mapping (Figure 8.2c) reproducing the orientation-field singularities. Finally, the quadmesh can be extracted by integer iso-line contouring as illustrated in Figure 8.2d. Each of these steps will be explained in the subsequent sections.

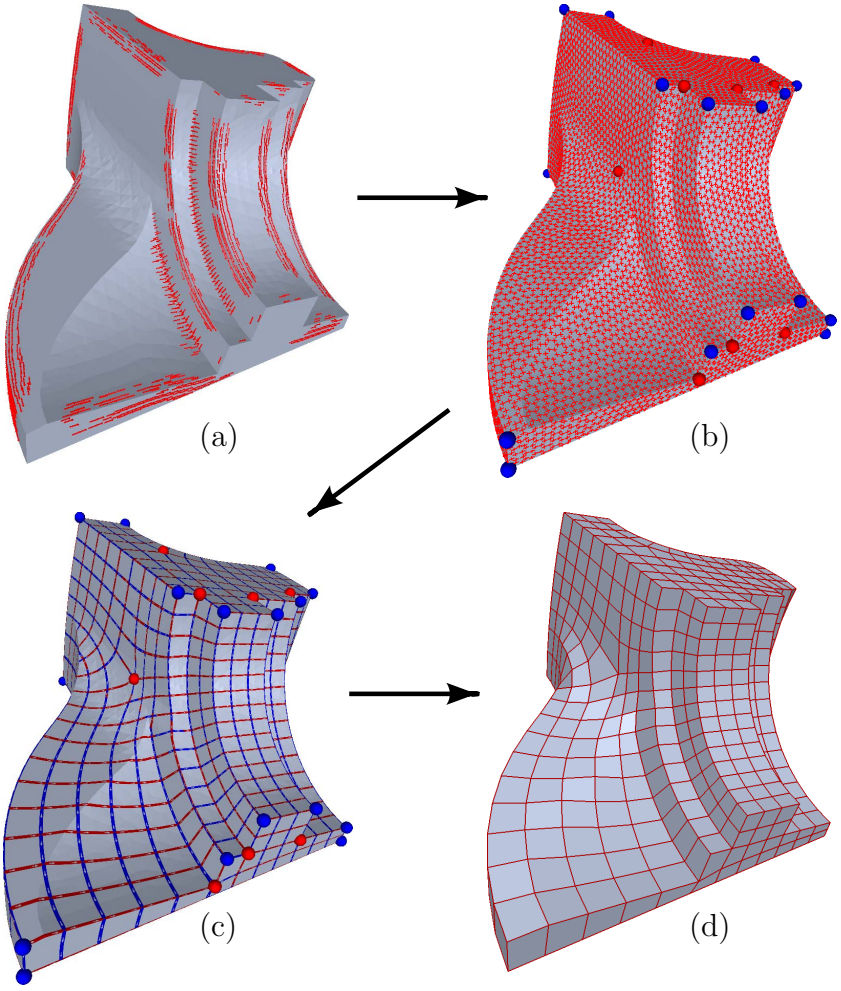


Figure 8.2.: Quadrangulation example: (a) A sparse set of conservatively estimated orientation and/or alignment constraints is selected on the input mesh by some simple heuristic or by the user. (b) In a global optimization procedure a cross field is generated on the mesh which interpolates the given constraints and is as smooth as possible elsewhere. The optimization includes the automatic generation and placement of singularities. (c) A globally smooth parametrization is computed on the surface whose iso-parameter lines follow the cross field directions and singularities lie at integer locations. (d) Finally, a consistent, feature aligned quadmesh can be extracted.

8.1. Filtering of Salient Orientations

In the vicinity of flat or umbilic points, the principal curvature directions are ill defined. Consequently, using the principal curvature directions as a dense guiding field for quadrangulation leads to suboptimal results. Typical artifacts are noisy directions with badly placed singularities or even clusters of unnecessary singularities. Generally, these artifacts cannot be removed by cross field smoothing algorithms, since the configurations often form local minima.
Therefore, in contrast to other methods, we aim at finding the smoothest cross field, interpolating only sparse directional constraints that can be found in a reliable manner.

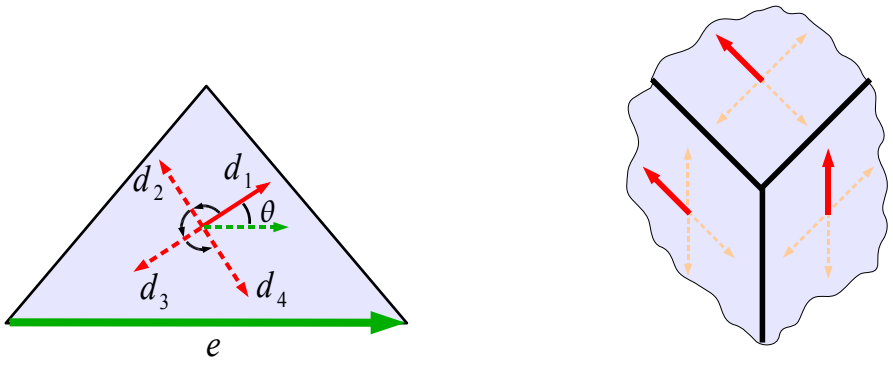
The directions we want to identify are in the spirit of feature lines, as computed in [HPW05]. However in our case a simple heuristic which robustly identifies parabolic regions is sufficient. Since parabolic regions are equipped with a well-defined orientation they are the best candidates to guide a quadrangulation. Parabolic regions can be identified by measuring the relative anisotropy of the principal curvatures

$$\tau = \frac{||\kappa_{max}| - |\kappa_{min}||}{|\kappa_{max}|} \in [0, 1]$$

which is defined to be zero, if κ_{max} is zero.

Computing meaningful curvatures on discrete triangle meshes is involved. A common technique is evaluating the shape operator [CSM03] of a geodesic disk near a point $\mathbf{p}$. But depending on the radius r we will get different estimates. To achieve a more stable result we compute for each point a set of shape operators S_r with different geodesic radii $r \in [r_0, r_1]$ and select the most promising one with a simple heuristic. A shape operator S_r is said to be valid if all shape operators in the interval $[r - w, r + w]$ have a relative anisotropy larger than a prescribed threshold τ_{min} and a mean curvature larger than K to exclude almost flat regions. For all points which provide a valid shape operator, we add a directional constraint. If there are multiple valid candidates for a single point we choose the one with the most stable direction, i.e. the one with the minimal angle deviation within its interval.

Fortunately all necessary coefficients of this heuristic have an intuitive meaning. Appropriate directions should be stable within a range depending on the target edge length h. Following this observation we choose $w = h/4$. Furthermore in our experiments we chose r_0 to be the average length of all triangle edges, $r_1 = h$, $\tau_{min} = 0.8$ and $K = 0.1/b_s$, where b_s is the radius of a bounding sphere. In general the quadrangulation result is not



(a) (b)

Figure 8.3.: (a) The four cross field directions in a triangle are parametrized by the angle θ w.r.t. a local reference edge e. (b) Depicts a smooth cross field in the vicinity of a cube corner, where the red arrows reflect the corresponding period jumps.

very sensitive w.r.t. these parameters, since similar cross fields can be generated with a large range of different sparse constraints, generated with slightly different parameters.

8.2. Orientation-field Generation

In this section we will use the elegant formalism for N-Symmetry direction fields [RVLL08] where a cross field ($N = 4$) on a triangle mesh $M = (V, E, F)$ is defined by an angle-field $\theta : F \mapsto R$ assigning a real number to each face and a period-jump field $p : E \mapsto \mathbb{Z}$ assigning an integer to each edge. The main idea is to use the angles θ to determine a single unit length vector-field which then extends to a symmetric cross field by applying three rotations of $\frac{\pi}{2}$ as shown in Figure 8.3a. Because a cross consists of four vectors between neighboring triangles it is necessary to specify which vector of the first cross is associated with which vector of the second cross. All these topological issues are handled by the period-jumps, as illustrated in Figure 8.3b for a smooth cross field near the corner of a cube. In this section we will summarize only the discrete results about cross fields that we need in this chapter. For more details see [RVLL08].

Measuring cross field smoothness After fixing the topology, measuring the smoothness of a cross field reduces to measuring the smoothness of one of the four rotation symmetric vector-fields.

The smoothness of a unit vector-field can be measured as the integrated squared curvature of the direction field. Following [RVLL08], on a discrete triangle mesh it turns out to be simply the sum of all squared angle differences between neighboring triangles:

$$E_{smooth} = \sum_{e_{ij} \in E} (\theta_i - \theta_j)^2$$

where θ_i is the angle of triangle i and neighboring angles are represented in a common coordinate frame, which is always possible by flattening both triangles along their common edge. However, for a surface with non-zero Gaussian curvature it is not possible to find a global coordinate frame. Therefore, a local coordinate frame is used for each triangle, where the x axis is identical to the first edge $\mathbf{e}$ of the triangle (Figure 8.3a). Thus, by incorporating the coordinate transformations between neighbors we can express the smoothness energy of a cross field:

$$E_{smooth} = \sum_{e_{ij} \in E} (\underbrace{\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij}}_{\theta_i \text{ w.r.t. frame } j} - \theta_j)^2 \tag{8.1}$$

where $\kappa_{ij} \in (-\pi, \pi]$ is the angle between both local frames and p_{ij} is the integer valued period jump across edge e_{ij}. The cross field index of a vertex can be computed as

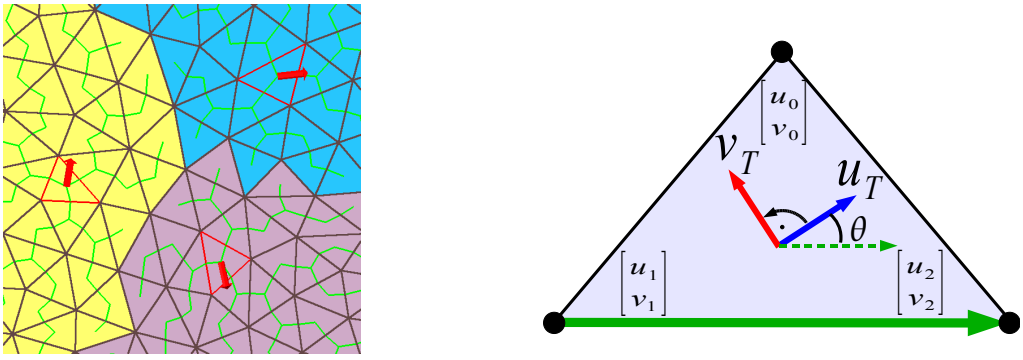
$$I(v_i) = I_0(v_i) + \sum_{e_{ij} \in N(v_i)} \frac{p_{ij}}{4}$$

with the constant integer valued base index

$$I_0(v_i) = \frac{1}{2\pi} \left(A_d(v_i) + \sum_{e_{ij} \in N(v_i)} \kappa_{ij} \right)$$

and $A_d(v_i)$ is the angle defect of vertex v_i. Only singularities of the cross field have a nonzero index which is always a multiple of $\frac{1}{4}$ [RVLL08], e.g. $\frac{1}{4}$ and $-\frac{1}{4}$ for quadrangulation configurations corresponding to valence 3 and 5 respectively.

Finding a smooth, interpolating cross field Equipped with these basic definitions we are ready to formulate the optimization problem. Given a mesh M and a subset of faces $F_c \subset F$ with constrained directions $\theta_i = \hat{\theta}_i$, we search for the smoothest interpolating cross field, i.e. we want to minimize (8.1). Accordingly we have to find an integer p_{ij}



(a) (b)

Figure 8.4.: (a) The three constrained faces (red) are the roots of dual spanning trees (green) covering the respective Voronoi cells. Each cell contains only one constraint and along all branches of the tree zero period jumps can be propagated without changing the total smoothness energy. (b) With the angle θ w.r.t. the local reference direction (green) the cross field directions $\mathbf{u}_T$, $\mathbf{v}_T$ can be extracted and used for the parametrization. In the computation two linear scalar functions (u, v) are sought whose gradients are oriented consistently with the cross field directions.

per edge and a real valued angle θ_i per face.

Reducing the Search Space: Up to here there is a whole space of equivalent minimizers to the energy (8.1). To understand this, assume we have already computed a minimizer which for one triangle provides the angle θ_0 and the three period jumps p_{01}, p_{02} and p_{03}. If we now rotate the vector by a multiple of $\frac{\pi}{2}$, i.e. set $\tilde{\theta}_0 = \theta_0 + k \cdot \frac{\pi}{2}$ and compensate this change by updating the affected period jumps to $\tilde{p}_{0i} = p_{0i} - k$, the smoothness energy is unchanged. We can repeat this procedure for all free triangles $f \in F \setminus F_c$. Consequently the solution can be made unique by fixing one period jump per free triangle to an arbitrary value, e.g. zero, without changing the energy of the minimizer. Care should be taken not to fix edges whose dual path connects two constrained faces, as done in [RVLL08], or closes loops because in these cases the cross field curvature along this path would be fixed to an arbitrary value and is not the intended result of the minimizer.

A valid set of edges, whose period jumps are allowed to be set to zero, can be found by constructing a forest of Dijkstra trees of the dual mesh as shown in Figure 8.4. Each

constrained face in F_c is the root of a separate tree such that no tree connects constrained faces. The number of fixed edges is exactly $|F \setminus F_c|$ since starting from the constrained faces each other face of the mesh is conquered by adding a single edge. Notice that no dual loop can be closed by a tree structure, such that we end up with a valid set of edges which can be fixed to zero period jumps without changing the energy of the minimizer.

Obviously there are many other valid sets of edges which could be fixed. The reason why we use trees living in the discrete Voronoi cells of the corresponding constrained faces is that this choice minimizes the length of a path to its corresponding constraint and so improves the accuracy of the greedy mixed-integer solver.

Additionally to the period jumps on tree edges each period jump between two adjacent constrained faces f_i and f_j can be fixed to $p_{ij} = round(2/\pi(\hat{\theta}_j - \hat{\theta}_i - \kappa_{ij}))$, since p_{ij} is only part of a single quadratic term in (8.1), which is independent from other variables.

In summary we end up with a mixed-integer problem consisting of $|F \setminus F_c| \approx 2|V|$ real valued variables θ_i and $|E| - |F \setminus F_c| \approx |V|$ integer valued variables p_{ij}.

Mixed-Integer Formulation: To apply the greedy mixed-integer solver from Chapter 5 it is sufficient to assemble the system of linear equations by setting the gradient of the energy (8.1) to zero:

$$\frac{\partial E_{smooth}}{\partial \theta_k} = \sum_{e_{kj} \in N(f_i)} 2(\theta_k + \kappa_{kj} + \frac{\pi}{2} p_{kj} - \theta_j) \stackrel{!}{=} 0 \tag{8.2}$$

$$\frac{\partial E_{smooth}}{\partial p_{ij}} = \pi(\theta_i + \kappa_{ij} + \frac{\pi}{2} p_{ij} - \theta_j) \stackrel{!}{=} 0 \tag{8.3}$$

Notice that the values on edges are antisymmetric, i.e. $p_{ij} = -p_{ji}$ and $\kappa_{ij} = -\kappa_{ji}$, which can lead to sign changes in equations (8.2) and (8.3). For all variables which are not fixed, we set up a row and assemble all of them into a single matrix. After applying our greedy mixed-integer solver, the result is a smooth cross field where the integer valued period jumps define type and position of all singularities. Figure 8.5 compares the result of our greedy solver with that of a direct rounding, where red and blue spheres represent singularities with negative and positive index respectively.

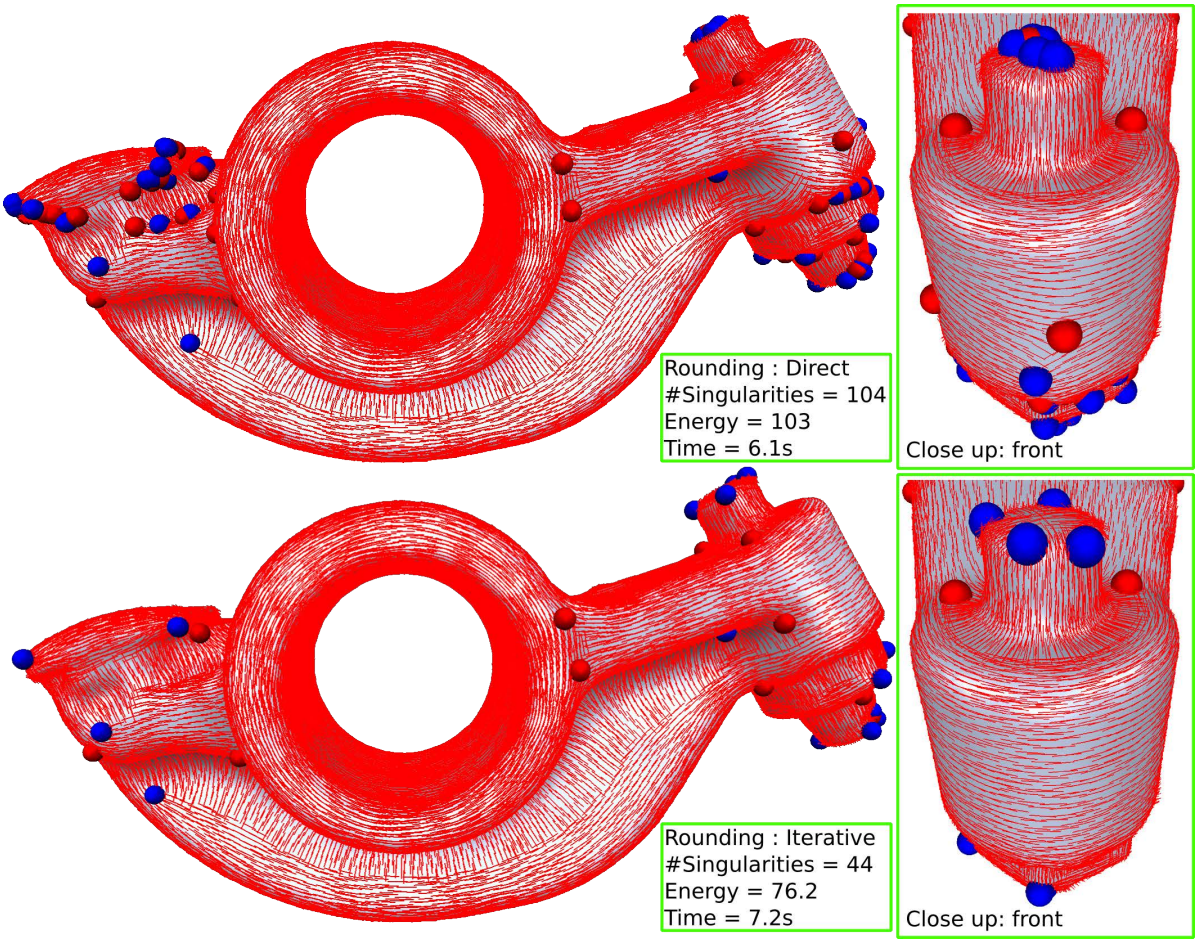


Figure 8.5.: Greedy rounding yields a smaller smoothness energy and fewer singularities (bottom), whereas the direct rounding produces unnecessary singularities and a higher energy (top). Note that these are the singularities and the field as they emerge from the solver, no singularity optimization has been carried out.

In practice we observed that some singularity positions, especially those in flat regions, can sometimes be improved by a local search algorithm, as described in the next section.

Local Search Singularity Optimization: In a postprocess we optionally check for each singularity, if the energy can be decreased by moving it to a neighboring vertex. Moving a singularity along an edge e_{ij} means changing the corresponding period jump p_{ij}. Notice, that by this operation only the right-hand-side of the linear system is changed. Consequently we can pre-calculate the sparse Cholesky factorization

of this matrix once and then compute solutions for different right-hand-sides efficiently [BBK05].

8.3. Sizing field computation

Depending on the application, the sizing field can be computed in different ways. The shape of the elements in the quad mesh can be influenced by the type of the sizing field which can be either isotropic or anisotropic. If squares are preferred, an isotropic sizing function should be chosen, while an anisotropic one offers the possibility to create rectangles by controlling two independent sizing values as was done in [ZHLB10].

The trivial constant sizing field is applied in the context of uniform remeshing where only a constant target edge length is specified. A second possibility is to choose the sizing w.r.t. the curvature in order to achieve a good approximation quality as proposed in [ACSD*03]. A variant of this strategy is to use *lfs* (local feature size), a more global surface characteristic that corresponds to both curvature and local thickness of the surface [AB99].

The third often-used strategy is to compute a sizing field which is compatible with the desired orientation field. To understand the rationale behind this methodology, imagine a cone with a smooth orientation field that diverges from the apex to the base. Clearly a quad mesh which interpolates these orientations, like e.g. a polar parametrization w.r.t. the apex, requires an increasing sizing function in the angular coordinate direction. As observed in [RLL*06], it is feasible to generate a quad mesh that exactly matches a cross field only if the curl of the cross field is zero. Therefore, if precise orientation reproduction is required, it is desirable to compute a sizing field that compensates the directional variations of the cross field by resizing the quads appropriately. Since no solution exists in general, in practice a sizing field that minimizes the curl is computed [RLL*06].

8.4. Orientation-field Parametrization

We now compute a global parametrization, i.e., a map from the given mesh $\mathcal{M}$ to some disk-shaped parameter domain $\Omega \in \mathbb{R}^2$. Since the parametrization should be piecewise

linear, it is sufficient to assign a (u, v) parameter value to each vertex — more precisely to each triangle corner — in the mesh.

The parametrization should be locally oriented according to the optimized cross field from Section 8.2 which implies that the gradients of the piecewise linear scalar fields u and v defined on the mesh $\mathcal{M}$ should minimize the local orientation energy

$$E_T \;=\; \|h\nabla u - \mathbf{u}_T\|^2 + \|h\nabla v - \mathbf{v}_T\|^2$$

for each triangle T. Here h represents the sizing field which controls the edge length of the resulting quad mesh. The vectors $\mathbf{u}_T$ and $\mathbf{v}_T$ are two orthogonal vectors in T corresponding to the cross field directions θ and $\theta + \pi/2$. Since the cross field is defined only up to rotations by $\pi/2$ we will have to specify which of the four possibilities we are picking in each triangle such that the proper compatibility conditions are satisfied across each edge in the mesh.

The global orientation energy is then defined as the integral of E_T over the entire mesh $\mathcal{M}$

$$E_{orient} \;=\; \int_{\mathcal{M}} E_T \, dA \;=\; \sum_{T \in \mathcal{M}} E_T \, area(T). \tag{8.4}$$

The minimizer of this quadratic functional is obtained by solving the sparse linear system which sets all the partial derivatives of E_{orient} to zero.

