Quad Layouts – Generation and Optimization of Conforming Quadrilateral Surface Partitions

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
Marcel Campen
aus Viersen, Deutschland

Berichter: Universitätsprofessor Dr. Leif Kobbelt
Universitätsprofessor Denis Zorin, Ph.D.

Tag der mündlichen Prüfung: 01.12.2014

Selected Topics in Computer Graphics

herausgegeben von
Prof. Dr. Leif Kobbelt
Lehrstuhl für Informatik 8
Computergraphik & Multimedia
RWTH Aachen University

Band 13

Marcel Campen

Quad Layouts

Generation and Optimization of
Conforming Quadrilateral Surface Partitions

Shaker Verlag
Aachen 2015

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: D 82 (Diss. RWTH Aachen University, 2014)

Printed in Germany.

ISBN 978-3-8440-3446-2
ISSN 1861-2660

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Abstract

The efficient, computer aided or automatic generation of quad layouts, i.e. the partitioning of an object's surface into simple networks of conforming quadrilateral patches, is a task that – despite its importance and utility in Computer Graphics and Geometric Modeling – received relatively low attention in the past. As a consequence, this task is most often performed manually by well-trained experts in practice, where quad layouts are of particular interest for surface representation and parameterization tasks. Deeper analysis reveals the inherent complexity of this problem, which might be one of the underlying reasons for this situation.

In this thesis we investigate the structure of the problem and the commonly relevant quality criteria. Based on this we develop novel efficient solution strategies and algorithms for the generation of high quality quad layouts. In particular, we present a fully automatic as well as an interactive pipeline for this task. Both are based on splitting the hard problem into sub-problems with a simpler structure each. For each sub-problem we design efficient, custom-tailored optimization algorithms motivated by the geometric nature of these problems. In this process we pay attention to compatibility, such that these algorithms can be applied in sequence, forming the stages of efficient quad layouting pipelines.

An important aspect of the quad layout problem is the complexity of the quality objective. The quality typically is a function of the layout's structural complexity, its topological connectivity, and its geometrical embedding, i.e. of discrete, combinatorial, and continuous aspects. Furthermore, application-specific demands can be quite fuzzy and hard to formalize. Our automatic pipeline follows a generic set of quality criteria that are common to most use cases. The best solution to make possible the optimization with respect to more specific design intents is to include the user in the process, enabling them to infuse their expert knowledge. In contrast to prevalent manual construction processes our interactive pipeline supports the user to a large extent. Structural consistency

is automatically taken care of, geometrically appropriate design operations are automatically proposed, and next steps that should be taken are indicated. In this way the required effort is reduced to a minimum, while still full design flexibility is provided.

Finally, we present novel methods for the computation of geodesic distances and paths on surfaces – for standard as well as anisotropic metrics. These play a critical key role in several parts of our pipelines and shortcomings of available solutions compelled the development of novel alternatives.

Acknowledgments

This thesis originated from my work at the Visual Computing Institute at RWTH Aachen University where I held a position as a doctoral researcher in the Graphics, Geometry, and Multimedia group. Funding was generously provided by the DFG Cluster of Excellence UMIC and the ERC Advanced Grant ACROSS.

First and foremost I want to express my gratitude to my advisor Leif Kobbelt for his distinguished scientific guidance and support, and for always granting me the invaluable freedom to investigate the topics that intrigued me the most. I am also deeply thankful to Denis Zorin for bringing in his profound expertise in the role of the second referee.

For nearly endless discussions and common philosophizing about "everything quad" I want to thank David Bommes. This ongoing dialog was a constant source of inspiration for the many ideas that build the foundation for the contributions presented herein. Henrik Zimmer, Hans-Christian Ebke, and Pierre Alliez likewise deserve my gratitude for their dedication to the frank discussion and progression of ideas and for sharing their valued thoughts.

It has always been a pleasure to collaborate with colleagues in the context of various projects. In particular, I am obliged to Marco Attene for the superb collaboration in the development of a state-of-the-art report and of conference and graduate school courses. Darko Pavić deserves thanks for his great cooperative work on my first research project. The supervision of undergraduate students likewise was a gratifying experience that not rarely sparked new ideas. In this regard I am particularly indebted to Martin Heistermann, Moritz Ibing, Max Lyon, and Patrick Schmidt. Jan Möbius deserves my gratitude for his incessant technical support during all of my projects.

Finally, I would like to sincerely thank all of my colleagues at the institute, my family and friends for providing the comfortable, winsome atmosphere – in work as in leisure – that certainly conduced to the success of this work.

Contents

1. Introduction

The physical world surrounding us can be seen as a collection of a large number of three-dimensional objects and subjects. Besides this physical world, an ever-growing variety of virtual processes or even entire virtual worlds is leveraged by modern computer technology. Examples range from systems for computer-aided design and manufacturing over rapid prototyping to simulation, from e-commerce and product visualization to games, movies, and other immersive entertainment applications. Not surprisingly, one of the most fundamental ingredients in such virtual scenarios are three-dimensional objects – whether virtual reproductions of physical entities or original representations of virtually generated ones. In any case, these objects need to be represented digitally in some form. One of the most common types of representation is via formal geometrical description of the object's surface. While the detailed internal, volumetric structure and constitution of an object can well be of interest for certain advanced applications, the surface is clearly the most prominent feature of an object: it is what we see of the object, it defines its shape, it separates interior from exterior, i.e. it defines the interface between the object and the surrounding world. For primitive objects like spheres, cylinders, or tori a single simple parametric expression suffices for its explicit representation, as depicted on the right for the example of the unit sphere. Other objects with more complex geometry or more complex topology, however, typically cannot practically be expressed in such simple closed form. This led to the development of *piecewise* surface representations, where each piece of a partition of a complex surface can be represented by a simple function. Analogously, this applies to maps from or onto complex surfaces, which are handled more easily in a piecewise manner.

$$p(\theta, \phi) = \begin{pmatrix} \cos\theta \sin\phi \\ \sin\theta \sin\phi \\ \cos\phi \end{pmatrix}$$

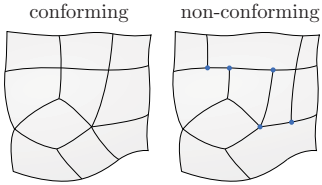
This thesis is concerned with the partitioning of an object's surface into pieces suitable for sophisticated forms of such surface and map representations, as detailed in the following.

Piecewise Surfaces

Examples of very common piecewise surface representations are:

- Triangle meshes (piecewise linear)
- Quad meshes (piecewise bilinear)
- Multiblock grids (piecewise regular quad grid)
- Bézier/Spline/NURBS networks (piecewise polynomial)

In the case of triangle meshes, the pieces (or *patches*) are trilateral, in all other cases quadrilateral, as illustrated in Figure 1.1. Furthermore, in all these cases the partition is *conforming*, i.e. such that patch corners coincide only with other patch corners (no *T-junctions*) as illustrated here.



Building such piecewise representations for an object's surface obviously involves two aspects: suitably partitioning the surface into pieces and finding a suitable representation for the geometry within each piece. For the case of triangle meshes, this is a well-researched, well-understood problem and solutions have been proposed already some time ago [AUGA08]. More recently, a lot of research has been devoted to the automatic generation of quad meshes and significant advances have been made [BLP*13]. For the other cases, however, while methods for determining a suitable representation per piece are available (e.g. on the basis of grid fitting or interval assignment [TA93, Mit00, BVK08], spline fitting [EH96, KL96, MK95, MBVW95, AAB*88], or subdivision surface fitting [LLS01]), the generation of appropriate partitions in the first place proved to be a hard problem which commonly implies tedious manual efforts in practice. In these cases the required partition is a so-called *quad layout* – which, while formally being structurally equivalent to a quad mesh, follows quite different geometric objectives. In particular, this necessitates different generation strategies: while a quad mesh basically

resamples a surface and its quality can be assessed locally, in the case of a quad layout the structural, topological aspect from a global point of view plays an important role.

It bears noting that there are further types of piecewise surface representation which do not rely on a quad layout, but on simpler-to-construct partitions with triangular or polygonal patches. Bézier triangles and certain types of box splines are examples of such representations. Their popularity in practice and support in software tools, however, is significantly lower compared to representations based on quadrilateral partitions. There are multiple reasons for this circumstance. One can argue from a historic point of view that in the early days of Computer-Aided Geometric Design in the automotive industry tensor-product Bézier and B-Spline patches with their quadrilateral domains already set the standard in this field [Far02]. But there are also solid theoretical reasons for preferring quad structures over triangular structures: for instance, it is well known from differential geometry that there are two orthogonal *principal curvature directions* at any point on a surface, the one of minimal and the one of maximal curvature [dC76]. The boundaries of quad layout patches can very naturally be aligned to these, with aesthetic as well as practical advantages [BLP*13].

Also several subdivision surface schemes are able to operate on complexes of arbitrary polygonal patches, but quadrilateral patches are advantageous for the smoothness of the resulting surfaces [Rei95] for prominent schemes [CC78, DS78].

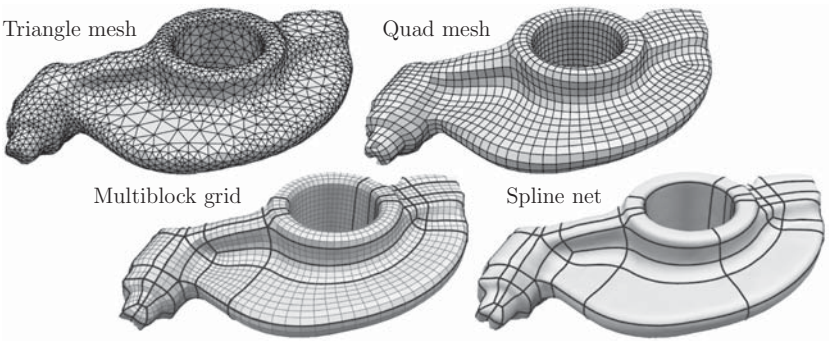


Figure 1.1.: Illustration of different kinds of *piecewise surface representations*. In the case of the triangle mesh the pieces are trilateral, in the other three cases quadrilateral.

Piecewise Maps

Not only for the representation of a surface itself, also for the representation of maps from or onto the surface (i.e. parameterizations, texture maps, displacement maps, etc.) is a quad layout of great value. When dealing with surfaces of three-dimensional objects, one most often deals with 2-manifold surfaces (with or without boundaries), i.e. they are locally homeomorphic to a disc, thus amenable to (local) parameterization over $\mathbb{R}^2$. The correspondences provided by such maps for instance allow to apply simpler 2D operations to the surface, although it is embedded in $\mathbb{R}^3$.

For such use cases, however, often a *global* parameterization is necessary, for which the whole surface needs to be homeomorphic to a disc. Hence, the surface needs to be cut. While a minimal *cut graph* [EHP02] is topologically sufficient, the resulting metric distortion in the map can be a problem for many applications.

Therefore, in practice often a chart atlas is used: the surface is cut (i.e. partitioned) into multiple charts which can be parameterized with low metric distortion. For various reasons (domain simplicity, transition simplicity, continuity across chart borders [RNLL10, MZ12]) it is advantageous if these charts are quads and these quads are conforming, i.e. again one is interested in quad layouts. Figure 1.2 illustrates a quad layout based piecewise map from the plane onto the surface.

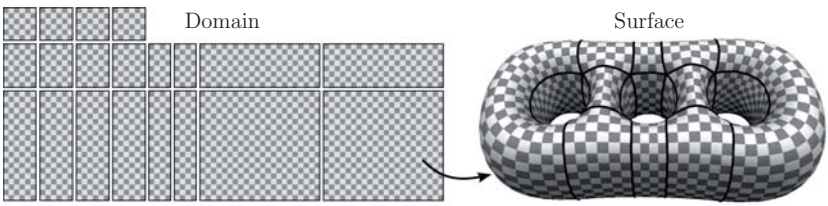


Figure 1.2.: Illustration of a *piecewise map* from the plane onto a surface using a quad layout based chart atlas.

Quad Layouts

The generation of such quad layouts, for purposes like surface or map representation, is the main topic of this thesis. A major source of motivation for the investigation of this field comes from the fact that the – often manual or at most semi-automatic – creation of partitions, in particular quad meshes, multiblock grids, spline networks, or subdivision base meshes, is an extremely time-consuming task in practice [HCB05, LRL06].

We will see that the mentioned hardness of the problem of automatic quad layout generation comes from two facts:

1. the optimization problem for the generation of high quality quad layouts is of mixed nature: it has continuous, discrete, and combinatorial or topological degrees of freedom (cf. Chapter 3),

2. the notion of quality of a quad layout is application-dependent, sometimes fuzzy, thus hard to formalize (cf. Section 2.2).

After we introduced foundations and requisite definitions in Chapter 2, we present our strategy to tackle this problem in Chapter 3. It is based on dividing the problem into three sub-problems whose degrees of freedom are more uniform and which can be solved in a serial manner by specialized, geometrically motivated optimization strategies. In order to deal with the variable notion of layout quality, we follow two different paths: in Part I a fully automatic quad layout construction pipeline is described, which relies on a very general notion of quality. We thus do not target a specific application, but present a generic method – notwithstanding that based on this fundament also specializations or adaptations to certain applications with their specific quality measures could be developed in the future. In some scenarios, however, the desired notion of quality can be quite fuzzy and hard to formalize – for example it may only exist (subconsciously) in some experts' minds based on years of experience, or it may be influenced by aspects of aesthetics. Therefore, in Part II we describe an interactive quad layout design system which grants the user the flexibility to incorporate their specific design intents. At the same time, the system highly supports and guides the user, taking care of consistency aspects and assessing geometric quality.

Thesis Structure and Contributions

Part I An efficient, specialized optimization strategy for each of the three sub-problems is presented, based on the publications *Dual Loops Meshing: Quality Quad Layouts on Manifolds* [CBK12] and *Quad Layout Embedding via Aligned Parameterization* [CK14b]. The resulting pipeline enables the automatic generation of quad layouts on surfaces.

Part II Alternative strategies are presented which allow for interactive design of quad layouts, based on the publication *Dual Strip Weaving: Interactive Design of Quad Layouts using Elastica Strips* [CK14a]. Putting the user into the loop enables respecting application-dependent quality criteria which can be hard to formalize but easier to judge for an expert user. We provide high-level interaction tools and visual guides which keep the amount of user effort low, while providing full flexibility.

Part III We describe novel strategies for the efficient approximation of (anisotropic) geodesic distances and paths on surfaces, based on the publications *Practical Anisotropic Geodesy* [CHK13] and *Walking on Broken Mesh: Defect-Tolerant Geodesic Distances and Parameterizations* [CK11]. Such computations are heavily made use of in our quad layout generation and optimization algorithms. The shortcomings of existing approaches compelled the development of novel techniques in this field.

Note The following chapters are largely based on the above mentioned articles [CK11, CBK12, CHK13, CK14a, CK14b], which have been published before. Excerpts thereof are used, enriched with additional information, and several aspects are elaborated in more detail. In all of these works this thesis' author, Marcel Campen, took the leading role in developing the concept, performing the implementation, and writing the article.

2. Quad Layouts

Since the early days of Computer-Aided Geometric Design the partitioning of a complex surface into quadrilateral patches (or conversely: its composition thereof) has been an essential requirement and fundamental challenge. The tensor product nature of smooth surface representations like Bézier, B-Spline, or NURBS patches [Far02] once established the need for such structures. Especially for reverse engineering purposes, the efficient creation of such partitionings has always been an important problem, as outlined by Li et al. [LRL06].

Quad meshes, which have seen an increase in popularity in recent years, sparked new interest in this problem [TPP*11, BLK11, CBK12]. This is due to the fact that *semi-regular* quad meshes (also known as *multiblock grids* [SB96]), which contain an underlying coarse quadrilateral base structure, i.e. which are a regular refinement of a quad layout, provide advantages for various application cases, as detailed in a recent survey [BLP*13] which deems them "the most important class [of quad meshes] in terms of applications". For instance, they enable the application of efficient adaptive and multi-level solver schemes [BDL10, DHM09] in the context of quad-based Finite Element simulation and the application of degree adaptation techniques in the context of Isogeometric Analysis [HCB05, Bom12]. Their high level of structuredness is of benefit for applications like mesh compression [AG03] and Fourier or Wavelet based processing [AUGA08]. In the field of character animation designers are interested in quad meshes with good so-called *edge-flow* [JLW10] (a concept closely related to a geometry-aware, simple base structure) as these tend to reduce visualization artifacts and distortions during deformation. In this context a simple layout can furthermore directly be exploited as convenient domain for purposes of texture and detail mapping [Bom12].

From a technical point of view the process of *quad layouting*, i.e. the partitioning of a surface into conforming quadrilateral patches, involves determining the layout's combinatorial structure as well as its geometric embedding in the surface. The embedding

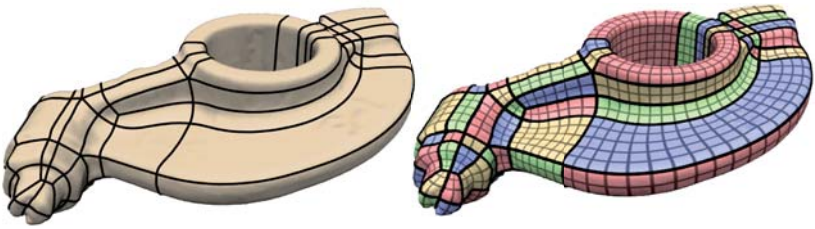


Figure 2.1.: A quad layout on a surface (left) and an iso-parameter line visualization of the parameterizations which embed its patches (right).

describes the locations of the layout patches' corners and borders on the surface as well as parameterizations of the patches. Figure 2.1 shows an example quad layout.

2.1. Foundations

We are going to deal with quad layouts on 2-manifolds, typically surfaces of three-dimensional objects:

Definition 2.1.1. (Surface) A compact orientable two-dimensional manifold $\mathcal{M}$ (with or without boundary $\partial\mathcal{M}$) we will call a *surface*.

For the purpose of discretization we will deal with a triangulation of a surface – to which we refer by $\mathcal{M}$, too, unless the context requires explicit distinction.

Layout Graphs and Embeddings

The notion of a quad layout, a conforming quadrilateral surface partition, is formalized using the following definitions:

Definition 2.1.2. (Layout graph) A multigraph $G = (N, A)$, which may contain multiple arcs between pairs of nodes as well as dangling arcs which are incident to only one node, defines the *graph* of a layout.

Definition 2.1.3. (Graph embedding) Each node h of G is associated with a map $f_h : 0 \to \mathcal{M}$ that assigns this node to a point on the surface. The image of this map we call *embedded node* where distinction from the abstract graph node is necessary – otherwise we simply call it *node*. Furthermore, each arc a of G is associated with a continuous map $f_a : [0, l_a] \to \mathcal{M}$, whose image is called *embedded arc* where distinction is necessary. The maps are such that $f_a(0) = f_h(0)$ and $f_a(l_a) = f_g(0)$ if h and g are the nodes incident to arc a, i.e. the curve formed by the embedded arc on the surface starts and ends at the points onto which the incident nodes are embedded. Furthermore, embedded arcs may only intersect at their endpoints and embedded dangling arcs end on $\partial\mathcal{M}$. The set of maps f then defines a *graph embedding* for G.

Figure 2.2 illustrates this. The embedded arcs partition the surface into regions (called *patches*) bounded by embedded arcs, embedded nodes, and possibly segments of $\partial\mathcal{M}$ – such patches bounded partially by $\partial\mathcal{M}$ are called *trimmed*, all others *full*. If all patches are homeomorphic to discs, the graph embedding formally is a *2-cell embedding (with boundary)*.

Definition 2.1.4. (Valence) The valence $\mathrm{val}(h)$ of a node h is the number of arcs incident to it. Note that one arc can be incident to a node (and contribute to the valence) two times. The valence $\mathrm{val}(p)$ of a patch p is the number of embedded nodes along the patch border. Note that one node can occur multiple times along a patch

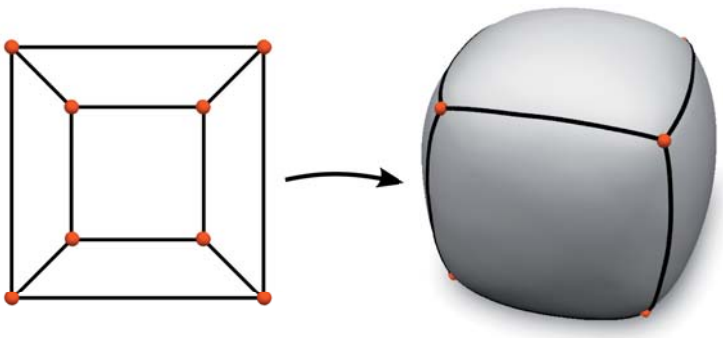


Figure 2.2.: Illustration of a layout graph (red nodes, black arcs) and an embedding of this graph on a surface.

border. Nodes and patches of valence 4 are called *regular*, otherwise *irregular*. Trimmed patches are considered regular if they could be completed to a patch of valence 4.

If all patches are regular, the 2-cell embedding of a layout graph can be extended to a *layout embedding*[1]:

Definition 2.1.5. (Layout embedding) In addition to the node and arc maps, each patch p can be associated with a continuous map $f_p : [0, w_p] \times [0, h_p] \rightarrow \mathcal{M}$ (or a restriction thereof in case of a trimmed patch) such that this map agrees with the maps of the incident arcs, e.g. $f_p(x, 0) = f_a(x)$ or $f_p(x, 0) = f_a(l_a - x)$ (depending on the orientation). The set of maps f then defines a *layout embedding* for G.

Figure 2.3 illustrates the embedding of a patch via its associated map f_p.

Definition 2.1.6. (Quad layout) A layout graph G together with a layout embedding f in surface $\mathcal{M}$ where all patches are regular is called a *quad layout* $\mathcal{L}$.

Figure 2.1 shows an example of such a quad layout.

[1] A layout embedding could also be specified in the presence of irregular patches (using different domains for the maps), but we do not need this more complex general case here.

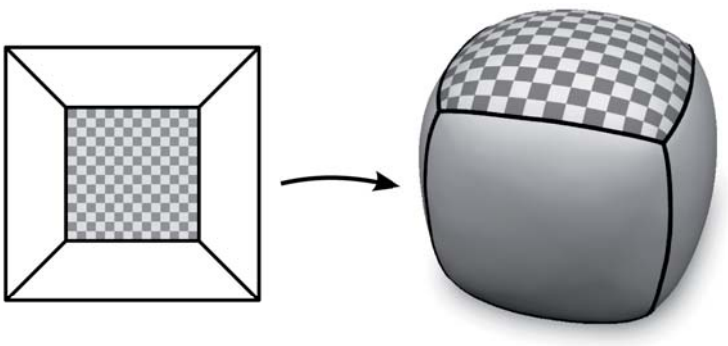


Figure 2.3.: Illustration of an individual patch embedding map.

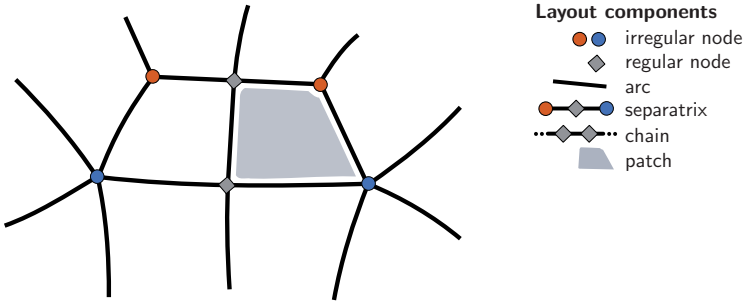
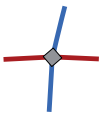


Figure 2.4.: Illustration of a layout's components.

Let us note that an alternative definition would be via cell complexes with quadrilateral 2-cells [Hat02].

Structural Properties



At regular nodes the four incident arcs can be partitioned into two pairs (depicted in red and blue on the right) of *opposite* arcs in the intuitive way. Based on this, separatrices can be defined:

Definition 2.1.7. (Separatrix) A chain of successively opposite arcs which starts at an irregular node and ends at a (not necessarily distinct) irregular node or $\partial\mathcal{M}$ is called (discrete) *separatrix*.

Note that we will later also deal with separatrices of cross fields and separatrices of parameterizations – concepts which are closely related but formally different.

Figure 2.4 illustrates the components of a quad layout.

Definition 2.1.8. (Layout minimality) If each arc of a quad layout is part of a separatrix, the layout is called *minimal*.

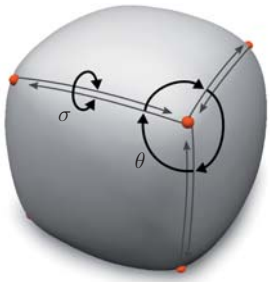
In a minimal layout, there are no chains of arcs which are cyclic or start and end at $\partial\mathcal{M}$.

Definition 2.1.9. (Base complex) The largest minimal sub-layout of a layout we call its *base complex*.

The base complex of a quad layout (or quad mesh) can be obtained by tracing all separatrices and then removing all untraced arcs.

Finally, besides the geometric representation of the layout via its embedding, a representation of just its topology is of use:

Definition 2.1.10. (Rotation system) A compact representation of the topology of a quad layout $\mathcal{L}$ is a pair (σ, θ) of permutations acting on the set of half-arcs (for each arc there exist two directed half-arcs, one attached to each incident node). σ is an involution that maps one half-arc to the other one of the same arc, and θ maps a half-arc to the next one in the clockwise (with respect to the orientation of $\mathcal{M}$) cyclic order of half-arcs incident to the same node. The pair (σ, θ) is called the layout's *rotation system*.



Dual Layouts

For the description and modification of the layout structure it can be of benefit to consider the dual layout:

Definition 2.1.11. (Dual) The combinatorial dual $\mathcal{D}$ of the cell complex specified by the quad layout $\mathcal{L}$ is called the *dual layout*.

$\mathcal{D}$ contains a *vertex* for each patch of $\mathcal{L}$, an *edge* for each arc of $\mathcal{L}$, and a *region* for each node of $\mathcal{L}$.[2] Except for boundary cases, $\mathcal{D}$ is a 4-regular cell complex, i.e. every non-boundary dual vertex has four incident dual edges. Hence, at every non-boundary vertex there are two pairs of *opposite* edges. The set of all edges uniquely decomposes into a disjoint collection of cycles (and, when boundaries exist, boundary-to-boundary chains) of successively opposite edges. Note that these dual edge cycles (chains) correspond to –

[2] We use the terms vertex, edge, and region for dual layouts in order to distinguish from nodes, arcs, and patches of primal layouts.

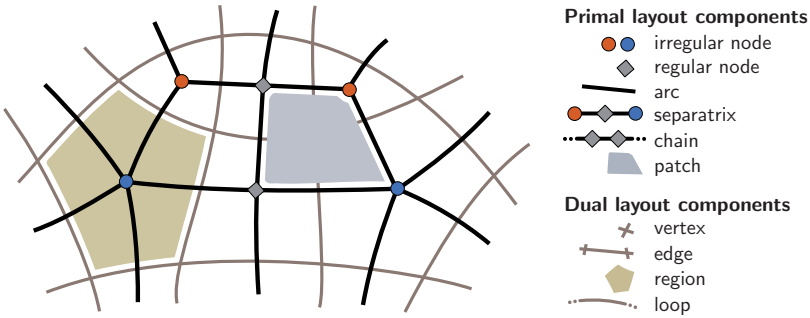


Figure 2.5.: Illustration of a layout's primal and dual components.

possibly non-simple, i.e. self-crossing – cyclic (or boundary-to-boundary) quad strips in $\mathcal{L}$. Geometrically, these dual edge cycles (chains) correspond to an arrangement of loops (or boundary-to-boundary curves) on the surface, cf. Figure 2.5; the loop intersections define the vertices and the loop segments between any two intersections define the edges of $\mathcal{D}$. The induced partition of $\mathcal{M}$ consequently defines the regions of $\mathcal{D}$ (cf. Figure 2.5). Note that a specific embedding for $\mathcal{D}$ is not inherent to the topological concept of duality.

2.2. Quality Criteria

As already mentioned, the *quality* of a quad layout is a measure that, at least to some extent, depends on the intended application. However, we can identify some generic aspects which are common to most application scenarios:

- **Geometric fidelity:** patches should map to planar rectangles with low parametric distortion
- **Structural simplicity:** the number of patches should be small

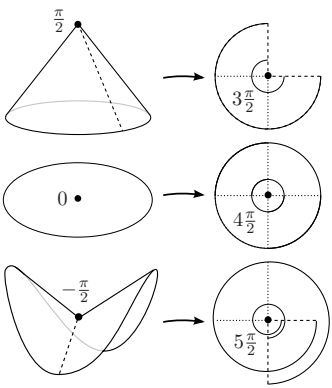
The geometric fidelity generally promotes mapping or representation quality, because the parametric distortion is a key factor in many applications, while the structural

simplicity gives preference to simpler surface representations, simpler mapping domains, or more flexibility for hierarchical structures (e.g. in the context of multi-block grids).

Unfortunately, both aspects tend to be opposing objectives. For instance, a genus 1 surface can always be covered with just a single quad patch, but then the parametric distortion can be arbitrarily high, depending on the geometric shape of the surface. On the other hand, low distortion can be achieved at the expense of a large number of small patches, as in the case of quad meshing [BZK09].

The general quality of the layout hence is a function of complexity and distortion, and an appropriate trade-off needs to be made. How this relation looks like in detail is application-dependent. Our automatic approach (cf. Part I) yields results where this trade-off generally seems reasonable or natural – and often even optimal on models where this can be judged objectively. Our interactive approach (cf. Part II) gives full flexibility to the user to take application-dependent aspects, even those that are subjective or hard to formalize, into account and to adjust this trade-off accordingly.

The two layout quality criteria *geometric fidelity* and *structural simplicity* imply local criteria for individual nodes, arcs, and patches. For instance, a patch corner ideally has an angle of $\frac{\pi}{2}$. Thus, a node of valence k ideally lies at a point where the surface has Gaussian curvature $2\pi - k\frac{\pi}{2}$ because around such a point the (cut) surface unfolds to a sector with inner angle $k\frac{\pi}{2}$ [PS98] as depicted here. Furthermore, an arc is ideally embedded as a geodesic curve, and arcs are short and few in number. It is obvious that these ideal states typically cannot be achieved, especially not simultaneously, due to interdependencies between the nodes, arcs, and patches and the surface geometry. These local criteria can, however, be helpful guides in the design of optimization strategies which then take the global interdependencies into account.



One further aspect is quite commonly of relevance for the layout quality:

- **Principal direction alignment:** the arcs and also the iso-parametric curves of the embedded patches should follow directions of principal curvature on the surface with low deviation.

The importance of this aspect for prominent use cases of quad layouts is well known [LRL06, ACSD*03, CSAD04]. Depending on the application, it serves maximizing surface approximation quality [D'A00], minimizing normal noise and aliasing artifacts [BK01], optimizing element planarity [LXW*11], or achieving smooth curvature distribution (due to their tensor-product nature common spline surface representations are prone to ripples (curvature oscillations) if aligned badly). Besides, principal direction alignment can also be of interest for aesthetic reasons. Due to their specific symmetries quad layouts and quad meshes have the natural ability to align to the orthogonal principal directions. In fact, this is one of the main reasons for preferring them over simplicial layouts [BLP*13].

The alignment of layout arcs or iso-parametric curves of the embedding maps to feature curves on the surface may be seen as a special case of this principal direction alignment, but it can be advantageous to consider this in a dedicated, stricter manner and to take alignment to surface boundaries into account as well.

2.3. Taxonomy

Besides quad layouts there exist several other concepts – quad meshes, subdivision base meshes, T-meshes – which are closely related but differ in terms of structure or quality objectives. To clarify our objectives, we here elucidate the distinction between these.

2.3.1. Quad Meshes

Structurally a quad mesh does not differ from a quad layout: it consists of conforming quadrilateral patches. The fundamental difference is the simplicity objective. When generating a quad layout the patch dimensions should be chosen automatically such that a simple layout is achieved. This can lead to patches of arbitrarily varying sizes and aspect ratios, depending on the geometry of the surface. Methods for generating quad meshes,

however, typically rely on a *a priori* specified target quad size and quad anisotropy (possibly varying over the surface) [RLL*06, DBG*06, KNP07, HZM*08, BZK09, ZHLB10, KMZ11, BCE*13, PPTSH14]. This is motivated by the fact that the nodes of a quad mesh are expected to already sufficiently encode the geometric information of the surface via their 3D positions because arcs and patches are typically just (bi)linear interpolations of this information. A quad mesh (e.g. for simulation or animation applications) thus typically needs to be much finer than a quad layout who's main purpose is to provide a partition of the surface but not to immediately represent it geometrically.

Due to these differences, quad mesh generation methods are typically not suited for the generation of quad layouts (cf. Section 2.4). Vice versa, however, quad meshes (in particular: highly demanded semi-regular quad meshes) can easily be generated from quad layouts generated by our algorithms (cf. Section 6.7).

2.3.2. Subdivision Base Meshes

Just like in the case of quad meshes, the complexity of base or control meshes for subdivision surfaces is mainly mandated by the fact that the nodes are expected to carry the full geometric information in their positions. However, the patch geometry is defined via more sophisticated means than simple (bi)linear interpolation, e.g. via bi-quadratic or bi-cubic splines in the case of the famous Doo-Sabin [DS78] and Catmull-Clark [CC78] subdivision schemes. Hence, in practice subdivision base meshes are typically coarser than quad meshes, but this of course depends on the concrete scenario and the underlying shape.

Furthermore, some subdivision schemes can deal with base meshes that contain irregular non-quad patches and in this sense allow more flexibility in the construction. These irregularities, however, lead to singularities with lower smoothness in the subdivision limit surface [Rei95].

Note that, just like in the above case of quad meshes, a quad layout can easily be refined (by splitting patches into regular grids of smaller patches) in order to achieve the patch resolution suitable for a subdivision base mesh application. The opposite, i.e. given a fine quad mesh, turn it into a coarser quad layout, by contrast, is highly challenging and requires sophisticated algorithms [DSC09, PPT*11, BLK11, TPP*11], which do not always manage to preserve a good patch quality and alignment.

2.3.3. T-Meshes

Dropping the requirement of conformity, i.e. that patch corners coincide only with other patch corners, we arrive at the class of so-called *T-meshes*. The name stems from the fact that a patch corner lying on the interior of an arc forms a T-junction. While such partitions are not suitable for, e.g., B-Spline surface representations due to continuity issues, generalized T-splines and T-NURCCs [SZBN03] build on exactly such meshes. The option to have T-junctions in the mesh, where chains of arcs just end, allows to more flexibly adjust the local resolution compared to a quad layout, where chains of arcs always are either cyclic or fully extend to irregular nodes or boundaries. The increased sparsity and compactness is an advantage for modeling and editing purposes [SCF*04, BVK08].

It is, however, important to note that this increased flexibility does not simplify the construction of a T-mesh compared to a quad layout – at least not in the context of T-spline related techniques. For this use case a T-mesh needs to be globally parameterizable, i.e. the assignment of non-zero knot intervals to edges of the mesh must be possible in a globally consistent manner [SZBN03]. This implies certain constraints regarding the constellation of T-junctions. These can only be fulfilled if the T-mesh can be obtained from a conforming partition (a quad mesh or a quad layout) by omitting a number of arcs (and nodes). While we are not aware of an explicit mentioning of this in the pertinent literature, existing approaches to automatic T-mesh construction respect this implicitly: they obtain a T-mesh as a subset of a quad mesh [MPKZ10, LRL06, SCF*04], as a subset of the iso-parametric curves of a surface parameterization [HWW*06], or by partial regular refinement of a quad mesh or layout [WZXH12, SZBN03].

Hence it is not unnecessarily expensive to use quad layout construction methods like ours to generate T-meshes, too, by (possibly after suitable refinement) removing nodes and arcs from the result where they are not necessary to achieve the desired quality.

2.4. State of the Art

In industrial workflows quad layouting is often performed manually by skilled, well-trained professionals in the animation and engineering sectors through the construction

of nets of surface curves. More recently, approaches that tackle (variants of) this problem in a (semi-)automatic manner have been proposed. We discuss them in the following.

An inherent issue is that a good quad layout generally is a compromise balancing layout simplicity, patch rectangularity, feature and principal direction alignment, and possibly further objectives, as described in Section 2.2. Existing automatic methods do not consider all of these aspects adequately, or appear to be strongly biased towards one particular objective.

2.4.1. Quad Meshing

For the generation of quadrilateral surface meshes a number of methods have been proposed. They are based on various principles, e.g., periodic parameterizations [RLL*06], Morse-Smale theory [DBG*06, HZM*08], or mixed-integer optimization [KNP07, BZK09, BCE*13]. These methods are mainly targeted at creating meshes with uniformly sized elements. While extensions for the incorporation of sizing, skewness, and aspect ratio fields have been presented [ZHLB10, KMZ11, PPTSH14], allowing to adjust the element properties to local properties of the input geometry, these fields have to be specified a priori. A recent survey [BLP*13] gives an overview over all of these techniques.

The base complex of meshes generated with these techniques could potentially be used as a quad layout. Unfortunately, most of these techniques do not provide means to control the base complex structure and its simplicity (cf. Figure 2.6); meshes with a two-level hierarchy, and thus a coarser base complex, are obtained by those mehods based on spectral surface analysis [DBG*06], but the coarser level is highly uniform by design, which of course restricts its simplicity. With our recent method [BCE*13] we increased the robustness of the quad remeshing process so far that it allows to generate relatively coarse quad meshes which could effectively be "misused" as quad layouts. When prescribing a very large target edge length, the correctness constraints implicitly (anisotropically) reduce the size of the individual elements such that a valid mesh or layout is obtained. The enforced small patches, however, conflict with the geometric objective and it can be observed that for large prescribed target edge lengths, the optimization is dominated by the aim of fulfilling the involved constraints and geometric quality occasionally suffers, resulting, e.g., in badly shaped patches and badly aligned arcs.

Figure 2.6.: Quad meshes created by a field-guided parameterization approach [BZK09] and their base complexes. On the left the arcs of the base complex are highlighted. In the middle the base complex patches are visualized by individual colors – on the right a very similar quad mesh of this object is depicted (same resolution, same number and type of irregular vertices) refined from a quad layout created by the techniques presented herein; it has a much simpler base complex.

2.4.2. Quad Layouting

Eck and Hoppe [EH96] proposed an early quad layout construction method for the purpose of spline patch fitting. Essentially, it is based on pairing the faces of a triangular layout. A similar approach is chosen by Boier-Martin et al. [BMRJ04]. Control over the geometric fidelity and alignment is quite limited due to the nature of these approaches. Daniels et al. [DSC09] propose to convert a triangle mesh to a quad mesh by one step of Catmull-Clark subdivision and then simplify the result using quad mesh decimation techniques. Driving this decimation in a way that the resulting coarse mesh is geometrically faithful and irregular vertices end up in plausible locations similarly remains a problem. In this context Panozzo et al. [PPT*11] use special kinds of pre-computed sizing fields (called *fitmaps*) to influence a decimation process.

Due to the complexity of the problem, several researchers proposed or assumed a manual [BVK08, KL96, MBVW95, AAB*88] or semi-automatic, assisted [TACSD06, TDN*12, JLW10] creation of quad layouts. This means the user (directly or indirectly) places nodes onto the surface and delineates the patch borders inbetween. This approach to the problem can be advantageous because layout design decisions might depend on the intended use of the layout and cannot always be derived from geometry alone.

However, manually designing a good all-quadrilateral layout turns out to be an intricate and cumbersome task even for experienced users due to the involved complex global interdependencies and structural constraints that are to be considered. To cite Takayama et al. [TPSHSH13] in this regard: "it is often quite challenging even for professional artists to manually design a perfect quad mesh on the first try. Since the quality [...] is a global property, the correction of a single mistake might require regeneration of the entire mesh.". They hence proposed a system which provides various helpful guides and automatisms to reduce the user's workload – for the case of subdivision base mesh design.

Automatic methods have furthermore been presented for related, simpler variants of the problem, like quad-dominant layouts [MK04, LKH08], polygonal layouts [CSAD04, MK05], feature graph layouts [NSP10], or non-conforming partitions [EGKT08].

2.4.3. Base Complex Simplification

The problem of obtaining a simple layout has also been approached "reversely": instead of creating a layout from scratch, one starts from a fine quad mesh with a possibly very complex base complex and attempts to successively simplify it by adjusting its structure. The base complex could then be used as quad layout for various purposes. Bommes et al. [BLK11] and Tarini et al. [TPP*11] followed this path. The underlying observation is that the separatrices (which form the base complex, cf. Section 2.1) of quad meshes generated by the methods discussed in Section 2.4.1 often wind around the underlying object many times, forming a rather fine base complex with many small patches as shown in Figure 2.6. Using sets of operators that modify the mesh connectivity while preserving the quad structure one can attempt to reduce the total length of these separatrices, thus the complexity of the base complex. Tarini et al. [TPP*11] build such operators from separatrix removal and re-insertion steps, Bommes et al. [BLK11] focus on eliminating certain helical configurations, which they identified as most relevant for base complex simplicity. Using greedy strategies the underlying huge search spaces are then explored in the search of modified meshes with a simpler base complex. In Section 5.4 we discuss the respective advantages and disadvantages in comparison to our approach.

General decimation and editing methods for quad meshes, e.g. [DSSC08, PZKW11], can also be seen as related, but they do not specifically target a coarse base complex.

3. Strategy

The problem of high-quality quad layout generation is hard to tackle using established standard optimization techniques ("black box solvers") due to its complicated nature: there are discrete, continuous, and combinatorial or topological degrees of freedom (DoFs):

- **discrete:** number and valence of nodes
- **continuous:** embedding of nodes, arcs, and patches
- **combinatorial/topological:** layout graph connectivity

This complexity requires us to design specialized, geometrically motivated strategies to enable the efficient computation of good solutions.

Our underlying strategy is to split the problem into three sub-problems, each of which has a simpler, more uniform (in terms of DoFs) structure. These sub-problems are then solved in three sequential algorithmic stages. The division into sub-problems allows us to rely on different optimization techniques (e.g. based on a linear equation system, non-linear gradient descent, combinatorial optimization, or an eigenvalue problem) for the different aspects of the problem.

If the DoFs were independent in between the sub-problems, the division would be uncritical – in the sense that sequential optimal solution of the sub-problems would yield the optimal solution of the entire problem. In the quad layout problem, however, the DoF dependency graph can typically be assumed to be very dense or even complete (unless the surface has multiple components). Therefore we have to take care in the design of the individual optimization stages that the sub-solutions produced by the early stages can still be completed to a full solution of high quality by the later stages. To this end we follow two paradigms, a forward and a backward paradigm:

- **forward anticipation:** anticipate how well later stages will be able to deal with the sub-solution which is being produced.
- **backward modification:** modify a sub-solution of an earlier stage if it cannot be dealt with well.

Obviously, this anticipation and modification can only be performed to a certain extent for the sake of efficiency – full anticipation would correspond to solving the entire problem already in the first stage, complete modification freedom would correspond to solving the entire problem in the last stage (independent of what the previous stages delivered).

3.1. Automatic Approach

The design of the individual stages of our automatic quad layout generation and optimization pipeline is outlined in the following.

3.1.1. Stage 1: Nodes

In the first stage (Chapter 4) the discrete degrees of freedom, i.e. an appropriate number of (irregular) nodes and their valences, are determined. It is based on constructing a non-standard connection [CDS10] from a smooth, principal curvature guided direction field on the surface. The irregular nodes and their valences are then derived from this connection's curvature distribution (or equivalently: the field's singularities). Due to the nature of such fields we achieve node configurations suitable for quad layouts and due to the nature of the field smoothness functional we achieve node valences that are geometrically appropriate. These properties have already been exploited in the related field of quad meshing [KNP07, BZK09].

3.1.2. Stage 2: Connectivity

In essence, our stage 2 (Chapter 5), which determines the combinatorial or topological degrees of freedom, is based on the careful construction of the combinatorial dual of

the quad layout from geodesic loops in a way that is sensitive to the input's geometry and ensures structural consistency. The process is driven by the objective of separating the nodes determined in stage 1 by dual loops, such that they eventually lie in regions with the desired valence. Dualization of the arrangement of dual loops then provides a connectivity for the nodes which is guaranteed to form only quad patches.

Our key contributions are:

- A novel framework to describe, identify, and manipulate structurally valid and geometrically faithful duals of quad layouts.
- Efficient methods for the geometry-aware construction of embedded loops that are the key components of the dual layouts.
- A practical greedy algorithm based on the framework to automatically and efficiently perform the layout construction.

3.1.3. Stage 3: Embedding

In stage 3 (Chapter 6) the continuous optimization is performed, i.e. the embedding of nodes, arcs, and patches is optimized. As all these individual embeddings interdepend (cf. Definition 2.1.5), we formulate an optimization problem based on a global, principal direction aligned parameterization which effectively combines all these embeddings. This problem is solved using an efficient custom-tailored optimization strategy based on alternating between global linear optimization and local non-linear optimization. The individual embeddings can then be extracted from the parameterization.

Our key contributions are:

- a description of how to add structural constraints to established optimization systems for principal direction fields and parameterizations, such that they can be used to optimize alignment-aware layout embeddings.
- a novel concept to continuously optimize layout node positions, directly driven by embedding quality, based on an efficient meta-optimization strategy.

3.1.4. Forward Anticipation

The paradigm of forward anticipation is implemented in the first two stages as follows:

Stage 1 → Stage 3 In stage 3 the embedding is optimized via a global parameterization which is guided by a principal direction field. In stage 1 the nodes are determined via exactly such a field, promoting suitability for the parameterization that is performed in stage 3.

Stage 2 → Stage 3 In stage 2 dual loops are constructed based on objectives that mimick those of the parameterization in step 3. For instance, principal direction alignment is taken into account and orthogonal loop crossings are favored.

3.1.5. Backward Modification

The paradigm of backward modification is taken into account as follows:

Stage 1 ← Stage 2 In stage 2 the dual loop construction algorithm has the possibility to merge nodes determined in stage 1 if they turn out to be inseparable, or separable only in geometrically inappropriate ways.

Stage 1 + 2 ← Stage 3 In stage 3 the algorithm has the possibility to perform so-called poly-chord collapses, removing arcs and nodes created in the previous stages in a structurally consistent manner, when they turn out to be superfluous.

3.2. Interactive Approach

Besides the automatic approach, we propose an interactive, computer-aided quad layout design system (Chapter 7). It grants the user the flexibility to incorporate their application-specific design intents but automatically takes care of consistency aspects and of assessing the geometric layout quality. Our system features novel interaction tools and provides guides to effectively support the design process. Just like the automatic approach, it is fundamentally based on a dual perspective with its advantageous consistency-ensuring aspects.

The three stage division, however, turns out to be unsuitable for the interactive scenario. Keeping stage 1 automatic and making stage 2 interactive is not an option, because the topology of the cross field from stage 1 already significantly restricts the design freedom regarding the dual loops. Making stage 1 interactive, too, i.e. having the user first create the field manually, would yield a very complicated, impractical workflow.

We thus combine stages 1 and 2, i.e. we derive the nodes as well as their connectivity from the dual loops constructed by the user. In order to enable this modification, we model the dual loops based on bending energy-minimizing *elastica* [BNR01] instead of the simpler field guided geodesics used in stage 2 of the automatic approach. This allows us to automatically propose geometrically and topologically suitable operations to the user, enabling an efficient workflow. The automatic stage 3 can then be used to determine an optimal embedding for the designed layout – but we also discuss an interactive alternative that grants control in this stage, too.

The key contributions can be summarized as follows:

- A fast method for the construction of *constrained elastica* on surfaces, without any restrictions on topology or homotopy.
- Support for symmetries in the dual loops/strips framework.
- Interaction tools and metaphors for incremental layout design, supported by automatic background computations.
- Evaluation of intermediate design states and visualization of the assessment to guide the user.

Figure 3.1 illustrates the automatic and interactive processing pipelines schematically.

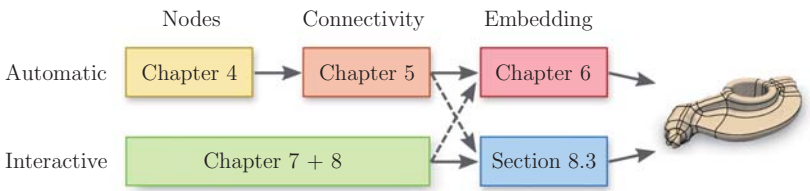


Figure 3.1.: Illustration of the automatic and interactive processing pipelines and their stages. Note that automatic and interactive stages can be combined (dashed arrows), i.e. the embedding of an interactively designed layout structure can be optimized automatically and vice versa.

Part I.

Automatic Generation

4. Nodes

In this first stage of the automatic pipeline a suitable number of layout nodes and their respective valences are to be determined. As discussed in Section 2.2 the ideal situation would be that the surface has isolated points s_i with a Gaussian curvature that is an integer multiple of $\frac{\pi}{2}$ while all other points of the surface have vanishing Gaussian curvature. Then one could place an irregular node n_i of valence $\text{val}(n_i) = k_i$ onto each point s_i with Gaussian curvature $2\pi - k_i\frac{\pi}{2}$ (regular nodes could be placed anywhere else as needed) and obtain developable (actually planar) patches with right angles. Obviously this is only possible on an extremely restricted class of surfaces, namely those of unions of boxes (in particular polycubes)[1]. On general surfaces this point-wise view of curvature is not very helpful. For instance, on smooth surfaces it is neither scale-invariant nor, in the discrete case, mesh resolution-invariant. A naïve approach in the style of "place an irregular node of valence k where the Gaussian curvature is in some sense close to $2\pi - k\frac{\pi}{2}$" is thus bound to fail. This holds even more due to the following important observation: we are not entirely free in the choice of nodes and their valences (at least for surfaces without boundary) – the topology of the surface implies a constraint via Euler's well known polyhedron formula, which likewise holds for the special case of quad layouts (with nodes N, arcs A, and patches P):

Theorem 4.0.1. (Euler's polyhedron formula)

$$|N| - |A| + |P| = \chi(\mathcal{M}) \tag{4.1}$$

[1] In fact, one can try to find a union of boxes which approximates the given surface (together with a continuous bijection), construct a quad layout on it, and map it onto the surface [HJS*14]. However, only a very restricted class of quad layouts can be obtained in this way: those whose set of dual loops can be partitioned into three subsets such that no two loops from the same subset intersect. Therefore the possibilities of achieving high patch quality and good alignment are limited.

where $\chi(\mathcal{M})$ is the surface's *Euler characteristic*, a topological invariant related to its genus. Considering each arc as two directed half-arcs (their number denoted by $|H|$) and exploiting the fact that in a quad layout there are exactly four times as many half-arcs as patches, we arrive at

$$|N| - |H|/4 = \chi(\mathcal{M}). \tag{4.2}$$

This can be restated in form of a sum over all vertices and their outgoing halfedges:

$$\sum_{n_i \in N} 1 - \mathrm{val}(n_i)/4 = \chi(\mathcal{M}). \tag{4.3}$$

We note that summands corresponding to regular nodes of valence 4 vanish. The sum over the irregular nodes' terms, however, needs to match the surface's Euler characteristic. This constraint needs to be considered in the optimization because otherwise the set of nodes does not admit a quad layout.

4.1. Connections

For these reasons a more global, integral view of curvature is necessary. This view is provided by *connections*[2] [CDS10]. A connection defines a notion of parallel transport on a surface and thus implies a notion of curvature. In particular, the *Levi-Civita* connection ∇_{LC} – the unique metric connection that transports tangent vectors without torsion – implies the usual Gaussian curvature. Intuitively, an arbitrary tangent vector at a point p on the surface, parallel transported along any simple closed curve on the surface using a connection, leads to another tangent vector at p. The accumulated *angle defect* between these two vectors (*holonomy*) defines the total curvature of the region enclosed by the curve. In this way the total curvature of any region is encoded in a connection.

The idea now is to construct a connection $\nabla_{\frac{\pi}{2}}$ which A) is (in the least-squares sense) as close as possible to Levi-Civita while B) implying a curvature which is an integer

[2]Formally, we are referring to *metric connections*, which preserve the surface's metric under parallel transport, i.e. an orthonormal frame is transported to an orthonormal frame.

multiple of $\frac{\pi}{2}$ everywhere (while it is 0 for all but a finite number of points s_i). Note that in the case of a union of boxes we will obtain $\nabla_{\frac{\pi}{2}} = \nabla_{LC}$. In any case, due to condition (B), we can easily derive a set of irregular nodes and their valences from this connection. Due to condition (A) we can expect that this irregular node configuration, while not ideal, is in some sense as good as possible for the given surface geometry. Most importantly, without need for any further measures, the set of irregular nodes derived from such connections is guaranteed to fulfill (4.3). This is a direct consequence of the Gauss-Bonnet theorem that states that the total curvature $\iint K$ of a surface without boundary (implied by any connection) is related to its Euler characteristic via

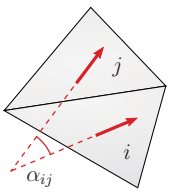
Theorem 4.1.1. (Gauss-Bonnet theorem for closed surfaces)

$$\iint K = 2\pi\chi(\mathcal{M}). \tag{4.4}$$

In the case of $\nabla_{\frac{\pi}{2}}$ we thus have $\chi(\mathcal{M}) = \frac{1}{2\pi}\sum_i K(s_i)$. This turns into (4.3) if $\frac{1}{2\pi}K(s_i) = 1 - \mathrm{val}(n_i)/4$ for each point s_i and its corresponding node n_i, and indeed this is the case for our desired choice of $\mathrm{val}(n_i) = k_i$ for $K(s_i) = 2\pi - k_i\frac{\pi}{2}$.

4.1.1. Optimization

In the discrete setting, i.e. on a triangle mesh, a connection can be expressed via *adjustment angles* α across the edges [CDS10], which express the tangential rotation a tangent vector undergoes as it is transported from one face to a neighboring face across an edge. The discrete Levi-Civita connection is characterized by all these angles being zero. As we are interested in a connection as close as possible to Levi-Civita, our objective is $\|\alpha\|_2 \to \min$, subject to condition (B).



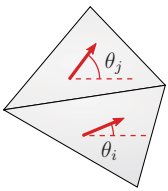
As detailed in [CDS10] condition (B) can be expressed via constraints for a set of basis face cycles which enforce that the adjustment angles exactly cancel the curvature of the Levi-Civita connection $\pm m_i\frac{\pi}{2}$. In that work the integers m_i are prescribed, allowing this optimization to be performed with a simple linear system solve. Here, we end up with a mixed integer problem because we only have the constraint that the curvature is *some* integer multiple of $\frac{\pi}{2}$, i.e. the m_i are free integer variables. Hence, a constrained mixed

integer solver is necessary for this optimization. Bommes et al. [BZK12] presented an efficient greedy solver strategy that could be applied here.

However, we are going to use an alternative formulation for the optimization, based on tangent direction fields, which is essentially equivalent but provides additional interesting insights and possibilities (in particular: consideration of principal direction alignment), as detailed in the next section.

4.2. Cross Fields

Instead of representing the per-edge adjustment angles as explicit variables, we can represent them implicitly as the angular differences between variable tangent directions in the incident faces, i.e. $\alpha_{ij} = \theta_i - \theta_j + \kappa_{ij}$, where θ are the angles of the tangent directions with respect to a local per-face reference system and the constant κ_{ij} aligns the reference systems.



We could then perform the following optimization:

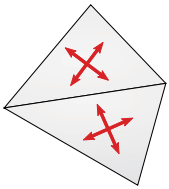
$$\sum_{e_{ij} \in E} (\theta_i - \theta_j + \kappa_{ij})^2 \rightarrow \min \tag{4.5}$$

Due to the relation between α and θ this is essentially equivalent to the connection based optimization problem – with one significant difference: constraints for a condition akin to (B) are inherent to this formulation. The resulting connection obviously transports vectors to themselves along any curve (just notice that by construction the field θ – and any constant rotation thereof – is parallel w.r.t. the connection given by α). Hence the implied curvature must be an integer multiple of 2π everywhere. The multiplicity, however, is fixed arbitrarily (formally and more precisely, it is implied by the indices of the field of chosen local reference systems) and we are actually interested in multiples of $\frac{\pi}{2}$, not 2π. Both problems can be solved by adding integer variables m to the system which allow to add any multiple of $\frac{\pi}{2}$ to the terms in (4.5):

$$\sum_{e_{ij}\in E}\left(\theta_i-\theta_j+\kappa_{ij}+m_{ij}\frac{\pi}{2}\right)^2\rightarrow\min \tag{4.6}$$

Note that adding such a variable for *each* edge is redundant – just as in Section 4.1.1 it is sufficient to add one variable per cycle of a face cycle basis. This is the case if we add a variable for each edge of a spanning tree of the edge graph, as also mentioned in [BZK09].

The connection $\nabla_{\frac{\pi}{2}}$ resulting from this optimization then fulfills (A) and (B). In particular, a vector is not necessarily transported to itself along any curve by this connection but may be off by a multiple of $\frac{\pi}{2}$. A *cross* (a set of four directions invariant to rotations by $\frac{\pi}{2}$), however, is always transported to itself, so we are in fact optimizing a cross field $\mathcal{C}$ for smoothness – a cross field which by example represents $\nabla_{\frac{\pi}{2}}$. Therefore it comes as no surprise that exactly the same optimization problem is used for the construction of smooth cross fields [HZ00, LVRL06, PZ07, RVLL08, RVAL09, LJX*10] for quad meshing purposes, e.g. in [BZK09, MPKZ10, LLZ*11, PTSZ11, ECBK14].