Cutting the mesh: In order to be able to compute a proper parametrization minimizing E_{orient} we have to cut open the mesh $\mathcal{M}$, such that we obtain a patch that is topologically equivalent to a disk. An additional requirement is that all singular vertices must lie on the cut, i.e. , at the boundary of the parameter domain. The reason is that the angle defect of a singularity cannot be represented by an inner vertex of the parametrization as depicted in Figure 8.6. We compute an appropriate cut graph in two steps.

First we start from a random triangle and grow a topological disk by constructing a dual spanning tree. Thus the primal of all non spanning tree edges is already a cut graph which transforms $\mathcal{M}$ into disk topology. The size of this cut graph can be significantly reduced by iteratively removing all open paths.

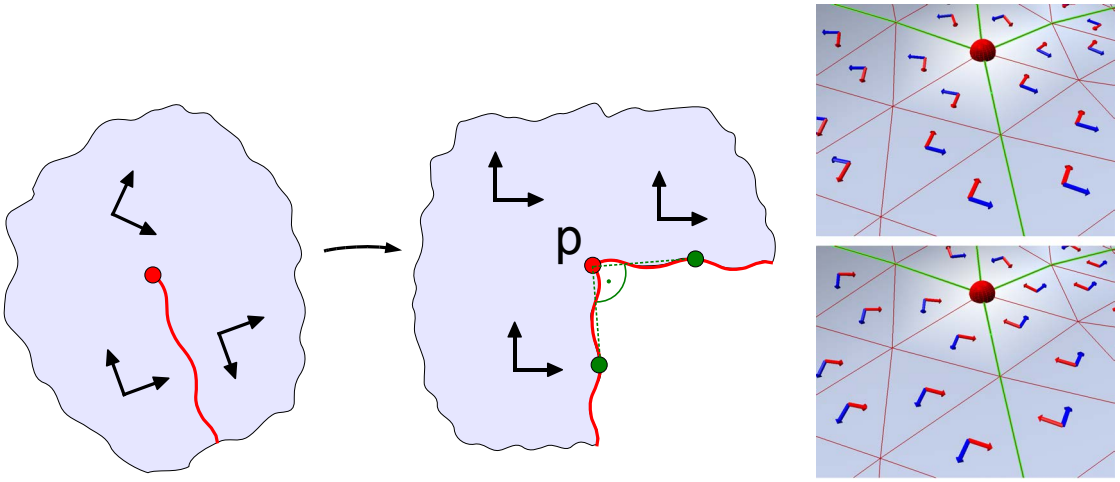


Figure 8.6.: (a) By placing a cut to a cone singularity **p** (here of index $\frac{1}{4}$) a distortion free unfolding of the patch is possible. (b) The upper image shows two directions of the cross field. In the lower image the mesh is cut into disk topology along the green edges, such that these directions can be consistently oriented on each side of the cut.

In the second step paths connecting each singularity to the cut graph are added. This can be done by successively applying Dijkstra's shortest path algorithm.

At the end of the two cutting steps we have a triangle mesh patch where all the singularities are located at the boundary. If a singularity is not a leaf node of the cut graph then it appears several times along the boundary. In order to compute a parametrization we have to find a planar embedding of this boundary polygon as well as all the interior vertices. The location of the mesh vertices in the parameter domain is computed by minimizing E_{orient}, however, there are a number of consistency constraints that have to be taken into account.

Integer location of singularities: By allowing a singularity to be in general position, it would cause an n-sided face instead of a valence-n vertex. Therefore to guarantee a pure quadrangulation, we have to snap all singularities to integer locations in the parameter domain. This means that the overall parametrization task is now a mixed-integer problem which we solve by our mixed-integer greedy solver from Chapter 5.

Cross boundary compatibility: In order to avoid visible seams across the cut paths on the surface we have to make sure that the quad structure on both sides of a cut edge is compatible. This is guaranteed by allowing only a grid automorphism as a transition function. This requires that the (u, v) parameter values on both sides of a cut edge are related by

$$(u', v') \;=\; Rot^{i}_{90}\,(u, v) + (j, k)$$

with integer coefficients (i, j, k).

The rotation coefficient in the transition functions can easily be computed by propagating a globally consistent orientation in the cross field, as illustrated in Figure 8.6 . Since after the cutting, all interior vertices of the mesh are regular, we can start at a random face and propagate its orientation in a breadth first manner to all the neighboring faces. This will establish a zero-rotation across all inner edges. The rotations Rot^{i}_{90} across the cut edges can be found by simply comparing the orientations in neighboring faces.

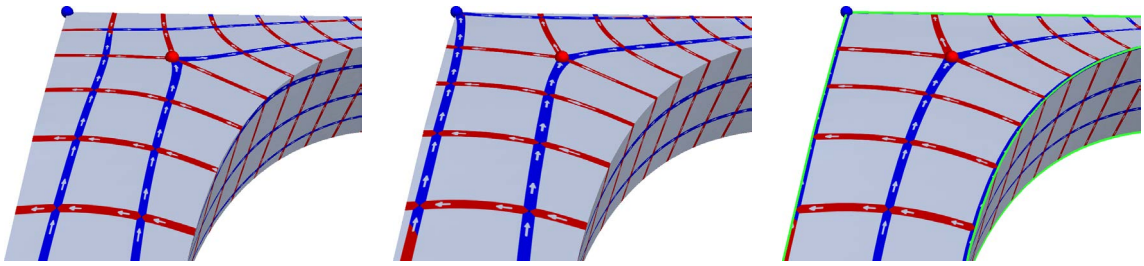
After fixing the rotations, the cross boundary compatibility conditions can be incorporated into the optimization scheme as linear constraints. Therefore for each cut edge $\mathbf{e} = \overline{\mathbf{p}\,\mathbf{q}}$ we introduce two integer variables $j_{\mathbf{e}}, k_{\mathbf{e}}$ to formulate the four compatibility conditions:

$$\begin{aligned}(u'_p, v'_p) &= Rot^{i_{\mathbf{e}}}_{90}\,(u_p, v_p) + (j_{\mathbf{e}}, k_{\mathbf{e}})\\ (u'_q, v'_q) &= Rot^{i_{\mathbf{e}}}_{90}\,(u_q, v_q) + (j_{\mathbf{e}}, k_{\mathbf{e}})\end{aligned}$$

Hence, in total we add two integer variables and eliminate four continuous variables per cut edge.

Applying our mixed-integer greedy solver to this parametrization task can be understood in an intuitive way. After computing an all-continuous solution, which corresponds to the unconstrained parametrization, we iteratively snap the singularities to integer locations.

Anisotropic Norm In practice precise orientation is often more important than exact edge length. The reason is that changing the orientation along a highly curved feature



(a) (b) (c)

Figure 8.7.: The parametrization in (a) is not aligned to the sharp edges of the object. Using the anisotropic norm the quads are allowed to stretch in order to better align with a given input field as shown in (b). In (c) alignment constraints have been imposed, leading to perfectly preserved features.

line, the quadrangulation quality will drop off dramatically due to normal noise. The orientation can be improved by less penalizing stretch which is in the direction of the desired iso-lines. This can be achieved by an anisotropic norm

$$\|(u,v)\|^2_{(\alpha,\beta)} = \alpha u^2 + \beta v^2$$

which penalizes the deviation along the major directions with different weights. Notice, such a diagonal metric is sufficient since we use $(\mathbf{u}_T, \mathbf{v}_T)$ as the local coordinate frame in each triangle.

$$E_T \;=\; \|h\nabla u - \mathbf{u}_T\|^2_{(\gamma,1)} + \|h\nabla v - \mathbf{v}_T\|^2_{(1,\gamma)}$$

with $\gamma \leq 1$. Figure 8.7b shows an example, where the orientation of the parametrization is improved by using the anisotropic norm.

Feature Line Alignment Sharp feature lines of the input mesh should be preserved in the quadrangulation. Given a subset $S \subset E$ of triangle mesh edges, the necessary alignment conditions can be incorporated in a straightforward way. First of all, alignment requires correct orientation. Therefore, while computing the cross field, all edges in S are used as orientational constraints in both adjacent triangles. Additionally to the correct orientation for alignment, a constant integer coordinate along the edge is necessary, which guarantees that this edge is preserved in the quadrangulation. Each alignment condition for an edge $\overline{\mathbf{p}\,\mathbf{q}}$ can be formulated independently. If the cross field direction

$\mathbf{u}_T$ is already oriented along the alignment edge, we end up with a simple condition for the v parameter values

$$v_p = v_q \in \mathbb{Z}$$

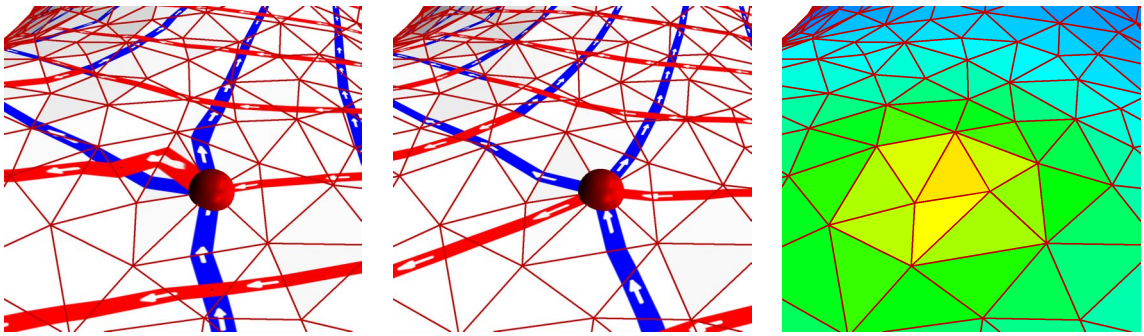
which ensures that $\overline{\mathbf{p}\,\mathbf{q}}$ is mapped to an integer valued iso-line. The $u = const$ case is handled analogously. Consequently, for each alignment edge a single variable can be eliminated and the remaining integer variables can be handled by the greedy mixed-integer solver. Figure 8.7c shows an example, where all feature edges are aligned.

Notice that for meshes with boundaries we can exploit the presented alignment functionality to guarantee that the boundaries are preserved in the quadrangulation and thus prevent jagged boundary lines (see Figure 8.10).

Singularity Relocation By computing a parameterization with the given sizing field h, new requirements have to be taken into account which cannot be anticipated by the cross field computation, since it is independent from h. Examples are singularities which are too close to each other, a boundary or a given alignment edge. Other aspects are symmetries which are irrelevant for a smooth cross field, but important for a quadrangulation. Therefore, to achieve maximal quality it can be necessary to relocate the singularities w.r.t. the requirements of the parameterization. This can be done with a local search algorithm similar to Section 8.2. Depending on how much time is available we can restrict the search to the best local candidate, i.e. , the closest neighbor in the parametrization, or evaluate the quality of all neighbors. In each step it is necessary to recompute the smooth cross field w.r.t. the relocated singularity as well as the parametrization. In the cross field computation the cross field indices are now prescribed by linear constraints. Movements are performed if the overall quality improves, i.e. , the energy (8.4) decreases.

The obvious drawback of this singularity relocation is its heavy computational cost. Fortunately in all of our examples the initial singularity positions were already sufficient. However, coarsely quadrangulating meshes with fine details will require singularity relocation.

Local Stiffening The parametrization is the result of a quadratic energy minimization. Thus, despite the global optimum, for a few triangles it might happen that the metric



(a) (b) (c)

Figure 8.8.: In (a) the minimized orientation energy produces flipped triangles, which can be removed by local stiffening (b). The chosen weighting is shown in (c) and decreases from orange to blue.

distortion gets very high or even worse, that the orientation of a mapped triangle flips. Figure 8.8 shows an example where such a problem occurs in the vicinity of a singularity. The idea of local stiffening is to add an adapted triangle weighting $w(T)$ into the energy formulation to penalize high local distortions, yielding:

$$E_{orient} = \sum_{T \in \mathcal{M}} w(T)\, E_T\, area(T)$$

This weighting, which is initialized to one, can be updated iteratively, as described in the following, until the quality of the parametrization is sufficient.

The metric distortion is characterized by the singular values σ_1 and σ_2 of the Jacobi matrix as described in [HLS07]. Furthermore to penalize flips we evaluate the orientation of a triangle

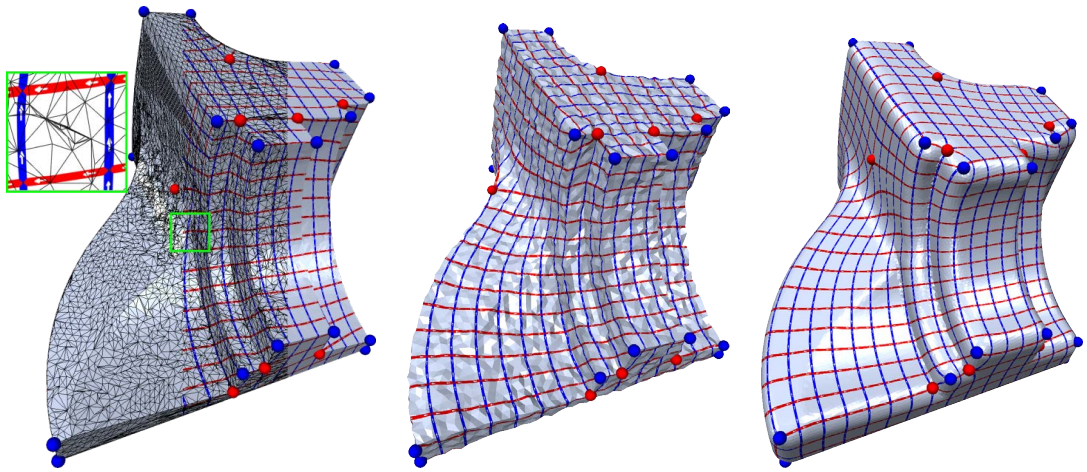
$$\tau = sign(det \begin{bmatrix} u_1 - u_0,\ u_2 - u_0 \\ v_1 - v_0,\ v_2 - v_0 \end{bmatrix})$$

where (u_i, v_i) are the vertex parameter coordinates in counter-clockwise ordering. We measure the local distortion of each triangle by

$$\lambda = |\tau \frac{\sigma_1}{h} - 1| + |\tau \frac{\sigma_2}{h} - 1|$$

which respects the edge length h. Finally, we update the weight of a triangle by evaluating a uniform Laplacian defined on the dual mesh

$$w(T) \leftarrow w(T) + min\{c \cdot |\triangle\lambda(T)|,\ d\}$$



(a) (b) (c)

Figure 8.9.: The presented algorithm is robust w.r.t. bad triangles (a) and can produce meaningful singularities in the presence of noise (b) and on smooth offset geometries (c).

with the proportionality constant c and a maximal allowed update of d, which we chose as $c = 1$ and $d = 5$ in all our examples. Notice that directly using the distortion instead of the Laplacian would not be a good idea. The reason is that the weighting would reflect the *global* stretch distribution, which is necessary for a globally consistent quadrangulation, instead of the desired *local* distortions. Subsequently, we increase the smoothness of the weighting field $w(T)$ by a few uniform smoothing steps, which in general leads to nicer quadrangulations.

8.5. Evaluation

The backbone of our approach is the mixed-integer solver introduced in Chapter 5, which is used for the computation of both the smooth cross field and the parametrization. Although it is often necessary to round tens of thousands of variables for the cross field computation, the timings in Table 8.1 show that this can be done efficiently using the proposed solver.

The example in Figure 8.5 shows that the greedy rounding leads to a significantly smoother cross field with less singularities compared to the direct rounding approach.

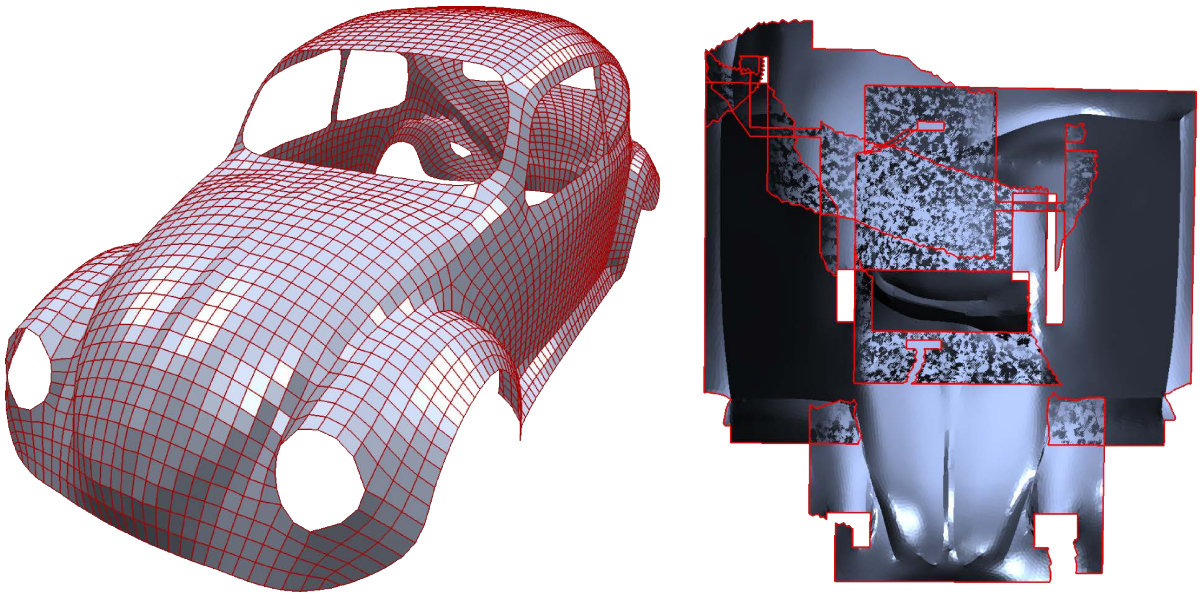


Figure 8.10.: Quadrangulation of the BEETLE model having 11 boundaries. On the right the parametrization is shown. Naturally, due to the occurrence of $-\frac{1}{4}$ singularities, parts of the flattening are overlapping.

All our experiments confirmed this behavior.

A further comparison between our approach and direct rounding is carried out in Figure 8.11. In both examples the same input cross field and target edge length have been used. The FANDISK comparison clearly shows the benefit of alignment on models with sharp feature edges, while the limitations of direct rounding are especially noticeable on the BOTIJO. On complex objects having many singularities or when remeshing with very coarse target edge length the direct rounding generates many "twists" and non-injectivities in the parametrization, such that the extraction of a hole-free quad mesh is not always possible. However, the combination of greedy rounding and local stiffening allow us to automatically generate consistent, hole-free quadrangulations at almost any resolution and with significantly less "twists".

The spectral approach [HZM*08] also produces oriented and aligned parametrizations with few singularities, however the Morse-Smale Complex sometimes fails to capture the detailed structure of the surface. This can lead to an unfavorable stretch of the quads affecting the angle as well as the edge length distribution. A comparison between

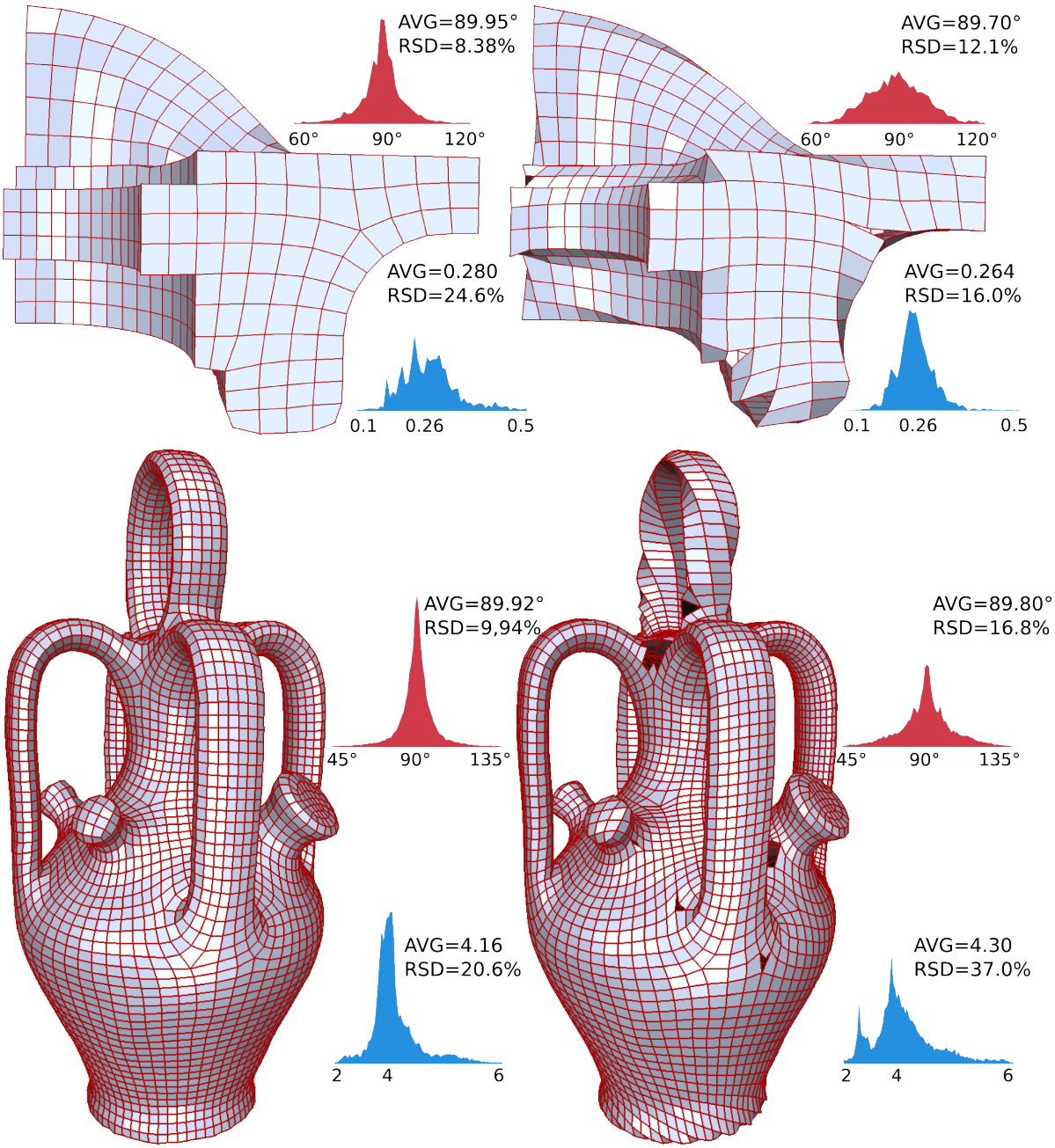


Figure 8.11.: A comparison between the our technique (left) and the direct rounding (right) for a sharp object (Fandisk) and a smooth object (Botijo). In both comparisons the same target edge length and the same cross field generated by our mixed-integer formulation were used.

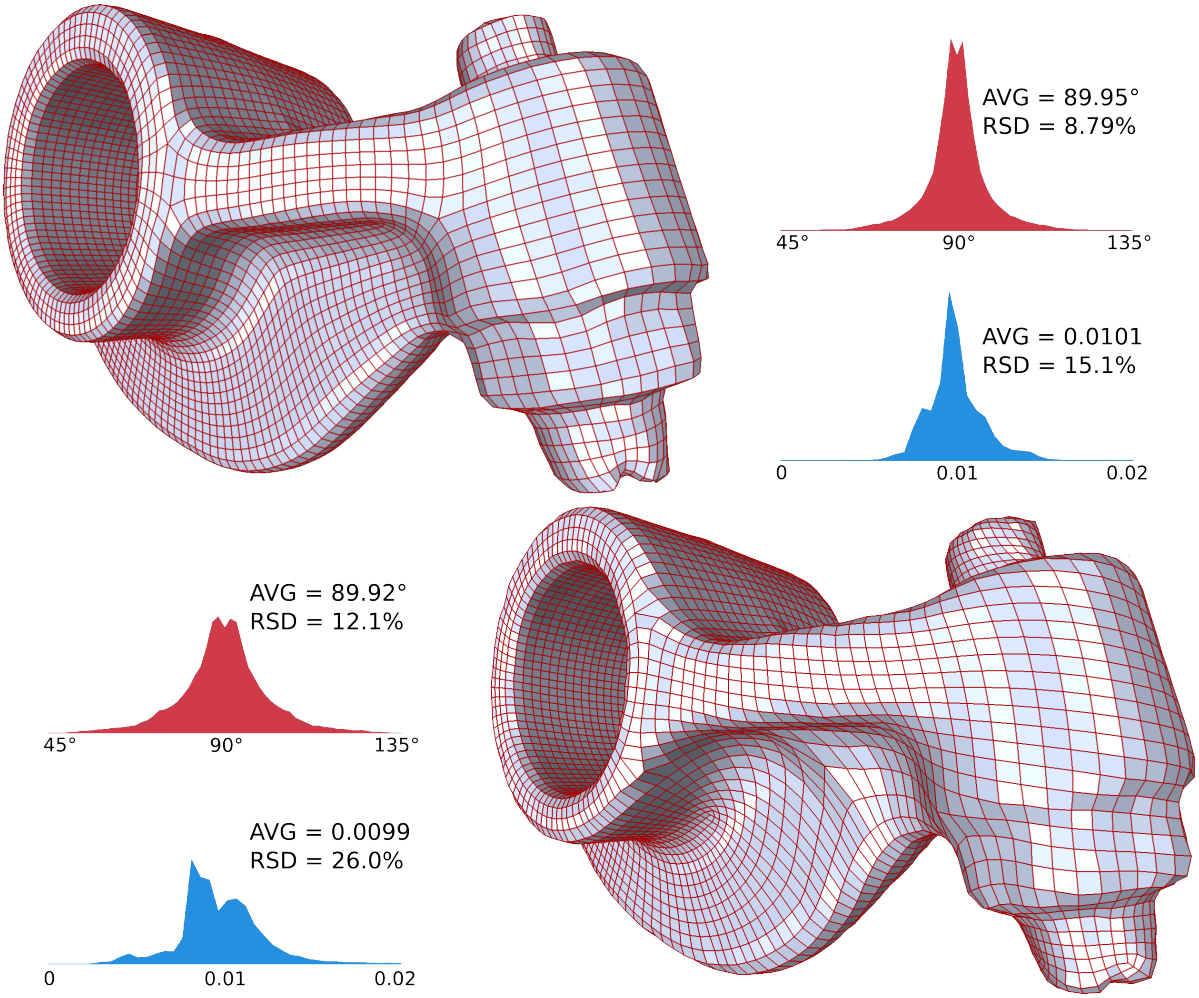
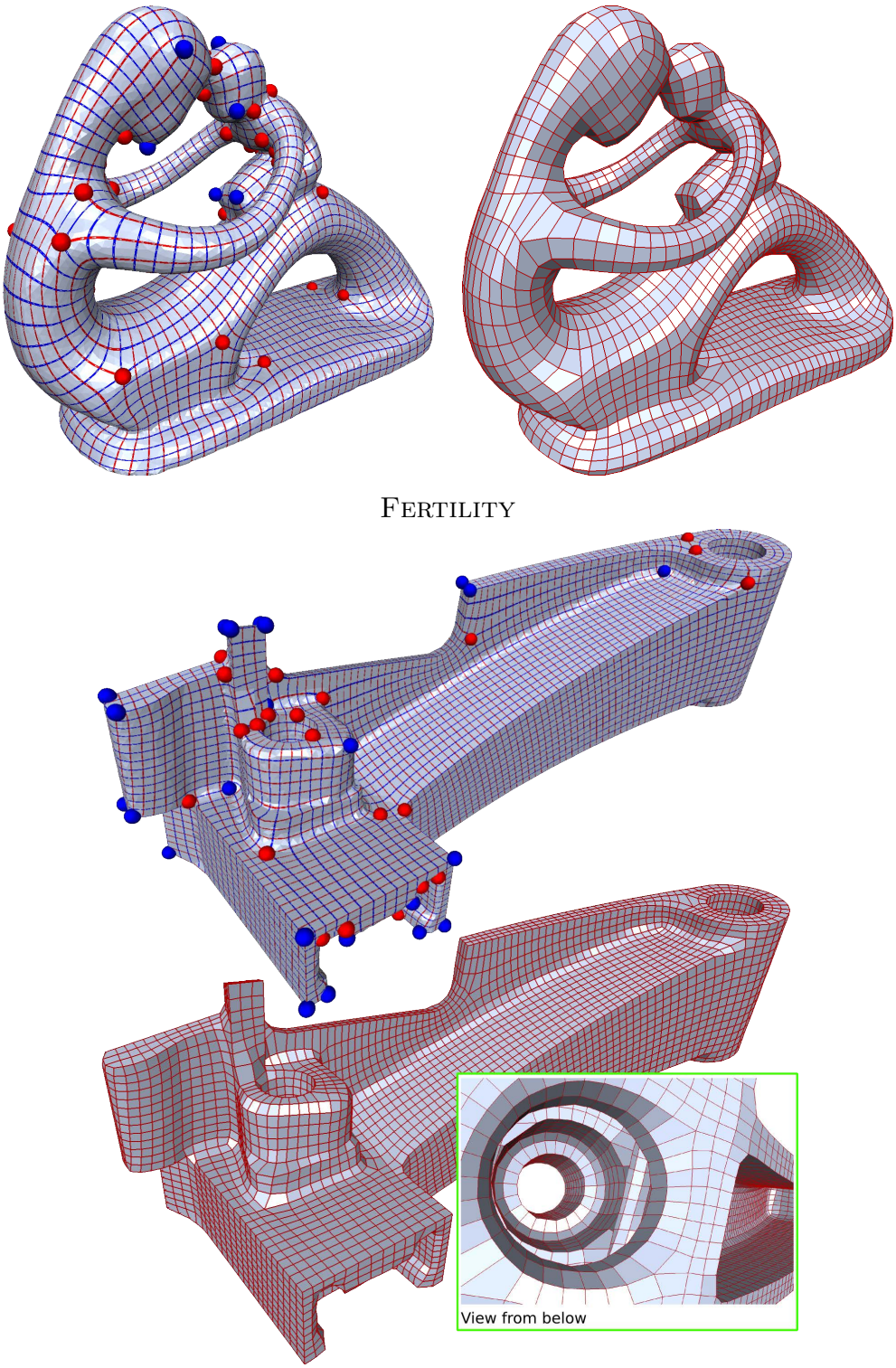


Figure 8.12.: RockerArm comparison between the technique described in this paper (top) and the spectral quadrangulation approach by [Huang et al. 2008] (bottom). The upper mesh has 9413 faces and 36 singularities, the lower one has 9400 faces and 26 singularities.

[HZM*08] and our approach is depicted by Figure 8.12.

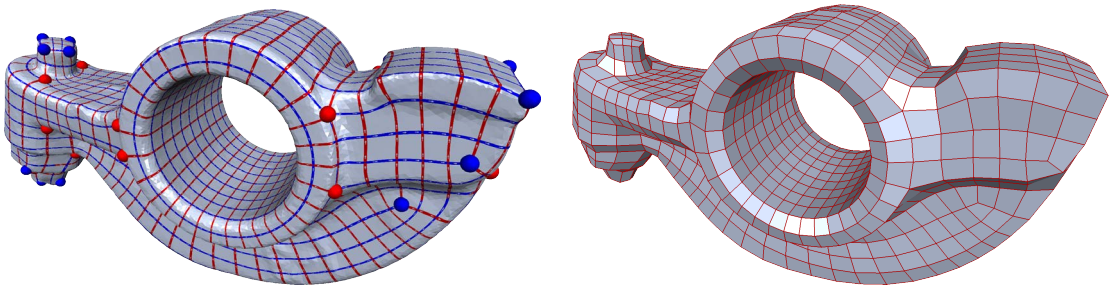
Quadrangulations computed by our technique typically have angle distributions with a sharp peak around 90° and an edge length distribution centered around the target edge length. However, for aligned meshes, like the Fandisk in Figure 8.11, further peaks, which reflect the unavoidable stretch may occur in the edge length histogram.



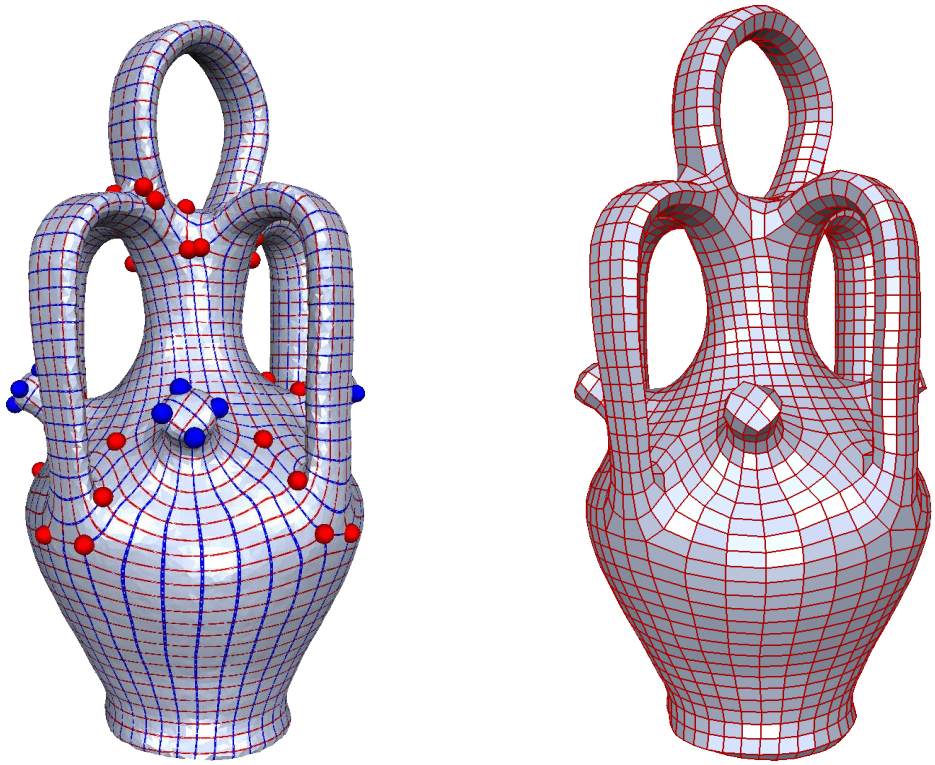
Fertility

Lever

Figure 8.13.: Results of our Mixed-Integer Quadrangulation approach I



Rocker arm



Botijo

Figure 8.14.: Results of our Mixed-Integer Quadrangulation approach II

The geometrically complex examples shown in Figure 8.13 and 8.14 underline the ability of our method to compute coarse, oriented quadrangulations with naturally placed singularities.

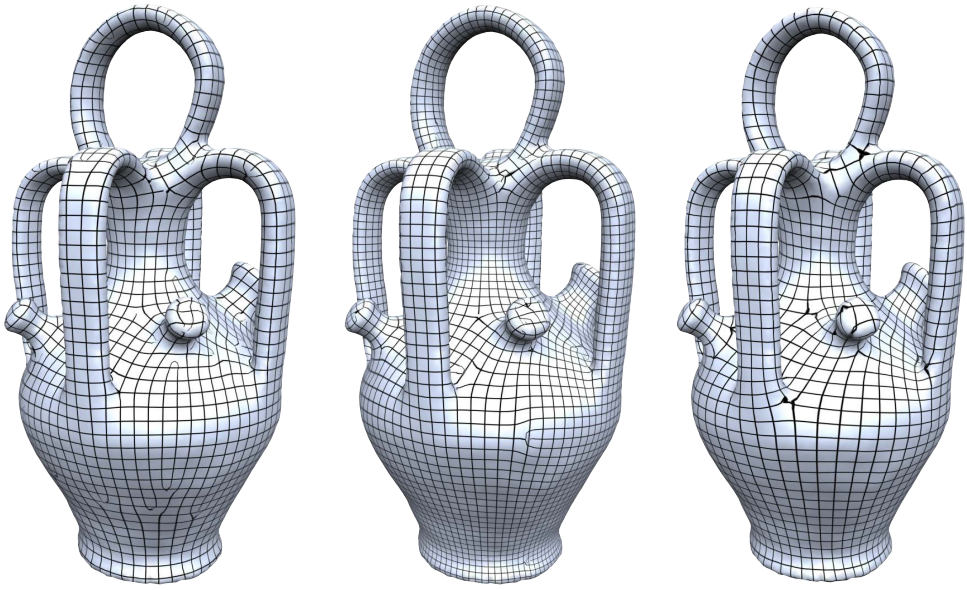
All examples were computed on a 3.0GHz standard PC, the statistics are shown Table 8.1. Interestingly the cross field computation is less demanding to compute than

	Solver Statistics						Quadmesh Statistics	
	Cross Field			Parametrization				
Model	Dim	#Int	Time	Dim	#Int	Time	#Sing	#Quad
Fertility	29342	10621	0.6s	27954	108	2.0s	48	3357
Lever	21113	8254	0.3s	19331	148	1.9s	83	7880
RockerArm	32843	12064	0.7s	41552	72	2.0s	36	1127
Botijo	25821	9611	0.6s	29994	164	2.7s	74	8395
Fandisk	17820	6447	0.2s	13776	19	0.3s	30	764
Beetle	46425	15827	0.7s	34705	82	1.5s	6	3778

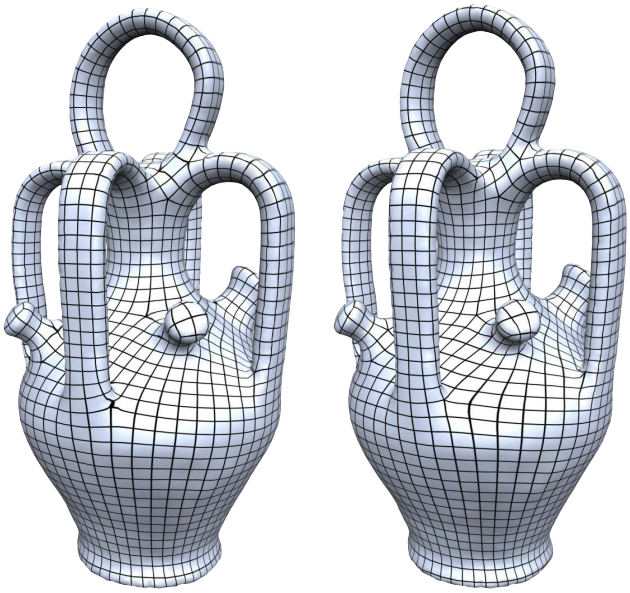
Table 8.1.: Statistics of the Greedy Mixed-Integer Solver used for computing the cross field (Section 8.2) and the parametrization (Section 8.4). Dim refers to the initial dimension of the linear system, #Int is the number of integer variables, #IS and #DS is the number of calls to iterative and direct solvers respectively. Time is the total time for the solution. Due to the global nature of the parametrization, the local and iterative search seldom lead to a gain of efficiency and therefore Time refers solely to the direct solver.

the parametrization, even though it requires practically two orders of magnitude more roundings. This effect is due to the locality of the cross field energy (Equation (8.1)). Rounding a period jump mainly affects a local neighborhood on the mesh and the solution can be efficiently updated by local Gauss-Seidel iterations. Whereas, rounding a corner point in the parametrization domain usually has global impact. Motivated by this observation and the typically low number of integer variables for the parametrization, we used two different parameter sets for the solver, as explained in Section 5.3.

Finally Figure 8.9 demonstrates the robustness of the mixed-integer quadrangulation approach w.r.t. different (degenerate) representations of a given object. The mesh in Figure 8.9a contains almost 1000 triangles with vanishing area (the close-up shows a part of the mesh where about 8 triangles are nearly collinear), the model in Figure 8.9b has been displaced by normal noise with a magnitude of 0.3% of the bounding box diagonal and the right most model (Figure 8.9c) was offseted, yielding a mesh with smoothed corners. These fandisks and most of the other triangle meshes used in this work (along with the extracted quad meshes) can be found in the supplementary material of the corresponding paper [BZK09].



(a) PGP [8s] (b) PGP+CC [10s] (c) QC [0.5s]



(d) MIQ [3s] (e) MIQ+ST5 [15s]

Figure 8.15.: Quad mesh synthesis comparison based on identical guiding fields: The PGP method provides the best length distortion at the cost of additional singularities (a). This effect can be reduced by a curl corrected sizing field (b). QC and MIQ are based on the same function space construction and consequently behave similarly with a clear trade-off between mapping distortion and runtime due to different heuristics for the estimation of integer DOF's (c) and (d). The mapping distortion can be further reduced by the iterative stiffening approach (e).

Comparison of parametrization based methods. In order to investigate the behavior of different parametrization based methods, Figure 8.15 compares the quad mesh synthesis of *Periodic Global Parameterization* (PGP) [RLL*06], *QuadCover-Surface Parameterization using Branched Coverings* (QC) [KNP07] and our *Mixed-Integer Quadrangulation* (MIQ) explained in this chapter against each other. For all methods the same guidance fields are used which consist in an orientation field produced by the PGP method and a constant sizing field. The only exception is Figure 8.15b where the sizing field was adjusted by the curl correction method proposed together with the PGP method in [RLL*06].

Figure 8.15 shows that all synthesis methods behave quite similar in regular regions, showing that the orientation and sizing fields have a strong influence on the result. As mentioned before, a suitable distribution of singularities in the orientation field is crucial for the success of the quad mesh synthesis step. In accordance to that, our experiment shows that interesting differences mostly occur close to singularities which will be the first aspect of our discussion. As expected, the PGP method generates additional singularities in order to capture the given constant sizing field, while QC and MIQ exactly reproduce the orientation field singularities at the cost of some length distortion. Some of the additional singularities can be compensated by a curl corrected sizing function as shown in Figure 8.15b, however, the PGP method does not provide explicit control. Clearly, the favored behavior strongly depends on the application. However, in most practical applications regularity and explicit control over singularities is preferred over moderate length distortion.

QC and MIQ search for an optimal mapping within the same function space and consequently their results shown in (c) and (d) are closely related. While QC is extremely fast since it requires only the solution of two sparse linear systems, MIQ is able to estimate integers that induce less distortion at the cost of increased runtime. The impact of the integer estimation technique strongly depends on how close singular vertices get in the quadmesh. If the goal is the generation of a very coarse quad mesh, it is very important to apply more expensive integer estimation schemes like those of MIQ, while for the generation of finely tessellated quad meshes a simple and fast heuristic like the one of QC is sufficient.

The last aspect we want to analyze here is the quality of the mapping. An often neglected aspect of parametrization based approaches are degeneracies in the mapping function (e.g. foldovers) which easily destroy the quad mesh consistency and necessitate a repair step comparable to that of PGP. One reason for such defects is that singularities in a parametrization behave similar to point constraints, which are well known to often introduce heavy distortions and foldovers. Although there is currently no fundamental solution to this problem, the *stiffening* heuristic of MIQ in practice often leads to sufficient results by iteratively updating a weighting function in order to minimize the maximal distortion. Figure 8.15e depicts the solution of MIQ with 5 stiffening iterations where especially the distortion around singularities is greatly reduced, again at the cost of an increased runtime.

In summary, for the quadmesh synthesis algorithms analyzed here, there is clearly a trade-off between speed and quality. While conceptually comparable, in practice the MIQ approach is often preferred over QC since, on the one hand, it naturally handles sharp features and boundaries and, on the other hand, the required greedy mixed-integer solver is freely available [BZK12], enabling a cost-efficient implementation.

While all field guided approaches discussed here lead to valence semi-regular meshes, one interesting direction for future research includes the design of methods that are directly able to generate semi-regular meshes with a coarse patch structure. Additional constraints described in [MPKZ10] offer a step in this direction. Another important aspect which would deserve some attention is the improvement of robustness. While MIQ with stiffening is able to generate valid mappings leading to quad meshes of moderate coarseness, the construction of degeneracy-free mappings for arbitrarily coarse sizing fields is still unsolved.

8.6. Flexibility

The orientation-field guided approach, as presented here, is designed to generate high-quality quadmeshes in a fully automatic manner. However, depending on the application scenario, it might be desirable to influence the meshing process in order to achieve a mesh with additional, maybe non-geometric, properties. Such special requirements arise for instance in animation and simulation, where the designer wants to optimize the mesh

for the intended deformations or simulation characteristics. In this section, we want to point out that our method is perfectly prepared to be used in such a semi-automatic, interactive quadmeshing environment. In the following, we list several practically important meshing criteria that can be influenced by the user. The idea consists in feeding the mixed-integer solver with additional linear constraints that represent user-desired mesh properties.

Element orientations: Instead of, or additionally to, the salient principal curvatures the user can easily specify local element orientations. A convenient way consists in drawing curves onto the surface that generate constrained orientations in each traversed triangle. Apart from the extended user-interface the algorithm requires no further modification.