It is worth noting that from this perspective the inherent fulfillment of (4.3) is a consequence of the Poincaré-Hopf theorem, relating a direction field's singularities to the surface's Euler characteristic:

Theorem 4.2.1. (Poincaré-Hopf theorem) Let $\mathcal{C}$ be a direction field on $\mathcal{M}$ and s_i its isolated singularities. Then

$$\sum_i \text{index}_{\mathcal{C}}(s_i)=\chi(\mathcal{M}). \tag{4.7}$$

The index of $\mathcal{C}$ at a point s_i is given by $\text{index}_{\mathcal{C}}(s_i)=\frac{1}{2\pi}K(s_i)$, where K is the curvature implied by $\nabla_{\frac{\pi}{2}}$. In the case of a cross field it thus is a multiple of $\frac{1}{4}$. For details on tangent direction fields, singularities and their indices we refer to [RVLL08, PZ07]. Relating (4.3) to (4.7) reveals a close connection which implies that the choice of a set of irregular nodes n_i which correspond to the singularities s_i of a direction field, with valences chosen as

$$1-\text{val}(n_i)/4=\text{index}_{\mathcal{C}}(s_i), \tag{4.8}$$

(which is equivalent to the choice discussed in Section 4.1) will always lead to a configuration fulfilling Euler's polyhedron formula.

These beneficial properties of cross fields have already successfully been eploited by several methods in the related field of quad meshing, e.g. in [KNP07, BZK09, ECBK14], which similarly derive irregular mesh vertices from cross field singularities – even though the Euler characteristic relation and its importance have not explicitly been mentioned in these works.

Note that the particular choice of relation between node valence and singularity index, which we previously justified by patch corner angle considerations, can also be seen to be natural in another way: the number $\text{sep}_{\mathcal{C}}(s_i)$ of *cross field separatrices* [PZ07] emanating from singularity s_i is related to the index in the same way, $1 - \text{sep}_{\mathcal{C}}(s_i)/4 = \text{index}_{\mathcal{C}}(s_i)$, as illustrated in Figure 4.1.

Figure 4.1.: Streamline visualization of a cross field in the neighborhoods of points with index $\frac{1}{2}$, $\frac{1}{4}$, 0, $-\frac{1}{4}$, $-\frac{1}{2}$. Streamlines emanating from a singularity are called *separatrices* and their number is related to the singularity's index via $1 - \text{sep}_{\mathcal{C}}(s_i)/4 = \text{index}_{\mathcal{C}}(s_i)$. In regular regions (middle) streamlines cross orthogonally.

4.2.1. Optimization

The optimization problem (4.6) can be solved using an efficient greedy mixed integer solver [BZK12] as described in [BZK09]. Recently, different formulations with alternative optimization strategies that do not require a mixed integer solver have been proposed [KCPS13, DVPSH14]. In these methods the integer degrees of freedom are essentially expressed implicitly [PZ07] rather than explicitly [LVRL06]. This, however, limits the types of singularities and thus node valences – in the case of a vertex-based formulation as in [KCPS13] to valence 3 and 5 – while the explicit representation allows for arbitrary valences. Furthermore, vector based interpolation [DVPSH14] can lead to degeneracies (zero vectors which cannot be back-transformed to crosses), while the angle based formulation always yields valid results. Hence we use the mixed integer formulation in our method.

Note, however, that indices should always be lower than 1 because otherwise the corresponding (necessarily positive) valence cannot be chosen according to (4.8) in the desired way. If singularities with index ≥ 1 occur (in pathological situations with Gaussian curvature of nearly 2π concentrated on one vertex) they can easily be split into lower index singularities [BZK09, PZKW11].

4.2.2. Principal Direction Alignment

The cross field perspective of the node optimization problem allows us to respect the forward paradigm (cf. Section 3.1.4) in this stage. The cross field can be seen as a kind of proxy for the parameterization (more precisely: its gradient field) that is going to be computed in the third stage. As we are going to construct a *principal direction aligned* parameterization, it is of benefit to likewise take an additional alignment objective into account already in the cross field construction. This influences the resulting singularity structure. In this way the nodes' suitability for the third stage is already considered here in the first stage. Specifically, we use the technique described in [BZK09] to select principal direction constraints based on the local surface anisotropy, and fix the crosses in the respective faces accordingly.

Note that one could go one step further and optimize a principal direction aligned parameterization (including free singularities) instead of "just" a principal direction aligned cross field (for instance in the spirit of [MZ13]). However, this would be more expensive: a parameterization corresponds to an *integrable* 4-symmetric vector field, and the integrability condition leads to a significantly harder problem. But even more important, the benefit is questionable because a parameterization computed here would still not match the one computed in stage 3: the connectivity information from stage 2, which is not yet predictable here, will be taken into account in stage 3, and it often has a larger effect than the integrability condition.

4.3. Results

Figure 4.2 shows the irregular nodes obtained for various models using the described method with principal direction alignment constraints.

4.4. Discussion

Regular Nodes Note that we, so far, only considered the irregular nodes – which play by far the most important role. In fact, here we can actually neglect regular nodes, which do not have any effect in (4.3) or on the geometric quality – they can emerge freely in later steps (basically implied by crossing separatrices, cf. Section 5.2.3).

Node Positions The purpose of stage 1 is to fix the discrete degrees of freedom – irregular node count and valences. As a by-product we get positions for these nodes. We can conveniently exploit this information in the next stages, but it is considered preliminary and we only use it as initialization – actual optimized positions are finally obtained in stage 3.

Guarantees (4.3) implies a necessary, not a sufficient condition, i.e. a set of nodes (derived from a cross field) which fulfill (4.3) does not generally admit a quad layout. However, as shown by Jucovič and Trenkler [JT73] and recently discussed by Myles et al. [MPZ14] only a single configuration fulfilling (4.3) but not admitting a quad layout exists: one valence 3 and one valence 5 node on a genus 1 surface. While our experiments as well as the lack of mention in the pertinent literature in the field-driven quad meshing area suggest that this is not an issue in practice, one may simply modify this problematic configuration, should it occur, e.g. by merging the two singularities [BZK09], thus removing the two nodes.

Noise We assume the input triangle meshes we operate on are free from noise artifacts. The Gaussian curvature variations on noisy surfaces can lead to an excessively large number of cross field singularities. The abstraction from such artifacts or small-scale geometric detail is beyond our scope, but the use of surface smoothing methods or alternative, scale-aware objectives for the field construction [RVAL09, ECBK14] is surely possible.

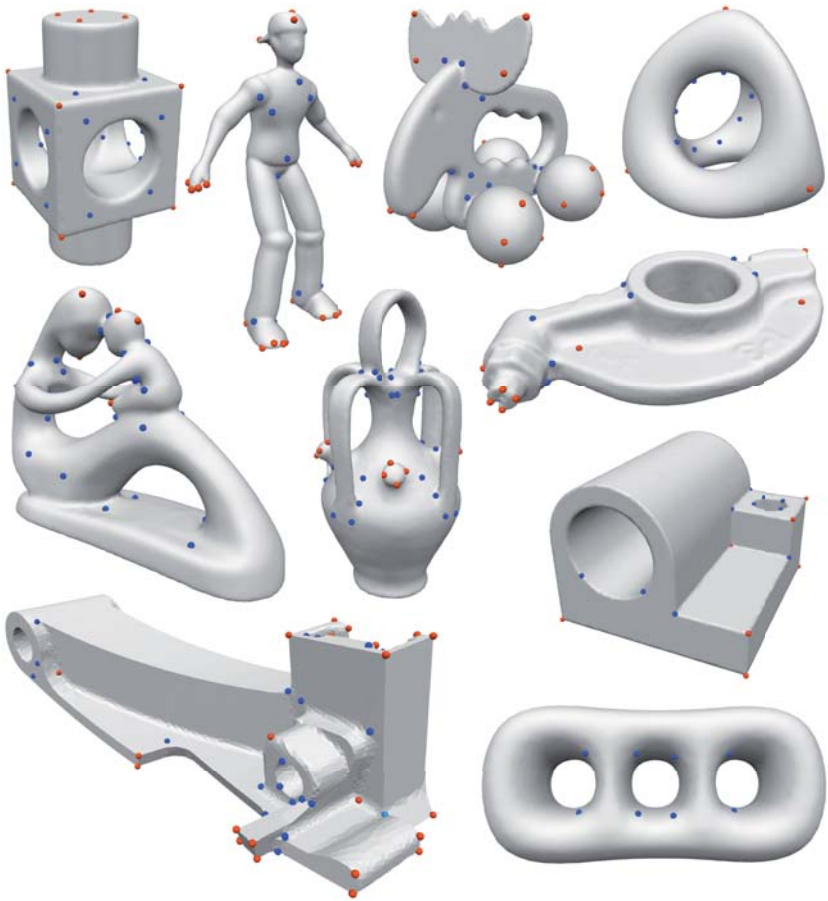


Figure 4.2.: Irregular nodes computed for various example models. Valence 3 nodes are displayed in red, valence 5 nodes in blue, and valence 6 nodes in cyan. Note how most valence 3 nodes lie in regions with positive Gaussian curvature and valence 5+ nodes in regions with negative Gaussian curvature, but where appropriate or necessary, e.g. in order to fulfill the Poincaré-Hopf theorem, nodes can also arise in flat regions.

5. Connectivity

Note: *This chapter is based on [CBK12].*

In stage 2 the connectivity of the irregular nodes created in stage 1 is to be determined. Probably the most obvious attempt to construct a quad layout given a set of nodes with prescribed valences consists in connecting these by incrementally constructing the separatrices (i.e. arcs or arc chains) from node to node. It turns out, however, that the structural consistency requirements ("only regular patches") are hard to control and satisfy when proceeding in this local, incremental manner: whenever a cycle in the graph of nodes and separatrices is closed during the process, strict conditions relating the valence of its patches to the number and valence of inlying nodes have to be met – otherwise additional irregular nodes would have to be introduced in order to still be able to obtain a quad-only layout. Besides the fact that these additional nodes might be geometrically implausible (causing low geometric fidelity), they further imply additional separatrices that are to be inserted. Hence, it can become difficult to generate a complete layout without introducing an excessive number of irregular nodes (causing high layout complexity). This primal, separatrix-based layout construction approach is furthermore highly order-dependent: early greedy decisions can render a layout unacceptable long before this is realized, as already discussed in Section 2.4.2. Takayama et al. [TPSHSH13] and Bommes [Bom12] likewise note these problems.

The key observation that allows us to avoid the problems of separatrix-based approaches is the fact that the layout consistency is much easier ensured when not working in the primal but in the dual setting (cf. Definition 2.1.11). In more detail, this means we do not directly construct the primal layout graph of separatrices connecting irregular nodes but its combinatorial dual, which we eventually dualize to obtain the primal. We construct this dual (embedded) graph from the arrangement of a set of crossing loops (closed curves) on the surface (cf. Figure 5.1 b,c for an example).

Murdoch et al. [MBBM97] already emphasized that the dual view of a quad mesh as intertwined loops is advantageous because these dual loops, due to their global nature, directly exhibit the global connectivity constraints involved – in contrast to the local primal arcs. Note that on surfaces with boundaries a dual layout contains boundary-to-boundary curves in addition to loops (cf. Section 2.1). To keep our exposition focussed we concentrate on the case without boundary first and elaborate on the general case in Section 5.3.

The central question now is how such a dual layout can be generated from the infinitely dimensional space of loops. In Section 5.1 we explain the theory behind our construction of arrangements of loops on surfaces that allows us to achieve layouts with geometric fidelity as well as structural consistency and simplicity. Afterwards, in Section 5.2, we describe the implementation in a discretized setting on triangulated surfaces.

5.1. Theory

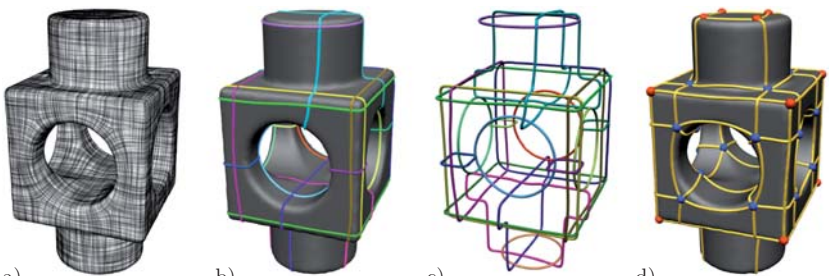


Figure 5.1.: Overview of the connectivity determination stage. Guided by a field of principal directions (a), a collection of transversally crossing loops on the surface is generated in a geometry-aware manner (b, c). These loops partition the surface into polygonal regions whose valences are intimately related to their total curvature. By dualizing the graph formed by these loops, we obtain the connectivity of a quad layout (d).

5.1.1. Dual Loop Arrangements

First, let us consider which properties an arrangement of loops needs to fulfill in order to induce a valid dual layout:

(D1) loop intersections are transversal (i.e. non-tangential),

(D2) no three loops intersect in one point,

(D3) each region has a valence of at least 1,

(D4) each region has disc topology.

The first two properties guarantee that each intersection induces a unique valence 4 vertex, the third ensures that no dual region has valence 0, which would imply an isolated primal node. The fourth point ensures a valid embedding with topologically simple faces.

If these properties are fulfilled, at least a valid dual graph is induced. Additionally we are interested in geometric fidelity. For this we need to take into account that a loose geometric relation between dual loops and the corresponding primal arcs can be assumed: because a dual loop runs through a strip of quad patches, the strip boundary arcs are roughly parallel to the dual loop. Considering this the quality criteria listed in Section 2.2 imply the following for the dual setting:

(D5) loop intersections should be close to orthogonal,

(D6) loops should follow principal curvature directions,

(D7) a region's valence should reflect its total curvature,

(D8) loops should be short and few in number.

Item (D5) encourages rectangular patches, (D6) encourages principal curvature alignment, and (D7) states a dual region valence-curvature relation analogous to the primal node valence-curvature relation (cf. Chapter 4), allowing for low patch-rectangle mapping distortion. Item (D8) directly translates into structural simplicity as the total length of primal arcs and dual edges is closely related in practice. Note that these items are soft requirements promoting quality while (D1)-(D4) are mandatory conditions.

5.1.2. Principal Curvature Fields

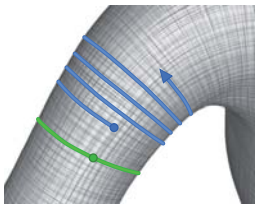
Constructing loops in a way that they are aligned with directions of minimal and maximal curvature directly serves fulfilling properties (D1), (D5), and (D6) (and indirectly also (D7)). Therefore, we base our construction on the cross field $\mathcal{C}$ computed in stage 1, cf. Chapter 4. It not only conforms with principal curvature directions in anisotropic regions, it furthermore is globally consistent and well-defined also in umbilic regions, which provides benefits for the following considerations. We will see that this cross field fundament also serves in fulfilling the other requirements because the irregular nodes appear in the field in the form of singularities, which encode Gaussian curvature information, as detailed in Section 4.1. In essence, the relation between node valence and Gaussian curvature implied by (4.3), (4.7), and (4.8) will ultimately serve to fulfill (D7). Figure 5.1 illustrates the basic idea of this stage.

5.1.3. Branched Coverings

Locally and away from singularities, the cross field $\mathcal{C}$ on $\mathcal{M}$ (which from now on we call F_C for distinction from a line field F_L and a direction field F_D) can be decomposed into four *direction fields*. This does not hold globally. However, as shown by Kälberer et al. [KNP07], we can construct a *four-sheeted branched covering surface* $\mathcal{M}_4$ of $\mathcal{M}$ with branch points induced by the singularities, and canonically lift the cross field F_C on $\mathcal{M}$ to a single smooth direction field F_D on $\mathcal{M}_4$. Analogously, we can construct a *two-sheeted branched covering surface* $\mathcal{M}_2$ of $\mathcal{M}$ and lift the cross field to a single smooth *line field* F_L on it. Figure 5.2 illustrates this schematically. Note that $\mathcal{M}_4$ is also a two-sheeted branched covering surface of $\mathcal{M}_2$. In the following we will deal with curves and loops on $\mathcal{M}_4$; note that they project uniquely onto $\mathcal{M}_2$ and $\mathcal{M}$ by means of the respective covering maps – for brevity we will not always mention this projection explicitly but simply refer to these loops or curves and their projection images by the same name. These covering concepts significantly simplify our further elaborations by allowing us to work with cross, line, or direction fields as needed.

5.1.4. Loops on Branched Coverings

Now consider what happens when we attempt to create a principal direction aligned loop through a point p on $\mathcal{M}$ by tracing. We choose one of the four directions from the local cross field by lifting p onto one of the four sheets of $\mathcal{M}_4$, trace a curve through the direction field F_D on $\mathcal{M}_4$ until we get back to the starting point[1], and project the curve from $\mathcal{M}_4$ to $\mathcal{M}$. The obvious problems are that the curve might never return, and if it does, it will probably be very long and far too complex to provide a useful loop for our purpose – we have to trade some deviation from the field for shortness. This is illustrated here on a cylindrical structure: the blue curve exactly follows the field and spirals over the surface, whereas the green curve slightly deviates from the field so as to be able to form a short loop.



Let $l : [0, b] \to \mathcal{M}_4$ denote a regular curve in arc-length parameterization which does not run exactly through a branch point (where the direction field is not defined). We define a combined measure c_α that rates such curves based on their direction fidelity and shortness, balanced by the parameter $\alpha \in [1, \infty)$:

$$c_\alpha(l) = \int_0^b \sqrt{\cos^2\theta(s) + \alpha^2 \sin^2\theta(s)}\, ds$$

[1]Note that, by tracing on $\mathcal{M}_4$, the curve is able to intersect itself on $\mathcal{M}$ – this is a highly desired behaviour, as quad meshes and layouts with non-intersecting dual cycles are only a restricted class.

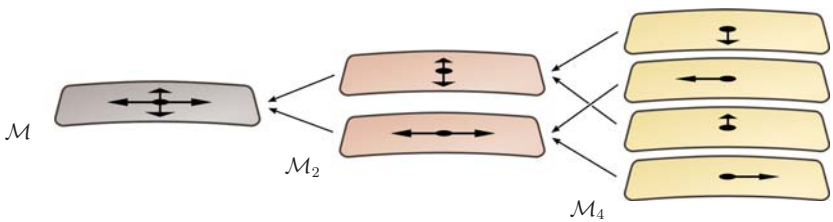
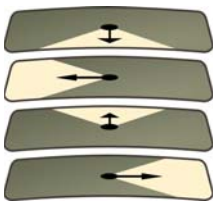


Figure 5.2.: Illustration of a small region of $\mathcal{M}$ with its branched coverings $\mathcal{M}_2$ and $\mathcal{M}_4$ and the cross, line, and direction field.

where $\theta(s)$ is the angle between the curves' tangent and the desired direction $F_D(l(s))$ at $l(s)$. A similar measure has also been employed by Tarini et al. [TPP*11] in the primal setting to rate separatrices. We are interested in non-trivial loops (closed curves) which minimize this measure for a prescribed α (called *minimal loops*) subject to certain constraints that we introduce in the following. As extreme cases, $\alpha = \infty$ leads to the abovementioned complex minimal loops, whereas $\alpha = 1$ leads to field oblivious geodesics. Values around 30 proved to provide a good balance for our purpose.

In order to obtain loops which are not only short and field-aligned, but furthermore can be combined to structurally valid loop arrangements we impose the following constraint: we restrict the set of non-trivial regular loops on $\mathcal{M}_4$ to the set L of *admissible* loops that fulfill $\theta(s) \in [-\frac{\pi}{4}, \frac{\pi}{4})$, i.e. we limit tangent deviations from the principal curvature directions. This effectively partitions the tangent space at each point into four "sectors", each describing the admissible loop tangents on one of the four sheets of $\mathcal{M}_4$. For $\alpha \gg 1$ minimal loops are very unlikely to violate this anyway, but imposing it as hard constraint allows us to guarantee important properties in the following. Furthermore, we define an arrangement of loops on $\mathcal{M}_4$ that have no intersections on $\mathcal{M}_2$ to be $\mathcal{M}_2$-*simple*. Note that $\mathcal{M}_2$-simple arrangements of loops meeting $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ contain no tangential intersections, thus fulfill (D1). This is illustrated in Figure 5.3 and elaborated in the proof of Theorem 5.1.1.



The following two theorems state the further beneficial properties of the $\theta(s) \in [-\frac{\pi}{4}, \frac{\pi}{4})$ constraint in conjunction with $\mathcal{M}_2$-simplicity. Let $\text{index}_{\mathcal{C}}(R)$ denote the sum of singularity indices lying in a region R:

Theorem 5.1.1. (Region valence-index relation) If a region $R \subset \mathcal{M}$ (bounded by loop segments) with disc topology in an $\mathcal{M}_2$-simple arrangement of admissible loops has k loop intersections on its boundary, then $\text{index}_{\mathcal{C}}(R) = 1 - \frac{1}{4}k$.

Proof. $\mathcal{M}_2$-simplicity and $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ guarantee transversality of all loop intersections on $\mathcal{M}$: a tangential intersection would imply equal tangent lines of the two loops at the intersection; as admissible tangent lines are disjunct on the two sheets of $\mathcal{M}_2$ above each point due to $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$, this would imply the intersection happens on one sheet of $\mathcal{M}_2$, contradicting $\mathcal{M}_2$-simplicity. Then, considering R is bounded by a cycle

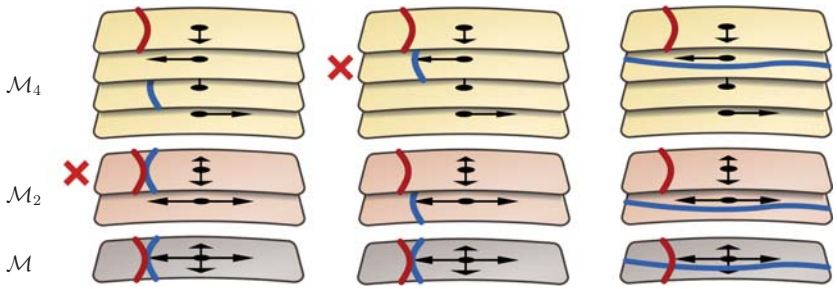
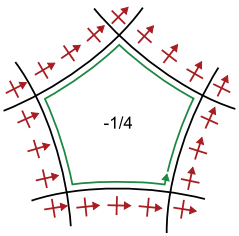


Figure 5.3.: Left: two loop segments (red, blue) have tangential contact on $\mathcal{M}$; this is possible as $\mathcal{M}_2$-simplicity is violated. Middle: the loops are $\mathcal{M}_2$-simple, but the $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ constraint is violated allowing for a tangential intersection. Right: $\mathcal{M}_2$-simplicity and the θ-constraint together only allow for transversal intersections.



formed by loop segments between k intersections (as depicted for the case $k = 5$), we determine the *turning number* [RVLL08] of the cycle to show the relation to $\text{index}_{\mathcal{C}}(R)$. Starting at an arbitrary intersection, we travel along the first segment to the next intersection. At that point the intermediate turning number t fulfills $-\frac{1}{4} < t < \frac{1}{4}$ since, due to the $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ constraint, the segment cannot make more than $< \frac{1}{4}$th of a full turn w.r.t. the field. Now turning to the next segment involves a rotation of the "sector" of allowed tangents by $\frac{\pi}{2}$, leading to $-\frac{1}{4} - \frac{1}{4} < t < \frac{1}{4} - \frac{1}{4}$ along this segment. Repeating this for all k segment switches to get back onto the start segment, we obtain $-\frac{1}{4} - \frac{1}{4}k < t < \frac{1}{4} - \frac{1}{4}k$ for the turning number of the whole cycle. As the turning number of a cycle necessarily is an integer multiple of $\frac{1}{4}$, we have $t = -\frac{1}{4}k$. It is related to $\text{index}_{\mathcal{C}}(R)$ by $t = \text{index}_{\mathcal{C}}(R) - 1$ [RVLL08], hence $\text{index}_{\mathcal{C}}(R) = 1 - \frac{1}{4}k$. □

This relation between singularity indices and region valences implies that whenever only a single singularity lies in a region, the region valence matches that of the node derived from the singularity in stage 1. It further shows that no further irregular nodes will be implied by the dual layout because regions with index sum 0 are always regular (i.e. bounded by four loop segments).

Now, the idea underlying the dual layout construction is to choose loops from the set L of admissible loops such that singularities get separated from each other, aiming at having each singularity lie in a separate irregular region – because then the primal layout obtained from the dual loops will have exactly the desired irregular nodes as determined in stage 1. We can achieve this separation in the following sense:

Theorem 5.1.2. (Singularity separation) In an $\mathcal{M}_2$-simple arrangement of loops any region R with $\text{index}_{\mathcal{C}}(R) > \frac{1}{4}$ containing multiple singularities of index $\frac{1}{4}$ or $\frac{1}{2}$, and any region R with $\text{index}_{\mathcal{C}}(R) < -\frac{1}{4}$ containing multiple singularities of index $-\frac{1}{4}$ can be split into regions with smaller $|\text{index}_{\mathcal{C}}(R)|$ by adding further admissible loops, without introducing irregular regions outside R and without violating $\mathcal{M}_2$-simplicity.

Proof sketch. Note that we can always add loops that (with infinitesimal distance) run along existing loops without violating $\mathcal{M}_2$-simplicity and without introducing any new irregular regions. The proof is based on the construction of one or more such loops that are then cut and reconnected within R. Figure 5.4 illustrates these constructions for the concerned cases. Note that reconnections inside the regions are possible in the shown ways because each subregion fulfills $\text{index}_{\mathcal{C}}(R) = 1 - \frac{1}{4}k$.

This means we can guarantee that singularities can suitably be separated by admissible loops for most practically relevant cases. The theorem further extends to several

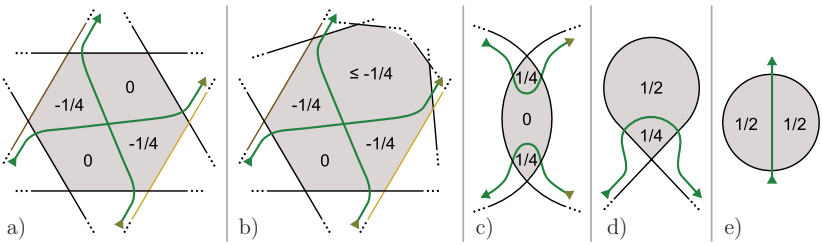


Figure 5.4.: Schematic illustration of the cases of Theorem 5.1.2: region with $\text{index}_{\mathcal{C}}(R) = -\frac{1}{2}$ (a), $< -\frac{1}{2}$ (b), $\frac{1}{2}$ (c), $\frac{3}{4}$ (d), 1 (e). Numbers indicate the $\text{index}_{\mathcal{C}}$ of the subregions. The green curve is a single loop that splits the region while running solely along other loops outside the region (a)-(d). In (e) the loop can be connected on the outside due to symmetry (the outside region has this same boundary loop).

cases which include multiple higher-order singularities (which generally are quite rare in practice). But note that even an exceptional case where singularities cannot be separated completely does not cause a problem: the dual loops always imply a valid primal quad layout – in which any non-separated nodes have effectively been merged (cf. Section 5.2.3). This can be seen as a form of adjustment of the input which did not well fit the current stage's objective, as laid out in Section 3.1.5.

Let us briefly summarize which insights we gained regarding fulfillment of the eight required/desired properties: (D1) and (D2) are ensured by $\mathcal{M}_2$-simplicity in conjunction with the $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ constraint, (D5) and (D6) are respected by minimizing c_α for $\alpha \gg 1$, (D3) can be achieved due to Theorem 5.1.2 if there are no singularities of index $\geq \frac{3}{4}$ (which normally is the case in practice, and can even be ensured by dividing such singularities), (D7) is fulfilled according to Theorem 5.1.1. The shortness part of (D8) is governed by using c_α-minimal loops.

For (D4) "each region has disc topology" there is no theoretical guarantee that this can always be achieved on objects of non-zero genus; in fact, one can artificially create pathological cross fields such that loops with arbitrarily large $|\theta|$ are necessary to cut the surface to discs (cf. the cross field depicted in Figure 4 left in [KNP07] or the one depicted in Figure 5 left in [MPZ14] for basic examples). According to all our experiments these cases do not arise from the employed principal directions guided field construction in practice. Nevertheless, holonomy constraints [LJX*10] can be employed in the field construction process to provide rigorous guarantees.