Irregular vertices: The irregular vertices of the quadmesh are automatically identified while searching for the smoothest orientation field that interpolates the constrained orientations. Following Section 8.2, the index $I(v_i)$, which defines irregular vertices, is linear related to the unknowns and therefore can be specified as a linear constraint. As a default, all indices are free and optimized by the mixed-integer solver. If the user explicitly wants to position irregular vertices, this can be easily achieved by adding the corresponding linear constraint. The same is true for regular regions, i.e. , vertices of index 0, which can be prescribed to prevent the generation of irregular vertices, e.g. in the vicinity of feature curves. If all indices are specified, the setting is identical to the zipping algorithm of [RVLL08], which indeed is a special case of our optimization approach. The idea of specifying a single irregular vertex directly generalizes to the integral index of a complete region. Hence, in the same way it is possible to specify the index-sum of several vertices.

Element sizing: Instead of the uniform (or otherwise generated) sizing field, the user can prescribe the sizing. This is, e.g. , helpful in an animation environment, where an increased sample density is typically desired in the vicinity of small details like the fingers of a character. However, since the generation is split into two parts, i.e. , orientation field and parametrization, the sizing cannot always be precisely reproduced (cf. [KMZ10]).

Feature lines: It is possible to prescribe complete chains of quadmesh-edges by drawing curves on the surface, which are integrated into the mesh and handled as feature lines.

In this way, not only the element orientation is controlled, but also the placement of samples.

Base-complex: The orientation-field guided approach typically leads to meshes of semi-regular valency, i.e. , a small number of irregular vertices but possibly a large number of base-complex faces. Directly incorporating the base-complex optimization within the orientation-field guided approach is a non-trivial task. However, it is easily possible to specify parts of the base-complex or even the complete base-complex manually by means of additional constraints. This is done by setting up linear constraints that ensure that two irregular vertices will be connected by a chain of quadmesh edges (cf. [MPKZ10]). Again the user is within the convenient situation that no information about the base-complex is required, but if available, it can be easily incorporated into the computation. If the complete base-complex is specified, the resulting optimization turns out to be closely related to the layout based quadmesh generation approach.

9. Geodesic Distance Fields

The computation of geodesic distances on a triangle mesh has many applications in geometry processing, ranging from segmentation and low distortion parametrization to motion planning and tool path optimization. Within the context of quadmesh generation, geodesic distance fields are useful to introduce metric aspects into the computation of a smooth orientation field as for instance to separate irregular vertices from each other or from feature curves. In most cases the true geodesic distance field is approximated by some fast marching method [KS98, NK02] which leads to acceptable results on nicely structured meshes and away from singularities of the distance function. However, such simple propagation schemes tend to become numerically unstable on not-so-nice meshes as they often occur in practical applications. Moreover, since they use the same mesh as a representation for the input geometry as well as the distance field, the precision is limited by the mesh resolution. Surazhsky et al. [SSK*05] present a practical imple-

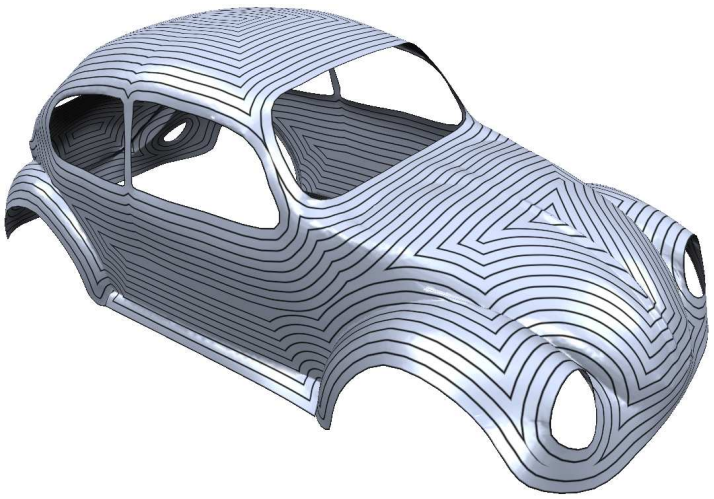


Figure 9.1.: The isolines of the geodesic distance field with respect to the boundaries of the car model are visualized.

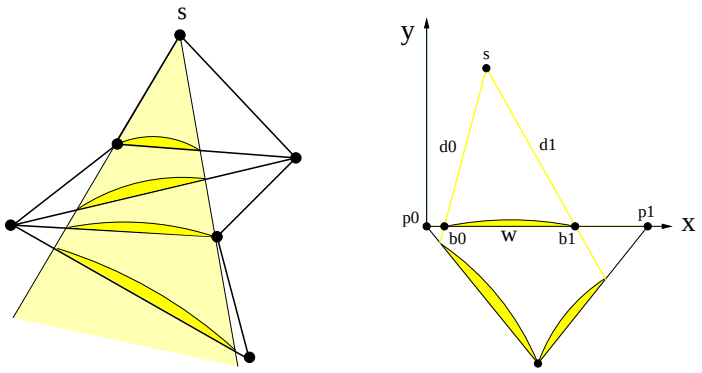


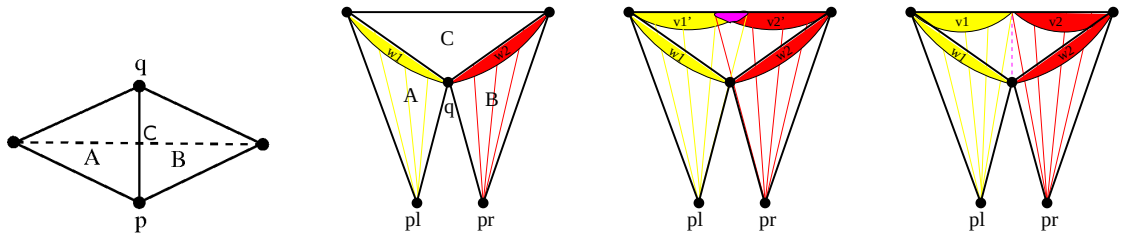
Figure 9.2.: (a) Starting on the point source **s** a (shaded) pencil of rays is propagated through three unfolded triangles along of straight lines. Each window is highlighted by an arc which is always on the edge side pointing to the source. (b) An edge aligned two dimensional coordinate system is used to compute new windows which are induced from window w.

mentation of the geodesic distance algorithm of Mitchell et al. [MMP87]. This was the first time that an exact geodesic distance computation has become applicable to arbitrary input meshes of practically relevant complexities. However, in this algorithm, the distance computation is initialized by one or more isolated points on the mesh and the distance is propagated from them - in the following, we present a summary of this algorithm. Unfortunately, for many practical applications this is too restricted. In general one would like to be able to compute the geodesic distance with respect to a curve on the surface, i.e., a polygon on the mesh since this allows us to take arbitrary boundary conditions into account. See Fig.9.1 for an example. In this chapter, based on [BK07], we derive an algorithm for this generalization.

The exact geodesic algorithm

Since our algorithm is an extension of [SSK*05] we briefly explain the basic principles and the resulting base algorithm.

In the plane, the geodesic distance coincides with the Euclidean distance. Hence, with respect to an isolated point, it is the square root of a quadratic function. On a triangle mesh, i.e. , on a piecewise planar surface, the geodesic distance with respect to a point



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

Figure 9.3.: (a) The geodesic distance field w.r.t point **p** is computed on a cap consisting of triangles A,B, and C. (b) Cutting along the edge $\overline{pq}$ unfolds the cap isometrically and enables the distance propagation in the plane through windows w_1 and w_2. (c) The temporarily propagated windows v_1' and v_2' overlap in the middle region. (d) The final windows v_1 and v_2 are properly cut to represent the piecewise geodesic distance along the edge.

turns out to be a piecewise function where in each segment the distance is given by the square root of a quadratic function plus an optional constant offset. This offset has to be introduced to properly handle saddle points on the surface.

The central idea of the algorithm [SSK*05] is to propagate exact distance information from one triangle to its neighbors with a Dijkstra-type algorithm. The key observation is that it is sufficient to store the piecewise distance function on the edges of the triangle mesh since this is sufficient for the propagation and also for the exact evaluation of distances everywhere on the surface.

For each edge of the mesh the algorithm maintains a list of segments, so-called *windows*. Each window defines the geodesic distance field within a pencil of rays covering both neighboring triangles (see Figure 9.2). When distance information is propagated across a triangle, the (incoming) windows have to be mapped to the opposite side. The propagation includes the proper intersection of windows, because unlike the planar case, on a surface propagated windows can overlap. Since the distance function is continuous, the intersection requires to find the point where the distance function values in both windows are identical.

We illustrate the procedure with a simple example. The cap of Figure 9.3a consists of three isosceles triangles A,B and C. Now we want to compute the geodesic distance field w.r.t the point $\mathbf{p}$. Since $\mathbf{p}$ is coplanar with the triangles A and B they are covered with a pencil of rays emanating from $\mathbf{p}$ through single *windows* w_1 and w_2. To propagate the distance information through w_1 and w_2 we cut the cap along the edge connecting $\mathbf{p}$ and $\mathbf{q}$ and unfold the triangles isometrically into the plane, i.e. all edge lengths and angles of the triangles are preserved (see 9.3b). In this setting $\mathbf{p}$ is doubled into $\mathbf{p}_l$ and $\mathbf{p}_r$. Now we are ready to propagate the pencil of rays defined by w_1 and w_2 across triangle C and create new temporarily overlapping windows v_1' and v_2' depicted in Figure 9.3c. Evaluating the distances induced by both pencils of rays, the windows can be intersected and properly cut to final windows v_1 and v_2 (see Figure 9.3d) which correctly represent the continuous piecewise geodesic distance function along the edge.

A nice feature of this *window* formulation is that all computations can be formulated in local two dimensional coordinates, i.e. only the mesh topology and scalar edge lengths are required. The necessary condition for this edge based algorithm is that geodesic paths can only pass through vertices with a total angle greater or equal than 2π, i.e. saddles and flat points. This result was proven by Mitchell et al. in [MMP87]. Saddle points and concave boundary points act as pseudo sources which generate additional new windows covering the geometric shadow of the locally expanded surface.

Base algorithm At first all source windows in the immediate vicinity of source points are created and pushed into a priority queue preferring shorter distances, because we want to compute the minimal geodesic distance. Notice that in general the result is independent of the propagation order but the priority queue ensures that windows are propagated as a wavefront which gives a strong speedup and makes the algorithm practical. Processing the queue, the current window is always propagated into the next unfolded triangle, where new windows are created (see Figure 9.2). When the front reaches saddle or boundary vertices new source windows are added. All new windows might overlap with already existing windows and must be intersected accordingly. The algorithm terminates when all edges are partitioned by the minimal geodesic distance windows, i.e. when the queue is empty. The pseudo code algorithm is presented below and all necessary computations are explained in more detail in the next sections.

Circular window propagation In the next section we will define a second type of windows, so from now on windows originating from point sources are called *circular*

```
sourceWs = createSourceWindows()
PQueue.add( sourceWs )
repeat
  curW   = PQueue.popFront()
  newWs  = propagate( curW )
  newWs += saddleAndBoundaryWs( curW )
  newWs  = intersect( newWs, oldWs )
  PQueue.add( newWs )
until queue.empty()
```

Algorithm 1: Exact Geodesic Field

windows. The starting point for a propagation is depicted in Figure 9.2b. Given a window w the corresponding edge $\overline{p_0p_1}$ is aligned to the x-axis of the local coordinate system with the origin in $\mathbf{p}_0$. Each window is described by a six tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$ with σ representing the optional constant offset between a pseudo source and a real source. The binary flag τ determines on which side of the x-axis the unfolded pseudo source $\mathbf{s}$ lies (symbolized in pictures by the arc). The window extents are encoded in b_0 and b_1 which are in the range $[0..|\overline{p_0p_1}|]$. Due to the fact that the distances d_0 and d_1 of the window endpoints from the pseudo source are known the unfolded position $\mathbf{s}$ can be reconstructed via circle intersection.

$$\begin{aligned} s_x &= \frac{1}{2}(b_0 + b_1 + \frac{d_0^2 - d_1^2}{b_1 - b_0}) \\ s_y &= -1^{\tau}\sqrt{d_0^2 - (c_x - b_0)^2} \end{aligned}$$

Using the local coordinates of $\mathbf{p}_3$ which are computed analogously to $\mathbf{s}$ the new windows are found by 2D ray intersection. There are different constellations which can lead to one, two or three (on saddle points) new windows.

Circular window intersection If two windows overlap and one provides a smaller distance everywhere the other is simply clipped against it. If both are minimal in part of the overlapping interval, both ranges are clipped to the point where both distance functions are equal. Notice that clipped windows have to be reinserted into the queue because their priority can change. Using the unfolded pseudo source $\mathbf{s}$ from the previous section, the distance function d_c of an arbitrary point $(p_x, 0)$ in the interval of a window

can easily be formulated. Due to the fact that $\mathbf{s}$ is not necessarily a real source (e.g. saddles induce pseudo sources) the distance σ from all traversed pseudo sources to the real source must be added.

$$d_c(p_x) = \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma \tag{9.1}$$

Trying to find the intersection of two such distance functions, namely $d_{c1}(p_x) = d_{c2}(p_x)$, the computation ends up as the solution of a quadratic equation $Ap_x^2 + Bp_x + C = 0$. In this case there is exactly one solution in the overlapping interval and the coefficients of the polynomial are

$$\begin{aligned} A &= \alpha^2 - \beta^2 \\ B &= \gamma\alpha + 2s_{1x}\beta^2 \\ C &= \frac{1}{4}\gamma^2 - |s_1|^2\beta^2 \\ \alpha &= s_{1x} - s_{0x} \\ \beta &= \sigma_1 - \sigma_0 \\ \gamma &= |s_0|^2 - |s_1|^2 - \beta^2 \end{aligned}$$

Generalization to arbitrary sources

Our goal is to generalize the original geodesic distance computation algorithm from isolated points to polygonal curves on the surface. In a planar configuration the Euclidean distance function to a polygonal curve can be partitioned into several segments. In some segments the distance function is, again, the square root of a quadratic function. Those segments correspond to the vertices of the polygon. In other segments, the distance function is just linear. These segments correspond to the edges of the polygon. See Figure 9.5 for an example.

Going from the plane to a piecewise planar triangle mesh, we can still propagate the distance function from one triangle to its neighbors by storing windows of the piecewise distance function on each edge. The only difference regarding the last section is that now we need to handle two different types of windows: the ones where the distance function is of the form (9.1) and the ones where the distance function is linear. The Dijkstra-type propagation algorithm then has to handle all kinds of window intersections: circular-circular, circular-linear, and linear-linear. In the following we will give the explicit formulae for the corresponding intersection points where the two distance functions

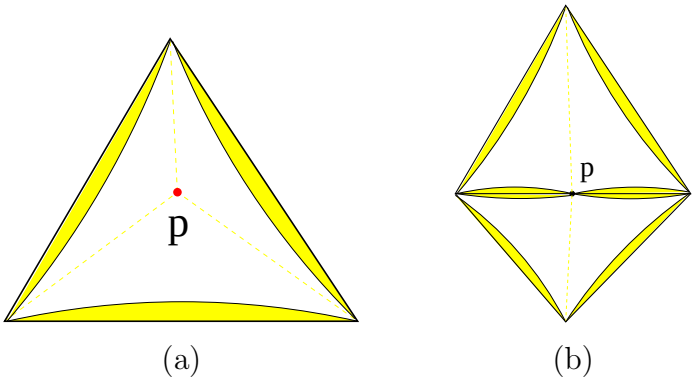


Figure 9.4.: (a) An arbitrary point source p on the surface induces three windows in the corresponding triangle. (b) The six windows of a point source on an edge.

coincide. Additionally we need the ability to create circular source windows induced by arbitrary points on the surface which will be discussed first.

Arbitrary points The original algorithm [SSK*05] was proposed to allow point sources only at vertex positions. However it is straightforward to overcome this limitation. Given an arbitrary point p on the surface the three edges of its containing triangle are initialized with windows emanating from this point as depicted in Figure 9.4. The new created windows are intersected with all other windows on an edge to handle multiple sources. Special care is needed for points lying exactly on an edge. In this case the edges of both triangles must be initialized.

Polygons on the mesh As seen in Figure 9.5 straight line segments induce linear and circular waves from its endpoints. Consequently we create linear and circular windows for each segment of a piecewise linear polygon. Exploiting the window intersection algorithms, already necessary for the window propagation, the overall initialization becomes very simple, because overlaps are handled consistently.

As a preprocess we subdivide the piecewise linear input polygon such that every segment lies entirely in one triangle. This can easily be done by inserting vertices on all intersections between triangle and polygon edges. Using this decomposition it is possible to handle each polygon segment independently. We illustrate the procedure with one line segment in a single triangle as depicted in Figure 9.6a. At first we add linear windows (green) whose extents are computed by intersecting orthogonal rays starting from the

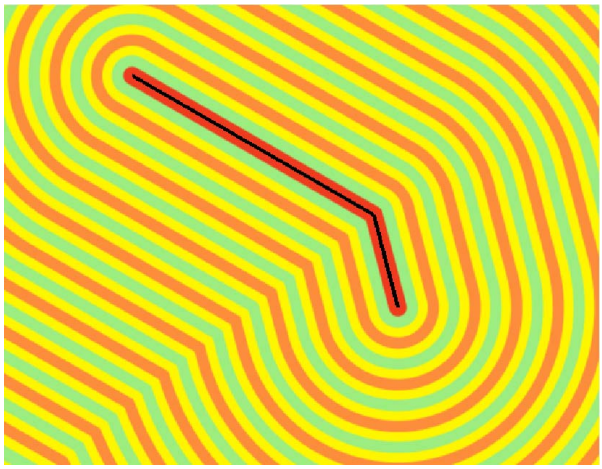


Figure 9.5.: The geodesic distance field w.r.t the black polygon. Linear waves emanate orthogonal to line segments and circular waves emanate from each endpoint of a line segment.

endpoints of the line segment with all triangle edges. Additionally, both endpoints induce circular windows (yellow) which are computed as described before. All new windows are again intersected with windows already registered to an edge. Notice that due to the exact equal distance intersection the result is still independent of the order in which the windows are added.

To complete the algorithm, we next describe the propagation and intersection of linear windows. Now each window is expressed as a seven tuple $(id, b_0, b_1, d_0, d_1, \sigma, \tau)$ in which the added type id is either *circular* or *linear*. In the case of a *circular* window we proceed exactly as described in Section 9. For *linear* windows the tuple components have analogous meanings. The key difference is that the emanating boundary rays of a window starting at $(b_i, 0)$ in local coordinates are computed in a different way. They do not intersect at a pseudo source center but are always parallel (see Figure 9.6b). The distance function over a linear window is a simple linear function fully determined by b_i and d_i.

Linear window propagation The starting point is depicted in Figure 9.6b. Similar to section 9 the window w covers the segment between b_0 and b_1 on the edge e. The x-axis is aligned to e and the y-axis lies in the plane of the triangle where the window should be propagated through. Using elementary geometric calculations the propagation

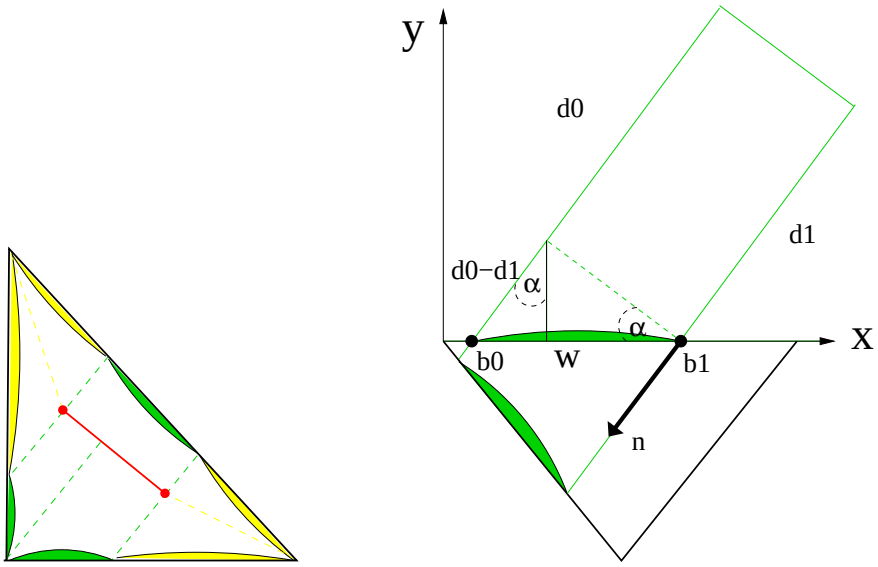


Figure 9.6.: (a) A line source within a triangle induces a set of linear (green) and circular (yellow) windows. (b) Computation of the propagation direction in local coordinates.

direction $\mathbf{n} = (n_x, n_y)$ can be computed in terms of the local coordinate system. The differences $|d_1 - d_0|$ and $b_1 - b_0$ define the angle between the linear front and the mesh edge:

$$\sin\alpha = \frac{-n_x}{|d_0 - d_1|} = \frac{|d_0 - d_1|}{b_1 - b_0}$$

Solving the previous equation for n_x, n_y can be computed by the theorem of Pythagoras:

$$\begin{aligned} n_x &= -\frac{(d_0 - d_1)^2}{b_1 - b_0} \\ n_y &= -\sqrt{(d_0 - d_1)^2 - n_x^2} \end{aligned}$$

Using these ray direction instead of the ray directions induced by the unfolded pseudo source the remaining part of the window propagation is identical to that of circular windows. Here overlaps of propagated windows can happen as well. For this reason the next paragraph describes all possible cases, namely linear-linear and circular-linear window intersections. Both reduce to the solution of a quadratic equation.

Linear window intersection Again there are two different cases for window intersections. The trivial one occurs when the distance function of one window is larger in the whole overlapping interval. In this case it is easily clipped against the other window. The more interesting case happens when the minimal distance function in the overlapping interval is composed of both windows. In this case there must be a point $(p_x, 0)$ where both distance functions are equal.

The distance function of a linear window along an edge is a simple linear function (cp. Figure 9.6b) which can be formulated in terms of $\mathbf{n}$ or directly using the window components. It fulfills the interpolation condition $d_l(b_i) = d_i$.

$$d_l(p_x) = p_x \underbrace{\frac{d_1 - d_0}{b_1 - b_0}}_{m} + \underbrace{\frac{b_1 d_0 - b_0 d_1}{b_1 - b_0} + \sigma}_{n}$$

Now we are ready to compute intersections of linear windows with linear and circular windows to find the separation point p_x on the corresponding edge:

1. linear-linear intersection

$$\begin{aligned} & d_{p1}(p_x) &&= d_{p2}(p_x) \\ \Leftrightarrow\ & p_x m_1 + n_1 &&= p_x m_2 + n_2 \\ \Leftrightarrow\ & p_x &&= \tfrac{n_2 - n_1}{m_1 - m_2} \end{aligned}$$

2. circular-linear intersection

$$\begin{aligned} & d_c(p_x) &&= d_l(p_x) \\ \Leftrightarrow\ & \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma &&= p_x m + n \end{aligned}$$

Squaring the previous equation leads to a quadratic equation $Ap_x^2 + Bp_x + C = 0$ with coefficients

$$\begin{aligned} A &= 1 - m^2 \\ B &= -2(s_x + m(n - \sigma)) \\ C &= s_x^2 + s_y^2 - (n - \sigma)^2 \end{aligned}$$

Notice that unlike the previous intersections here exist possibly two valid solutions which can lead to a trisection of the overlapping interval. In this case the cut circular window lies in the middle of two disconnected parts of the linear window.

Approximation algorithm

The propagation of distance information across many triangles leads to an increasing number of windows per edge because windows split up at vertices. A large number of windows increases the time as well as the space complexity of the algorithm. So the idea for the ε-Approximation-Algorithm in [SSK*05] is to merge neighboring windows on an edge whenever the induced relative error is acceptable. Allowing for example a relative error of $\varepsilon = 0.1\%$ leads to visually indistinguishable results but enables the processing of huge models with several millions of faces which are far too complex for the exact algorithm. Again the proposed linear windows fit naturally in the original framework and share all properties necessary for window merging. Before we describe the merging of linear windows we shortly review the basic principles and the case of circular windows. For details see [SSK*05].

To guarantee consistency of the geodesic field some conditions must be checked before merging two neighboring windows.

1. **Directionality:** Both windows propagate into the same direction.
2. **Visibility:** The pencil of rays of the merged window must at least cover all rays of the original windows so that no gaps arise.
3. **Continuity:** The distance at the endpoints bounding the merged window must be preserved to conserve distance field continuity.
4. **Type:** Both windows must be of the same type, e.g. planar or circular.

Additionally the user can prescribe a relative error bound ε_U so that only those merges are performed where the relative difference between the distance function of the new window $d'(p_x)$ and the original piecewise distance function $d_{lr}(p_x) = d_l(p_x) \cup d_r(p_x)$ is smaller than ε_U, i.e.

$$\frac{|d_{lr}(p_x) - d'(p_x)|}{d_{lr}(p_x)} \leq \varepsilon_U$$

Merging of circular windows Taking two neighboring circular windows

$$\begin{aligned} w_l &= (id, b_{0l}, b_{1l}, d_{0l}, d_{1l}, \sigma_l, \tau_l) \\ w_r &= (id, b_{0r}, b_{1r}, d_{0r}, d_{1r}, \sigma_r, \tau_r) \end{aligned}$$

which meet at the common point $(b_{1l}, 0) = (b_{0r}, 0)$ the merged window w' is already determined up to σ' due to the necessary conditions:

$$\begin{aligned} id' &= id \\ b'_0 &= b_{0l} \\ b'_1 &= b_{1r} \\ d'_0 &= d_{0l} + \sigma_l - \sigma' \\ d'_1 &= d_{1r} + \sigma_r - \sigma' \\ \tau' &= \tau_l = \tau_r \end{aligned}$$

The continuity constrain restricts w''s pseudo source $s' = (s'_x, s'_y)$ to lie on a conic curve $s'^2_y(s_x)$. Because of the positivity of the d'_i and the visibility constraint the valid domain of this conic curve is further restricted. If it is the empty set, the merge is disallowed and in all other cases the smallest possible σ' is chosen (see [SSK*05] for details and how to evaluate the approximation error).

Merging of linear windows The distance values d_i of a linear window can always be transformed so that the corresponding pseudo source distance σ vanishes. So w.l.o.g. two neighboring linear windows

$$\begin{aligned} w_l &= (id, b_{0l}, b_{1l}, d_{0l}, d_{1l}, 0, \tau_l) \\ w_r &= (id, b_{0r}, b_{1r}, d_{0r}, d_{1r}, 0, \tau_r) \end{aligned}$$

which join at the common point $(b_{1l}, 0) = (b_{0r}, 0)$ can be merged into a linear window

$$w' = (id, b_{0l}, b_{1r}, d_{0l}, d_{1r}, 0, \tau_l = \tau_r)$$

which satisfies all necessary constraints and is fully determined by the original windows. Notice that the visibility constraint is always fulfilled because diverging linear windows can only occur in combination with an additional point source or a saddle. The maximum approximation error is obtained at the joining point and can be computed as

$$\varepsilon = |1 - \frac{d_{1r}(b_{1l} - b_{0l}) + d_{0l}(b_{1r} - b_{1l}))}{d_{1l}(b_{1r} - b_{0l})}|$$

Adaptive refinement

The algorithm presented in the last section is able to compute the exact geodesic distance field on a triangle mesh with respect to an arbitrary polygon embedded on the mesh. However, the distance information is not given explicitly but rather through a set of windows defined on the edges of the mesh. For most geometry processing algorithms this implicit information has to be made explicit. The standard approach to do this is to simply sample the distance function at the mesh vertices and then use a linear interpolant on each face as an approximation of the original distance field. In order to have some guarantee about the approximation tolerance, we have to refine the mesh in regions where this tolerance is violated. Usually this happens in the vicinity of the geodesic medial axis. To decide where to refine we compare the exact geodesic distance on edges with the linear interpolant and check if a user-defined threshold is exceeded. In this case we split the edge and insert a new sample point.

The geodesic distance field is smooth with constant gradient magnitude everywhere except for the geodesic medial axis. By properly placing the newly inserted vertices on the medial axis (i.e. at the maximum distance value on the cut locus) we can avoid excessive local refinement. This feature sensitive placement leads to optimal convergence and is in the spirit of [KBSS01].

Since edge splits in arbitrary order lead to poor triangles, we employ a strategy similar to adaptive red-green triangulations. An important feature is that our refinement does not change the underlying geometry and can be seen as a pure up-sampling of the original geodesic distance field. Due to this fact no re-computation of the geodesic field is necessary. The geodesic distance has to be updated only for those edges that are newly inserted. The edge-based refinement and the evaluation procedure are described in more detail in the next sections.

Edge-based refinement In each refinement step we evaluate for each edge the maximal deviation between the exact distance function given as a piecewise function along the edge and the linear function interpolating the exact distance only at the edge endpoints. If this maximal deviation exceeds a user-defined threshold the edge is tagged for refinement and the corresponding point p_{max} is cached as the optimal splitting position. Simply splitting all tagged edges would result in poor triangle quality. We aim at applying a one-to-four split (see Figure 9.7) of triangles lying entirely in the refined region. The one-to-four split operator can be composed of edge split and edge flip operations.

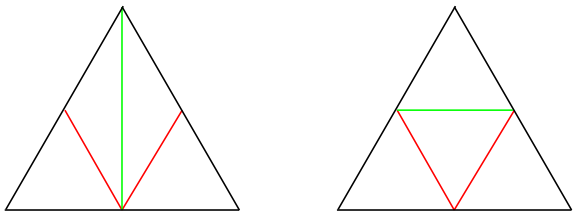


Figure 9.7.: Implementation of a one-to-four split of a triangle using only edge split and edge flip operators. (a) Each edge of the black triangle is first split in arbitrary order. (b) The green edge, characterized by two adjacent triangles with only one original edge segment, is flipped to complete the one-to-four refinement.

For one triangle this requires the splitting of all edges (in arbitrary order) and the flipping of one specific edge (see Figure 9.7). To increase the number of regular one-to-four splits we iteratively tag all edges which are adjacent to triangles with already two tagged edges. This edges will be split on their midpoint. Subsequently all tagged edges are split at their cached split positions and all necessary flips are done. Identifying which edges should be flipped is easy if we mark all new created edges as red during the splitting process. If both triangles of a red edge are bounded by exactly two (the edge itself plus one additional) red edges the edge must be flipped.

Evaluation of interpolation error The Geodesic Distance Function along a mesh edge e is defined piecewisely and consists of linear and circular segments corresponding to linear and circular windows. To compute the maximum deviation between this exact function and the linear interpolant defined by the exact distances on the edge vertices it is possible to first evaluate the maximal deviation for each segment individually and then take the overall maximum.
In the case of a linear segment the evaluation is simple. The difference between two linear functions is again a linear function and so the maximum is always on the boundary of the corresponding linear window.
In the case of a circular window the maximum can be computed analytically. The difference of both distance functions along the edge

$$E(p_x) = \sqrt{(p_x - s_x)^2 + s_y^2} + \sigma - (ax + b)$$

Table 9.1.: Timings

Model	#Faces	Time exact	WPE exact	Time 0.1%	WPE 0.1%
Plane	422	4ms	2.40	2ms	1.2
Fandisk	12k	1.90 s	9.06	0.12s	1.6
Car	34k	3.03 s	7.06	0.91s	3.4
David	8M	-	-	165s	1.3

has a single extremum at

$$q_x = s_x - a\sqrt{\frac{s_y^2}{a^2-1}}$$

If q_x is not in the valid interval $[b_0..b_1]$ of the window the maximal deviation is on the boundary of the window as in the linear case.
The optimal position for a new sample point is exactly the position p_{max} where the deviation is maximal. Allowing split points to lie arbitrarily close to the edge endpoints leads to degenerate triangles. In practice we clamp the splitting position to be in the range of $25-75\%$ of the edge length. Additionally if the optimal position lies between $12.5-25\%$ or $75-87.5\%$ we adjust the new vertex so that the optimal position lies exactly on the midpoint of the new created edge because this leads to better triangulations. Given the optimal sample position $t \in [0..1]$ the update is as follows:

$$t \mapsto \begin{cases} 0.25 & 0 \leq t < 0.125 \\ 2t & 0.125 \leq t < 0.25 \\ t & 0.25 \leq t \leq 0.75 \\ 2t-1 & 0.75 < t < 0.875 \\ 0.75 & 0.875 \leq t \leq 1 \end{cases}$$

Results

We demonstrate the results of our algorithm on models of different complexity. Table 9.1 shows the corresponding timings for the computation of the exact and approximated geodesic fields which were generated on an AMD 64 3500+ system with 2GB of RAM. Additionally the average number of windows per edge (WPE) is listed. On the David and the Fandisk model we computed the geodesic field w.r.t. the red polygonal curves on the surface (see figure 9.9). The visualization uses a 1D texture to transfer the linear

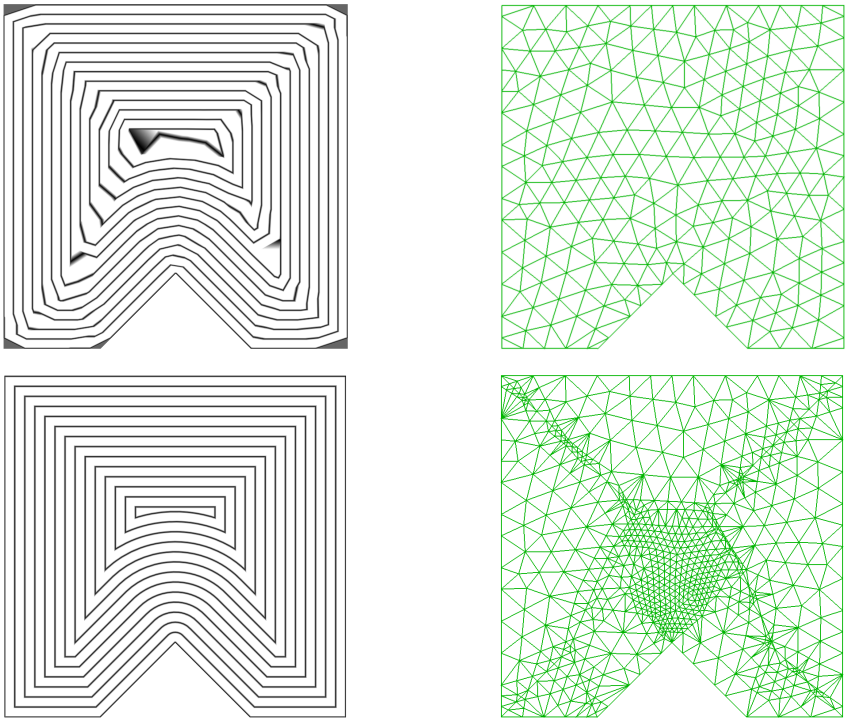


Figure 9.8.: plane (422 faces)

interpolant of the geodesic field into a color. For the car model depicted in Figure 9.1 we computed the geodesic field for the boundary and applied the adaptive refinement to get an satisfactory visualization. The refined mesh is showed in Figure 9.9. Obviously most of the mesh refinement occurs in a thin local neighborhood near the medial axis of the geodesic field. The plane model in Figure 9.8 illustrates the quality gain of our adaptive refinement in more detail. The upper row shows the original mesh with the corresponding linear interpolant of the geodesic field. Even though the mesh structure looks nice, the result is very noisy near the medial axis and shows large errors. Applying the presented adaptive refinement we gain a high quality explicit representation of the geodesic field shown in the lower row together with the generated mesh structure. The approximation error reduced by a factor of 100 while the number of faces increased by a factor of 4.

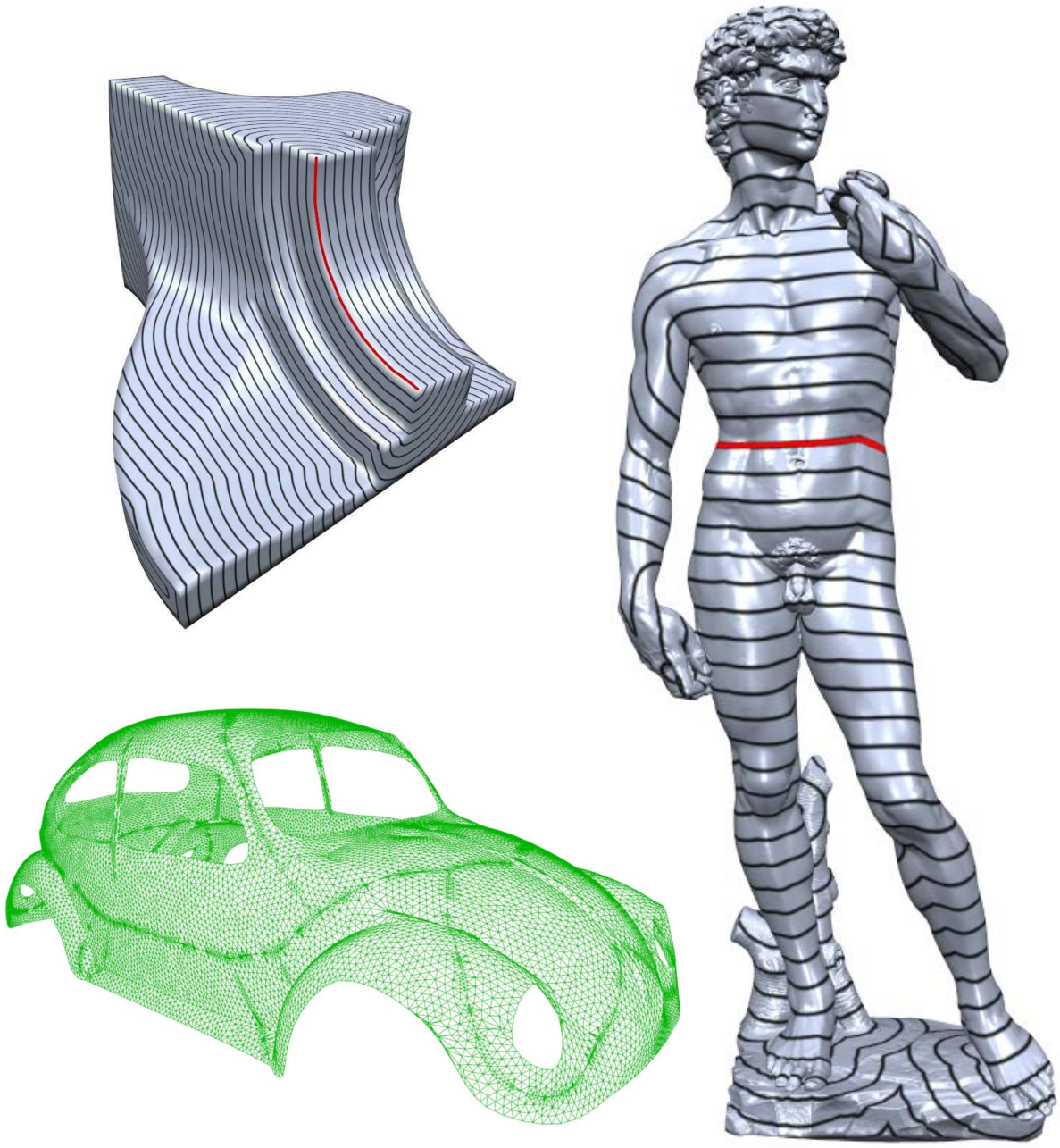


Figure 9.9.: Fandisk (12k faces), Car (34k faces) and David (8M faces)

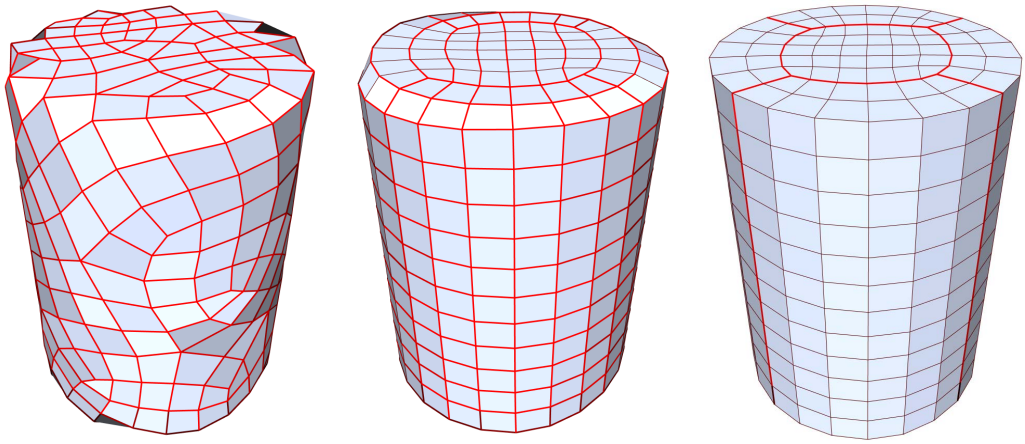
Part III.

Quadmesh Optimization

The third part of this thesis deals with quadmesh optimization. In contrast to the previous part, not only the output consists in a quadmesh but the input is a quadmesh as well. Such an optimization is appealing from the robustness point of view. Instead of being confronted with the problems of global parametrization like foldovers, the optimization can be driven by a well designed set of robust operators like edge flip or edge collapse. While such an approach based on local operators can be applied successfully in the context of triangle meshes, the situation is more complicated in case of quadrilateral meshes. It can easily be shown that local operators always introduce additional irregular vertices and thus usually long chains of well combined local operations would be required to find a high-quality quadrangulation. Consequently, a greedy optimization based on local operators typically gets stuck in a local minimum, where the distribution of irregular vertices is far away from being optimal.

However, instead of optimizing the overall quadmesh quality, we build on top of the previously presented parametrization based quadmesh generation results. While the resulting meshes already exhibit a good structure in terms of semi-regular valency, they typically lack a high-quality base-complex. In Chapter 10, which is based on [BLK11], we present a novel class of global operators, called *grid-preserving operators*, that are able to change the global connectivity within a quadmesh without altering its irregular vertices. Based on this global operators, a greedy strategy is performed that iteratively eliminates helical configurations which negatively affect the base-complex. Accordingly, the combination of both of our approaches for quadmesh generation and structure optimization leads to a fully automatic pipeline that is able to generate, in practice often desired semi-regular meshes.

10. Structure Optimization



(a) (b) (c)

Figure 10.1.: Comparing different structural quality: (a) A completely unstructured mesh with bad quads and a dense base-complex (in red). (b) Appropriate singularities and oriented quads improve the mesh, but due to a quad-loop winding down the cylinder the base-complex is still dense. (c) While preserving singularities and orientations, the base-complex is optimized and topologically equivalent to a cube

Providing a high-quality quadrilateral mesh with a coarse base complex is of great interest, since a coarse base complex induces a simple patch layout which is desired for e.g. fitting of NURBS-patches or as a base mesh for subdivision. In general, computing quadrangulations which provide on the one hand a nice stretch distribution in terms of angles and anisotropic edge lengths and on the other hand a coarse base complex is an unsolved problem. Parametrization based techniques usually lead to nicer stretch distributions due to well adapted singularities and edge orientations, but unfortunately they often posses a rather fine base complex. On the contrary, decimation based algorithms are able to generate coarse base complexes, however, this benefit usually comes at the cost of inappropriate placed singularities or edge orientations, inducing high stretch dis-

tributions in a finer subdivision.

Our strategy is to start with a quadrangulation already equipped with appropriate singularities and a nice stretch distribution and then try to improve the base complex as much as possible while keeping the singularities fix. Notice that apart from the base complex there is no other straightforward coarse quadrangulation with the same singularities as in the input mesh. Due to the global topological restrictions we cannot define a concept analogous to the Delaunay triangulation to achieve a coarse quadrangulation of the singularities.

In the following we present an algorithm to improve the *base complex* $\mathcal{B}(Q) = (\mathcal{V}, \mathcal{E}, \mathcal{Q})$ of a given quadrilateral mesh Q as introduced in Section 2.1. As an example, Figure 10.1 shows three different quadrilateral meshes, where the base complex is highlighted in red. Our algorithm is able to perform the optimization from Figure 10.1b to 10.1c.

Definitions To explain our approach we require the notion of a parametric line, as introduced below. Remember that a vertex v_i is called *regular* if it has valence 4, otherwise it is a *singular* vertex. Topologically a regular vertex is the crossing of two coordinate lines in a 2D Cartesian grid and therefore we can easily build a right-handed local coordinate system at such a vertex by cyclically labeling the adjacent edges in counter-clockwise order with u, v, $-u$ and $-v$ as depicted in Figure 10.2. However, notice that such a labeling is only possible within a singularity-free local region since e.g. walking counter-clockwise around a valence 3 singularity would mean that a formerly labeled u edge becomes a v edge contradicting with the initial label.

A *parametric line* is generated by tracing a local coordinate direction through regular vertices or more formally a connected sequence of edges, such that two subsequent edges e_i and e_j are always connected through a regular vertex where both edges belong to the same local parametric direction, i.e. they are either $\{u, -u\}$ or $\{v, -v\}$ (see Figure 10.2). Finally a *regular parametric loop* is a closed parametric line where all traversed vertices are regular. Notice, that the base complex is the union of all parametric lines which start and end at singular vertices.