In conclusion, we know that it should be *possible* to construct desired dual layouts from admissible loops, and the remaining question is *how* this can effectively be done, i.e. how to choose *few* admissible loops from the set L that fulfill $\mathcal{M}_2$-simplicity and suitably separate singularities. For this we propose a greedy strategy as described in the following section.

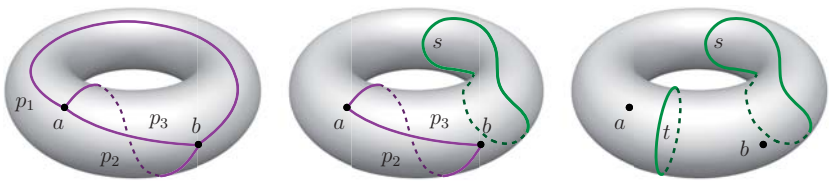
5.1.5. Greedy Loop Selection Strategy

The primary objective for loop selection is the separation of cross field singularities. In favor of layout simplicity this should be achieved with few loops. We construct the arrangement of loops in an incremental manner by iteratively adding further loops that

do contribute to the goal of singularity separation. In this process, to ensure $\mathcal{M}_2$-simplicity, a new loop may not intersect the previously constructed ones on $\mathcal{M}_2$. Let $L(A)$ denote the subset of loops from L which, on $\mathcal{M}_2$, do not intersect the loops of an arrangement A.

For the purpose of deciding whether a loop contributes to singularity separation, imagine the set of all paths on $\mathcal{M}$ connecting a pair (a, b) of singularities without crossing any loop of an arrangement A (cf. Figure 5.5). As separation is a topological concept, we are not interested in these paths' geometry but solely in their homotopy classes. Let $H(A, a, b)$ denote the set of these classes, and $H(A) = \cup_{a \neq b \in S} H(A, a, b)$, where S is the set of all singularities. A loop $l \in L(A)$ is said to be (A, h)-*cutting* if $h \in H(A)$ but $h \notin H(A \cup \{l\})$. Intuitively, l actively increases the "level of separation" by cutting all connections of class h when adding it to A. Figure 5.5 illustrates this abstractly on a simple torus; Figure 5.8 shows a less abstract yet simple example (in the discretized setting).

Note that $H(A)$ can be infinite (and a single loop can be (A, h)-cutting for an infinite number of classes h). In Section 5.2.2, which deals with an implementation of the following algorithm, we describe how this is handled in practice.



$h_{1/2/3} \in H(\varnothing, a, b)$ $\quad$ $h_{2/3} \in H(\{s\}, a, b)$ $\quad$ $H(\{s, t\}, a, b) = \varnothing$

Figure 5.5.: Left: three paths $p_{1/2/3}$ of three exemplary homotopy classes $h_i = [p_i]$ connecting singularities a and b are depicted. The loop arrangement A is still empty. We have $h_{1/2/3} \in H(\varnothing, a, b)$. Middle: All paths of h_1 cross the loop s, thus $h_1 \notin H(\{s\}, a, b)$. Hence, s is $(\varnothing, h_1)$-cutting. Right: All paths of h_2 and all paths of h_3 cross loop t, thus $h_{2/3} \notin H(\{s, t\}, a, b)$. Hence t is $(\{s\}, h_2)$- and $(\{s\}, h_3)$-cutting. Furthermore, now $H(\{s, t\}, a, b)$ is empty, no more paths between a and b not crossing $A = \{s, t\}$ exist.

Surely, we are not interested in arbitrary cutting loops but minimal ones: let $l_{\min}(A,h)$ denote the c_α-minimal (A,h)-cutting loop. Then, for some arrangement A, $L_{\min}(A) = \cup_{h\in H(A)} l_{\min}(A,h)$ is the set of all minimal loops whose addition to A would contribute to further singularity separation.

Algorithm: We iteratively add such minimal cutting loops to increase the level of separation based on a greedy approach:

$$\begin{aligned} A^0 &= \varnothing \\ A^{i+1} &= A^i \cup \{ \operatorname*{argmax}_{l\in L_{\min}(A^i)} c_\alpha(l)\} \end{aligned}$$

The iteration is repeated until $H(A^i)$ is empty or $l_{\min}(A^i,h)$ is undefined for each $h \in H(A^i)$ because there are no further cutting loops (in cases where not all singularities are separable).

The greedy choice of the maximum instead of a minimum might seem unnatural, but it typically leads to better results. First notice that even the maximum (with respect to c_α) out of the loops of $L_{\min}(A^i)$ is still a minimal cutting loop – for each path homotopy class the set $L_{\min}(A^i)$ contains only the c_α-minimal of the loops that cut it. Hence the choice of the maximum does not lead to unnecessarily long and field-misaligned loops as it might appear at first sight – it still selects from the best loops that are available

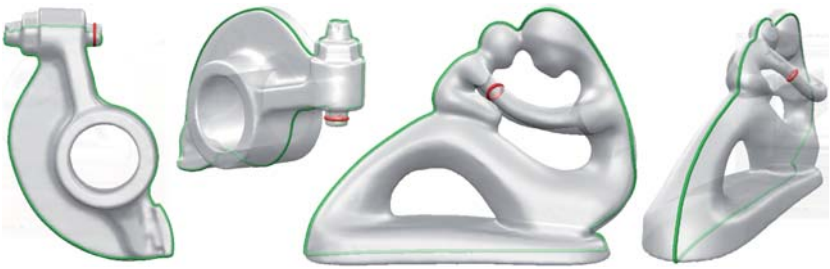
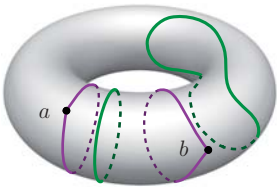


Figure 5.6.: Visualization of the maximum (green) and minimum (red) of all minimal cutting loops $L_{\min}(A^0)$ in the RockerArm and Fertility examples from Section 5.4. Notice that the longer loops are typically concerned with the global structure, whereas the short ones capture rather local features. The maximum-first greedy approach thus, in some sense, pursues a coarse-to-fine strategy.

to achieve separation of certain singularities. See Figure 5.6 to get an idea of how such loops look like.

Consider the class $h_{\max} = \arg\max_{h \in H(A^0)} c_\alpha(l_{\min}(A^0, h))$, i.e. the class that, among all homotopy classes, requires the c_α-largest loop to get cut. This loop is $l_{\min}(A^0, h_{\max})$ and $h_{\max}$ cannot be cut by any c_α-smaller loop; in fact, if we first add some loops cutting other classes, then $l_{\min}(A^j, h_{\max})$ for $j > 0$, i.e. the best loop available to cut $h_{\max}$ after j iterations, can easily be c_α-larger due to $L(A^{i+1}) \subset L(A^i)$ by definition. Following the maximum-first strategy, the long loops necessary for separation of some singularities are secured early, in contrast to a minimum-first strategy where other loops already restrict the space of $\mathcal{M}_2$-simple loops available to cut $h_{\max}$ in the end. Hence, as confirmed in our experiments, this maximum-first strategy typically leads to arrangements with a smaller total sum as well as lower variance of the loops' c_α-values.

The objective of achieving regions with disc topology seamlessly integrates into this algorithm: for each singularity a the set of homotopy classes of non-contractible paths connecting a with itself, i.e. $H(A, a, a)$ without the trivial class of contractible loops, is simply included in the set $H(A)$. $H(A)$ will then not become empty before in A all singularities lie in disc topology regions. This is illustrated here based on the example from Figure 5.5 right: two still remaining non-contractible paths self-connecting a and b are depicted.



5.2. Discretization & Implementation

We now present the techniques necessary for a practical implementation of the proposed method on triangulated surfaces $\mathcal{M}$. In this context the terms *vertex*, *edge* and *face* will refer to the triangle mesh $\mathcal{M}$ rather than to dual layout components. The field F_C, with its lifted variants F_L and F_D, is computed and represented discretely per face as described in Chapter 4. To simplify parts of the implementation by avoiding special cases, we assume no two singularities are directly adjacent, i.e. incident to a common mesh edge – this can easily be ensured by splitting the edge, inserting another vertex.

The branched covering triangle meshes $\mathcal{M}_2$ and $\mathcal{M}_4$ are constructed as described by Kälberer et al. [KNP07].

5.2.1. Anisotropic Front Propagation

Loops minimizing c_α are anisotropic geodesics and, given a start point p, can efficiently be computed by anisotropic front propagation on $\mathcal{M}_4$. In this setting α plays the role of a global anisotropy coefficient that, together with the direction field, forms the tensors defining an anisotropic local surface metric guiding the propagation. This process can be understood as a "fuzzy" curve tracing: the fuzzy curve travels fast in the principal direction and slower the more it deviates (cf. Figure 5.7). When the front reaches p again, the minimal loop is found.

Sethian and Vladimirsky [SV04] presented a method for anisotropic front propagation with nice convergence properties, but on coarse meshes the relative error as well as the runtime is rather high for large anisotropy coefficients. Dijkstra's classical shortest path algorithm applied to the mesh graph of $\mathcal{M}_4$, with edge weights computed according to the metric c_α, is much faster, but the error is even worse for large anisotropy coefficients. A detailed analysis of the quality and performance of available algorithms is given in Chapter 9.

For this reason we created a novel variant, Short-Term Vector Dijkstra, which drastically increases the accuracy by propagating distances not only along edges, but rather over vectorial chains of edges up to discrete length k. This conceptually increases the angular resolution of the propagation process, as illustrated in Figure 5.7 left. We found $k = 4$ to be sufficient, working very well for our purposes (cf. Figure 5.7 right). The algorithm is presented in detail in Chapter 9. We here use a variant of this algorithm which does not build the edge chains ad hoc. Instead we precompute all edge chains up to length k and their weights and store these meta-edges in a graph P. On this graph the algorithm reduces to Dijkstra's original algorithm. Using this explicit representation of the graph, we can take care of the propagation constraints more easily, as detailed in the following. Furthermore, for each meta-edge in P we remember corresponding edges in $\mathcal{M}_4$, such that a loop computed in P can then be represented as a cycle of edges in $\mathcal{M}_4$.

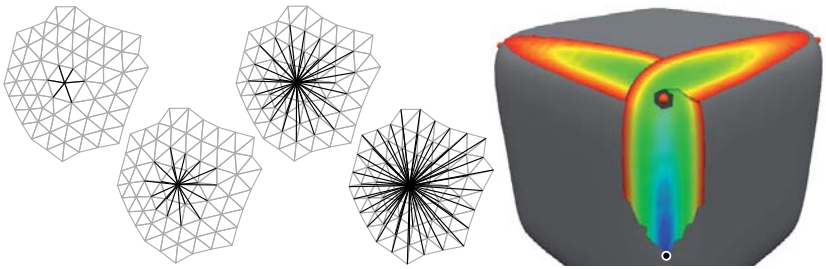
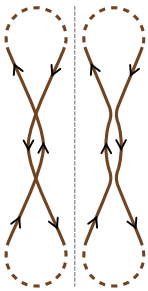


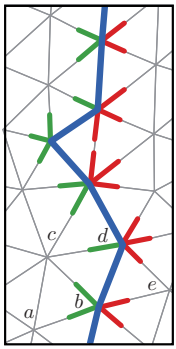
Figure 5.7.: Left: 1-, 2-, 3-, and 4-disc edges of a vertex. Right: Distance visualization of anisotropic propagation on a rounded cube mesh. Since we propagate on $\mathcal{M}_4$ the front can cross over itself at the branchings and pursue different courses on the different covering sheets, according to the respective field directions.

Propagation Constraints The $\theta \in [-\frac{\pi}{4}, \frac{\pi}{4})$ constraint is easily imposed by discarding those edges from the propagation graph P which violate it. $\mathcal{M}_2$-simplicity of the layout is achieved by forbidding the front to cross existing loops on $\mathcal{M}_2$. Additionally, $\mathcal{M}_2$-simplicity of the new loop itself is to be ensured. In the continuous case this is guaranteed by its minimality, but in the discrete setting constructed loops could have small local self-intersections on $\mathcal{M}_2$. These can easily be resolved as depicted: at the intersections, the loop is cut, the parts alternately reversed, and reconnected to a non-intersecting (just grazing) one.



Preventing the front from crossing already existing loops could easily be accomplished by removing the loops' underlying vertices (and incident edges) from the propagation graph. This, however, directly limits the local maximum density of constructed loops to the resolution of the triangulation. We avoid potential dead-ends in the propagation by allowing multiple loops to run over the same graph edges locally – they just may not cross in the sense of $\mathcal{M}_2$-simplicity: during front propagation each graph vertex at the front memorizes for each already existing loop whether the path to this vertex currently runs along the loop on $\mathcal{M}_2$ and whether it moved onto it from the left or from the right. Propagation may then only proceed further along the loop or back to the side it came from, effectively preventing crossing. This can efficiently be implemented by equipping each constructed loop with so-called *whiskers*:

two sets of half-edges, those incident to the loops' vertices from the one and the other side, respectively, as illustrated on the right in green and red. The "memory" per front vertex then consists of two flags per existing loop, that are set or reset whenever propagation proceeds over a green or red whisker, respectively, depending on the half-edge orientation. As an example consider the case of the front propagating from vertex a to b. The green flag is set at b due to moving over a green whisker. This disallows further propagation over a red whisker to e; propagation to d carries the green flag over to d, and when propagating to c the flag is not carried on due to moving over a green whisker reversely.



5.2.2. Singularity Separation

Being able to construct the loops of L in an $\mathcal{M}_2$-simple manner, we now implement the greedy approach presented in Section 5.1.5. Recall that we need to consider an infinite number of connecting paths in a potentially infinite number of homotopy classes. We employ two efficient heuristics to make this tractable, and add a simple post-validation procedure to be able to guarantee correctness even in case they fail. In practice, they perform so well that post-validation is required to come into action only very rarely. Even then, the worst case is that the result is not greedy-optimal in the described sense, but still structurally correct.

In each step of the greedy construction we need to determine loops which cut certain homotopy classes. To be able to do this efficiently, in the beginning we construct so-called *separation indicators* (SIs) for each pair (a, b) of singularities – paths representing a homotopy class each (discretely embedded in the mesh graph). Based on the intersection constellation of a loop and an SI we are able to estimate whether the loop cuts the whole homotopy class (heuristic I). Figure 5.8 shows an example of these SIs. While on genus 0 objects (with trivial homotopy group) we have a single SI for each pair, on objects of genus $g > 0$ the homotopy group is infinite; for practicability we restrict ourselves to $2g + 1$ SIs constructed as described by Erickson and Whittlesey [EW05] for the case of loop homotopy bases (cf. Figure 5.5 left for the simplest example of such $2g + 1$ SIs between two singularities on a genus 1 object).

The rationale behind this is the observation that SIs of further homotopy classes (winding around surface handles multiple times, etc.) are very likely to be cut by loops that cut their simpler siblings, anyway (heuristic II). The mentioned method [EW05] is adapted from the loop case to our case of paths by not using the distance field of one loop root point but two distance fields, one of each path end point, i.e. the two singularities to be connected, in the maximum spanning tree construction (cf. [EW05] for details on this). This yields $2g$ SIs. If $a \neq b$, an additional one, not crossing the cutgraph involved in the construction at all, is created simply by Dijkstra's shortest path algorithm (a step which is not necessary in the case $a = b$ dealt with by the original method due to the triviality of this one class).

Given an SI s and a loop l, we estimate whether l cuts all connections from s's class [s] (heuristic I). If l does not cross s, we can be sure that it does not; if it crosses s an odd number of times, we can be sure it does. In the case of an even number of crossings, we cannot definitely decide locally – whether [s] is cut completely depends on the global interplay of multiple loops. Instead of performing expensive global checks in this regard, a heuristic, exploiting that l is created on $\mathcal{M}_4$, proved to be highly reliable: we check how many of the four pre-images of s on $\mathcal{M}_4$ are crossed by l. If only one is crossed we consider s not cut, if more are crossed we consider s cut. The rationale is that between two crossings, l has to travel around multiple singularities for these two crossings to lie on different sheets – implying intersections of l with further loops in between. If all crossings are on the same sheet, we usually have a situation as depicted on the right, not signifying a true cut.

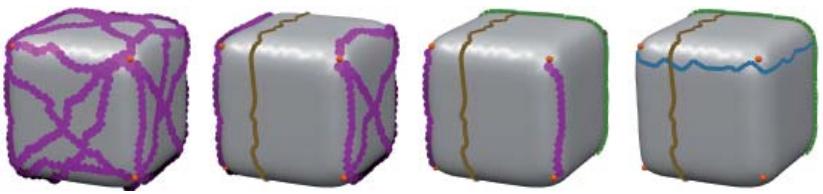


Figure 5.8.: Visualization of separation indicators (pink) between 8 singularities on a cube: all, and those that remain uncut after the addition of the first, second, and final third loop, respectively.

Now, for each SI s, we aim at finding the minimal loop $l_{\min}$ cutting it. As this loop necessarily crosses s, we can start a front propagation from each graph vertex v on s and compute the minimal loop through each of them. More precisely, we start from two of the four pre-images of v on $\mathcal{M}_4$ – the two other sheets with opposite directions only lead to the same loops in reverse. The c_α-smallest of those that cut s then is $l_{\min}$ of s, called the *candidate* for s.

Instead of computing the candidate for each SI, i.e. the whole set $L_{\min}$ (cf. Section 5.1.5), we can heavily increase efficiency in the spirit of lazy evaluation by exploiting the fact that we are always interested in the c_α-largest candidate in each greedy step. If a candidate of s cuts other SIs, their candidate does not (yet) need to be computed as it can only be equal or smaller. To provide an example of the gain in efficiency typically provided by this optimization: for instance on model FERTILITY only 30 candidates are computed in the beginning, they already cut all 10,152 SIs; 11 more are later (re)computed (in repetitions of step 4 of the following algorithm).

Algorithm: With the initial set of candidates C we can then perform an iteration of the greedy algorithm, starting with an empty arrangement A, in the following manner:

1. Add the largest candidate c: $A \leftarrow A \cup \{c\}$, $C \leftarrow C \setminus \{c\}$,
2. remove all SIs that are cut by c,
3. discard candidates from C that are no longer $\mathcal{M}_2$-simple with respect to A,
4. for SIs that are not cut by any curve from C compute new candidates (again in a lazy manner) and add them to C.

These four steps are repeated until no more candidate is available.

To appropriately handle rare cases where in the end some singularities remain unseparated due to the two heuristics, we perform a post-validation: we iteratively check whether paths between two singularities still exist by simple flood-filling within the regions of A (not crossing any loop of A), add these paths as new SIs, and perform further greedy steps (beginning with step 4 to compute new candidates) until no more paths or candidates exist.

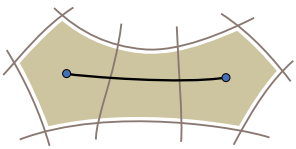
5.2.3. Layout Primalization

The constructed dual layout implies the topological structure of the primal layout: each dual region corresponds to a primal node, and region adjacency implies node connectivity.

Note that this connectivity is not of purely combinatorial nature, but in addition has a topological aspect: two nodes cannot only be connected or not connected, the connection's homotopy (with respect to $\mathcal{M}$ punctured at the irregular nodes [MPKZ10]) also plays a role. In particular, two nodes can be connected by multiple arcs from different homotopy classes. This information needs to be encoded properly, such that the following stage can rely on an unambiguous representation.

To this end we select one arbitrary vertex $\nu(h)$ per region as representative for the corresponding primal node h – in irregular regions containing one singularity it is of course natural to chose the corresponding vertex. Then, for each pair of adjacent regions, we compute a chain $\gamma(a)$ of successively adjacent mesh faces[2] connecting the nodes of these regions, representing the corresponding primal arc a. A simple breadth-first search on the dual mesh graph, restricted to two adjacent regions and only allowed to cross the separating dual loop once, is sufficient to generate these face chains – because their geometric quality does not matter for the subsequent stage, only their homotopy class. The maps ν and γ then completely encode the layout's topological structure – via a prototypical, discrete embedding of the layout graph in $\mathcal{M}$.

For the purpose of visualization of the layout connectivity, it can be useful to take the geometry into account in the computation of representative arc paths. Instead of using the simple breadth-first search, we can compute the connection as a region-restricted geodesic path with respect to the anisotropic curvature-driven metric (cf. Section 5.1.4). Furthermore, instead of computing paths for the individual arcs, we can compute paths for whole separatrices (sequences of arcs from irregular node to irregular node) through the corresponding dual *corridors* as depcited alongside. The regular nodes can then be posi-



[2]The choice of a face chain instead of a perhaps more intuitive edge chain simplifies the handling of arcs in the next stage.

tioned onto the intersections of these separatrix paths, leading to a smoother geometric representation for visualization purposes. The actual *global* optimization of the geometric embedding (based on ν and γ), however, is dealt with in Chapter 6.

5.3. Extensions

Boundaries On surfaces with boundaries, dual layouts contain dual curves which run from boundary to boundary in addition to dual loops. In order to enable the automatic construction of such curves where appropriate we simply modify the loops construction procedure described in Section 5.2.1: instead of propagating the front on $\mathcal{M}_4$ from the seed into the field direction until it hits the seed again, we propagate two fronts in opposite directions (i.e. on two different sheets of $\mathcal{M}_4$) until they either hit each other on $\mathcal{M}_2$ *or* both hit the boundary – whatever happens first. Tracing back through the two distance fields yields a loop in the former case and a boundary-to-boundary curve in the latter case. The separation concept that drives the loops selection strategy is also easily extended to the boundary case: in addition to the separation indicators for singularities we include boundary separation indicators (shortest paths between each pair of boundary loops) such that no two boundaries lie within the same region in the end, as this would topologically merge them in the layout.

Feature Curves For cases where $\mathcal{M}$ contains feature curves, we can make a simple extension to the algorithm to allow for better alignment of arcs to features. Given a set of curves representing the features (e.g. selected using dihedral angle thresholding for the simplest case of sharp features) we use them as directional constraints in the cross field construction as described by Bommes et al. [BZK09] such that the cross field is aligned with the features. We then lift each curve to the one sheet of $\mathcal{M}_2$ the line field of which is aligned with the curve tangent, and modify the definition of $\mathcal{M}_2$-simplicity to include that loops may also not intersect these lifted feature curves on $\mathcal{M}_2$. This prevents loops from crossing feature curves in a manner that these curves would not lie within a dual corridor in the final loop arrangement – which would make aligning some separatrix to the feature impossible. After performing the greedy algorithm with this change, separatrices are then not constructed as geodesics but constrained to features where applicable.

It is worth noting that, while this procedure prevents arcs from crossing features unsuitably, it does not guarantee that *each* feature segment is captured by a layout arc in the final layout. In fact, different strategies would be necessary already in the node determination stage in order to enable complete alignment in the presence of complex global feature configurations as recently shown by Myles and Zorin [MZ13, MPZ14]. The evaluation of possibilities to combine their ideas with the dual loops concept for the generation of quad layouts with full control over alignment to complex features and boundaries is an interesting avenue for future investigation.

Additional Loops The presented algorithm creates layouts A that are topologically minimal in the sense that no loops are added that do not contribute to singularity separation. This is a desirable property, allowing us to obtain simple layouts. But depending on the geometry of $\mathcal{M}$, the addition of some further loops can potentially have a much larger impact on the improvement of geometric fidelity than on the concomitant decrease of simplicity as it can alter the node connectivity in the primal layout – which might allow for better curvature and feature alignment. Loops whose addition to A can cause such changes can be identified in the following way: each SI lifts to two curves on $\mathcal{M}_2$; we say an SI is *doubly-cut* if both of these are cut by the loops of A on $\mathcal{M}_2$. A further loop causes a change to node connectivity if its addition to A doubly-cuts an SI that, so far, was only singly-cut. This is based on the observation that singly-cut SIs might appear in the primal layout as separatrices (in the form of curves homotopic to the SI), while doubly-cut SIs do not. Figure 5.9 shows a prototypical example of the situation.

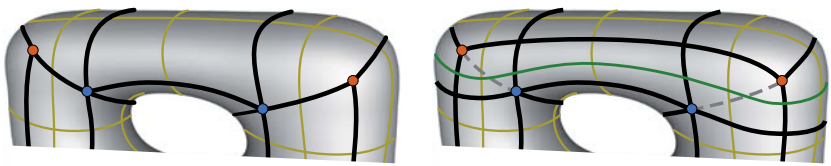


Figure 5.9.: Left: Primal quad layout (black) induced by a loop arrangement (yellow). Right: Adding a further loop (green) alters the irregular node connectivity: the dashed connections vanish, caused by the cuts (by yellow and green loops) on both sheets of $\mathcal{M}_2$.

In order to exploit this possibility, after the greedy algorithm has been performed, we can, where possible, construct candidate loops doubly-cutting the SIs that, so far, are not. Then the quality improvement can be assessed for each of these loops by rating the separatrices in the corresponding primal layouts based on their cross field deviation. Starting with the one providing the highest improvement, loops can then iteratively be added until no further loop is admissible or no further gain possible. The assessment can quickly be made based on the initial embedding, or, more accurately, based on the final improved embedding. Notice that it is also possible to remove a loop from A if, by the addition of others, it becomes redundant (= all SIs cut by the loop are also cut by other loops).

5.4. Results

In this section we show some exemplary results obtained by our implementation of the presented method. Figure 5.11 depicts the result layouts, the statistical facts are summarized in Table 5.1.

For the mere purpose of comprehensible layout connectivity visualization, the arcs and nodes shown in Figure 5.11 have been smoothed using a simple variant of the approach presented in [TPP*11], which is based on iterated parameterizations (cf. Figure 5.10), as detailed in [CBK12]. In the next chapter our novel, more sophisticated layout geometry optimization is presented.

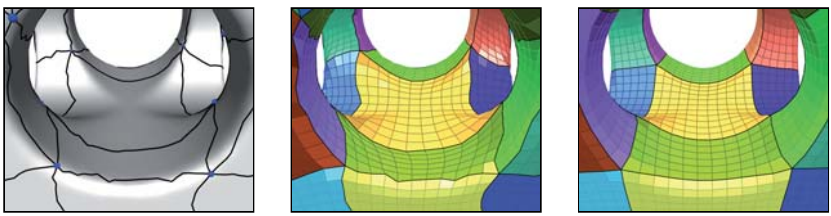


Figure 5.10.: A quad layout with initial embedding (shortest paths on the mesh graph), an initial harmonic per-patch parameterization, and the final optimized parameterization from which smooth arcs can be derived to more clearly visualize the connectivity.

Model	Singularities	Loops	Nodes	Patches
TetraThing	24	18	80	84
Fertility	48	24	92	98
Block	48	20	72	76
3Holes	16	12	16	20
Elk	52	22	86	86
Joint	24	16	77	79
Guy	40	18	170	168
RockerArm	30	15	115	115
Botijo	72	36	213	221
Lever	83	36	269	275

Table 5.1.: Statistics of the constructed quad layouts shown in Figure 5.11.

With our current implementation the processing from the initial construction of the coverings with the fields, over creation of SIs and execution of the greedy algorithm, to the primalization of the layout takes from a few seconds to a few minutes on a commodity PC (model complexities between 20K and 122K faces). The processing is clearly dominated by the front propagations performed to construct candidate loops – which can easily be parallelized.

As feature constraints (cf. Section 5.3) only the trivially detectable sharp edges on Joint and Lever have been used – advanced detection techniques for smooth feature edges were not employed.

Comparison We applied our method to several datasets also taken as input by Tarini et al. [TPP*11] and Bommes et al. [BLK11] for their simplification methods. While on model RockerArm the patch count of our result is higher by a factor of 1.17 compared to the result of Tarini et al., on model 3Hole the same obvious layout was achieved, and our layouts of models Joint, Botijo, Fertility, and Lever are simpler by factors of 1.4, 2.1, 2.6, and 2.8, respectively (the Botijo and Lever results of Tarini et al. can be found in their supplemental material). In comparison to Bommes et al. the layout complexities in terms of patch count achieved by our Dual Loops Meshing approach are even simpler by factors of 1.5, 3.3, 4.7, and 5.4 on models RockerArm, Lever, Botijo, and Fertility, respectively. This generally large difference is explained by

the fact that their method is based on removing only certain helical configurations from the base complex.

Surely, the possibility cannot be ruled out that the greater layout simplicity might come at the expense of geometric quality to some extent, but visual inspection and comparison of the results does not reveal distortions which would generally be considered intolerable or out of proportion.

The general advantage of the simplification-based methods is their explicit control of the geometric quality: the deviation from the start configuration can be kept track of and be limited as desired. Furthermore, any previous state can always be used as a fallback solution. In this sense such methods can be considered safer than from-scratch-construction approaches like ours – especially when geometric fidelity is of higher importance than simplicity.