The consideration of dual parametric lines instead of primal ones is advantageous. For each primal parametric line we can always identify two parallel dual parametric lines,

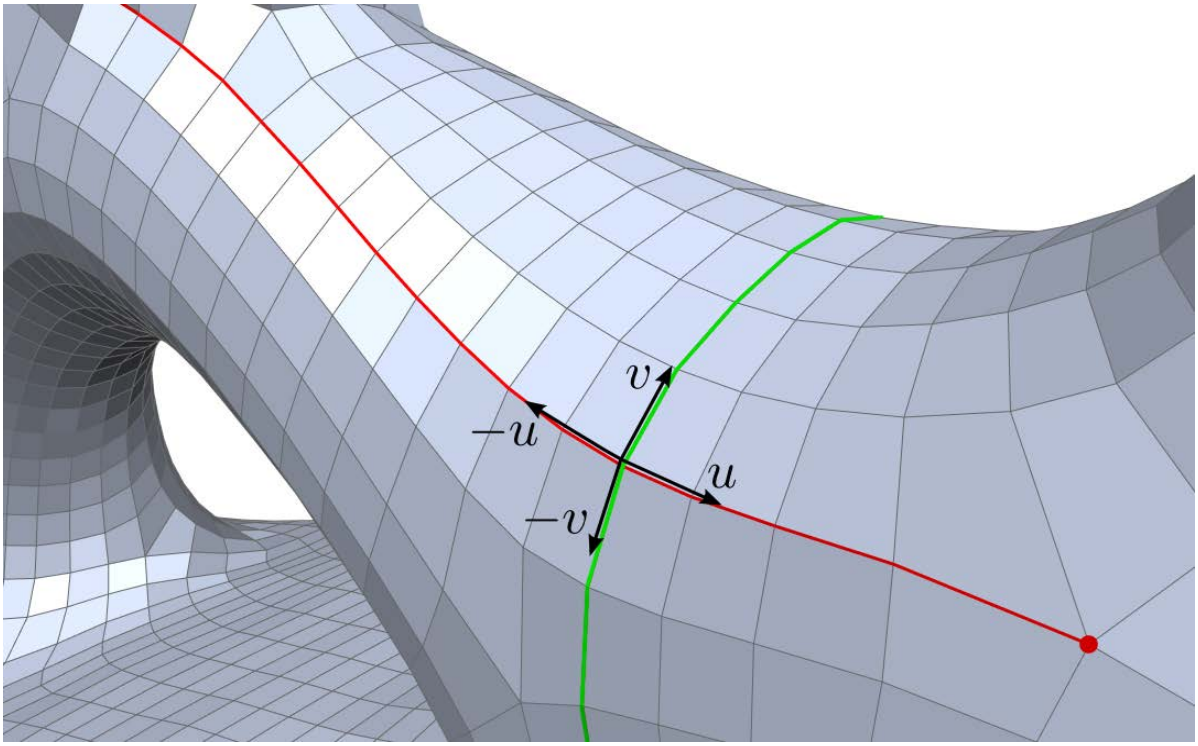


Figure 10.2.: Each regular vertex induces a natural coordinate system by counter-clockwise labeling the outgoing edges with u,v,$-u$,$-v$. Parametric lines, as shown in red and green can be extended until they end in a singularity (red point).

while the contrary is not always true due to the fact that primal parametric lines end at singularities. Consequently, using dual parametric lines or dual *parametric loops*, which are *quad-loops* in the primal meshes and called *poly-chords* in [DSSC08], increase the set of candidates for our grid preserving operator (see Section 10.1).

We next propose a novel operator which is fundamental for our base complex optimization, since it offers a new class of global operations which preserve quadrilaterals and are optionally able to preserve singularities.

10.1. Grid-Preserving Operators

Changing the local connectivity within a quadrilateral mesh without introducing non-quadrilateral elements or new singularities is a delicate task. And even worse, no local operation exists to perform such a modification. However, since such an operation is

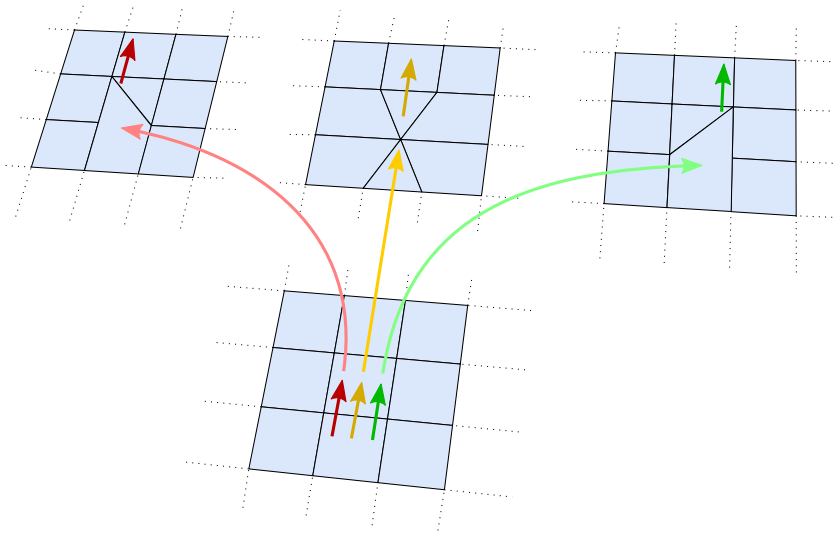
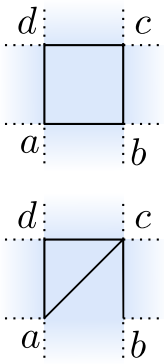


Figure 10.3.: The three atomic operations of a dual half-edge: A *shift left* step (red arrow) releases the vertex on the left side of the dual half-edge and shifts it towards the next vertex, generating a triangle and a pentagon. In a *collapse* step (yellow) the edge is *collapsed* into a single vertex. The *shift right* (green) is the counterpart of *shift left*, releasing the right vertex. After applying one step we move to the next dual halfedge as indicated by the arrows.

highly desirable, it is worth to examine the problem in more detail.



Assume that we have a closed quadrilateral mesh without boundaries and that we want to change the connectivity within a single quadrilateral with points a, b, c and d such that a is connected to c instead of b, as depicted in the figure to the right. The problem is that after executing this edge-flip, we end up with a triangle and a pentagon. If the corresponding quad-loop is self-intersection free, one solution would be to propagate the edge-flip along the whole (always closed) quad-loop such that in the end the triangle and the pentagon cancels out. Unfortunately not all quad-loops are intersection free and even if they are, this combined operation is completely determined by the quad-loop structure and leaves no freedom to control which areas of the mesh should preferably be modified. This property is in conflict with the requirement to protect parts of the mesh which contain

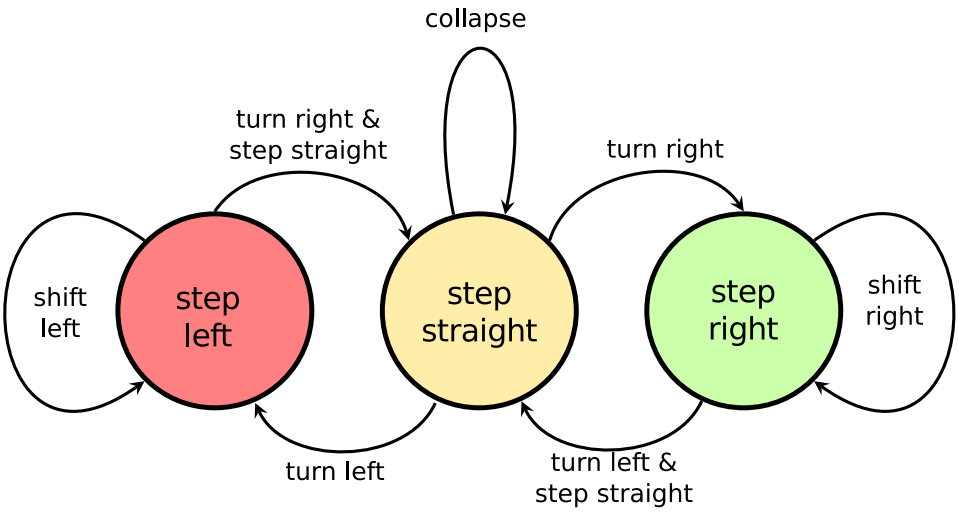


Figure 10.4.: The finite-automaton describes all valid possibilities to combine the three atomic operations. Each closed dual path on the mesh, which is closed within the finite-automaton preserves the all-quadrilateral structure without introducing new singularities.

important features or regions of good quality.

To obtain more degrees of freedom we propose to combine the above edge-flip operation with a collapse operation in such a way, that we can create a much larger variety of possible operators, but still can guarantee to preserve the quadrangular structure of the input mesh. Figure 10.3 shows the three necessary atomic operations, namely *shift left*, *collapse* and *shift right* which are visualized with a red, yellow and green arrow respectively. All three operations can be associated with a dual halfedge and combined along a dual path in the way shown in the finite automaton in Figure 10.4 in order to form a valid grid-preserving operator (*GP-operator*). The most important property of such a *GP-operator* is that it does not introduce new singularities or non-quadrilateral elements.

This means, if we start at one mesh edge in the *step left* state we can do as many *shift left* steps as we want by following the dual path in the same direction where all crossed edges are shifted. To leave the *step left* state, within a face we can turn right and change the state to *step straight*. From here we can either move straight and collapse as many edges as desired, or turn right and apply the *shift right* operator, or again turn left and apply the *shift left* operator. Altogether, using this state machine, we can traverse a

dual path which is assembled of straight steps, sidesteps to the left and sidesteps to the right, but we can never step back, i.e. turn twice into the same direction (cp. Figure 10.5).

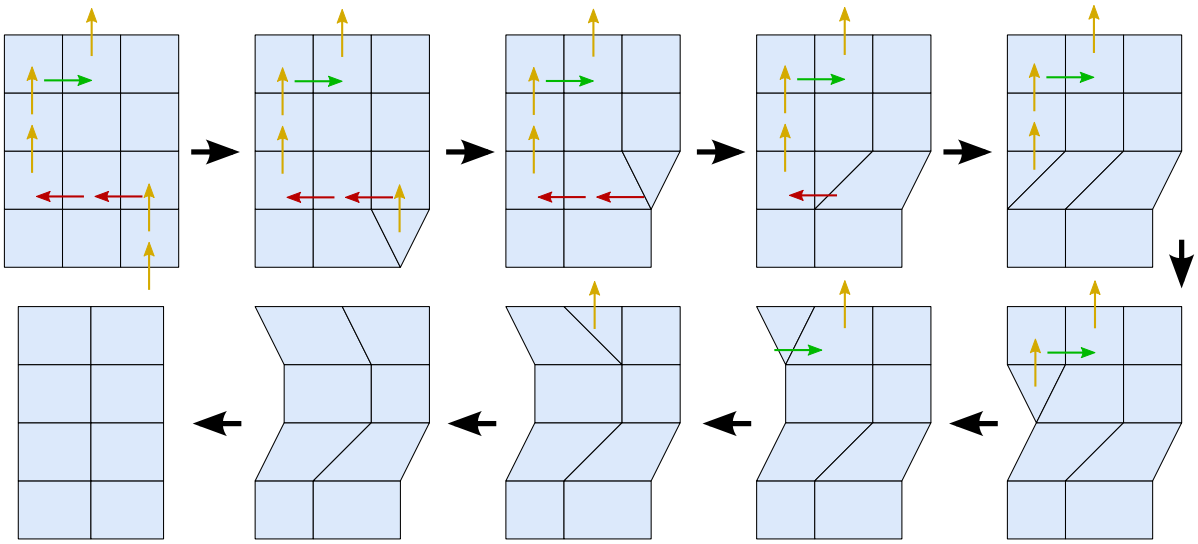
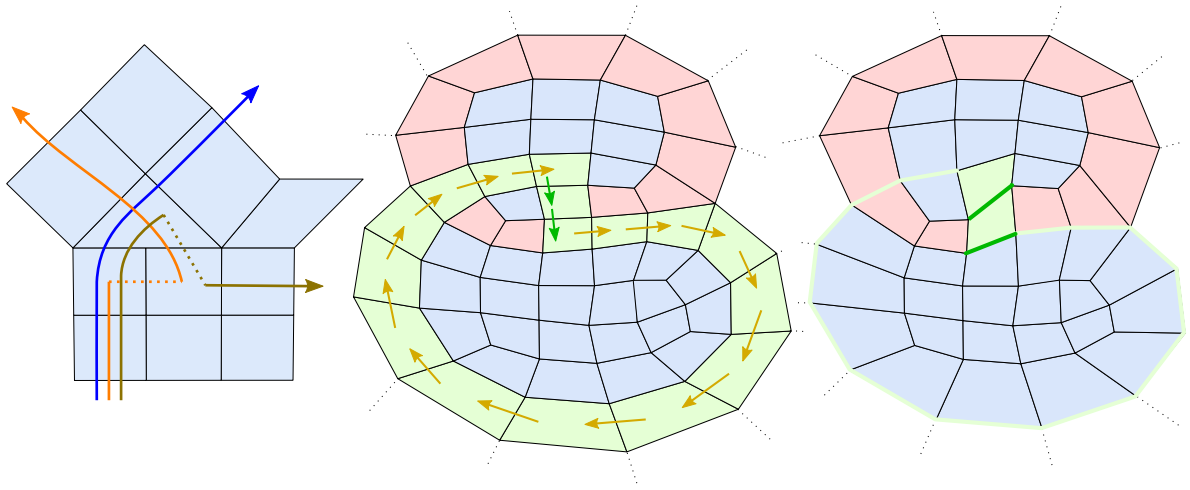


Figure 10.5.: Example of a valid dual path combining the three atomic operations according to the state machine. In the absence of singularities the resulting topology is equivalent to the removal of a single column of quads (cp. last step).

While this might seem to be quite restrictive it fortunately is not. The reason is that we can exploit the singularities within the mesh to change the walking direction, e.g. walking around two valence three singularities is the same as turning by an angle of π in a regular grid. Consequently, navigating between and around the singularities offers a large variety of possible paths. Figure 10.6a gives an example of this behavior.

To guarantee that in the end all triangles and pentagons cancel out, it is necessary that the dual path is closed within the state machine, meaning that there is a transition from the state at the last dual half-edge to the state at the first dual half-edge. Notice that this is exactly the case when the closed dual path circuits a group of singularities such that the total rotation becomes an integer multiple of 2π. For illustration, Figure 10.6b shows such a path and the resulting quadrangulation after applying the corresponding operations.



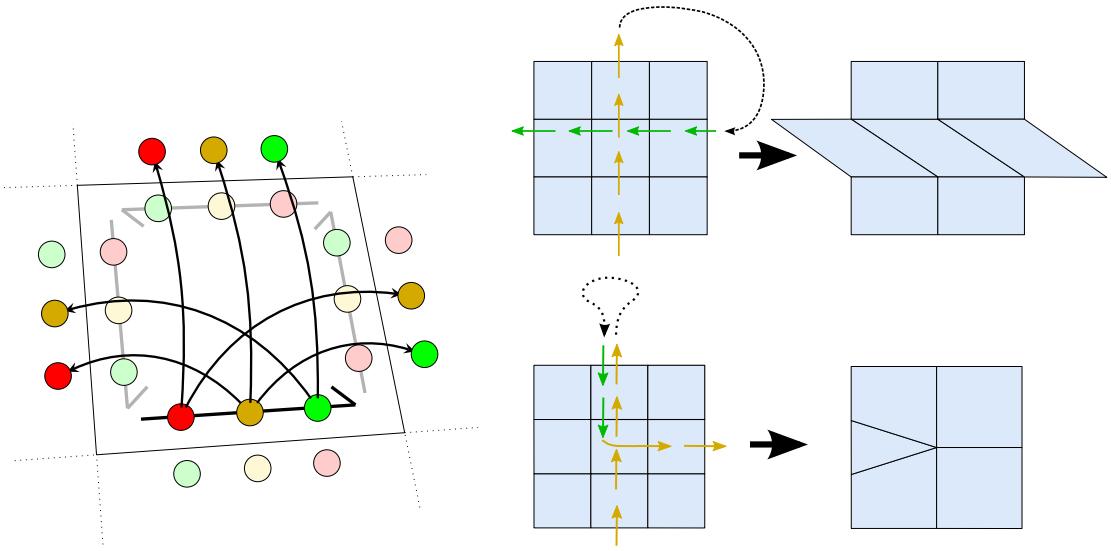
(a) (b)

Figure 10.6.: (a) Side steps (dashed lines) can control the walking direction by navigating between singularities. (b) The dual path through the green quadrilaterals, consisting of *collapse* steps (yellow) and *shift right* steps (green), is a valid GP-operator (left). Executing the corresponding atomic operations results in a new quadrilateral mesh with the same singularities (right). Notice that the GP-operator has closed the red quad-loop.

Going back to our introductory question, we are now able to give a more satisfying answer. If we want to shift the edge between a and b to an edge between a and c while maintaining a quadrangulation without additional singularities, we can start at the edge between a and b with the state *shift right* and walk along any closed dual path compatible to the state machine and perform the induced atomic operations. Which one of those candidate operations is the best strongly depends on the application in mind.

A natural choice is to minimize the overhead, i.e. , the number of additional atomic operations which are necessary to close the path. This can be found by enumerating all possible paths generated by the state-machine with increasing length until the shortest cycle is found. Obviously this approach leads to an exponential complexity which is useless for practical applications.

The state-machine graph: In order to efficiently find a cycle which is compatible to the state machine, we first assemble a directed graph, as depicted in Figure 10.7a. In this



(a) (b)

Figure 10.7.: (a) Illustration of the state-machine graph: By creating three vertices for each dual half edge we can encode the different states *shift left* (red), *collapse* (yellow) and *shift right* (green). Adding directed edges corresponding to transitions within the finite-state automaton we obtain a graph where all paths that belong to chains of operations are compatible with the finite-state automaton by construction. (b) The upper part of the figure shows a valid while the lower one depicts an invalid crossing configuration.

graph all cycles are compatible with the state-machine by construction. The idea is that the graph possesses three different vertices for each dual halfedge of the quadrilateral mesh which encode the three different states. Adding directed edges which reproduce the transitions of the state machine as illustrated in Figure 10.7a we achieve a directed graph with the desired property. All cycles in this graph correspond to dual paths on the quadrilateral mesh which are closed within the mesh as well as in the state machine.

In this graph a shortest cycle through a start vertex can be found by a simple and efficient breadth-first search. However there is one drawback compared to the explicit exponential algorithm of the state-machine. Since the graph is static, it does not capture the changes made by previous operations of the same path. Clearly we cannot *shift* an edge which was already collapsed, although such a path exists in the graph. Therefore we have to do a post-evaluation of the cycle in order to check whether it belongs to a realizable set of operations or not. If it is not realizable, we iteratively modify the graph

and perform new searches, until we have found a valid cycle or the algorithm terminates without finding one. In contrast to the breadth-first search the iterative process cannot guarantee to find a shortest path. However, as our practical experiments showed, it is at least a good compromise between quality and performance.

Illegal configurations within a cycle are typically induced by a corresponding dual path on the quadrilateral mesh that visits a face more than once, e.g. by performing more than one operation on a single edge or first shifting an edge and then performing any other operation while walking through the face. The only two exceptions where it is allowed to visit a face twice are first collapsing through a face in two orthogonal directions and second collapsing through a face in one direction and then shifting through the face in the orthogonal direction. In both cases the static graph structure still leads to valid paths.

If an illegal cycle is found, we first identify the first illegal configuration where a face is visited twice, leading to a pair of graph vertices v_i and v_j, which are in conflict by visiting v_i first. To modify the graph, we remove all graph vertices and adjacent edges which are incompatible for the path up to vertex v_i and then restart the search from v_i.

Feature and singularity preservation: A nice property of the graph representation is that we can exclude all unwanted atomic operations by simply removing the corresponding graph vertex and all its adjacent edges from the graph. This is for example useful to disallow the merging of neighboring singularities or the shifting of feature edges.

Moreover it is possible to disallow the merging of singularities which are not directly connected. Such a merging could possibly happen if the breadth-first search leads to a cycle which collapses several edges connecting two singularities. Of course we do not want to forbid the collapse of all edges between the singularities. Therefore such illegal configurations are identified in the post-evaluation phase and as before we restart the search with a modified graph, where the last collapse leading to the illegal merge was removed. The same procedure can be used to prevent that two distinct feature lines collapse into one.

Using GP-operators: In summary the concept of a GP-operator offers a variety of different structural modifications, which by construction do not introduce new singularities or non-quadrilateral elements. Notice that the well-known poly-chord collapse used in [DSSC08] is one special case of a GP-operator which only consists of edge collapses.

Here we suggested to extend a desired local operation to a full GP-operator by the minimal number of additional operations. However, depending on the desired structural optimization many other choices are conceivable, leading to other graph search algorithms like e.g. a Dijkstra or Hamiltonian cycle.

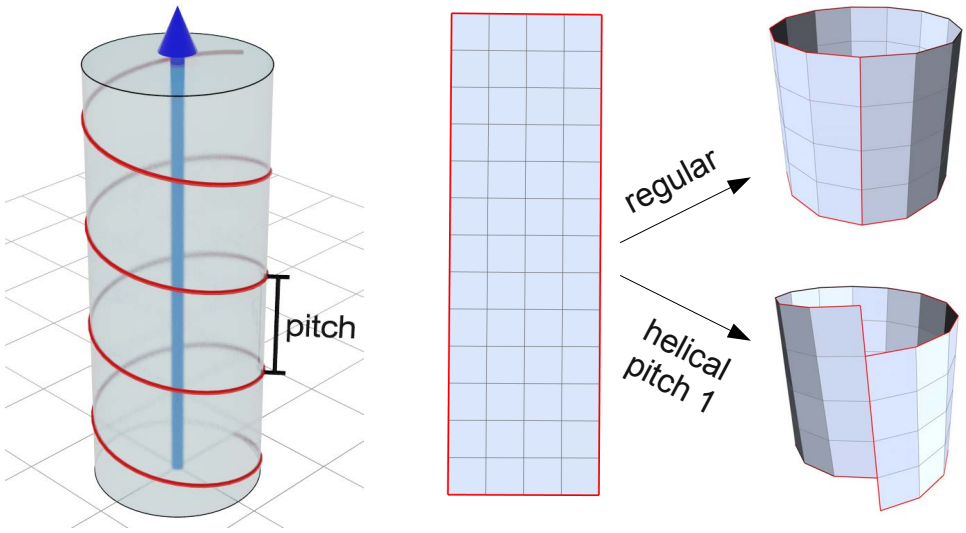
A nice feature of the graph based construction is the flexibility to optionally guarantee the preservation of singularities and/or (sharp) features of the input quadrangulation by just removing some of the graph vertices.