On the downside, it is not easy to drive such simplification processes in a well-targeted manner; roll-back mechanisms can be necessary to undo series of previous steps in order to escape from dead-ends. The robust automatic generation of the quad meshes taken as input is a further non-trivial subtask. These input meshes' edge directions are then implicitly taken as geometrically optimal reference and structural simplification aims at minimal deviation from that. However, assuming the input mesh is generated in a common field-guided manner, e.g. driven by principal directions, this reference will often already deviate from the original field (in the sense that quad mesh edges are not well-aligned with the field everywhere) as such fields rarely are integrable and can only be aligned to approximately during meshing. The finite density of the mesh further contributes to this deviation.

The Parameter α While variation of α could be expected to be useful for controlling the layout complexity to some extent, it should be mentioned that this relation is not linear (nor fully monotonic) due to the discrete decisions implicitly made by the routing of the loops. Especially for small α (<10) loops tend to become too unaligned with the field, this incompatibility sometimes leading to less plausible, complex loops being necessary towards the end of the algorithm. On the other hand, increasing α beyond ~ 70 usually does not lead to any more changes due to accuracy limitations of the discrete propagation process. The setting $\alpha = 30$ proved to work very well quite generally and all the presented layouts have been obtained with this fix setting.

5.5. Discussion

The presented method allows for the creation of simple all-quadrilateral patch layouts by means of the construction of loop arrangements as duals of these layouts. This dual perspective beneficially exhibits the global structural implications involved in quad layouts more directly. Our method is built on a theoretical framework based on suitably restricted loop spaces on branched coverings of the underlying surface induced by principal curvature cross fields. It allows us to guarantee valid dual layouts which furthermore show good geometric fidelity. In the end, the primal layout connectivity can easily be derived from the dual layout.

The proposed approach employs a greedy strategy for practicability. Employing a non-greedy, globally optimizing loop selection would be very attractive – but is not straightforward due to the interdependencies caused by the necessary $\mathcal{M}_2$-simplicity. Further, while control over the simplicity of the primal layout is directly given in the dual setting, control over the geometric fidelity is, to some extent, indirect due to the loose relation between dual and primal geometry.

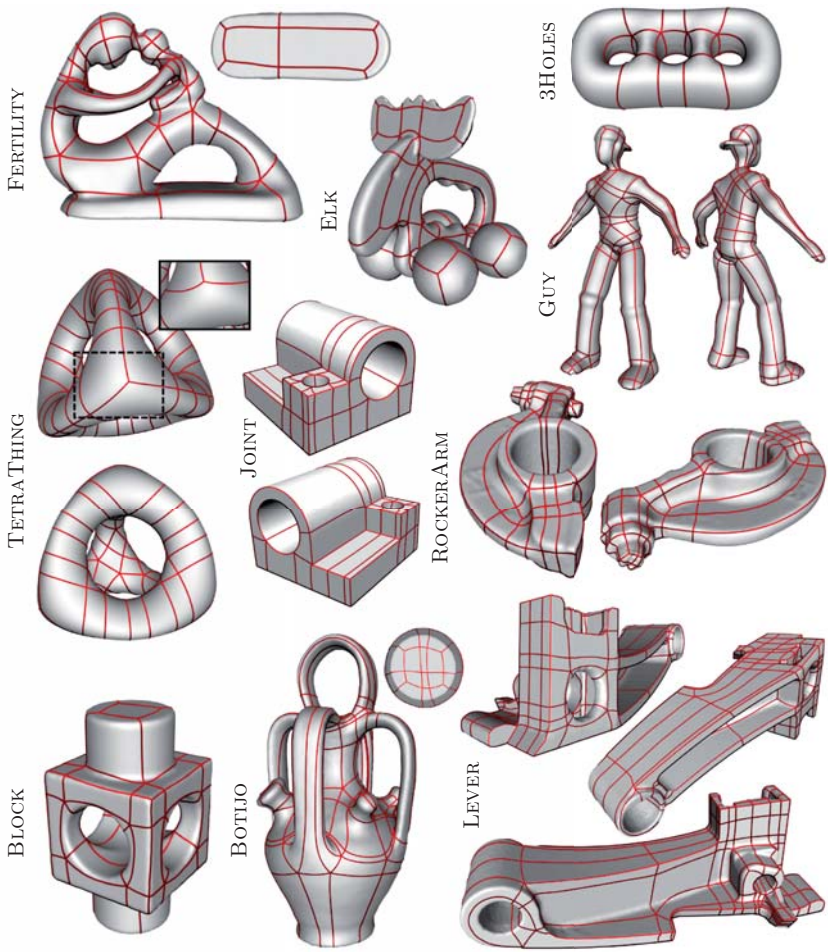


Figure 5.11.: Quad layout connectivity generated by our method. The inset at example TetraThing shows the result before an additional loop which increased separatrix/field alignment was added after the greedy algorithm (as described in Section 5.3). The Fertility and Botijo insets show bottom views of the models.

6. Embedding

Note: *This chapter is based on [CK14b].*

In the third and last stage of our method the layout's geometry, i.e. its embedding in the surface is to be optimized. As input from the previous stage we take the created layout graph G with its prototypical discrete embedding given by ν and γ (cf. Section 5.2.3).

It is worth noting that this input may also be taken from a manual layout construction process, based on drawing a sketch on the surface, potentially supported by assisting systems (cf. Section 6.2). With this generic setup, our stage 3 cannot only be used in

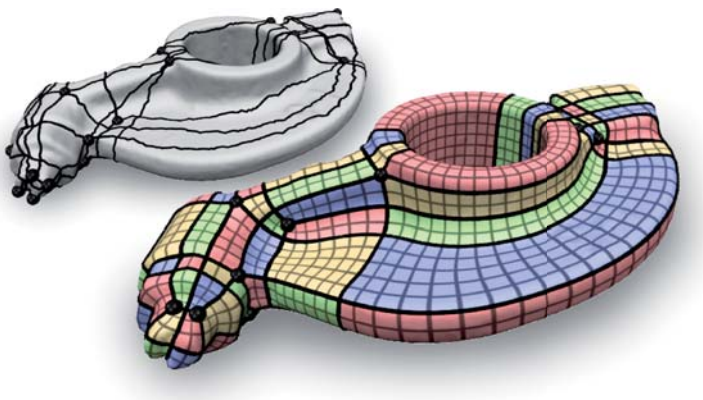


Figure 6.1.: Given a rough layout graph partitioning a surface into quadrilateral regions (top), our stage 3 creates a quad layout embedding and patch parameterization optimized for low distortion and alignment to principal directions and surface features.

automatic scenarios, it can likewise support the process of manual layouting, which is still very common in the industry, as outlined by Li et al. [LRL06]. Starting from an initial embedding roughly sketched by the user, who can now focus more on the structural aspect and less on geometric precision, our method takes on the process of meticulously positioning the layout's nodes and routing its arcs across the surface so as to achieve low overall patch distortion. Figure 6.1 shows an example input layout graph with rough embedding and the result of our optimization. This is in contrast to simpler aids which operate in an isolated manner, like automatically straightening jaggy arcs to geodesics [LRL06], neglecting the complex consequences for patch distortion. The potential problems are illustrated in Figure 6.2 on the layout from Figure 6.1. Hence, our method takes an integrated, global approach, that takes the interdependencies between adjacent arcs and patches into account. It is based on optimizing a *global* parameterization of the surface, structurally constrained by the layout topology. This parameterization can be seen as the union of the individual node, arc, and patch maps f (with suitable transition functions) which specify the layout's embedding as described in Section 2.1. They are thus optimized in a combined manner. In the end the individual embedding maps can be extracted from the global parameterization.

The most important, essential difference of our method compared to previous approaches to layout embedding optimization is the fact that our formulation takes alignment to principal curvature directions into account (cf. Figures 6.1 and 6.11). In Section 2.2 we already described the importance of this aspect.

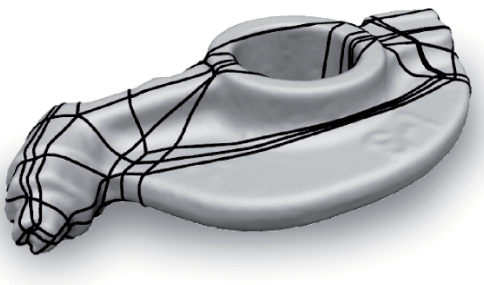


Figure 6.2.: The input layout graph from Figure 6.1 with geodesic arcs. Such isolated per-arc optimization can clearly lead to problems.

6.1. Overview

Our method takes as input a quad layout graph with an initial embedding on a surface. The requirements on the initial embedding's quality are very weak: from the arcs' embedding we only derive topological properties of the layout, i.e. their routes over the surface are not crucial, only their homotopy, as detailed in Section 5.2.3.

Guiding Field (Section 6.3) In order to achieve alignment to principal curvature directions (where reliable), we build upon *field-guided parameterization*. In order to also support the case where the input layout comes from other sources than our stages 1 and 2 described in the last chapters, we do not rely on reusing the cross field used in these stages, but describe how to compute one which is compatible with the input layout, independent of its origin. We examine the structure of the input layout graph and build a topologically compatible space of cross fields on the surface. In this space we then find a smooth cross field which aligns to specified directions.

Aligned Parameterization (Section 6.4) Based on this we globally parameterize the surface subject to connectivity constraints that enforce the given layout structure, such that an optimized embedding for the layout's arcs and patch interiors could be extracted from the parameterization – layout nodes, however, so far remain fixed.

Node Optimization (Section 6.5) A meta-optimization strategy then optimizes the embedding of the layout's nodes: these are relocated based on the gradient of the parameterization's objective functional with respect to their positions, so as to arrive at a local optimum of global embedding quality. We describe how this repositioning can be performed continuously, not restricting node positions by the underlying triangulation. It is demonstrated that this concept is beneficial for general quad meshing applications, too.

Gradient Computation (Section 6.6) Compelled by the complexity of the objective functional's true gradient, we describe a fast, easy-to-implement estimator and demonstrate its effectiveness.

In Figure 6.3 these stages of our algorithm are illustrated.

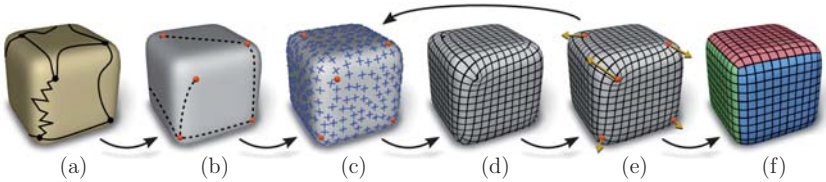


Figure 6.3.: Given a rough (manually or automatically generated) sketch of a layout with quadrilateral patches (a), the space of topologically compatible cross fields with suitable singularities is determined (b). Based on reliable principal curvature directions (and possibly feature information) a smooth, interpolating cross field is then created (c). Guided by this field and constrained by the layout connectivity, an aligned global parameterization is generated (d). After optimization of the layout node positions by a non-linear gradient descent strategy (e), the optimized embedding for the layout can be extracted, including smooth patch parameterizations (f).

6.2. Related Work

So far, the specific problem of optimizing and parameterizing a given layout while considering alignment to principal directions has not been addressed in the literature. There are, however, several related works which address either layout parameterization *without* alignment or aligned parameterization *without* underlying layout structure.

Aligned Parameterization

Numerous methods for global surface parameterization have been presented [FH05]. Most related to our work are more recent methods that aim for alignment (of iso-parametric curves) to specified directions on the surface. Ray et al. [RLL*06] introduced a method from this class, which minimizes the deviation between the parameterization functions' gradients and a cross field representing principal directions. The same goal can be achieved based on a Hodge-type decomposition of the field [KNP07].

Furthermore, there are concepts for parameterization-based quad meshing; we refer to a recent survey [BLP*13]. They are related in that they obtain an (aligned) parameterization with conforming quadrilateral patches, but are best suited for the generation of

fine quad meshes (except for, to some extent, our recent [BCE*13]) and, even more important, the structure is a product of the algorithm itself. We strive for parameterization with a *predetermined*, potentially *coarse* layout structure.

Layout Parameterization

While already some time ago approaches to automatic quad layout generation have been made [EH96, BMRJ04, DSC09], quad layouting is still often performed manually by skilled professionals in order to inject the subtle domain-specific requirements [LRL06]. This process involves positioning nodes and connecting them using paths across the surface, thus specifying the layout graph's structure and embedding through delineation of its patch boundaries [AAB*88, MBVW95, KL96, TPSHSH13].

To at least alleviate the burden of having to tediously specify nice arcs which form patches that can be parameterized with low distortion, several methods for layout parameterization [TACSD06, DBG*06, BVK08] allow for the optimization of the layout's arcs' embedding during their parameterization process. More problematic are the nodes: suboptimal placement not only causes unnecessarily high mapping distortion, it can also give rise to local non-injectivity.

For improvement, node relocation based on iterated relaxation of local neighborhoods can be used – for simplicial [GVSS00, PSS01, KLS03, SAPH04, KS04, PTC10] as well as quad layouts [DBG*06, THCM04, TPP*11, CBK12]. While this approach is sufficient on very smooth surfaces, anisotropically curved regions certainly call for explicit alignment of the parameterization with principal directions as already detailed in Section 2.2. A comparison of alignment-oblivious and alignment-aware layout optimization is provided in Section 6.8.1. Note that it is not straightforward to exchange the functionals used by these local relaxation strategies for alignment-aware ones. For reasons of computational tractability, auxiliary constructs like guiding fields are necessary to allow for the efficient formulation of functionals for aligned parameterization [BLP*13], which the abovementioned frameworks are unprovided for. Hence, the quest for alignment-aware quad layout embeddings calls for a novel strategy.

In addition to explicitly addressing this, our method further handles (aligned and unaligned) surface boundaries and can deal with certain partially specified layouts. This is not immediately possible with most of the above methods.

6.3. Guiding Field

In order to obtain a guiding field for the subsequent parameterization step, we construct a smooth cross field $\mathcal{C}$ on $\mathcal{M}$ that topologically conforms with the quad layout $\mathcal{L}$ (Section 6.3.1) and geometrically follows principal directions and sharp features of $\mathcal{M}$ (Section 6.3.2).

Let g denote the genus and b the number of boundaries of $\mathcal{M}$. Further, s is the number of irregular (valence $\neq 4$) nodes in G.

6.3.1. Guiding Field Topology

For each irregular node of $\mathcal{L}$ we need one irregular point (*singularity*) in $\mathcal{C}$ at the position of the node. The space of cross fields with these singularities has $2g+b+s-1$ topological degrees of freedom [RVLL08] which we need to fix in order to restrict to cross fields topologically compatible with $\mathcal{L}$, otherwise no non-degenerate parameterization will be possible. The degrees of freedom are the *turning numbers* of $\mathcal{C}$ along $2g$ homology generator cycles, b boundary cycles, and around s singularities – minus one, which depends on the others via the Poincaré-Hopf theorem.

In the discrete setting one specifies the turning numbers for cycles of faces (or equivalently dual edges) of $\mathcal{M}$. For an irregular node h, the turning number t necessary for the face cycle clockwise around $\nu(h)$ (cf. Figure 6.4 left) is simply determined from the valence $\mathrm{val}(h)$ of h as $t = -\frac{1}{4}\mathrm{val}(h)$. For nodes on boundary vertices no turning number needs to be prescribed.

A set of $2g$ homology generator cycles of $\mathcal{L}$ can easily be computed using the homotopy basis construction algorithm for combinatorial surfaces described in [EW05]: A spanning tree T of the layout graph G and a spanning tree T^* of the dual graph which does not cross T are computed. $2g$ edges of G will then not be contained in T nor crossed by edges of T^*. Connecting these $2g$ edges to the root vertex of T through T yields the $2g$ arc cycles. For a cycle of arcs $c = a_1 a_2 \cdots a_n$ (with consistent orientation of the a_i) we concatenate the corresponding face chains $\gamma(a_i)$ (red in Figure 6.4), while inserting clockwise face fans (blue in Figure 6.4) in between (i.e. at the nodes) to make the resulting face cycle $\gamma(c)$ continuous. As the combinatorial surface $\mathcal{L}$ is homotopic to $\mathcal{M}$,

these face cycles are homology generator cycles of $\mathcal{M}$. We count the number n_a of arcs (emanating from the n involved nodes) this cycle crosses. The turning number of the cycle then is fixed to $t = \frac{1}{4}(n - n_a)$; in Figure 6.4 bottom left the depicted face cycle c crosses 2 arcs (dashed) which emanate from the 2 involved nodes, thus $t = 0$ in this case.

For each boundary loop d of $\mathcal{M}$ we find the cycle of arcs $c = a_1 a_2 \cdots a_n$ closest to d, i.e., such that no node lies between c and d, and choose its orientation such that d lies right of c when traveling along c (in order to avoid the need for special case handling when nodes of the cycle lie on the boundary). We then determine the turning number for $\gamma(c)$ as in the previous case. Figure 6.4 right illustrates this for the gray mesh boundary loop d: $t = \frac{1}{4}(4 - 4) = 0$ in this case.

These turning numbers implicitly define the cross field's topology. It can explicitly be expressed using *period jumps* on $\mathcal{M}$'s edges. Ray et al. [RVLL08] present a zipping algorithm (Crane et al. [CDS10] an alternative formulation) that, given the above turning numbers, determines all period jumps accordingly. Exactly the requested cross field topology (in particular no additional singularities) arises from this algorithm. Setting period jumps along the initial arcs according to their continuity type [TACSD06] is an

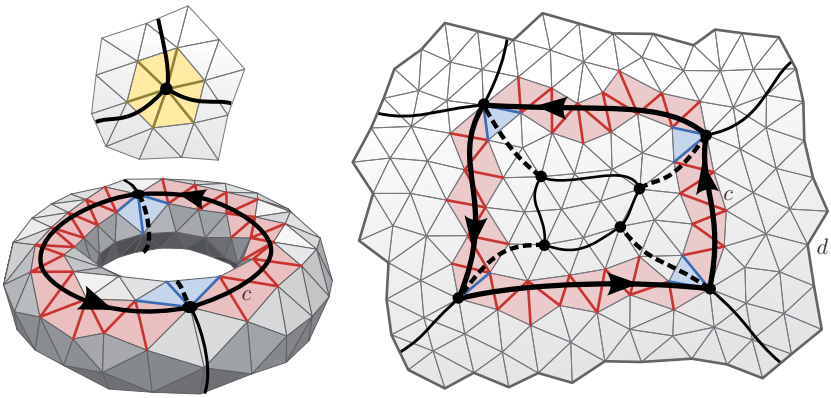


Figure 6.4.: Visualization of the face cycles used for fixing the *turning numbers* of singularities (top left), homology generators (bottom left), and of boundaries of the mesh (right).

equivalent alternative (which, however, only works for complete layouts, not partial ones, cf. Section 6.9).

6.3.2. Guiding Field Smoothness

Within this topologically fixed space we now strive to find a smooth $\mathcal{C}$ that interpolates sparse directional constraints, corresponding to reliable principal directions, feature curve directions, or user-specified design intents. Like in Chapter 4 we use the strategy described by Bommes et al. [BZK09] for determining regions with reliable principal directions.

It is not inherently clear by which of $\mathcal{C}$'s four directions a directional constraint is to be interpolated. If this information is available, as could be for user-specified constraints, the smoothest interpolating cross field (i.e. the one with minimal discrete field curvature energy [RVLL08]) is obtained by solving a simple linear system as described by [RVLL08] (which we modify to use soft constraints [RVAL09], with a high penalty factor of 100 which, while mostly achieving accurate alignment to constraint directions, provides some freedom around singularities). Otherwise, we have to do the assignment of the field to the constraints "modulo rotation by multiples of $\frac{\pi}{2}$", which is easily expressed using additional integer variables in the system. A mixed-integer solver is then initially used to obtain a suitable assignment.

6.4. Aligned Parameterization

Now we formulate a global parameterization problem to simultaneously optimize the embedding maps f_a and f_p (cf. Section 2.1) of arcs and patches guided by $\mathcal{C}$ – the optimization of the nodes' embedding is described in Section 6.5. The parameterization $\mathcal{P} = (u, v)$ is represented piecewise linearly using three (u, v) parameter tuples per triangle – one for each corner. Let $\mathbf{u}_t$ and $\mathbf{v}_t$ denote the orthogonal unit tangent vectors in triangle t which correspond to the first and second direction of the cross field $\mathcal{C}$ in t. The objective functional [BZK09] we use to obtain $\mathcal{P}$ is

$$E = \sum_{t \in T} \Big(||\nabla_t u - \mathbf{u}_t||^2 + ||\nabla_t v - \mathbf{v}_t||^2 \Big) A_t \;\rightarrow\; \min \tag{6.1}$$

where T are $\mathcal{M}$'s triangles, A_t the area of t, and $(\nabla_t u, \nabla_t v)$ the (per-triangle) gradients of the sought parameterization. The per-triangle parameterizations are interlinked via transitions, which we require to be rigid transformations. Across an edge between triangles s and t we thus have a constraint

$$(u, v)_t = R(r_{st})(u, v)_s + (j_{st}, k_{st}) \tag{6.2}$$

with a rotation R by angle r_{st} and a translation (j_{st}, k_{st}). The rotation angles r are deduced a priori directly from the period jumps [KNP07], naturally as multiples of $\frac{\pi}{2}$. The transitions (6.2) with fixed r and variables (j, k) are then incorporated as linear constraints into (6.1) using elimination of variables.

Note that in contrast to related quad meshing methods it is unnecessary to impose integer constraints, which require mixed integer optimization strategies, neither on (j, k) nor on the singularity parameters. Hence the parameterization $\mathcal{P}$ can be obtained using a single linear system solve.

A parameterization of this kind induces a *base complex*, which can be extracted by tracing iso-parametric curves (*separatrices*) from the nodes (cf. Figure 6.5). With *iso-parametric* we mean that either the u or v parameter is constant along the curve when taking transitions into account. We now further constrain the parameterization problem (6.1) using *node connection constraints*, derived from the structure of the layout, to accomplish that this induced base complex is structurally equivalent to $\mathcal{L}$ (cf. Figure 6.5 right). This ultimately allows us to derive an embedding for $\mathcal{L}$ from $\mathcal{P}$.

6.4.1. Node Connection Constraints

We have to ensure for each arc that the two incident nodes lie on a common iso-parametric curve of $\mathcal{P}$ – the restriction of $\mathcal{P}$ to this curve then gives us the arc embedding map f_a. For this we need to consider the transition functions along the face sequence $\gamma(a)$ representing an arc a. Let $\gamma(a)_\vdash$ denote the first face in the sequence, $\gamma(a)_\dashv$ the last one. Further, let $\gamma'(a)$ be the sequence of edges in between the faces of $\gamma(a)$. Let $\tau(a)$ be the concatenation of transition functions of the edges in $\gamma'(a)$, i.e. $\tau(a)$ maps from the patch $s = \gamma(a)_\vdash$ to the patch $t = \gamma(a)_\dashv$ along arc a. Then we need to require

$$\big[\tau(a)(u, v)_s\big]_{\lambda(a)} = \big[(u, v)_t\big]_{\lambda(a)} \tag{6.3}$$

for either $\lambda(a) = u$ or $\lambda(a) = v$, where this subscript extracts the u- or v-component of the tuple. This can be incorporated into (6.1) as linear constraint [MPKZ10]; we only need to determine the *arc labeling* λ, i.e. whether an iso-u or an iso-v constraint should be used for an arc.

We could make this decision by rating an arc's alignment to the u- and v-directions in an unconstrained parameterization [MPKZ10] or based on local angle considerations [TACSD06]. While this can be sufficient locally for single arcs, we need to setup constraints for all arcs. If even just a single decision is inconsistent with the others, the constrained problem would admit only degenerate solutions.

Consistent Arc Labeling

In order to ensure global consistency, we do not consider each arc individually, but first construct a complete, consistent prototypic $\{\bar{u}, \bar{v}\}$ labeling $\bar{\lambda}$ of the arcs based solely on the combinatorial structure of $\mathcal{L}$. Then only one global geometric decision is necessary: check whether the total alignment of all $\bar{u}$-arcs to the **u**-direction and $\bar{v}$-arcs to the **v**-direction of $\mathcal{C}$ is better than the opposite choice, and exchange the prototypic labels for $\{u, v\}$ labels λ accordingly.

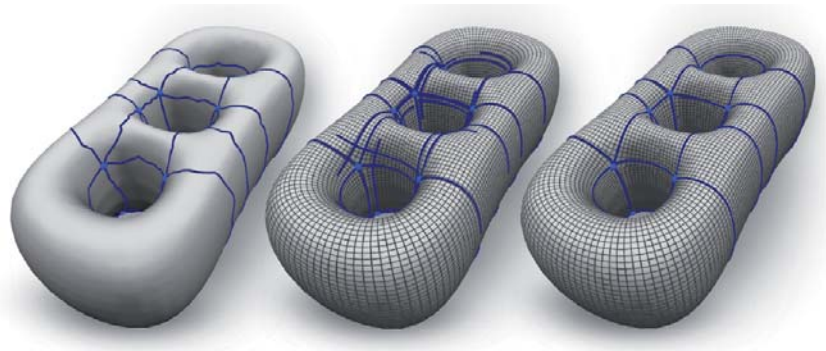


Figure 6.5.: Left: input layout. Middle: intermediate state of tracing iso-parameter curves from the singularities in an unconstrained parameterization to obtain the base complex. Right: base complex of a parameterization with node connection constraints; now structurally equivalent to the input.

The prototypic arc labeling is a map $\bar{\lambda} : H \to \{\bar{u}, \bar{v}\}$ which assigns symbols $\bar{u}$ and $\bar{v}$ to the set H of half-arcs. The idea is to basically assign the same label to both half-arcs of an arc (each representing one end of the arc) and alternating labels to half-arcs incident to the same node in cyclic order. Here the notions of "same" and "alternating" are again meant to take transitions into account. This is formalized as follows for a half-arc a using the rotation system (σ, θ) of $\mathcal{L}$:

$$\bar{\lambda}(\sigma(a)) = \tau_\sigma(a)\,\bar{\lambda}(a) \tag{6.4}$$

$$\bar{\lambda}(\theta(a)) = \mathrm{Rot}(\pi/2)\,\tau_\theta(a)\,\bar{\lambda}(a). \tag{6.5}$$

where τ_σ and τ_θ are the respective transitions: let a^+ and a^- be the two half-arcs of an arc a, a^+ attached to the node at a's beginning, a^- attached to the end node. $a^\times$ denotes an arbitrary half-arc. Let $\tau_\sigma(a^+) = \tau(a)$, i.e. the concatenation of transition functions of the edges in $\gamma'(a)$, and $\tau_\sigma(a^-) = \tau(-a)$. Let $\tau_\theta(a^\times)$ be the concatenation of the transition functions of the edges between faces $\gamma(a^\times)_\vdash$ and $\gamma(\theta(a^\times))_\vdash$ in clockwise order around the incident vertex. We let these transitions act on the symbols $\bar{u}$ and $\bar{v}$ in the intuitive manner: the symbols are mapped to themselves if the rotational part is an even multiple of $\frac{\pi}{2}$, and to each other if it is an odd multiple. The translational part is ignored.

When we now assign $\bar{\lambda}(a) = \bar{u}$ to one half-arc a (or one half-arc per component if G is not connected), the labeling can be extended to all half-arcs by recursive application of equations (6.4) and (6.5). It is due to the intimate connection between the turning numbers of $\mathcal{L}$, the period jumps of C, the transition functions, and τ_σ / τ_θ (which are all built upon each other) that the resulting labeling is independent of the label propagation order, thus globally consistent with respect to (6.4) and (6.5).

With constraints (6.3) in effect, we then compute $\mathcal{P}$ (6.1) whose base complex is now structurally equivalent to $\mathcal{L}$ (cf. Figure 6.5 right). Notice that using this particular setup (global parameterization with layout structure constraints) optimal parametric extents of the individual patches emerge freely. This is in contrast to all previous layout parameterization approaches (cf. Section 6.2), which rely on fixed (unit) patch extents or initial estimates, followed by an iterative adjustment procedure. In particular, we thus do not depend on the quality of the arcs' initial embedding; it is only the arcs' path homotopy classes (with respect to the surface punctured at the singularities) that matters [MPKZ10].

6.4.2. Embedding Extraction

A global parameterization $\mathcal{P}$ with a base complex structurally equivalent to $\mathcal{L}$ naturally induces an embedding for $\mathcal{L}$ (cf. Figure 6.5 right). The embedded arcs are found as iso-parametric curves starting from the nodes until another node is reached.