In the following sections we will use GP-operators to improve the quality of the base complex by identifying and repairing helical mesh configurations.

10.2. Helices

Topological Helices in Quadrilateral Meshes The most intuitive way to think of topological helices in quadrilateral meshes, which we will call *q-helices*, is, to imagine their construction out of a rectangular part of the Cartesian grid as illustrated in Figure 10.8b. First start creating a cylinder in the usual way, by keeping one side of the rectangle fixed in space, wrapping the opposite side of the rectangle around the first one and gluing together pairs of boundary vertices which belong to equal parametric lines.

If we instead connect vertices from different parametric lines of the rectangle, we are able to create a single new parametric line, which winds upwards or downwards in the grid with a constant orthogonal offset. Hence, we have constructed a discrete helical structure. In this structure we can identify all the properties of a usual helix. The *pitch* h of the helix is is the distance between two neighboring windings, while the *turn length* τ is the arc length of a single turn. For a q-helix both values are integers, since all distances are measured in the grid-metric of the quadrangular mesh, which means that all edges (and dual edges) have a length of one. The winding number γ which counts the number



(a) (b)

Figure 10.8.: (a) A left-handed helix winds up the blue axis. (b) By wrapping a rectangular quad-patch and and gluing two sides, we can create a cylinder. Shifting the sides against each other before gluing, we end up with a topological q-helix equipped with the same properties as in the continuous case.

of turns can be computed by dividing the total length l by the length of one turn $\gamma = l/\tau$.

The *orientation* of a helix is reflected in the sign of the pitch. Following the right hand grip rule, a right-handed helix has a positive pitch, while the pitch of a left-handed helix is negative. Notice that the handedness of a helix is an intrinsic geometric property and does not depend on the chosen coordinate system.

After describing the construction of q-helices, in the next paragraph we will derive a criterion which can be used to identify q-helices in quadrangular meshes. Some example helices are shown in Figure 10.9.

As discussed in the previous section, we want to work with helices of the dual mesh. More precisely a q-helix $H^d = [e_0^d, \ldots, e_n^d]$ with pitch $h \in \mathbb{Z}$, turn length $\tau \in \mathbb{Z}$ and winding number $\gamma \in \mathbb{R}$ is an ordered set of connected dual edges e_i^d forming a dual

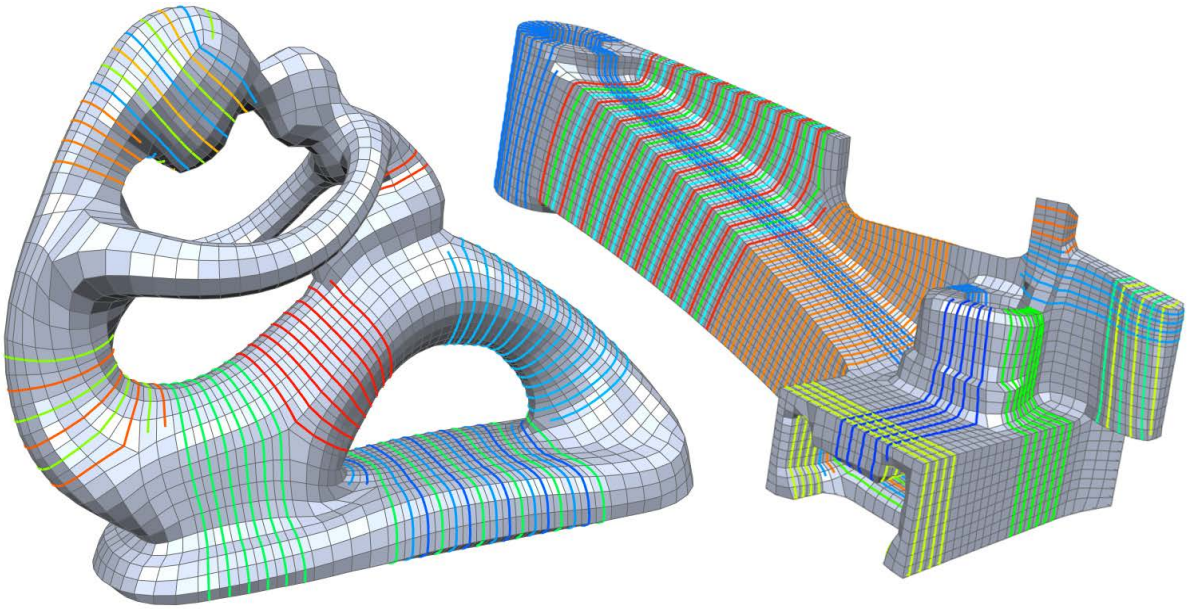


Figure 10.9.: Exemplarily some q-helices are shown as colored dual parametric lines. Notice that q-helices with a pitch greater than 1 often form bundles of interleaved helices.

parametric line and fulfilling the following q-helix property:

Within a q-helix it is equivalent to either walk τ steps along the helix or alternatively do h side-steps to the left. Here equivalent means that not only the position but also the orientation coincides.

Mathematically the above statement implies that there is a regular region without interior singularities around H^d, where it is possible to choose a consistent frame. Consequently q-helices cannot have any self-intersections.

For our mesh optimization task not all helices which fulfill the above definition are of interest. Therefore it is useful to define the interesting subset to be so called *minimal q-helices*. They are characterized through two properties: For a minimal q-helix the pitch h is always smaller or equal to the turn length τ and secondly there is no subset of dual edges belonging to the q-helix, which form a separate q-helix with smaller pitch. The first criterion excludes approximately half of all q-helices, because for each q-helix there exists an orthogonal q-helix living in the same regular region, where the values of pitch and turn length are exchanged. The second criterion excludes helices which

contain other helices with smaller pitch, not well suited for our optimization.

As illustrated in the introductory example in Figure 10.1, q-helices subdivide the base-complex into narrow stripes. Therefore in the next section we will discuss how to remove them from the quadrilateral mesh.

To remove a q-helix we can apply exactly the inverse operation of the construction example of Figure 10.8b, which means cutting the mesh along the helix, shifting the vertices of one side of the cut, and gluing them with their new partners. However, on a closed mesh the situation is a little bit more complicated. In order to preserve the quad-structure we have to compute a full GP-operator, as introduced in Section 10.1, where the desired shifting operations are a sub-part of the complete operation. Furthermore we have to make sure that no other shifting of horizontal edges within the cylindrical mesh area of picture Figure 10.8b are done by the GP-operator. Since the graph construction of the GP-operator does not allow multiple operations on a single edge, we repair helices with pitch > 1 iteratively by applying the following algorithm.

Removing a q-helix H with pitch 1 can be done in four steps.

1. Set up the graph G representing the state-machine for the input quadrilateral mesh.
2. Identify an open dual path $D = [d_0, \ldots, d_m]$ consisting of shift steps which are necessary to remove the helix.
3. Remove all vertices from G which correspond to shift operations which are in conflict with the correction of D, i.e. all shifting steps of horizontal edges in the cylindrical region which do not belong to D.
4. Execute the iterative path search described in Section 10.1 from vertex d_m to vertex d_0 in G to extend D to a GP-operator. If such an operator exists, perform the induced atomic operations. Otherwise it was not possible to remove H.

In general we have different possibilities to choose the correction path D. Each possibility is a column of quadrilaterals within the cylindrical region. We randomly choose one of those candidates and only in cases where we do not find a path, we iteratively test the other ones.

10.3. Greedy Algorithm

Given the above GP-operator to remove a single q-helix, it is straightforward to design a greedy algorithm which removes as many q-helices as possible.

An important side condition within this algorithm is that we forbid all operations which worsen mesh areas which have a nice topological structure. More precisely we identify all dual edge-loops without self-intersections, i.e. all minimal q-helices with pitch 0, and only search for GP-operators which do not destroy them by shifting a neighboring parallel edge. Furthermore we disallow increasing the pitch of all present q-helices with a winding number greater or equal 2. This somehow arbitrary choice is justified by the observation that helices with at least two complete windings most likely increase the base-complex quality and therefore it is often advantageous to protected them from worsening. Both modifications can easily be done by removing graph vertices as explained in section 10.1.

Altogether our base-complex optimizing greedy algorithm works in the following way (cp. Figure 10.10):

1. Identify all minimal q-helices $\{H_i\}$ within the input quadrilateral mesh.

2. Greedily remove the helix with the largest winding number with the algorithm explained in Section 10.2.

3. Apply smoothing to reduce the geometric distortion introduced by shift steps.

4. Go back to step 1. until there is no removable q-helix left.

A naive search for q-helices would first check for all dual vertices whether their orthogonal dual parametric lines intersect each other. If this is the case, the first intersection is a q-helix candidate and we can verify whether the necessary conditions of Section 10.2 are fulfilled and extend the q-helix in both directions as far as possible. By precomputing for each dual half-edge the corresponding parametric line, a local position index on this line and the next self-intersection on this parametric line, the detection of q-helices becomes much faster.

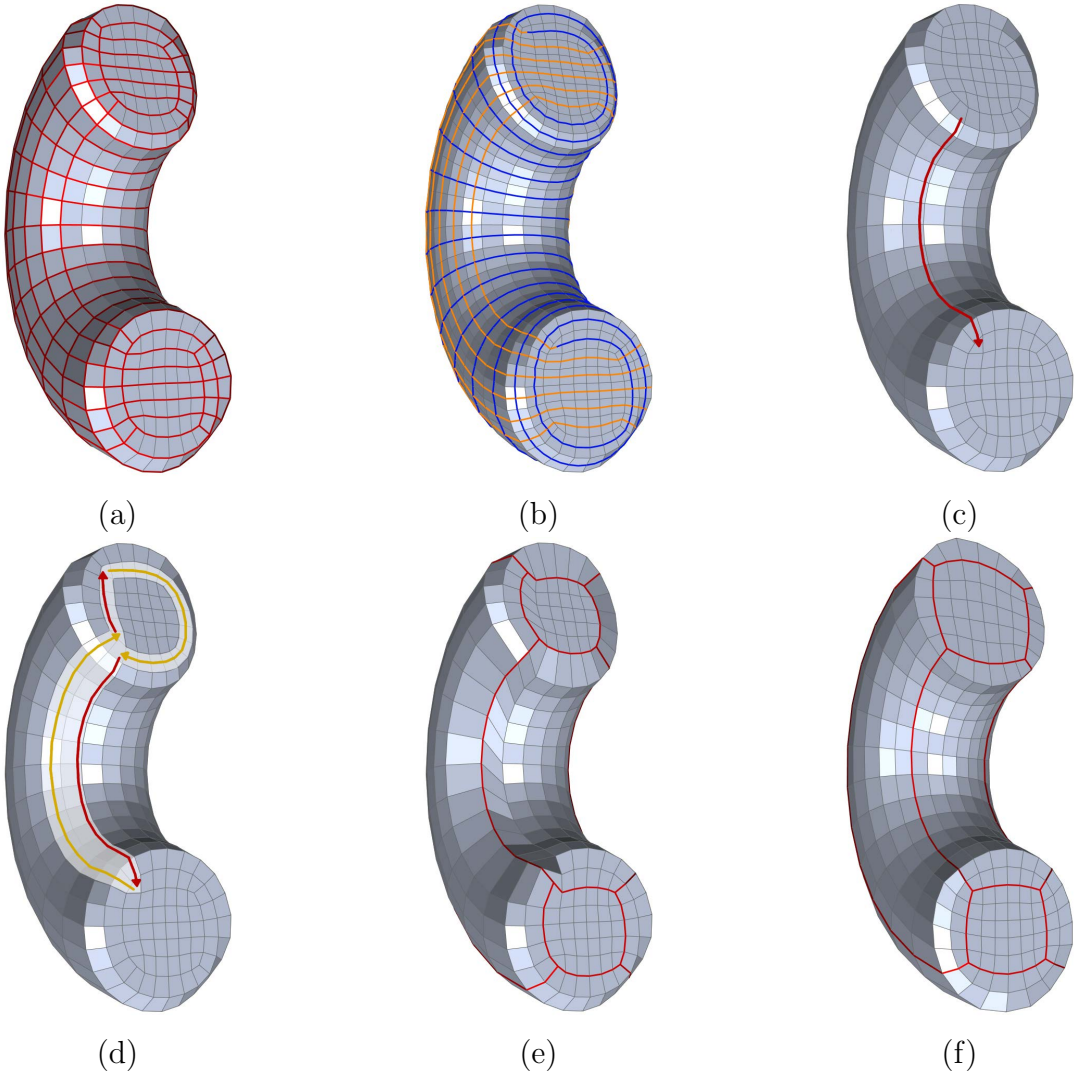


Figure 10.10.: Algorithm example: Figure (a) shows the input mesh with a fine base-complex. Two q-helices (blue and yellow curve) are identified in (b) and the correction path shown in (c) and belonging to the blue helix is extended to a GP-operator in (d). Figure (e) shows the mesh after applying all induced atomic operations. This single operation is sufficient to remove both helices leading to the desired (coarse) base complex. Finally tangential smoothing improves the per element quality (f).

For the smoothing we apply a very simple explicit variant of [ZBX05] as done in [DSSC08] which is able to handle features appropriately. In general it would be possible to leave this step out, however shift operations will locally create unaesthetic angles.

Therefore if not only the topological result is of interest, a tangential relaxation is preferable.

10.4. Evaluation

For the evaluation of our base-complex optimization technique, we apply the method to several quadrilateral meshes generated with the method presented in Chapter 8. As a quantitative evaluation we compare the number of helices and the quality of the base-complex of the input mesh against the optimized mesh as shown in Table 10.1. The quality of the base-complex is measured by the number of its quadrilateral patches , i.e. the number of quadrilaterals that remain after removing all regular parametric lines. All results were computed within a few minutes on a standard PC.

	Input			Output		
Model	#Hel	#F	#F in BC	#Hel	#F	#F in BC
Fandisk	19	764	408	5	506	144
DrillHole	24	3077	1368	7	1948	216
RockerArm	17	3180	1226	3	1678	178
Fertility	46	3357	2271	1	2387	526
Botijo	42	8395	4957	7	5472	1034
Lever	49	7886	5578	10	5850	907
Jet	52	36472	23303	23	31296	1492

Table 10.1.: Statistics of the base-complex optimization: We compare the number of helices # Hel, the number of quadrilaterals of the mesh # F and the number of quadrilaterals of the base-complex # BC between the input and the optimized mesh of several models.

For all meshes most of the q-helices could be removed leading to a significant reduction of the base-complex size. On the Fandisk model the optimization method reduces the size of the base-complex from 408 to 144 quadrilaterals. Furthermore we experimentally collapsed all face-loops that did not lead to singularity merges or collapsing features. In this experiment the base complex could be even reduced to 90 quadrilaterals, as shown in the right most picture of the Fandisk in Figure 10.11. However, this reduction comes at the cost of moving the valence five singularity onto the feature line on top of the

FANDISK which is not optimal and induces unwanted stretch.

Another additional experiment was performed on the FERTILITY model, where the right most picture in Figure 10.11 shows the result of a base-complex optimization where the merging of singularities was allowed. Here the size of the base-complex could be reduced from 526 to 222 but again the overall distortion of the patches increased as a result of the merged singularities. Whether such aggressive reductions are useful depends strongly on the desired application.

The BOTIJO and the LEVER model both have a larger number of singularities leading to rather many separating lines despite the removal of most of the helices. But still the decoupling of quad-loops is advantageous for many applications enabling for example a better control of anisotropic edge-lengths.

Limitations. The presented algorithm works in a greedy fashion and therefore it is no surprise that we cannot guarantee optimality. Due to the iterative graph search it is even not guaranteed to find a suitable GP-operator if one exists. Our experiments showed that prioritizing the q-helices by their winding-number usually leads to good results, but we also experienced counter examples where a different ordering performed better.

Furthermore the resulting base-complex is strongly dependent on the number and placement of singularities in the input, since we do not change them. In particular for unstructured quadrilateral meshes like the cylinder in Figure 10.1a it cannot be expected to achieve a coarser base-complex without adequately adjusting the singularities.

While the topological optimization is completely robust and parameter free, the mesh smoothing may occasionally lead to geometric instabilities. Replacing the explicit smoothing by a superior parametrization based method which e.g. exploits the optimized base-complex could be an interesting research topic for the future.

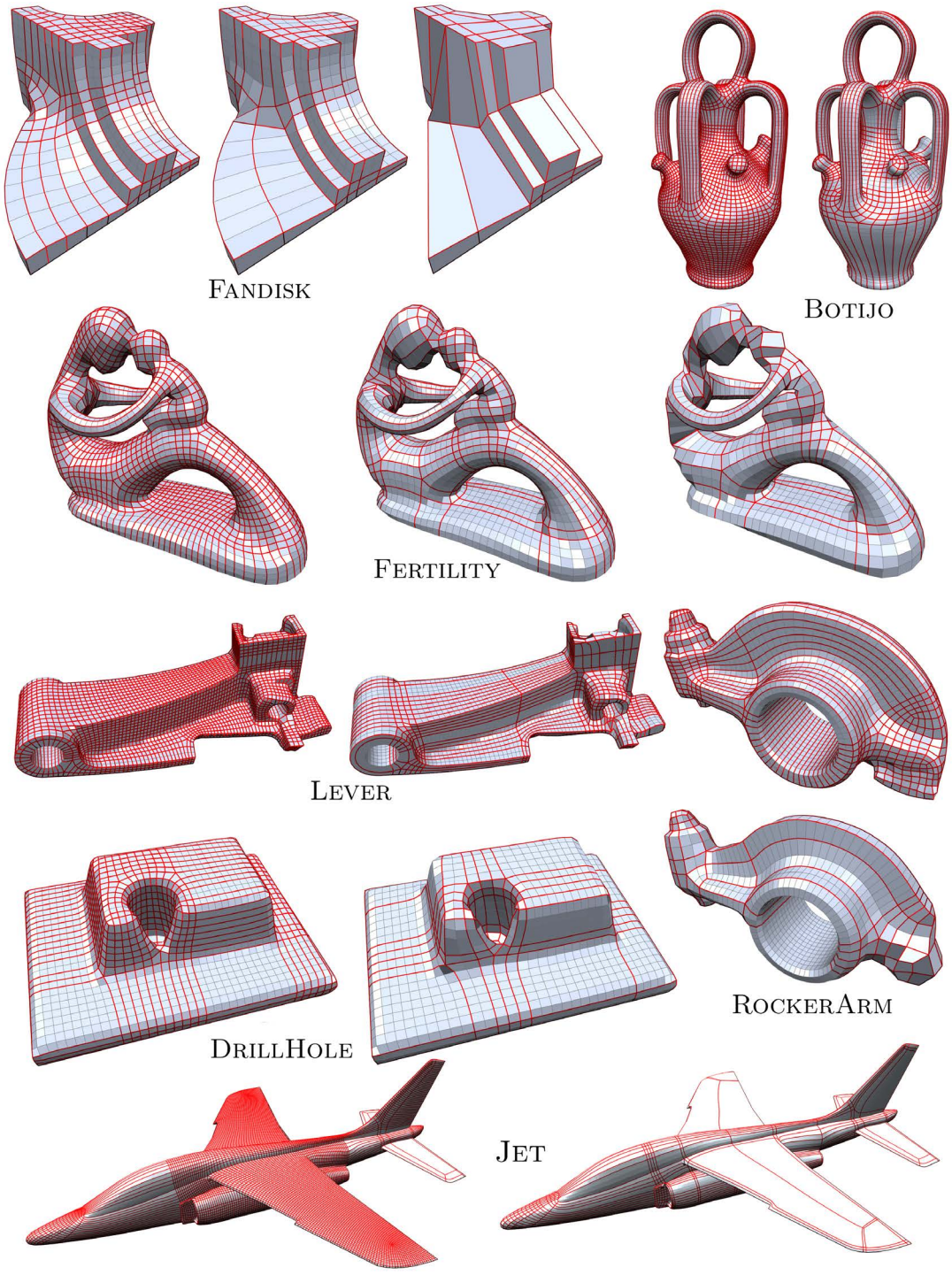


Figure 10.11.: Comparison of various example meshes before and after our base-complex optimization. The red lines indicate the base-complex, i.e. all parameter lines emanating from the singular vertices. For the Fandisk model the third result is a maximal reduction of quadloops without merging singularities, while the third picture of Fertility was created by allowing singularity merges within the helix removal step.

11. Conclusion

In this thesis we studied the generation of quadrilateral surface meshes and proposed novel methods designed for the application in animation and simulation environments. In the following we will summarize our main contributions and give an outlook on interesting future research directions.

Summary

One of the key ingredients of parametrization based quadmesh generation consists in a fast optimization strategy for mixed-integer problems. Consequently, in the first part of this thesis we analyzed general solution strategies for mixed-integer problems and developed a novel highly efficient adaptive optimization algorithm, specially designed for the requirements in geometry processing. Since state-of-the-art solution strategies are far too time consuming, this optimization algorithm is crucial for the parametrization based generation of quadrilateral meshes. One outcome of this thesis is a publicly available implementation of our optimization algorithm, which is able to rapidly approximate huge quadratic mixed-integer problems.

The second part of this thesis dealt with the intended goal of our work, i.e. quadrilateral surface mesh generation. While our layout based approach generalizes previous globally smooth parametrization approaches in order to increase robustness within a reverse engineering environment, it is still limited by the requirement of a predefined rectangular patch layout. Since the automatic generation of such high-quality patch layouts is a very complicated and yet unsolved problem, our second approach, i.e. the orientation-field guided method, tackles the quadmesh generation problem in a different way. Although fully automatic parametrization based construction of a high-quality quadmesh seems to be intractable at a first glance, it turns out that by splitting the overall optimization in an orientation and a metric part, leads to very good results. Both corresponding sub-problems can be solved efficiently by our mixed-integer optimization

algorithm and the resulting algorithm represents an important contribution to the state-of-the-art. Several approaches build on top of our results [KMZ10, MPKZ10, LXW*11], extend it to different input data [LLZ*11, PTSZ11] or other mesh types [NPPZ10]. The most important aspect of our method consists in its flexibility - quadmeshes can either be generated fully automatic or interactively designed by iteratively adding high-level constraints. From this point of view, several other methods which always require guidance, like e.g. the layout based approach, can indeed be seen as special cases of our general strategy.

Quadrialteral meshes generated with our orientation-field guided approach exhibit a well-behaved distribution of irregular vertices. However, their global connectivity often does not lead to a coarse partitioning into rectangular patches. Since several practical applications, as for instance texture and displacement mapping, benefit from a semi-regular mesh, i.e. one with a coarse patch layout, the third part of this thesis was devoted to a global structure optimization. We proposed a novel set of global operators that can be applied to turn a mesh with semi-regular valency into a real semi-regular mesh.

In summary, the combination of both of our proposed algorithms enables a complete mesh generation pipeline, which leads to quadmeshes that fulfill the practical quality requirements of animation and simulation environments.

Outlook

Compared to triangle mesh generation, quadrilateral mesh generation is still relatively unexplored. Consequently, interesting directions for future research can be found in almost all topics addressed in this thesis. While the orientation-field guided approach based on splitting is tempting due to its performance, it implies the drawback of decoupling the orientation computation from the element sizing. For applications which require a precise control over element size or a strongly graded mesh, a better integrated approach which is able to optimize the rotational and metric part simultaneously would be desirable. The same holds for the optimization of the base-complex, which currently

is done in a post-processing step.

Another important aspect for future research is robustness. The proposed stiffening approach mostly prevents flipped triangles in the parametrization, however, without providing any guarantees. While our greedy rounding strategy in combination with stiffening is sufficient for the generation of moderately coarse meshes, in particular the generation of very coarse quadmeshes is delicate. It might happen, that irregular vertices are snapped to the same location in parameter space and thus lead to degeneracies in the mapping function. Finding an efficient formulation that is guaranteed to find valid integer-grid mappings in reasonable time belongs to the most important open questions.

We proposed the new class of GP-operators in order to eliminate helical configurations within a quadmesh. However, we believe that these operators are much more general and could be used for other optimization tasks as well. The combination of GP-operators with other known quadmesh operators into an optimization framework like [TPC*10] seems to be straightforward, however, finding a good prioritization is non-trivial and requires further investigations.

Maybe the most important research direction consists in generalizing the ideas of quadrilateral surface mesh generation to the equivalent volumetric problem, i.e. hexahedral volume meshing. Hexahedral meshes are broadly applied in simulation although their generation is very time consuming due to the absence of a fully automatic approach. The parametrization part of the orientation-field guided approach can be generalized to the volumetric case [NRP11]. However, the automatic generation of 3D orientation-fields is more complicated. Although promising ideas were developed recently [HTWB11], the problem of finding topologically consistent 3D orientation-fields is still unsolved.

Bibliography

[AB99] Amenta N., Bern M. W.: Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry 22*, 4, 1999, 481–504.

[ACSD*03] Alliez P., Cohen-Steiner D., Devillers O., Lévy B., Desbrun M.: Anisotropic polygonal remeshing. In *ACM SIGGRAPH 2003 Papers*, New York, NY, USA, 2003, SIGGRAPH '03, ACM, pp. 485–493.

[AGK] Aurenhammer F., Graz T. U., Klein R.: Voronoi diagrams. In *Handbook of Computational Geometry*, Elsevier Science Publishers B.V. North-Holland, pp. 201–290.

[AUGA05] Alliez P., Ucelli G., Gotsman C., Attene M.: *Recent Advances in Remeshing of Surfaces*. Research report, AIM@SHAPE Network of Excellence, 2005.

[AW11] Alexa M., Wardetzky M.: Discrete laplacians on general polygonal meshes. In *ACM SIGGRAPH 2011 papers*, New York, NY, USA, 2011, SIGGRAPH '11, ACM, pp. 102:1–102:10.

[BBK05] Botsch M., Bommes D., Kobbelt L.: Efficient linear system solvers for mesh processing. In *IMA Conference on the Mathematics of Surfaces*, 2005, Martin R. R., Bez H. E., Sabin M. A., (Eds.), vol. 3604 of *Lecture Notes in Computer Science*, Springer, pp. 62–83.

[BK07] Bommes D., Kobbelt L.: Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. In *VMV*, 2007, pp. 151–160.

[BKP*10] Botsch M., Kobbelt L., Pauly M., Alliez P., Lévy B.: *Polygon Mesh Processing*. AK Peters, 2010.

[BLK11] Bommes D., Lempfer T., Kobbelt L.: Global structure optimization of quadrilateral meshes. *Comput. Graph. Forum 30*, 2, 2011, 375–384.

[BLP*12] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D.: State of the art in quad meshing. In *Eurographics STARS*, 2012.

[BVK10] Bommes D., Vossemer T., Kobbelt L.: Quadrangular parameterization for reverse engineering. In *Proceedings of the 7th international conference on Mathematical Methods for Curves and Surfaces*, Berlin, Heidelberg, 2010, MMCS'08, Springer-Verlag, pp. 55–69.

[BZK09] Bommes D., Zimmer H., Kobbelt L.: Mixed-integer quadrangulation. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, New York, NY, USA, 2009, ACM, pp. 1–10.

[BZK12] Bommes D., Zimmer H., Kobbelt L.: Practical mixed-integer optimization for geometry processing. In *Proceedings of the 7th international conference on Curves and Surfaces*, Berlin, Heidelberg, 2012, Springer-Verlag, pp. 193–206.

[CBK12] Campen M., Bommes D., Kobbelt L.: Dual loops meshing: Quality quad layouts on manifolds. *ACM Trans. Graph. 31*, 4, 2012.

[CDHR06] Chen Y., Davis T. A., Hager W. W., Rajamanickam S.: *Algorithm 8xx: CHOLMOD, supernodal sparse Cholesky factorization and update/-downdate*. Technical Report TR-2006-005, University of Florida, 2006.

[Cha91] Chazelle B.: Triangulating a simple polygon in linear time. *Discrete Comput. Geom. 6*, 5, Aug. 1991, 485–524.

[Chv73] Chvátal V.: Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics 4*, 4, Apr. 1973, 305–337.

[CMS97] Cignoni P., Montani C., Scopigno R.: A comparison of mesh simplification algorithms. *Computers & Graphics 22*, 1997, 37–54.

[CP05] Cazals F., Pouget M.: Estimating Differential Quantities using Polynomial fitting of Osculating Jets. *Computer Aided Geometric Design 22*, 2, 2005, 121–146.

[CSM03] Cohen-Steiner D., Morvan J.-M.: Restricted delaunay triangulations and normal cycle. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, 2003, pp. 312–321.

[D'A00] D'Azevedo E. F.: Are bilinear quadrilaterals better than linear triangles? *SIAM Journal on Scientific Computing 22*, 1, Jan. 2000, 198–217.

[DBG*06] Dong S., Bremer P.-T., Garland M., Pascucci V., Hart J. C.: Spectral surface quadrangulation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, 2006, pp. 1057–1066.

[DEG*99] Demmel J. W., Eisenstat S. C., Gilbert J. R., Li X. S., Liu J. W. H.: A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications 20*, 3, 1999, 720–755.

[DFG99] Du, Faber, Gunzburger: Centroidal voronoi tessellations: Applications and algorithms. *SIREV: SIAM Review 41*, 1999.

[DG86] Duran M. A., Grossmann I. E.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Math. Program. 36*, 3, Dec. 1986, 307–339.

[DISSC08] Daniels II J., Silva C. T., Shepherd J., Cohen E.: Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, December 2008, 148:1–148:9.

[DMSB99] Desbrun M., Meyer M., Schröder P., Barr A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1999, SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 317–324.

[DSC09a] Daniels II J., Silva C. T., Cohen E.: Localized quadrilateral coarsening. In *SGP '09: Proceedings of the Symposium on Geometry Processing*, Aire-la-Ville, Switzerland, Switzerland, 2009, Eurographics Association, pp. 1437–1444.

[DSC09b] Daniels II J., Silva C. T., Cohen E.: Semi-regular quadrilateral-only remeshing from simplified base domains. In *SGP '09: Proceedings of the*

Symposium on Geometry Processing, Aire-la-Ville, Switzerland, Switzerland, 2009, Eurographics Association, pp. 1427–1435.

[DSSC08] Daniels J., Silva C. T., Shepherd J., Cohen E.: Quadrilateral mesh simplification. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, New York, NY, USA, 2008, ACM, pp. 1–9.

[EKS*10] Eigensatz M., Kilian M., Schiftner A., Mitra N. J., Pottmann H., Pauly M.: Paneling architectural freeform surfaces. *ACM Trans. Graph. 29*, July 2010, 45:1–45:10.

[FLHCO10] Fu C.-W., Lai C.-F., He Y., Cohen-Or D.: K-set tilable surfaces. *ACM Trans. Graph. 29*, July 2010, 44:1–44:6.

[FM84] Fournier A., Montuno D. Y.: Triangulating simple polygons and equivalent problems. *ACM Trans. Graph. 3*, 2, Apr. 1984, 153–174.

[GGK98] Gotsman C., Gumhold S., Kobbelt L.: Simplification and compression of 3d meshes. In *In Proceedings of the European Summer School on Principles of Multiresolution in Geometric Modelling (PRIMUS*, 1998, Springer, pp. 319–361.

[GLLR11] Gurung T., Laney D., Lindstrom P., Rossignac J.: Squad: Compact representation for triangle meshes. *Computer Graphics Forum 30*, 2, 2011, 355–364.

[GRB11] Gu Z., Rothberg E., Bixby R.: Gurobi optimizer 4.5: http://www.gurobi.com, 2011.

[Gro02] Grossmann I. E.: Review of nonlinear mixed-integer and disjunctive programming techniques. *Methods 3*, 3, 2002, 227–252.

[GSC*04] Glymph J., Shelden D., Ceccato C., Mussel J., Schober H.: A parametric strategy for free-form glass structures using quadrilateral planar facets. *Automation in Construction 13*, 2, 2004, 187 – 202. Conference of the Association for Computer Aided Design in Architecture.

[GSS96] Groenwold A. A., Stander N., Snyman J. A.: A pseudo-discrete rounding method for structural optimization. *Structural and Multidisciplinary Optimization 11*, 3, 1996, 218–227.

[HCB05] Hughes T. J. R., Cottrell J. A., Bazilevs Y.: Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering 194*, 39-41, 2005, 4135–4195.

[HLS07] Hormann K., Lévy B., Sheffer A.: Mesh parameterization: theory and practice. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, 2007, p. 1.

[HPW05] Hildebrandt K., Polthier K., Wardetzky M.: Smooth feature lines on surface meshes. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland, 2005, Eurographics Association, p. 85.

[HTWB11] Huang J., Tong Y., Wei H., Bao H.: Boundary aligned smooth 3d cross-frame field. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, New York, NY, USA, 2011, SA '11, ACM, pp. 143:1–143:8.

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2000, ACM Press/Addison-Wesley Publishing Co., pp. 517–526.

[HZM*08] Huang J., Zhang M., Ma J., Liu X., Kobbelt L., Bao H.: Spectral quadrangulation with orientation and alignment control. *ACM Trans. Graph. 27*, 5, 2008, 1–9.

[IBM12] IBM: Ilog cplex optimizer 12: http://www.ibm.com, 2012.

[KBSS01] Kobbelt L. P., Botsch M., Schwanecke U., Seidel H.-P.: Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2001, SIGGRAPH '01, ACM, pp. 57–66.

[Kin97] Kinney P.: Cleanup: Improving quadrilateral finite element meshes. In *6th International Meshing Roundtable*, 1997, pp. 437–447.

[KMZ10] Kovacs D., Myles A., Zorin D.: Anisotropic quadrangulation. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, New York, NY, USA, 2010, SPM '10, ACM, pp. 137–146.

[KNP07] Kälberer F., Nieser M., Polthier K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3, Sept. 2007, 375–384.

[Knu01] Knupp P. M.: Algebraic mesh quality metrics. *SIAM J. Sci. Comput. 23*, 1, Jan. 2001, 193–218.

[KS98] Kimmel R., Sethian J. A.: Computing geodesic paths on manifolds. In *Proc. Natl. Acad. Sci. USA*, 1998, pp. 8431–8435.

[LJX*10] Lai Y.-K., Jin M., Xie X., He Y., Palacios J., Zhang E., Hu S.-M., Gu X.: Metric-driven rosy field design and remeshing. *IEEE Trans. Vis. Comput. Graph.*, 2010, 95–108.

[LKH08] Lai Y.-K., Kobbelt L., Hu S.-M.: An incremental approach to feature aligned quad dominant remeshing. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, 2008, pp. 137–145.

[LL10] Lévy B., Liu Y.: Lp Centroidal Voronoi Tesselation and its applications. *ACM Transactions on Graphics 29*, 4, 2010.

[Llo82] Lloyd S. P.: Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*, 2, 1982, 129–137.

[LLS01] Litke N., Levin A., Schroeder P.: Fitting subdivision surfaces. In *IEEE Visualization 2001*, October 2001, pp. 319–324.

[LLZ*11] Li E., Lvy B., Zhang X., Che W., Dong W., Paul J.-C.: Meshless quadrangulation by global parametrization. *Computer and Graphics*, 2011.

[Lue01] Luebke D. P.: A developer's survey of polygonal simplification algorithms. *IEEE COMPUTER GRAPHICS AND APPLICATIONS 21*, 2001, 24–35.

[LXW*11] Liu Y., Xu W., Wang J., Zhu L., Guo B., Chen F., Wang G.: General planar quadrilateral mesh design using conjugate direction field. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, New York, NY, USA, 2011, SA '11, ACM, pp. 140:1–140:10.

[MK04] Marinov M., Kobbelt L.: Direct anisotropic quad-dominant remeshing. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, Washington, DC, USA, 2004, IEEE Computer Society, pp. 207–216.

[MMP87] Mitchell J. S. B., Mount D. M., Papadimitriou C. H.: The discrete geodesic problem. *SIAM J. Comput. 16*, 4, Aug. 1987, 647–668.

[MMWW02] Marchand H., Martin A., Weismantel R., Wolsey L.: Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics 123*, 1-3, 2002, 397–446.

[MPKZ10] Myles A., Pietroni N., Kovacs D., Zorin D.: Feature-aligned t-meshes. *ACM Trans. Graph. 29*, 4, 2010, 1–11.

[MTTT98] Meyers R. J., Tautges T. J., Tuchinsky P. M., Tuchinsky D. P. M.: The "hex-tet" hex-dominant meshing algorithm as implemented in cubit. In *In Proceedings, 7th International Meshing Roundtable 98*, 1998, pp. 151–158.

[NK02] Novotni M., Klein R.: Computing geodesic distances on triangular meshes. In *The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2002 (WSCG'2002)*, Feb. 2002.

[NPPZ10] Nieser M., Palacios J., Polthier K., Zhang E.: Hexagonal global parameterization of arbitrary surfaces. In *ACM SIGGRAPH ASIA 2010 Sketches*, New York, NY, USA, 2010, SA '10, ACM, pp. 5:1–5:2.

[NRP11] Nieser M., Reitebuch U., Polthier K.: Cubecover- parameterization of 3d volumes. *Comput. Graph. Forum 30*, 5, 2011, 1397–1406.

[NW99] Nocedal J., Wright S. J.: *Numerical Optimization*. Springer, August 1999.

[NW06] Nocedal J., Wright S. J.: *Numerical Optimization*, 2nd ed. Springer, New York, 2006.

[PTC10] Pietroni N., Tarini M., Cignoni P.: Almost isometric mesh parameterization through abstract domains. *IEEE Transaction on Visualization and Computer Graphics 16*, 4, July/August 2010, 621–635.

[PTSZ11] Pietroni N., Tarini M., Sorkine O., Zorin D.: Global parametrization of range image sets. *ACM Transactions on Graphics, Proceedings of SIGGRAPH Asia 2011 30*, 6, 2011.

[PW08] Pottmann H., Wallner J.: The focal geometry of circular and conical meshes. *Adv. Comp. Math 29*, 2008, 249–268.

[PZ07] Palacios J., Zhang E.: Rotational symmetry field design on surfaces. *ACM Trans. Graph. 26*, 3, 2007, 55.

[PZKW11] Peng C.-H., Zhang E., Kobayashi Y., Wonka P.: Connectivity editing for quadrilateral meshes. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA) 30*, 6, 2011.

[Ren03] Renard Y.: gmm++, Generic Matrix Methods. http://home.gna.org/getfem/gmm_intro.html, 2003.

[Rin88] Ringertz U. T.: On methods for discrete structural optimization. *Engineering Optimization 13*, 1, 1988, 47 – 64.

[RLL*06] Ray N., Li W. C., Lévy B., Sheffer A., Alliez P.: Periodic global parameterization. *ACM Trans. Graph. 25*, October 2006, 1460–1485.

[RLS*11] Remacle J.-F., Lambrechts J., Seny B., Marchandise E., Johnen A., Geuzaine C.: Blossom-quad: a non-uniform quadrilateral mesh generator using a minimum cost perfect matching algorithm. *International Journal for Numerical Methods in Engineering*, 2011. accepted.

[Rus06] Russell M.: Cutting planes for mixed integer programming. *Optimization*, 2006.

[RVAL09] Ray N., Vallet B., Alonso L., Levy B.: Geometry-aware direction field processing. *ACM Trans. Graph. 29*, December 2009, 1:1–1:11.

[RVLL08] Ray N., Vallet B., Li W. C., Lévy B.: N-symmetry direction field design. *ACM Trans. Graph. 27*, 2, 2008, 1–13.

[SDW*10] Shepherd J. F., Dewey M. W., Woodbury A. C., Benzley S. E., Staten M. L., Owen S. J.: Adaptive mesh coarsening for quadrilateral and hexahedral meshes. *Finite Elements in Analysis and Design 46*, 1-2, 2010, 17 – 32. Mesh Generation - Applications and Adaptation.

[SSK*05] Surazhsky V., Surazhsky T., Kirsanov D., Gortler S. J., Hoppe H.: Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3, July 2005, 553–560.

[SZBN03] Sederberg T. W., Zheng J., Bakenov A., Nasri A.: T-splines and t-nurccs. *ACM Trans. Graph. 22*, 3, July 2003, 477–484.

[TACSD06] Tong Y., Alliez P., Cohen-Steiner D., Desbrun M.: Designing quadrangulations with discrete harmonic forms. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland, 2006, SGP '06, Eurographics Association, pp. 201–210.

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: Polycube-maps. *ACM Trans. Graph. 23*, Aug. 2004, 853–860.

[TPC*10] Tarini M., Pietroni N., Cignoni P., Panozzo D., Puppo E.: Practical quad mesh simplification. *Computer Graphics Forum (Special Issue of Eurographics 2010 Conference) 29*, 2, 2010, 407–418.

[TPP*11] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P.: Simple quad domains for field aligned mesh parametrization. *ACM Transactions on Graphics, Proceedings of SIGGRAPH Asia 2011 30*, 6, 2011.

[VZ01] Velho L., Zorin D.: 48 subdivision. *Computer Aided Geometric Design 18*, 5, 2001, 397 – 427. Subdivision Algorithms.

[WB06] Wächter A., Biegler L. T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program. 106*, May 2006, 25–57.

[WP94] Westerlund T., Pettersson F.: *A Cutting Plane Method for Solving Convex MINLP Problems.* Åbo Akademi, 1994.

[YYPM11] Yang Y.-L., Yang Y.-J., Pottmann H., Mitra N. J.: Shape space exploration of constrained meshes. *ACM Trans. Graph. 30*, 6, Dec. 2011, 124:1–124:12.

[YZJ03] Yu X., Zhang S., Johnson E.: A discrete post-processing method for structural optimization. *Engineering with Computers 19*, 2, 2003, 213–220.

[ZBX05] Zhang Y., Bajaj R., Xu G.: Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In *In Proceedings, 14th International Meshing Roundtable*, 2005, John Wiley & Sons, pp. 449–468.

[ZCBK12] Zimmer H., Campen M., Bommes D., Kobbelt L.: Rationalization of triangle-based point-folding structures. *Comp. Graph. Forum 31*, 2pt3, May 2012, 611–620.

[ZHLB10] Zhang M., Huang J., Liu X., Bao H.: A wave-based anisotropic quadrangulation method. *ACM Trans. Graph. 29*, July 2010, 118:1–118:8.

[ZS00] Zorin D., Schröder P.: *Subdivision for Modeling and Animation*. Tech. rep., SIGGRAPH 2000, 2000. Course Notes.

Curriculum Vitae

David Bommes

E-Mail	David.Bommes@gmail.com
Date of Birth	31.12.1979
Place of Birth	Korschenbroich, Germany
Citizenship	German

Academic Education

Apr. 2006 – Oct. 2012	Doctoral Student at RWTH Aachen University, Computer Graphics Group. Degree: Dr. rer. nat. Supervisor: Prof. Dr. Leif Kobbelt.
Oct. 2000 – Mar. 2006	Computer Science Studies at RWTH Aachen University. Degree: Dipl. Inform. Supervisor: Prof. Dr. Leif Kobbelt.

Publications

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, Denis Zorin: *State of the art in quad meshing*, Eurographics STARS, 2012

Marcel Campen, David Bommes, Leif Kobbelt: *Dual Loops Meshing: Quality Quad Layouts on Manifolds*, ACM Transactions on Graphics, Volume 31, Number 4, (Proc. of SIGGRAPH), 2012

Henrik Zimmer, Marcel Campen, David Bommes, Leif Kobbelt: *Rationalization of Triangle-Based Point-Folding Structures*, Computer Graphics Forum, Volume 31, Number 2, 611-620, (Proc. of EUROGRAPHICS), 2012

David Bommes, Timm Lempfer, Leif Kobbelt: *Global Structure Optimization of Quadrilateral Meshes*, Computer Graphics Forum, Volume 30, Number 2, 375-384, (Proc. of EUROGRAPHICS), 2011

David Bommes, Henrik Zimmer, Leif Kobbelt: *Practical Mixed-Integer Optimization for Geometry Processing*, Lecture Notes in Computer Science, (Proc. of MMCS'10), 2010

David Bommes, Henrik Zimmer, Leif Kobbelt: *Mixed-Integer Quadrangulation*, ACM Transactions on Graphics, Volume 28, Number 3, (Proc. of SIGGRAPH), 2009

David Bommes, Tobias Vossemer, Leif Kobbelt: *Quadrangular Parametrization for Reverse Engineering*, Lecture Notes in Computer Science, (Proc. of MMCS'08), 2008

David Bommes, Leif Kobbelt: *Accurate Computation of Geodesic Distance Fields for Polygonal Curves on Triangle Meshes*, Proc. of Vision Modeling, and Visualization, 151-160, 2007

David Bommes: *Physically based Segmentation of 3D Objects*, Diploma Thesis at RWTH Aachen University supervised by Prof. Dr. Leif Kobbelt and Prof. Dr. Christian Bischof, 2006

Mario Botsch, David Bommes, Leif Kobbelt: *Efficient Linear System Solvers for Mesh Processing*, IMA Conference on the Mathematics of Surfaces, 62-83, 2005

Mario Botsch, David Bommes, Christoph Vogel, Leif Kobbelt: *GPU-Based Tolerance Volumes for Mesh Processing*, Pacific Conference on Computer Graphics and Applications, 237-243, 2004