The individual patch parameterizations, i.e. the maps f_p, over rectangular domains $[0, w_p] \times [0, h_p]$ can be derived easily: if no non-trivial transitions lie in the patch region it can directly be read from $\mathcal{P}$ as the surface between the four bounding iso-parametric curves emanating from the corner nodes is parameterized over some rectangle $[a, b] \times [c, d]$ by $\mathcal{P}$, thus only a translation and possibly a scaling is necessary. If transitions are involved, we need to express all parameter values with respect to a common chart – as each patch is disc-homeomorphic and contains no singularities in its interior, this is possible without ambiguity. We start from an arbitrary triangle *seed* lying within the patch and from there conquer all others while transferring them to *seed*'s parameter system. Let f_{st} be the transition from face s to face t, Id the identity transformation, s.uv the set of (u, v) parameters of the corners of s, and Q a simple FIFO queue.

Q.push([*seed*, Id])
while not Q.empty
  $[t, f] \leftarrow Q$.pop()
  for $s \in T \mid$ adjacent to t **and** not yet processed
    $g \leftarrow f \circ f_{st}$
    s.uv $\leftarrow g(s$.uv$)$
    Q.push($[s, g]$)

Note that if instead of individual patch parameterizations one *global* parameterization without transitions within the patches is desired (e.g. for visualization purposes), the transitions must necessarily be located at the patch borders. This is achieved by making the mesh conform with the layout by splitting the faces crossed during the arc tracing, effectively inserting edge strips that coincide with the arcs. Then performing the above procedure for one seed per patch directly results in all transitions getting shifted to the patch borders.

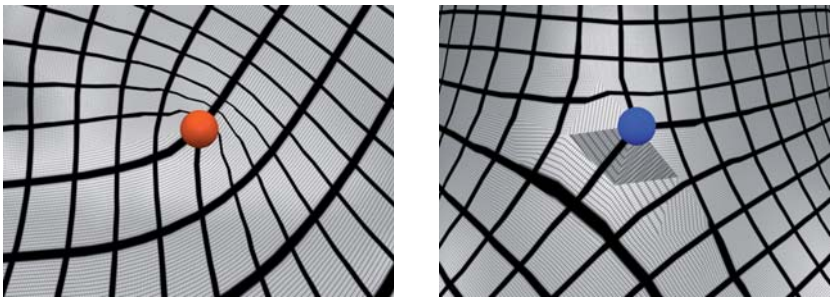


Figure 6.6.: Typical parameterization distortions due to suboptimal node singularity positions. Most problematic are non-injectivities due to fold-overs, i.e. triangles parameterized with reverse orientation (shaded darker). Intuition already suggests that moving the nodes towards the bottom left would improve parameterization quality.

6.4.3. Optional Extensions

We would like to point out that the functional (6.1) can be extended in several ways. For instance, an anisotropic norm can be used for improved field alignment [BZK09] – we generally use an anisotropy ratio of 10. Furthermore, a sizing field, computed so as to reduce the curl of the sized cross field to allow for better alignment [RLL*06], can be taken into account – we used this option for all examples.

6.5. Node Optimization

While the described constrained parameterization procedure optimizes the embedding of arcs and patches, the nodes remain fixed due to the very nature of the setup. This not only restrains the achievable quality of the resulting embedding, it further gives rise to large distortions or even local non-injectivities due to fold-overs, i.e. triangles parameterized with inverse orientation (cf. Figures 6.6, 6.7, and 6.9). This is because fixed nodes in our setup behave much like isolated point constraints in, e.g., harmonic parameterizations – which are well known for their problematic effects on distortion and injectivity [YLY*12].

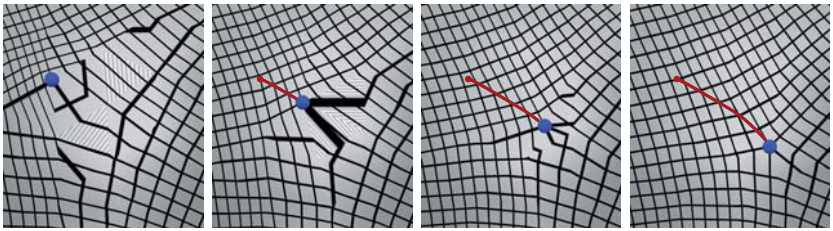


Figure 6.7.: Visualization of the trajectory (red) of a node (blue) as it moves in negative gradient direction $\mathbf{d}$. In the beginning severe distortions and inversions are present which successively vanish in the course of the movement.

A popular approach to this problem is the use of *stiffening* [BZK09, MPKZ10]. It tries to iteratively remove non-injectivities by increasing a penalty factor for the affected regions in the objective functional. While this reduces the problematic effects, it does this at the cost of actually increasing the residual, i.e. the parameterization is pushed away from the least-squares solution deemed optimal by (6.1) – higher overall distortion is traded for local improvements (cf. Section 6.5.3). We advocate a strategy that instead moves the nodes so as to arrive at a solution with *lower* residual. Thus, while taking care of the distortion problems, this strategy simultaneously optimizes the nodes' embedding, basically by exploiting the information the distortion provides. Figure 6.7 demonstrates this proposition's reasonability. This basic idea of node relocation has been made use of in previous methods – however, these are not suited for our setting using an aligned parameterization as detailed in Section 6.2.

Technically speaking, we are going to optimize (6.1) not only w.r.t. the parameters u and v (thus the patches' and arcs' embedding) but also w.r.t. the geometric positions of nodes on $\mathcal{M}$. We tackle this non-linear problem using a strategy which optimizes A) with respect to u and v (with fixed nodes) and B) with respect to the node positions (with fixed u,v) in an *alternating* manner. In this way the large problem A ($O(|\mathcal{M}|)$ variables) can still be solved as a simple linear problem as described in the previous sections. The smaller non-linear problem B ($O(|\mathcal{L}|)$ variables) we address using a gradient descent strategy, as detailed in the following.

6.5.1. Gradient Descent

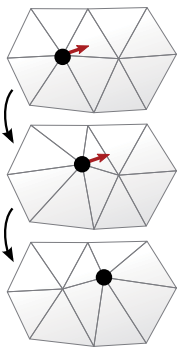
In order to locally move a node h in such a way that the residual of (6.1) (which we now just call E for brevity) decreases, we need to move this node in direction

$$\mathbf{d}(h) = -\left(\frac{\partial}{\partial x} E(h_x, h_y),\ \frac{\partial}{\partial y} E(h_x, h_y) \right) \tag{6.6}$$

i.e. in a gradient descent manner. Here (x, y) are 2D coordinates in some local coordinate chart of $\mathcal{M}$ around h and (h_x, h_y) expresses the current position of node h accordingly. Note that E depends on x and y because nodes are embedded in vertices, i.e. node positions are vertex positions. In Section 6.6 we address the computation of $\mathbf{d}(h)$ as well as of a suitable corresponding descent step length $l(h)$ in detail.

Node Movement

Assume node h is currently at position p. To determine the new position p' on $\mathcal{M}$ we trace a straightest geodesic g of length $l(h)$ starting at p in direction $\mathbf{d}(h)$ [PS98] (stopping if a boundary is reached). If h lies on a feature curve, we first project $\mathbf{d}(h)$ onto it and restrict the tracing to this curve.

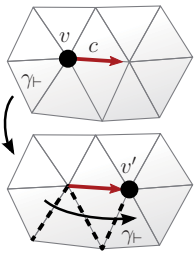


Remember that nodes need to be mapped to vertices, but p' does in general not lie on a vertex. Snapping the node to the closest vertex is not a good solution as it provokes discontinuous behavior and hampers convergence. Instead we enable a virtually continuous movement by (temporarily) relocating one of the vertices incident to the face on which p' lies, so as to provide a suitable support for h. In contrast to the insertion of a new vertex this does not introduce low valence vertices and leaves the mesh structure unchanged. The choice among the incident vertices is made such that geometric alteration of $\mathcal{M}$ caused by the relocation is minimal. Vertices that have another node embedded in them and vertices on sharp features are excluded from the choice.

Let v be the vertex at position p onto which h is mapped before the move and v' be the vertex chosen to be relocated to p'. If $v = v'$, we simply relocate it to p'. If $v \neq v'$, the original position of v is restored and v' moved to p'. In this case further adjustment

is necessary: the cross field singularity corresponding to the node needs to be moved to v'. In detail, we determine an edge chain c connecting v with v' along the geodesic g. We then adjust the chain's edges' period jumps so as to reflect the new singularity location.

Furthermore, the connection constraints (6.3), i.e. the composite transitions τ and labels λ, have to be updated so as to reflect the new situation. For simplicity we assume all arcs incident to a node start in the same face, i.e. $\gamma_{\vdash}$ of all incident arcs (oriented in outgoing direction) is the same. To now update the current τ and λ we select an arbitrary face incident to v' as new $\gamma_{\vdash}$, accumulate the transitions of the edges (dashed in the inset figure) crossed when going from the old to the new face along the edge chain c, and apply them to τ and λ.



Iteration Strategy

We solve problem A (optimizing cross field and parameterization), determine gradient and step length for each node, and move them one step as described in the previous section. This is iterated. We stop when the next step would increase the residual, i.e. we execute the step, and output the previous solution in case the residual increased. Further fine-tuning of the node positions can be achieved by repeating this with decreased step size. We used this option for the shown examples, halving the step size each time and stopping after max. 5 halvings or 25 total steps.

6.5.2. Selective Optimization

At the user's discretion (or based on additional information) we can selectively exclude nodes as well as arcs from the automatic optimization process to keep them in their intended original state. For a node this is as easy as disregarding it in the gradient descent. For an arc we need to ensure that the corresponding iso-parametric curve coincides. We achieve this using constraints similar to the node connection constraints: we do not just require the arc's end points to lie on a common iso-parametric curve, but also all the points where the arc crosses mesh edges. The parameters of these

crossing points are easily expressed as linear (convex) combinations of the parameters of the edges' incident vertices. In a completely analogous manner the parameterization can also be forced to align to given feature curves, surface boundaries, or other user-specified curves on the mesh.

6.5.3. Discussion

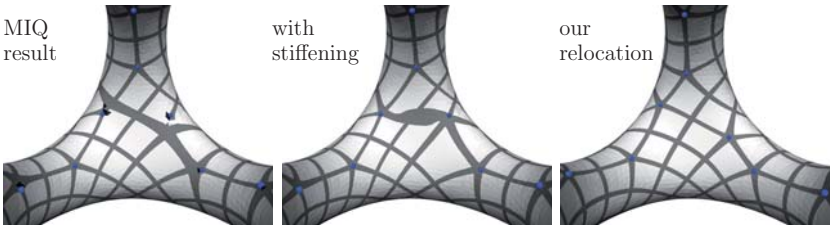
The key features of our node relocation approach are the largely triangulation-invariant movement and continuous positioning of nodes. This is in contrast to the discrete singularity relocation mentioned by Bommes et al. [BZK09] which has the "obvious drawback of heavy computational cost" and can get stuck in situations where all possible moves to neighbor vertices lead to a higher residual although a continuous path of decreasing residual exists inbetween. Both aspects can be seen in the following table comparing runtime t and final residual E_{final} (after max 1h) of both relocation methods applied to examples from Figure 6.10.

	Ours		Bommes et al.	
Model	t (s)	E_{final}	t (s)	E_{final}
Trihole	24	430	742	457
Rockerarm	75	842	>3600	1099
Elk	18	751	>3600	762
Fertility	40	522	>3600	648
Block	42	140	>3600	182

Nieser [Nie12] proposes a variant that efficiently estimates parameterization improvement based on the cross field's curl improvement. Note that this is not appropriate for our highly constrained problem where a major part of the residual is caused by the constraints, not the field's curl.

Notice that our relocation strategy does not rely on a layout – it can likewise be used for *singularity relocation* in general quad meshing scenarios based on energy (6.1). In this context singularity locations are typically determined a priori [KNP07, BZK09], with only limited awareness of implications for the parameterization. Figure 6.8 demonstrates the advantage of relocating them (directly driven by parameterization quality) over the alternative of using stiffening.

Limitations of Node Movement No matter which node relocation strategy is used, potentially some fold-overs (i.e. non-injectivities) remain after convergence, especially when the layout is coarse and ignores significant geometric features, or when nodes or



$E = 1112$, 55 folds $E = 1289$, 0 folds $E = 507$, 0 folds

Figure 6.8.: Left: Parameterization computed using the MIQ approach [BZK09]. The residual is 1112 and there are 55 fold-over triangles. Middle: Stiffening is able to get rid of the folds while *increasing* the residual by 16%. Right: Our relocation is likewise able to get rid of the folds while beneficially *decreasing* the residual by 54%.

arcs are fixed by the user in bad positions. Subsequent stiffening proved to often be a helpful remedy in such case. A more reliable solution, however, is to use strict triangle orientation constraints, e.g. as in [Lip12] or [BCE*13]. We employ the latters' linear tri-sector constraints to get rid of folds in the last iteration.

6.6. Gradient Computation

The gradient (6.6) of (6.1) depends on x and y in multiple ways: they appear directly in the discrete gradient operator ∇ and in A_t, but indirectly also in the cross field, i.e. in $\mathbf{u}$ and $\mathbf{v}$, as well as in the parameterization (u, v). As these dependencies are via the solution of (constrained) minimization problems, a closed-form expression of gradient $\mathbf{d}(h)$ is not available. One possibility to compute (or approximate) it is to use numerical differentiation, e.g. using finite differences: $\frac{\partial}{\partial x} E(h_x, h_y) \approx (E(h_x + \varepsilon, h_y) - E(h_x, h_y))/\varepsilon$. This amounts to moving vertex $\nu(h)$ by a small ε, re-solving the cross field and parameterization systems, and evaluating the residual. While simple, this needs to be done two times (∂x and ∂y) per node in each step, thus clearly is a costly procedure. Another option is to use exact *algorithmic differentiation*. Exploiting recent results which allow for the efficient differentiation of functions involving systems of linear equations [NL12], we are able to handle our E. This technique only requires the equation

systems to be solved two times per step (once normal, once adjoint) yielding the gradient w.r.t. all nodes at once. Implementation, however, is relatively demanding.

As practical alternative we devised a gradient estimator, described in the following, which is both, easy to implement and very efficient, avoiding additional system solves altogether.

6.6.1. Efficient Estimator

Let's consider the energy functional (6.1) again. The position (x, y) of a node appears directly in ∇_t and A_t, but also in the cross field and the parameterization. If we neglect these indirect dependencies, i.e. consider $\tilde{E}$ where $\mathbf{u}$ and $\mathbf{v}$ as well as (u, v) are fixed, we can analytically derive

$$\frac{\partial}{\partial x}\tilde{E} = \sum_{t \in T(h)} 2\left(\frac{\partial \nabla_t}{\partial x} u\right)^{\mathsf{T}} \left(\nabla_t u - \mathbf{u}_t\right) A_t + \frac{\partial A_t}{\partial x} ||\nabla_t u - \mathbf{u}_t||^2 + \cdots$$

where we omitted the second analogous half. Note that the sum is only over triangles $T(h)$ incident to node h, as all other terms vanish due to independence from x. The corresponding approximate gradient $\tilde{\mathbf{d}}(h)$ can thus very efficiently be evaluated based on only the 1-ring neighborhood.

To obtain a local 2D (x, y) coordinate system for h's 1-ring, we employ the commonly used geodesic polar map [WW94], effectively flattening the 1-ring to the plane while preserving radial lengths and relative angles by uniformly scaling the inner angles incident to h such that they sum to 2π; origin and axes in the plane can be chosen arbitrarily.

The approximate gradient $\tilde{\mathbf{d}}(h)$ is surprisingly well-behaved in terms of gradient direction, even in configurations with strong distortions and inversions where one could easily expect the local per-triangle gradients to be severely perturbed and non-informative. Tests on several hundreds of nodes in numerous layouts showed that the average angular deviation between $\mathbf{d}$ and $\tilde{\mathbf{d}}$ is around 4°, with very few outliers (typically rather small gradients) which showed deviations up to 35°. Figure 6.9 gives an impression of the amount of deviation. We observed the magnitude of $\tilde{\mathbf{d}}$ to be around 1.5 times larger than that of $\mathbf{d}$ (caused by neglecting the dependence of the parameterization on (x, y)). The average magnitude deviation between $\frac{2}{3}\tilde{\mathbf{d}}$ and $\mathbf{d}$ was only 6% in our tests.

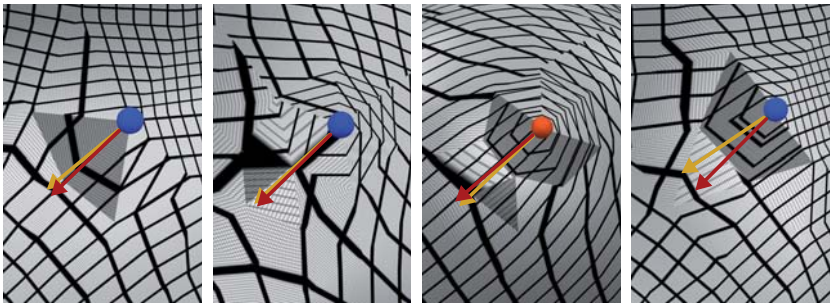
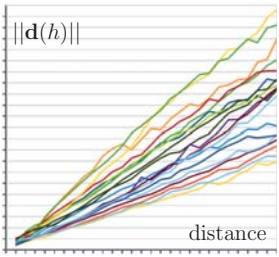


Figure 6.9.: Visualization of descent directions: negative gradient $\mathbf{d}$ (yellow) and our estimator $\tilde{\mathbf{d}}$ (red). The angular deviation is typically very low, even in complicated regions with severe distortions and inversions (shaded darker).

6.6.2. Step Length

In addition to the gradient direction, we also need to determine an appropriate step length for the iterative gradient descent procedure. We empirically found that $E(h_x, h_y)$ grows roughly quadratically with the geodesic distance of node h from the position where $E(h_x, h_y)$ attains a minimum – at least in the range of relevance, i.e. unless we maliciously move h way beyond its adjacent vertices, twisting the layout. This means $||\mathbf{d}(h)||$ is approximately proportional to the distance of h from its locally optimal position – as can be seen in the inset graph for 20 different nodes in an example layout. Note that the proportionality factor between $||\mathbf{d}(h)||$ and the distance is scale independent: if we scale a model by a factor f, its area and the residual E are scaled by f^2 – distances and $\mathbf{d}(h)$ are both scaled by f. Hence we can directly rely on the gradient magnitude to determine a suitable step length.



We observe in the graph that there is some variation among the nodes – different slopes. These are not due to the model's scale or varying local density of the layout, but depend on variations in local surface and layout region shape in a complex manner. As the range of variation is not very large (the graph already shows a rather extreme

case), we can easily account for this using a conservative global damping factor, i.e. we use $l(h) = \alpha||\mathbf{d}(h)||$ (respectively: $\alpha\frac{2}{3}||\tilde{\mathbf{d}}(h)||$) as step length for node h. A value of $\alpha = 0.75$ proved to perform very well in practice – we used it in all the examples. As a safeguard, we limit each node's movement to its current cell in a simple Dijkstra-based node Voronoi diagram on $\mathcal{M}$.

6.7. Quad Mesh Generation

One particular use case of quad layouts is the generation of quad meshes via subsequent refinement, as the block structure of the resulting meshes provides specific advantages [BLP*13]. One option is *a posteriori* subdivision of each patch parameterization into an $m \times n$ grid, where m and n comply between neighboring patches. This is the case if we choose $m = \lceil w/q \rceil$ and $n = \lceil h/q \rceil$, where (w, h) is the patch's parametric extent and q the quad mesh target edge length.

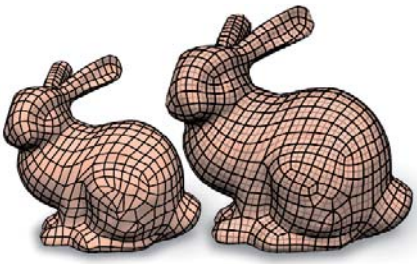
Mixed-Integer Option When rather coarse meshes are to be generated it is advantageous for uniformity to require the parameterization to yield patch sizes (w, h) such that w and h are integer multiples of q (sparing the above rounding) already during the optimization. This is accomplished by requiring the parameters (u, v) of nodes and the translations (j, k) of transitions (6.2) to be integer multiples of q, resulting in a mixed integer problem [BZK09]. It proved advantageous to first optimize node positions in the relaxed (non-integer) setting, then solve for the integers, and then further optimize nodes with respect to the fixed integers. This avoids discontinuities during the gradient descent. Finally, a quad mesh can be extracted from the parameterization [EBCK13].

6.8. Results

We applied the proposed method (using the efficient gradient estimator) to input layouts computed by stage 1 and 2 described in the previous chapters, as well as to some manually sketched layouts (Fertility a, Face, Mask, Elephant, and Botijo). Figure 6.10 shows the input and output layouts, Table 6.1 the corresponding statistics.

The patch parameterizations are visualized using $m \times n$ grid textures (as described in Section 6.7).

We used CHOLMOD [CDHR08] for the equation systems and IPOPT [WB06] to handle the anti-fold constraints. Runtime is dominated by the repeated systems solving – the initial one-time work concerning layout parsing, cross field topology, and connection constraints, as well as the per-step gradient estimation and node movement carry no significant weight. The complexity of the layout has only little influence (cf. Table 6.1, BUNNY), as we move all nodes at once between two solves.



We observed nice convergence properties of the node optimization strategy: in every case more than half of the total decrease in residual happened during the first three steps.

6.8.1. Comparison

Most previous approaches to complete embedding optimization are based on the concept of fixing a node's 1-link and relaxing the interior [GVSS00, PSS01, KLS03, SAPH04, KS04, DBG*06, TPP*11]. We compare the results of an implementation of the most recent one (TPP, the final stage of [TPP*11]) to ours. TPP has some restrictive requirements that prevented us from applying it to all examples, e.g. patches must nowhere be narrow (below 1-2 triangle mesh edge lengths), neither initially nor during the course of the optimization, unaligned boundaries cannot be handled, etc.

Table 6.2 shows the conformal energy $E_{\text{confor}} = \frac{1}{A}\int_{\mathcal{M}} \left(\frac{\sigma_1}{\sigma_2} + \frac{\sigma_2}{\sigma_1}\right)$ [FH05] (A being $\mathcal{M}$'s surface area, σ_1 and σ_2 the singular values of the parameterization's Jacobian), average angular deviation E° between ∇u and ∇v and the principal directions weighted by squared principal curvature difference $(\kappa_1 - \kappa_2)^2$, and conjugacy error $E_{\text{planar}} = \frac{1}{\sqrt{A}}\int_{\mathcal{M}} \text{II}\big(\nabla u/||\nabla u||, \nabla v/||\nabla v||\big)$ which, based on the surface's second fundamental form II [LXW*11], quantifies how non-planar the (infinitesimal) elements of a quad mesh generated from the parameterization would be.

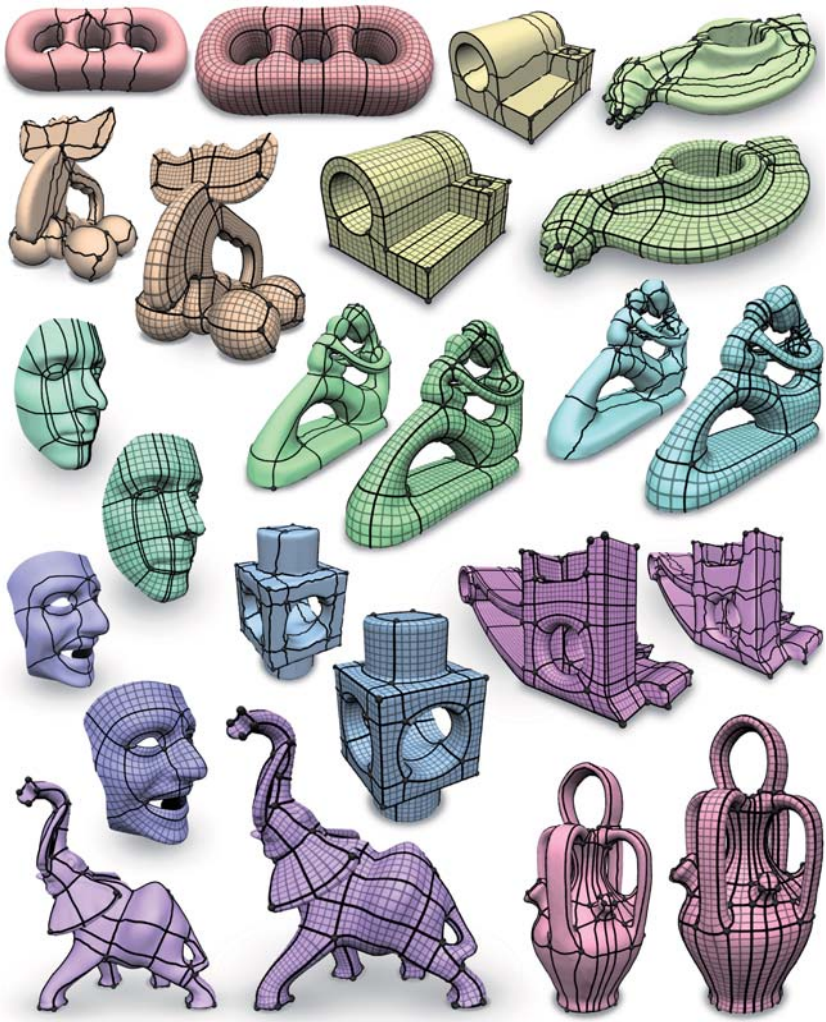


Figure 6.10.: Input and result layouts. Feature alignment has been used for models JOINT (yellow) and LEVER (pink), boundary alignment on eyes and mouth of the MASK model (blue). Irregular nodes are displayed as little spheres for visual orientation.

Model	faces	patches	time (s)	E_{init}	E_{final}	folds*
Trihole	30K	20	24	1087	430	
Elk	18K	86	18	3058	751	1
Joint	43K	79	31	221	167	
Rockerarm	70K	115	75	2669	842	1
Face	25K	50	11	1531	588	
Fertility a	28K	72	40	1889	522	
Fertility b	28K	98	33	4538	662	5
Block	36K	76	42	1523	140	
Mask	9K	30	8	3709	2732	
Elephant	40K	104	69	3336	1398	2
Lever	20K	275	18	2092	1055	2
Botijo	30K	167	37	1946	675	
Bunny	51K	1063	67	8033	3935	

Table 6.1.: Statistics: mesh and layout complexity, total optimization time, residual before and after node optimization (note the significant decrease in each case), *number of fold-over faces if orientation constraints would not have been used; with them (or stiffening) all results are fold-over free.

	Ours			TPP		
Model	E_{confor}	E_{planar}	E°	E_{confor}	E_{planar}	E°
Trihole	2.047	1.40	6.2	2.026	2.12	9.5
Rockerarm	2.069	1.93	5.8	2.085	2.62	8.7
Fertility	2.049	1.57	4.2	2.066	2.57	7.4
Block	2.013	1.24	3.2	2.022	2.39	7.2
Elk	2.065	1.29	4.3	2.106	2.56	8.6
Botijo	2.060	1.57	3.3	2.074	2.72	5.6
Elephant	2.145	2.71	7.4	2.109	4.16	12.0

Table 6.2.: Comparison of the results of our method and TPP of [TPP*11] based on conformal energy E_{confor}, average angular misalignment E°, and conjugacy error E_{planar}.

Note that a "random" parameterization would have E° around 22.5°; as the input layouts have been constructed with principal directions in mind, also TPP shows smaller deviations, but the alignment-awareness of our method consistently led to lowest E° (note that non-integrability of the cross field prevents $E^\circ = 0$ in general). In combination

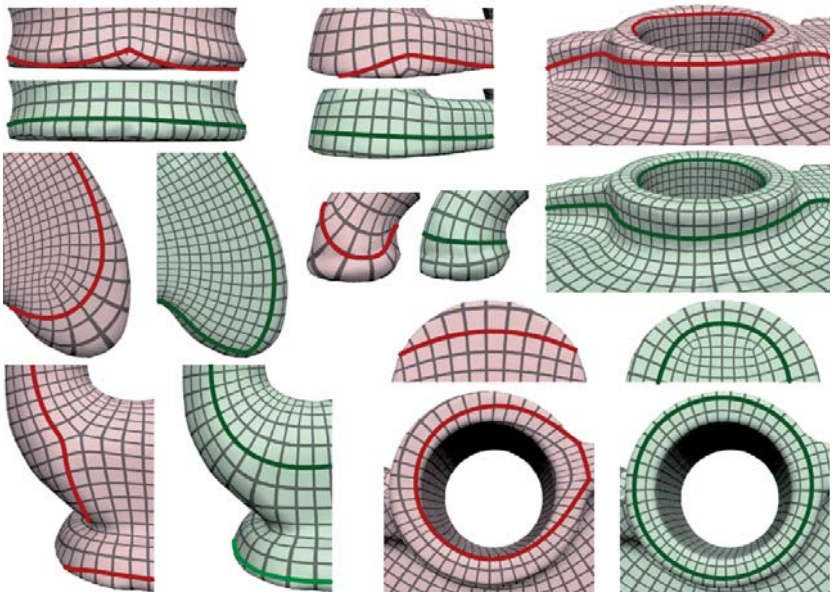


Figure 6.11.: Zoom-ins comparing TPP (red) and our method (green). Some isocurves are highlighted to ease visual alignment quality comparison.

with good conformality (no large difference between both methods) our method also generally achieved better E_{planar} values. Figure 6.11 illustrates the differences.

6.9. Discussion

Our stage 3 simultaneously performs a geometric optimization and parameterization of quad layouts on surfaces. What sets our approach apart from related methods is its unique property of yielding aligned layout parameterizations, taking surface anisotropy and features into account. A key contribution is a novel efficient technique to continuously optimize node positions, directly driven by the objective of minimizing distortion and misalignment. We have seen that its applicability extends to related areas like quadrangular remeshing.

Alignment

The presented method assumes that the input layout is intended for a principal direction aligned embedding. While this clearly is the case when our stages 1 and 2 are used, a designer who manually sketches a layout might arbitrarily (depending on the application intentionally) deviate from that. In such cases our method might not be the ideal choice, as demonstrated in the following.

Principal direction conflict Figure 6.12c/d shows an input layout which largely contradicts the principal directions. The conflict between the objectives of layout structure preservation and alignment leads to large distortions.

High valence nodes Around nodes with high valence (in regions without correspondingly high Gaussian curvature) there are typically quite some distortions. While the valence 8 node in Figure 6.12a is surrounded by a distorted but still valid parameterization, there are fold-overs around the valence 12 node in Figure 6.12b – fold-overs which cannot be prevented even by orientation constraints: the six triangles surrounding the valence 12 node need to span an angle of 6π in parameter space, but in a valid piecewise linear (PL) parameterization all inner angles are less than π. Note that this is a general problem of PL triangle parameterizations [NP09] (however, only a local one, restricted to the 1-ring of singularities).

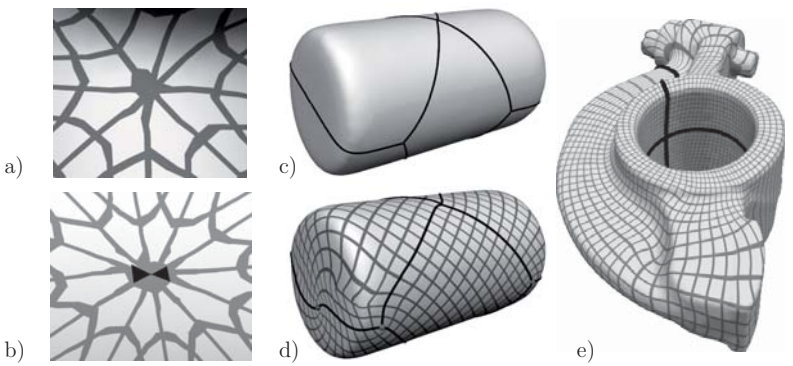


Figure 6.12.: Problematic input configurations: high valence nodes, principal direction conflicts, overly coarse structure.

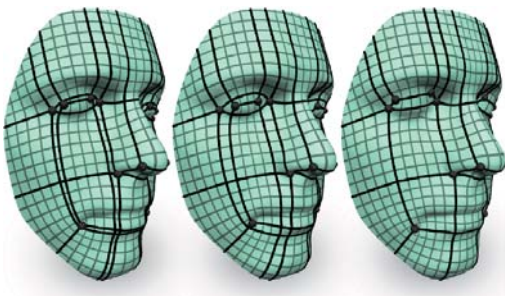
Overly coarse structure The layout in Figure 6.12e is much coarser than the feature structure of the underlying model. This necessarily causes large alignment deviations, thus distortions or even fold-overs. In this example still a valid embedding was found, but there is no general guarantee.

Notice that in these cases it is questionable whether a method aiming for principal direction alignment is the right choice. It seems unlikely that such layouts are actually intended to be aligned, or that there even exists a solution with reasonable alignment. Optimization methods not aiming for alignment (cf. Section 6.2) are then preferable.

Structure Modification

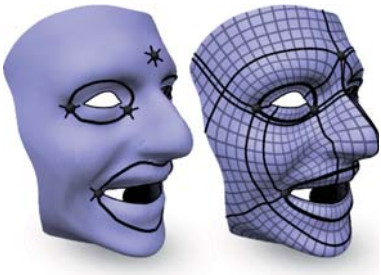
In a few cases we observed adjacent nodes ending up quite close together (cf. the green FACE model). Typically not only individual pairs move towards each other, but rather all pairs bordering a *dual loop* or *poly-chord*. This is plausible, considering that the parametric distance of each such pair is equal. As the resulting narrow patches can be undesirable depending on the application, node spacing constraints – similar to the node connection constraints (6.3) – can be of value. For each arc a from s to t the parametric distance of its end nodes can be computed as $\left|\left[\tau(a)(u,v)_s\right]_i - \left[(u,v)_t\right]_i\right|$ with $i \in \{u,v\}$, $i \neq \lambda(a)$. If this drops below a desired minimal spacing ϵ (e.g. dictated by the quad mesh target edge length) during the gradient descent, one can add a constraint $\left[\tau(a)(u,v)_s\right]_i = \left[(u,v)_t\right]_i \pm \epsilon$ to ensure that this distance is kept in following iterations.

Alternatively, one could understand the narrowing as indication that merging the approaching nodes is advantageous. We can constrain the parametric distance to zero when it drops below ϵ, effectively enforcing a *poly-chord collapse*. Like the node merging in Section 5.1.4 this can be seen as a form of adjustment of the intermediate result from stage 2, as motivated in Section 3.1.5. Both options are illustrated here next to the standard solution (left).



Partial Layouts

It is also worth noting that partial layouts, where only a subset of all arcs is specified (together with irregular nodes and their valences), can be taken as input, too. Missing arcs then naturally emerge from the parameterization (as shown in the inset) and their routes may indicate suitable layout completions. Considering this in conjunction with the fast convergence properties, we imagine using the presented techniques in an assisted (primal) layout drawing system with interactive feedback.



Regular Node Optimization

Let us mention an option concerning the optimization of regular nodes in a layout. While irregular nodes (implying singularities) have an essential influence on the field construction and parameterization system structure, regular nodes do not. Using a simple modification, the optimization of the embedding of regular nodes can thus already be achieved during the parameterization step. We can simply remove a regular node's explicit occurrence in the node connection constraints by instead concatenating the four involved connection constraints in two pairs (unless this concatenates a constraint with itself), just as if the regular node was not present in the layout, but two arcs crossing in its place. The regular nodes' embedding is then optimized *implicitly* in the parameterization process: a node's new position can be found at the intersection of the two respective iso-parametric curves.

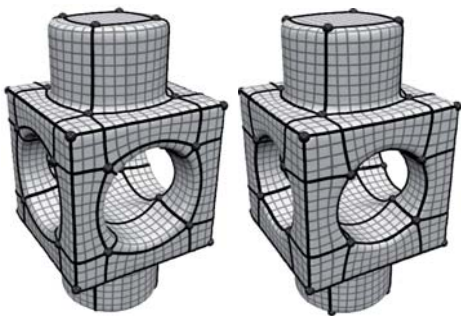
As the number of nodes to be moved in the gradient descent procedure does not significantly affect the total runtime, this option does not necessarily increase efficiency. It can, however, speed up convergence when the initial embedding of regular nodes is worse than that of irregular nodes – which can, for instance, be the case if it is a mere byproduct of the rough initial embedding of crossing arcs – as in our stage 2.

It is important to notice that when this option is used the structure of the resulting layout is not guaranteed to be equivalent to the input. One can think of constellations where the arcs in the final parameterization could cross in another way than they do in the input (except for layouts involving numerous dangling arcs, we have not come across such a situation). This can be seen as a form of implicit structure optimization, but can be undesirable if preservation of the structure is mandatory.

Directional vs. Positional Alignment

Our layout embedding optimization process generally aims for alignment of iso-parametric curves (hence also arcs and final patch parameterizations) to principal directions on the surface. Another form of alignment is that of *positional* alignment: one might want to fit an arc or a specific iso-curve of the parameterization to a specifically located curve on the surface. We described how to do this for the cases of fixing arcs to feature curves (cf. Section 6.5.2).

The optimization does, however, not explicitly aim at specifically positioning arcs onto non-sharp, smooth features or similar curvature extrema (unless fixed manually). While such alignment might be desirable from an aesthetic point of view, let us point out that it would often be suboptimal in terms of isometry and alignment. The inset demonstrates this on the BLOCK model which has numerous smooth feature curves:



On the left is the result of our optimization where some arcs align to smooth features, some do not – because this is the optimum in terms of isometry and principal direction alignment (as measured in a combined form by the parameterization energy functional). On the right we show a version where the nodes have manually been repositioned so as to achieve further arc-to-feature alignment, in particular around the holes. While this can be of interest for certain applications, in a general sense the left result is to be considered better: the parameterization residuum,

reflecting isometric distortion and misalignment, is higher by a factor of 2.3 on the right. In particular, forcing an irregular node and two incident arcs to lie on a smooth feature curve implies patch corner angles of $\leq 60°$ at valence 5 nodes and of $180°$ at valence 3 nodes – far from the general optimum of $90°$.

Part II.

Interactive Design

Note: *This part is based on [CK14a].*

In Section 2.2 we discussed the issue of layout quality objectives which can be hard to formalize, subjective, driven by aesthetics, etc. The related field of quad mesh generation faces a very similar issue: based on some quality measure a compromise between alignment, orientation, and element shape needs to be made [BZK09]. A solution which found success in practice in this field was the inclusion of the user in the process – instead of aiming for full automation. Major 3D sculpting packages like Pixologic's ZBrush or Pilgway's 3D-Coat have recently been equipped with quad remeshing features which follow this paradigm and rely on high-level user interaction, e.g. regarding the specification of alignment and element sizing. This allows the user to tune the result to meet the given requirements – even if no formal description thereof, or no specialized optimization method therefor is available.

Unfortunately, while methods for quad mesh generation allow for user influence (regarding edge flow, irregular vertex configurations, element sizing, element anisotropy, etc.) [BZK09, ZHLB10, KMZ11], even though not always at interactive rates, this is not provided for in the automatic pipeline presented in Part I, nor in other methods targeted at the problem of quad layout generation (e.g. [EH96, BMRJ04, TPP*11]).

In this part we present an interactive quad layout design system that explicitly takes the user into the loop and grants full flexibility to incorporate application-specific design intents.

7. Metaphors and Guides

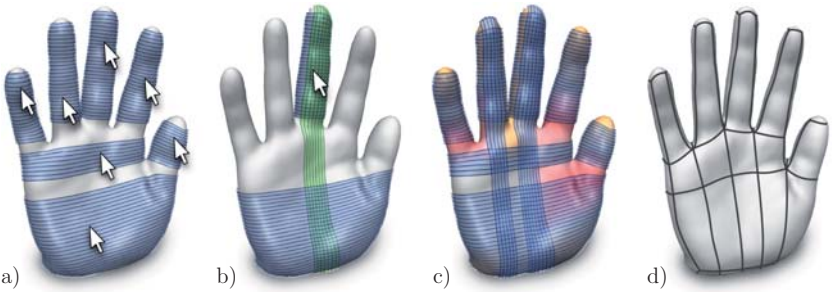


Figure 7.1.: Overview of our *Dual Strip Weaving* approach for the design of quadrilateral patch layouts. a) When hovering over the object, the user is immediately presented with the best elastica strip (visualized using a stripe pattern) at the current pointer position. It can be selected and fixed with a single click. b) Fixed strips (blue) constrain the design space; only compatible strips are offered next (green). c) Indicators based on color-coding and stripe patterns guide the user to regions where modifications are recommended for the benefit of layout quality. d) Finally, a quad layout structure can be derived from a collection of strips.

7.1. Concept

A design process with a user in the loop necessarily proceeds incrementally. This raises the question of what nature the increments should be, or in other words:

What should be the atomic operations provided to the user?

In established modeling tools (mainly for subdivision base meshes) this is often answered like: the user can create nodes, arcs, and patches, and on a somewhat higher level multiple patches can be created at once using splitting, extrusion, and similar operators. In these cases, the user deals directly with the individual elements of the layout.

A drawback of this approach is its fine granularity: the operators are very local. Quad meshes and quad layouts, however, generally have a rather constrained global structure [MBBM97]. As this fact is not reflected in the local operators, it is up to the user to plan ahead such that in the end all the locally constructed elements meet up globally in a desirable manner. Large parts of the design might have to be redone when the set of created quads comes to form a non-quad region which cannot be quadrangulated without adverse side-effects (irregular vertices, excessive refinement, etc.). To cite Takayama et al. [TPSHSH13]: "it is often quite challenging even for professional artists to manually design a perfect quad mesh on the first try. Since the quality [...] is a global property, the correction of a single mistake might require regeneration of the entire mesh."

For this reason, in stage 2 of our automatic pipeline (cf. Chapter 5) we made use of global dual loops, which allowed us to ensure layout consistency more easily. We here propose to use **the creation of an entire dual loop as *the* atomic operation** also for interactive design purposes. In order to make the interaction more intuitive and comprehensible we, however, equip the topological dual loops with a geometric dimension, turning them into *dual strips*. In contrast to the established local operators, this operator has a kind of built-in global consistency (no non-quad patches are ever generated), effectively reducing the burden of "planning ahead" for the user.

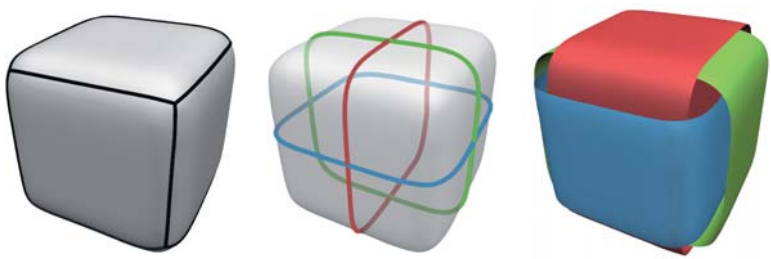


Figure 7.2.: Basic concepts: quad layout, dual loops, dual strips.

7.1.1. Dual Strips

A dual loop corresponds to a cyclic chain of quads in the primal layout, cf. Section 2.1. The union of all the quad patches of such chain we call *dual strip*, cf. Figure 7.2. We may say a dual loop is the spine of its dual strip. Note that a quad patch of the primal layout corresponds to an intersection region of two dual strips, crossing transversally. From this point of view our goal of partitioning a surface into quad patches is equivalent to **doubly-covering the surface with dual strips** where all strip intersections are transversal.

A practical analogon which illustrates this idea is basketry, i.e. the weaving, or more specifically plaiting, of baskets from strips of plant materials like bark, straw, or flax. The figure on the left (courtesy of Jonas Hasselrot) shows an example. Notice how every quad is covered by two crossing strips. At the bottom corners of the basket it can be observed that irregular "nodes" (here of valence 3) can be formed using this technique, too. In this sense, the underlying conceptual idea of our system can be seen as weaving dual strips on a given surface until it is covered. Note, though, that our strips are of variable width and the layer-alternation of the woven strips, which serves stability in practice, is of no meaning in our case. This process of *Dual Strip Weaving* is computationally supported in our system, e.g. by proposing optimal routes or choosing appropriate widths for strips.

Note that building the layout based on dual loops or strips does not restrict the class of designable layouts in any way – for every quad-only layout there is an equivalent collection of dual loops/strips.

7.2. Interactive Workflow

We begin by describing the design workflow in order to provide a high-level understanding of the system and its core concepts. As stated above, the creation of a dual strip is

the fundamental operation in our system. Instead of having the user draw such strips by hand, the central idea of our system is to compute suitably optimized dual strips automatically and propose them to the user. Intuitive tools to select, edit, or model these strips are then made available to provide full flexibility, while still keeping the user workload low. Technical details of the involved algorithms for dual strip computation follow in Chapter 8.

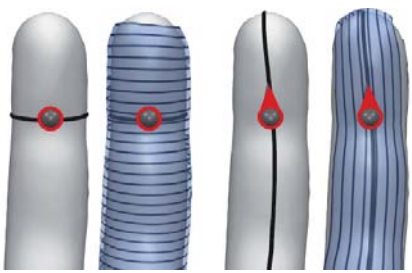
7.2.1. Tools and Metaphors

Simply **hovering** over the surface with the mouse pointer, the user is immediately presented with the best possible dual strip which runs through this point. This dual strip is constantly updated as the user moves the pointer. See Figure 7.1a.

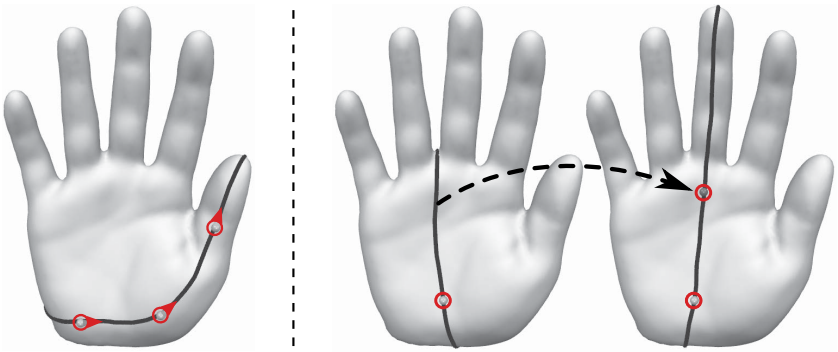
Using the **mouse wheel** the user can furthermore browse good alternative dual strips which run through the same point but take different routes over the surface, in order of descending quality.

These automatically proposed strips (computed and rated as described in Section 8.1), provide a rich fundament already sufficient for the construction of complete dual layouts. In order to provide full design flexibility to the user we furthermore enable the modeling and editing of dual strips using the following metaphors. In these manipulation modes it proved advantageous for clarity to display the strip in form of its representative spinal loop (the automatically chosen strip extent is not subject to user modification anyway; it serves purposes of "coverage" visualization).

By **clicking**, the user creates an anchor point and the best dual strip through this point is shown (the one which was already shown when the user hovered over this point). By **dragging** a directional anchor is created instead, producing a dual strip which runs through this point, interpolating the specified direction. The direction can also be altered interactively in order to explore the space of possible dual strips through the anchor point.



By clicking (or dragging) at further points on the surface, additional anchors (of positional or directional kind) can be placed. The best dual loop **interpolating** these anchors is then shown.



Another metaphor that can be used to conveniently edit the shape of a dual loop is **grabbing**. The user can grab the current loop at any point and drag it to another position, implicitly creating an additional anchor to be interpolated. This can be seen in analogy to modern route planning applications whose interfaces offer similar grab-and-drag tools to manipulate proposed routes.

When the user begins creating a new dual strip, the existence of already created strips is taken into account. The best way for two strips to cross is orthogonally; at least they should be crossing transversally, not touching tangentially, cf. Section 5.1.1. The system only proposes strips which respect this, favoring orthogonality, and which are furthermore not topologically equivalent to already existing ones, cf. Figure 7.1b.

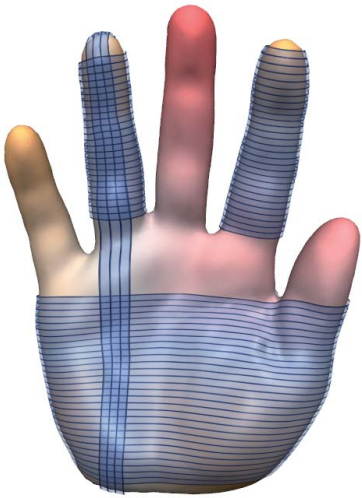
7.2.2. Assessment & Feedback

The provided tools enable the flexible design of dual strip collections, which via dualization imply a primal quad layout structure (determination of its actual geometric embedding in detail is considered in Section 8.3). Especially when designing rather coarse layouts it can, however, be quite hard to visually judge the appropriateness and quality of the dual strip collection during design, because large patches that wrap around

the underlying object are not visible in their entirety. We thus equip our system with visual indicators guiding the user in this regard.

The dual strips are visualized by a pattern of **quasi-parallel loops** in order to indicate their orientation ("edge flow"). Where two strips cross, these patterns overlap, forming a **grid**. This indicates that the corresponding region is *doubly-covered* as desired.

The user does not need to doubly-cover every part of the surface – uncovered regions only hint at sub-optimally shaped quads in the primal layout, but this can be a desired trade-off for layout simplicity. However, every region enclosed by loops at least needs to be homeomorphic to a disc, otherwise no valid primal layout is implied. Regions with **non-disc topology** are hence highlighted in red to indicate that further strips need to be added.



When no region is red anymore, the set of dual strips is *topologically sufficient* and implies a valid primal quad layout. The addition of further strips is, however, often desired in order to refine the layout and achieve better geometric layout quality. Our system guides the user to those regions where this is particularly advisable. Regions which contain multiple pronounced **local extrema of Gaussian curvature** are highlighted in orange, indicating that it would be beneficial (though not mandatory) to add further strips there. This is motivated by the fact that such extrema are best represented by separate (irregular) nodes of the quad layout (thus separate dual regions) – having (some of) them lie in the interior of a quad patch would lead to low geometric patch quality.

In addition, we highlight in yellow those regions with a **mismatch between curvature and valence**: Ideally, the valence v of a region should be related to its total curvature K by $K = (4 - v)\frac{\pi}{2}$ for the sake of patch quality, cf. Chapter 4. If the actual valence differs by more than 1, we indicate this for information. Note that this is mainly relevant for highly edited strips – the automatically proposed strips rarely lead to such situations due to the inherent favoring of orthogonality and alignment (cf. Section 8.1).

8. Elastica Strips

The practicality and efficiency of the described interactive workflow depends on the geometric and structural quality of the automatically generated loops and strips. Given a surface $\mathcal{M}$, a dual loop on it should ideally (cf. Section 5.1.1)

1) be aligned with principal directions of $\mathcal{M}$ and

2) have low geodesic curvature to facilitate a good quad shape.

Furthermore, for the sake of layout coarseness, it should rather

3) be short than wind around the whole object several times.

It should further respect, i.e. interpolate, the specified anchors of positional and directional kind (cf. Section 7.2). We generate dual strips in two steps: a dual loop is constructed to serve as spine (Section 8.1) and is then extended to an appropriate dual strip (Section 8.2).

8.1. Elastica on Surfaces

8.1.1. Field-Guided Geodesic Loops

In stage 2 of our automatic pipeline, cf. Chapter 5, we modeled dual loops as anisotropic geodesics with respect to a prescribed guiding cross field, which was computed in stage 1. This field is aligned to stable principal directions and smooth otherwise, and in this way jointly promotes the alignment (1) and low geodesic curvature (2) of the constructed loops.

Unfortunately, this approach is not amenable to interactive user-guided design. The prescribed field already considerably restricts the space of representable layouts: dual loops constructed by this approach are essentially "bound" to the field. Hence, the user is not free to create loops as desired – geometrical as well as topological restrictions apply. While there seems to be enough flexibility to automatically find *a* reasonable result layout, chances are that the layout intended by the user is not representable. In particular, the singularities of the field already completely define the number and approximate position of the result layout's nodes (up to local merging). But not only the number and configuration of nodes is fixed, also the connectivity of these nodes, i.e. the arcs of the layout, is subject to restrictions induced by the fixed underlying field. In detail, only those loops along which the prescribed field has zero holonomy can be created [LJX*10].

For design purposes such broad restrictions are hardly communicable to the designer. In the following we hence present a field-less approach that allows for arbitrary dual loops. It is furthermore able to take multiple user design constraints (the specified anchors) into account.

8.1.2. Elastica Loops

A suitable model for the (closed) curves $\ell : [0, L] \to \mathcal{M}$ in arc-length parameterization we are looking for is the objective

$$c(\ell) = \int_0^L 1 + \alpha\, q_{\ell(t)}(\ell'(t)) + \beta\, \kappa_\ell(t)^2 dt \to \min \tag{8.1}$$

subject to (positional and directional) constraints. Here the term 1 penalizes length, $q : T\mathcal{M} \to \mathbb{R}$ penalizes deviation from principal directions (cf. Section 8.1.5), and the last term penalizes geodesic curvature $\kappa_\ell(t)$. A curve minimizing the bending energy $\int \kappa_\ell(t)^2 dt$ subject to positional and directional constraints is called *elastica* (or, depending on the context, *spline*), going back to Leonhard Euler and Jacob Bernoulli. Our functional in addition includes means to soft-constrain local direction via q, with the parameters α and β expressing the relative weighting of these objectives.

On surfaces, embedded elastica can be found using various methods of variational nature, like Active Contour and Snake models [LL02, BWK05] or the constrained spline optimization of [HP04]. These methods, however, require an initialization and then

Figure 8.1.: Our optimization algorithm for elastica on surfaces has *free homotopy*: depending on the orientation of the directional constraint (red dot with arrow), loops from differing homotopy classes are obtained because the optimum is searched over all classes.

strive to find a local optimum in the same homotopy class as the initial curve or loop[1]. To maximally support the user, we want to avoid the need for an initialization and would in particular like to find the optimum over all homotopy classes.

Hence, we design an algorithm, based on combinatorial optimization, which is able to find (approximations of) global optima of certain curve functionals, and this over all curve homotopy classes, i.e. without prescribed homotopy (cf. Figure 8.1). It is based on finding constrained minimum weight cycles in special graph structures.

To this end, all metric information must be modeled as graph edge weights. Unfortunately, in the case of a surface graph (e.g. a triangulation of the surface) only first-order differential quantities can be taken into account in such an approach: the weight of an edge has to be computed in advance from local information only, i.e. we can at most build the difference of its two end node positions and compute the weight from this vector's length and direction (and the node positions). Curvature, as in our objective (8.1), obviously is not accessible per-edge in this way.

8.1.3. Elastica Graph

If, however, we take a graph whose nodes are not a sample of the surface $\mathcal{M}$, but of the tangent bundle $T\mathcal{M}$ of $\mathcal{M}$, the nodes already contain first-order information, such that

[1]The spline method of [PBDSH13] does not require initialization, but is essentially based on (pseudo) geodesics, leading to similar homotopy class restrictions. In particular, it cannot construct (non-degenerate) loops from just one constraint point – the most relevant case here.

second-order properties, i.e. curvature, can be computed per edge: given two adjacent nodes with position and tangential direction each, we are able to evaluate $\int \kappa(t)^2 dt$ for an imagined connecting curve interpolating this Hermite data.

We are free to choose any sampling of $T\mathcal{M}$ and a connectivity to construct a graph approximation of the tangent bundle. A natural and intuitively accessible way is to take some digraph g approximating $\mathcal{M}$ and then form its *derivative* g' (also called *line graph* or *adjoint*) [Bei68]. A node of this graph is a *directed* edge of g and can hence readily be identified with a point $p \in \mathcal{M}$ (the edge's midpoint) and a direction $d \in T_p\mathcal{M}$ (the directed edge's direction). Two g'-nodes are connected iff the corresponding directed g-edges are adjacent, forming a directed path of length two. For the special case of planar regular grid graphs such line graph or related product graph constructions have successfully been applied for curvature-regularized image segmentation purposes [SC07, SMC11].

As underlying graph g we choose an extended neighborhood graph built from a triangulation $\mathcal{T} = (V, E, F)$ of $\mathcal{M}$, where two nodes are connected (by two directed edges, realized as geodesics) iff they have graph distance $\leq k$ in $\mathcal{T}$. We found $k = 4$ to sufficiently increase the angular resolution such that the resulting elastica are visually smooth. In this case, the cardinality of the node sets is related as follows: $|V_g| = |V|$, with an average valence of 60, and $|V_{g'}| \approx 60|V|$, with an average valence of 60. Note that g'-edges connecting g-edges which form a large geodesic angle can be omitted (e.g. the red and purple ones in Figure 8.2) – the corresponding curves have large geodesic curvature, thus are expendable in light of the bending energy minimization goal. We use a generous limit of 30°, reducing the average valence of g'-nodes to 10.

Curved Edges The question remains how a g'-edge $e' = (e_0, e_1)$ should be interpreted geometrically, i.e. which curve's geodesic curvature should be measured to obtain edge weights for e'. Let's first assume a planar configuration. One option is to compose a circular arc of maximal radius and a straight line segment, connecting the midpoints of e_0 and e_1 (cf. Figure 8.2). For this unique curve we calculate $\int \kappa^2 = 2\gamma \tan \frac{\gamma}{2} / \min(|e_0|, |e_1|)$, where γ is the angle between g-edges e_0 and e_1 at their common node, $|\cdot|$ the length of an edge. As for its approximation

$$\frac{\gamma^2}{\min(|e_0|, |e_1|)} =: \kappa^2(e')$$

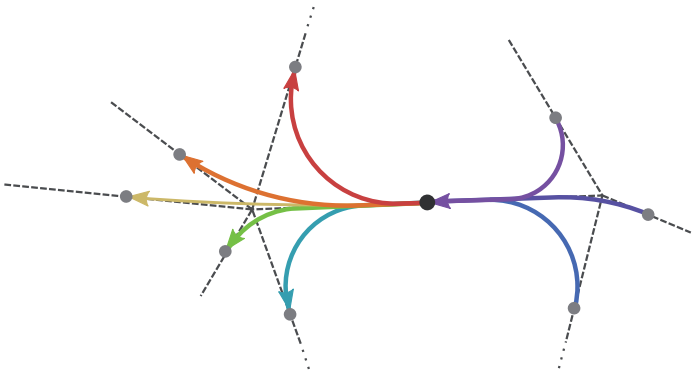


Figure 8.2.: Illustration of one directed node's (black) incoming and outgoing curved edges in the elastica graph g'. The underlying surface graph g is shown dashed. The g'-nodes lie at g-edge centers.

(using Taylor expansion $\tan(\gamma) = \gamma + O(\gamma^3)$) favorable convergence properties have been proved (i.e. using graph refinement, continuous elastica can be approached) [BNR01], we opt for this choice. To generalize to non-planar configurations we only need to measure γ as *geodesic* angle on $\mathcal{M}$, i.e. in a tangent plane at the common node of e_0 and e_1. This effectively unrolls the configuration to the tangent plane, masking out normal curvature, such that only the geodesic curvature is measured as intended.

The length of a g'-edge e' realized as described above evaluates to

$$|e'| = \frac{\gamma \min(|e_0|, |e_1|)}{2 \tan \frac{\gamma}{2}} + \frac{||e_0| - |e_1||}{2}.$$

The function q from (8.1), promoting principal direction alignment, we evaluate for e' using the trapezoid rule, sampling at the incident g'-nodes n_0, n_1 (midpoints of g-edges e_0 and e_1), leading to

$$q(e') := \frac{1}{2} \left(q_{n_0}(e_0) + q_{n_1}(e_1)\right) |e'|.$$

Using these definitions we can build a discrete version of (8.1) for the closed curve formed by a cyclic chain E of g'-edges:

$$c(E) = \sum_{e' \in E} |e'| + \alpha\, q(e') + \beta\, \kappa^2(e') =: \sum_{e' \in E} w(e') \tag{8.2}$$

8.1.4. Constructing Discrete Elastica

Minimizers of (8.2) are minimum weight cycles in the elastica graph g'. The global optimum can hence for instance be found using a variant of the Floyd-Warshall algorithm. However, we are not interested in this unconstrained optimizer, but would like the loop to interpolate the specified anchors to account for user influence. Therefore, we design an algorithm based on nested minimum weight path problems which can very efficiently be solved using Dijkstra's algorithm.

In detail, we would like to take into account an arbitrary number of positional and directional constraints to be fulfilled by a loop ℓ. Let $\Phi = (\phi_0, \dots, \phi_{n-1})$ with $\phi_i = (p_i, d_i) \in T\mathcal{M}$ be a list of $n > 0$ constraint points that shall be passed by ℓ in order, with tangent parallel to d_i where $d_i \neq 0$ ($d_i = 0$ signifies a purely positional constraint). Let v_i be the g-node closest to p_i and ξ_i the set containing just the outgoing g-edge of v_i closest to parallel with d_i if $d_i \neq 0$, and the set of all outgoing g-edges of v_i otherwise.

For each two subsequent constraints ϕ_i, ϕ_{i+1} (all indices taken mod n), we compute intermediate elastica curves according to (8.2): for each pair $(a, b) \in \xi_i \times \xi_{i+1}$ (remember: a, b are edges in g and nodes in g') we compute the shortest path (taking the g'-edge weights w into account) from a to b in g' and record its cost as $\bar{w}(a, b)$. Note that if $|\xi_i| = 1$ or $|\xi_{i+1}| = 1$, the shortest paths for all pairs can be computed with just one run of Dijkstra's algorithm[2] (being a single-source all-destinations type algorithm). Only if both constraints have unspecified direction, $\min(|\xi_i|, |\xi_{i+1}|)$ runs are necessary. Being independent, all runs can conveniently be performed in parallel.

Then we form a metagraph. Its set of nodes is $\cup_{0 \leq i < n} \xi_i$, and the set $\cup_{0 \leq i < n} (\xi_i \times \xi_{i+1})$ specifies its edges (cf. Figure 8.3). These metaedges are weighted by $\bar{w}$. In this metagraph we find the minimum-weight cycle. Due to the special structure of the metagraph, we can do this more efficiently than by using the Floyd-Warshall algorithm. If at least one ξ is a singleton, the cycle is found using a run of Dijkstra's algorithm on the metagraph from this one element back to itself. Otherwise, this is done for each element of the smallest ξ and the minimum taken. In any case, for fixed n and k, the complexity of the complete elastica construction algorithm is $O(|V| \log |V|)$, V being the set of vertices of $\mathcal{M}$'s triangulation.

[2] A variant that does not return the empty path if $a = b$ must be used.

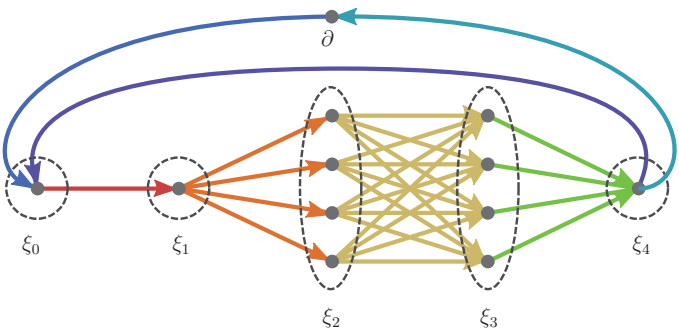


Figure 8.3.: Metagraph construction for multi-constraint elastica. Example with 2 positional (ξ_2, ξ_3) and 3 positional+directional (ξ_0, ξ_1, ξ_4) constraints. The boundary node ∂ allows for the construction of *dual curves* in addition to dual loops.

Concatenation of the intermediate elastica which correspond to the metaedges of the minimum-weight cycle yields the optimal loop interpolating Φ: in case that $d_i \neq 0$ at each constraint, the metagraph is a single cycle and the optimal loop trivially composed of all the pairwise elastica curves; otherwise, if there are constraints with unspecified direction, this algorithm finds the minimum over all (combinations of) possible directions. In the case of one positional constraint only, the $|\xi_0|$ trivial runs of Dijkstra's algorithm on the metagraph yield $|\xi_0|$ loops (all crossing the anchor point in different directions), which can be offered as alternatives to the user.

It is worth noting that there are quad layouts which contain dual loops crossing themselves. Also such loops can readily be obtained using the described algorithm without any additional measures.

8.1.5. Principal Direction Alignment

The functional $q : T\mathcal{M} \to \mathbb{R}$ is used to penalize deviation of loop directions from principal directions. We compute (unit) principal direction vectors $d_{\min}, d_{\max}$ and principal curvatures $\kappa_{\min}, \kappa_{\max}$ from the eigenvectors and eigenvalues of the shape operator [CSM03]. The alignment of a unit tangent vector t with a principal direction can naturally be

measured using the inner product $|\langle t, d\rangle|$ on M, the deviation using the reciprocal. We measure deviation to *either* principal direction using

$$\mathrm{dev}_p(t) := \max\left(|\langle t, d_{\min}\rangle_p|, |\langle t, d_{\max}\rangle_p|\right)^{-1} - 1,$$

which is zero in case of perfect alignment.

We weight this deviation with the local shape anisotropy factor $(|\kappa_{\min}| - |\kappa_{\max}|)^2$ as in [KCPS13] and obtain

$$q_p(t) := (|\kappa_{\min}| - |\kappa_{\max}|)^2 \mathrm{dev}_p(t),$$

which suitably vanishes in umbilic regions with ill-defined principal directions.

8.1.6. Feature Curves

It is usually desirable to align patch boundaries of a quad layout to sharp feature curves on the surface. To enable this, dual loops should not cross such features with small crossing angles, but ideally orthogonally. We replace $\mathrm{dev}(p, t)$ defined above by $\mathrm{dev}(p, t) := |\langle t, d_{\max}\rangle_p|^{-1} - 1$ on feature curves to achieve this. Now orthogonally crossing loops are favored, and the penalty increases to infinity as the crossing angle goes to zero.

8.1.7. Boundaries

So far we considered dual *loops*, i.e. closed curves. On surfaces with boundary $\partial\mathcal{M}$, we also need to deal with non-closed curves which end at $\partial\mathcal{M}$. For this, we add one additional node ∂ representing all boundaries to the above metagraph construction. Edges (a, ∂) and (∂, b) are added for each $a \in \xi_{n-1}$ and each $b \in \xi_0$, weighted by the cost of the shortest path between a/b and any g'-node in E_∂, where E_∂ is the set of all g-edges incident to a boundary vertex of $\mathcal{M}$'s triangulation $\mathcal{T}$. Where distinction is necessary, we call the resulting elastica *dual curves* instead of dual loops.

The user can choose whether the final quad layout should be aligned to the boundary, or whether non-aligned (trimmed) patches are acceptable. In the first case, dual curves should meet the boundary orthogonally. This is achieved by treating the boundary as a feature curve, using the weighting described in Section 8.1.6.

8.1.8. Symmetries

In case $\mathcal{M}$ has a global (exact or approximate, extrinsic or intrinsic) symmetry, it is likely that the user wishes to design an accordingly symmetric layout. Global symmetry can be of reflectional or rotational kind. In the first case, $\mathcal{M}$ consists of two, in the latter of two or more symmetric parts $\mathcal{M}_i$. We provide the convenient option to perform the design on only one part, $\mathcal{M}_0$, automatically transferring the layout to the other parts. We make no assumptions about the symmetry transformations ψ_i mapping $\mathcal{M}_0$ onto $\mathcal{M}_i$ except for continuity; they can be specified by the user or be determined using (semi-)automatic methods [MPWC13]. Note that the dual curves on $\mathcal{M}_0$ must meet certain constraints such that they merge into continuous smooth loops on $\mathcal{M}$. We achieve this as follows, see also Figure 8.4.

Rotational In case of rotational symmetry, the boundary $\partial\mathcal{M}_0$ has two symmetric parts which are identified by the map ψ_1. In g' we realize this identification by means of virtual edges, connecting respective g'-nodes adjacent to $\partial\mathcal{M}_0$. A loop found in g' containing one or more of these virtual edges then represents one or more dual curves on $\mathcal{M}_0$ whose ends lie on positions on $\partial\mathcal{M}_0$ which are identified by ψ_1 in a pairwise manner (cf. Figure 8.4d). Hence, mapped to all parts via ψ, these curves join seamlessly (cf. Figure 8.4e).

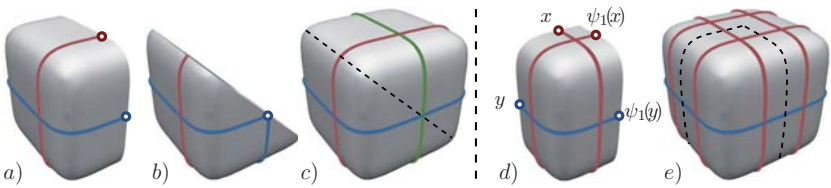
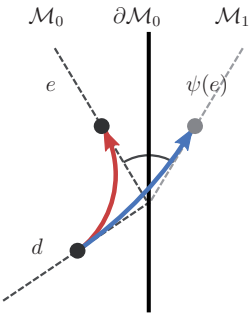


Figure 8.4.: Symmetry: (a) shows one representative half of the reflectionally symmetric object (c), dual curves end orthogonally on its boundary and thus form smooth loops when mapped to both halves. In (b) one half for another reflectional symmetry is shown, the blue curve is reflected on the boundary and thus forms two smooth crossing loops on (c). (d) shows one of four parts of a rotational symmetry, left and right halves of its boundary are identified, thus technically the two red curves are one loop, as is the blue curve. Mapped to all four parts, continuous loops are formed (e).

Reflectional In case of reflectional symmetry we can make the following observation: a symmetric loop on $\mathcal{M}$ must either cross $\partial\mathcal{M}_0$ orthogonally, or it must cross another (not necessarily distinct) loop on $\partial\mathcal{M}_0$ – an analogous property was recently discussed for symmetric cross fields [PLPZ12]. The orthogonal case can be handled just like an aligned boundary as described above. For the other case we must ensure the existence of both crossing curves at once. Conceptually, instead of letting a curve end at $\partial\mathcal{M}_0$, we reflect it at this boundary and let it also form the second curve. Technically, weights of g'-edges between g-edges incident to $\partial\mathcal{M}_0$ are computed differently: the red g'-edge (d, e) in the figure on the right is assigned the weight computed for the blue curve $(d, \psi(e))$ formed with the reflection of the g-edge e. In this way, curves resulting from the Dijkstra approach are either closed and contained in $\mathcal{M}_0$, meet $\partial\mathcal{M}_0$ orthogonally and end there, or meet $\partial\mathcal{M}_0$ non-orthogonally and are reflected. Mapped to both halves of $\mathcal{M}$ via ψ, these curves form symmetric loops (or curves), smoothly crossing the symmetry's stationary line.



8.2. From Loop to Strip

While (infinitesimal) dual loops are sufficient to define the layout's topological structure, the lack of a geometric dimension may overly stress the user's imaginative powers. Hence, we appropriately expand dual loops, constructed as described in the previous section, to dual strips in order to better visualize the state and to guide the process of interactive design as described in Section 7.2.

Formally, in analogy to a dual loop (cf. Section 8.1.2), a dual strip is a continuous map $s(u, v) : [-l, r] \times [0, L] \to \mathcal{M}$ of a rectangle (or a subset thereof in case of surface boundaries) onto the surface, with $s(\cdot, 0) = s(\cdot, L)$ in case of a closed dual strip. Its dual loop/curve (spine) is its zero-u-isocurve $s(0, \cdot)$. An alternative intuitive view of a dual strip is as a continuum of adjacent dual loops – which represent the one-parameter family of u-isocurves of s. The stripe pattern used to illustrate the strips (cf. Figure 7.1) simply represents a regular sampling of this continuum.

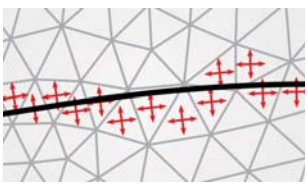
Conceptually, we build a dual strip by taking a dual loop and broadening it by extending the map up to appropriate bounds l and r, i.e. up to the point where the shape of $\mathcal{M}$ causes too large distortions in s. As the acceptable degree of distortion depends on the user's intent, the boundaries of the dual strips are to be seen as fuzzy indicators rather than definite patch borders. Because, a priori, the bounds l and r are unknown we compute an unbounded, global, distortion-minimizing map $S : \mathcal{M} \to \mathbb{R}^2$ (cf. Section 8.2.1) from which all individual strip maps s can be extracted. This common map S is recomputed whenever the user inserts a new strip.

Note that one could follow the simpler strategy of expanding strips in a local, incremental manner – until some stopping criterion is met – e.g. based on a geodesic wavefront emanating from the loop [Sch13]. Besides the difficulty of defining a suitable local stopping criterion, this would result in u-isocurves with a constant spacing, implied by the constant gradient magnitude of a geodesic distance field. Depending on the model's shape this leads to isocurves with high geodesic curvature, thus to unnatural stripe patterns and strip boundaries (e.g. on the arc of model FIVE or the arms of model FERTILITY in Figure 8.5). The global map based approach, by contrast, yields stripes which much better resemble the edge flow of a globally coherent quad mesh by its very nature.

8.2.1. Global Parameterization

We require the global map S to be aligned with all dual loops, i.e. the loops should be isocurves of S. This implies that, in general, S needs to have singularities. Such parameterization with alignment and singularities can automatically be computed using an instance of cross field based parameterization [KNP07].

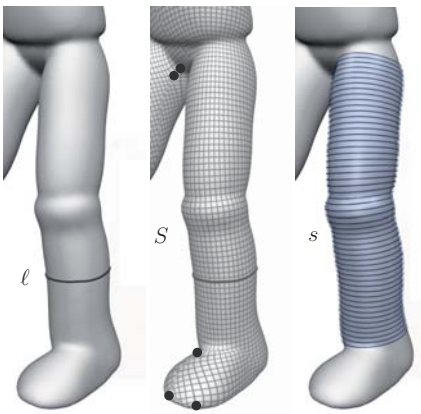
For the generation of a suitable globally smooth cross field we make use of the representation vector concept [PZ07], which allows for smoothness optimization via simple sparse linear system solves [KCPS13, DVPSH14] – in contrast to the popular angle-based formulation [BZK09] without any integer constraints. In each triangle of the triangulation $\mathcal{T}$ of $\mathcal{M}$ which is crossed by a dual loop we set a unit representation vector which represents a cross aligned with the loop. These representation vec-



tors are then harmonically interpolated over $\mathcal{T}$, a cross for each triangle recovered from the result, and the field singularities extracted, analogous to the recent description as a special case (4-RoSy) in [DVPSH14].

Then we compute a global, chart-based parameterization which is optimized for isometry and alignment of isocurves to the constructed cross field. This can be done efficiently, again using a single sparse linear system solve, in a least-squares manner as described in [KNP07, BZK09] (note that in contrast to these works we do not have to perform any integer rounding). To achieve exact alignment to the dual loops we add (linear) alignment constraints to the optimization problem, forcing all edge intersections of a loop to lie on a common isocurve as described in Section 6.5.2.

By construction, each dual loop is an isocurve[3] of S. To both sides of a loop ℓ we then conquer the continuum of adjacent isocurves, effectively extracting the strip map s as illustrated on the right. We stop when a singularity in S is reached (or, in the extreme case of a singularity-free field, the entire surface is conquered). The singularities quite naturally indicate appropriate bounds for the strips because they represent portions ($\pm k\,\pi/2$) of total Gaussian curvature, in particular extrema like corners – which should not come to lie in the interior of quad patches for the benefit of patch developability (e.g. enabling low-distortion patch parameterization). Encountered degeneracies or fold-overs, which can sometimes appear in S, can gracefully be dealt with by simply stopping there, too. As these appear adjacent to singularities on principle, the impact is minuscule. Note that the field singularity constellation can also be exploited to easily perform the region coloring related to Gaussian curvature extrema (cf. Section 7.2.2).



[3]Formally, the chart-based parameterization S has transition functions across chart boundaries implied by the cross field. The term "isocurve" is implicitly meant to take these transitions into account.

Note that the cross field has no influence on the dual loop construction (in contrast to the approach described in Chapter 5); it merely serves the strip map creation for visualization purposes – and helps to ensure loop and strip transversality as described next.

8.2.2. Strip Compatibility

Whenever a strip has been created by the user, it should constrain the design space appropriately, as already mentioned in Section 7.2. Within the area covered by the strip we thus remove (i.e. disable) all those nodes from the elastica graph whose associated direction forms an angle $\leq 45°$ with the isocurves of the strip map s. Now, when the user hovers over the area covered by the strip, 1) no strips which are topologically equivalent are proposed, and 2) the underlying dual loops do only cross transversally, as required for validity.

8.3. Primalization

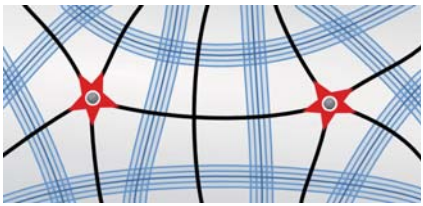
The collection of dual strips defined by the user uniquely determines the structure of the primal quad layout. For the determination of the layout's detailed geometry, i.e. its actual embedding in $\mathcal{M}$, we then have multiple (automatic, manual, assisted) options.

The method described in Chapter 6 can of course directly be applied for this task. It, however, assumes the layout is intended for principal direction alignment. If this is not the case, a method which does not aim for alignment could be the solution of choice. For instance, in the final stage of their method, [TPP*11] apply a technique to optimize the embedding of a quad layout in such a manner. Like our embedding optimization, it is driven by a quality metric based on parameterization of the layout's patches. These approaches proceed in a fully automatic manner.

Alternatively, in order to give control to the user also in this stage, a manual strategy can be followed: the user positions the layout's nodes in the regions inbetween the dual loops and draws layout arcs through the corridors formed by the loops in order to connect them. Note that no hard topological decisions need to be made in this context, the question is only where a node should be positioned within a region and where within

a dual loop corridor a patch boundary should lie. Nevertheless, this can be a tedious task.

We provide assistance by automatically proposing good node positions and arc routes – which may serve as starting point for adjustments through the user. Each region (enclosed by dual loops) of valence $\neq 4$ contains at least one cross field singularity. We initially position a region's primal node onto this singularity if there is just one, otherwise into the geodesic center of the region. We then compute elastica curves within the corridors between the dual loops and use them as routes for the connecting arcs. This is illustrated here schematically. The manipulation tools from Section 7.2.1 can be applied for these curves, too. The use of elastica is motivated by the fact that the objectives stated for dual loops in Section 8.1 often hold for primal arcs, too. Note that in regions of valence 4 two curves cross, implying a natural position for the corresponding regular node. This latter approach was used for the examples shown in the following.



8.4. Results

Figure 8.5 shows layouts designed by non-expert users with the described system. The video accompanying the article [CK14a] gives further impressions of the design process.

Parameters We normalized all input models (to bounding box diagonal 1), such that model-independent parameters can be used. We used $\alpha = 1$ and $\beta = 0.25$ in (8.2) for all examples.

Timings We run our design system on a commodity PC. To get an impression of the system's performance, let's consider the practical scenario of an input mesh with 30,000 faces for example. The elastica loop construction is output sensitive: good dual loops with low cost are found within a few milliseconds; when multiple anchors induce very high cost, construction time can increase to around 1 or 2 seconds. The subsequent generation of the cross field takes around 200 ms, the parameterization and strip extraction around

400 ms. For optimal responsiveness, we display a loop immediately after its computation, while the field and strip generation is performed asynchronously in a background thread. In total, this allows to provide an interactive interface – only expensive loops restrain fluency to some extent. The user interaction time per model of Figure 8.5 for the design of all strips was between 10 seconds and 5 minutes.

8.5. Discussion

We proposed a novel, alternative concept for the design of quad layouts. In contrast to established systems it builds upon global operators based on dual loops and strips. The core technical component of our approach is a novel combinatorial algorithm for the constrained optimization of elastica loops embedded in surfaces. We further described how our system can support the designer in the presence of symmetries, boundaries, and feature curves.

In our system, optimal dual loops are constructed and then extended to dual strips. It could be of benefit if the strip geometry, in particular its width, could also already be considered in the loop optimization: wide strips could be favored over very narrow ones to the benefit of layout coarseness. However, it is unclear how this problem could be tackled algorithmically, and currently it seems unlikely that such integrated optimization could be performed at interactive rates, as desired in our system.

While we ensure that each loop crosses all other loops transversally, favoring orthogonality, a loop can be brought to cross *itself* with arbitrary angles; there is no mechanism in the loop optimization algorithm that could provide control over self-crossings and prevent small angles there. This is not a noticeable hurdle in practice because the principal direction alignment term in (8.1) favors orthogonality also in this situation, but depending on the anchors set by the user, self-crossing with bad angles cannot be ruled out. However, it is easy to automatically detect such a loop and highlight it accordingly, asking for adjustment.

Our current implementation offers a fully interactive workflow on models with several tens of thousands of triangles. In practice sometimes models with hundreds of thousands or millions of triangles are to be dealt with. The geometric fidelity of these complex models, however, is not relevant for the layout in many use cases – at least not for

its topological structure. This opens the door for an efficient employment of simplified proxies. The layout design could be performed on a decimated model version and the result mapped back to the detailed original. This could even be done in a transparent manner, i.e. the elastica construction and strip expansion is performed on a coarse model in the background but the (interpolated) result displayed to the user on the original model. The final embedding optimization would be done on the detailed model again. We leave in-depth investigation of suitable simplification and mapping strategies (e.g. along the lines of [LSS*98]) and the analysis of the resulting interaction quality to future research. While, as a sideline, this proxy strategy offers a way to abstract from small-scale geometric detail, it could also be valuable to investigate ways to offer scale control for the layout design process independent of the model resolution. For this the shape operator could be computed with a correspondingly large integration radius, the curvature of elastica be measured based on analogously smoothed tangent planes, and scale-aware methods for cross field and parameterization construction [RVAL09, ECBK14] be employed for the strip expansion on the desired level of detail.

Figure 8.5.: Quad layouts designed with our interactive method. Shown are intermediate design stages with dual strips and region coloring, followed by the final quad layout. Next to each layout the number of dual strips and the (typically much larger) number of primal layout arcs is specified.

Part III.

Geodesy

9. Anisotropic Geodesics

Note: *This chapter is based on [CHK13].*

A fundamental operation made use of in the presented quad layout construction method is the computation of geodesic distances and geodesic paths. For instance, it occurs in the computation of separation indicators (cf. Section 5.2.2) or Voronoi regions (cf. Section 6.6.2), but most importantly and fundamentally in the computation of dual loops (cf. Section 5.1.4). In the latter case, distances are to be measured, and paths to be constructed, with respect to a non-Euclidean, anisotropic metric.

In the following we show how (and how well) known distance computation methods proposed for the isotropic standard scenario can deal with such general metrics, analyze the inherent issues, and discuss the results that can be achieved. For metrics with a high degree of anisotropy (like the one used in the dual loops construction, cf. Section 5.1.4) it turns out: typically either the runtime gets impractically high or the accuracy low. This is problematic for our use case because we rely on the anisotropic distance computation as the most fundamental operation – and it is used a large number of times.

For improvement we propose *Short-Term Vector Dijkstra*: an algorithm that can intuitively be understood as Dijkstra's classical shortest path algorithm equipped with a vector-valued short-term memory. It is fast and easy to implement while providing a practical level of accuracy even for metrics with a high degree of anisotropy.

9.1. Basics

We consider a 2-manifold $\mathcal{M}$ (possibly with boundary) equipped with smoothly varying norms $\|\cdot\|_{g_x}$ on the tangent spaces $T_x\mathcal{M}$. As these norms allow us to infinitesimally

measure distances on $\mathcal{M}$, for a continuously differentiable curve $\zeta : [0,1] \to \mathcal{M}$ we can define its total length through integration as $\ell(\zeta) = \int_0^1 \|\zeta'(t)\|_{g_{\zeta(t)}} \, \mathrm{d}t$. With this we can define the *intrinsic metric* g measuring geodesic distances between two points $p, q \in \mathcal{M}$ as the infimum over the lengths of all curves ζ on $\mathcal{M}$ connecting p with q, i.e.

$$g(p,q) = \inf_{\zeta} \{\ell(\zeta) : \zeta(0) = p, \zeta(1) = q\},$$

and in this way obtain a so-called *length metric space* $(\mathcal{M}, g)$.

Note that in the special but common case that $\|\cdot\|_{g_x}$ is induced by some inner product $\langle \cdot, \cdot \rangle_x$ on $T_x\mathcal{M}$ for each $x \in \mathcal{M}$, i.e. $\|v\|_{g_x} = \sqrt{\langle v, v \rangle_x}$, g is a Riemannian metric. If $\mathcal{M}$ is further embedded in Euclidean space – as is the case in most geometry processing scenarios – the Euclidean dot product implies a natural inner product via its restriction to $T_x\mathcal{M}$. We call the corresponding norm *standard norm*, denoted $\|\cdot\|_x$, and the induced Riemannian metric *standard metric*.

Relative to the standard metric and norm we characterize a metric g and its underlying norm based on their quotient $r(v,x) = \|v\|_{g_x} / \|v\|_x$ (for $v \neq 0$) as follows:

- If $r(v,x) \equiv r(x)$, i.e. it is independent of v, we call the metric g *isotropic* as there is no directional dependency.
- If $r(v,x) \equiv r(x) \not\equiv 1$, we call the metric g *weighted* by the weight field $r(x)$.
- If $r(v,x) \not\equiv r(x)$, i.e. there is some directional dependency, we call the metric g *anisotropic*.

The value $\gamma(x) = \max_v r(v,x) / \min_v r(v,x)$ defines the local degree of anisotropy or simply *local anisotropy* and $\gamma = \max_x \gamma(x)$ is the *maximum anisotropy*.

While the standard norm realizes the intuitive notion of geodesic distance and is most common in applications, more general anisotropic norms that incorporate directional dependencies gained increasing interest in recent years. Applications range from Meshing [BPC08, CBK12] and Segmentation [BPC08, SJC09], over Path Planning [RR90, LMS99] and Shape Matching [SJC09], to (mainly in 3D) Medical Imaging [PWKB02, PWT05, BCLC09, BC11]. The anisotropy can be related to principal curvature directions, terrain steepness, surface vector fields, or MRI diffusion tensors, to name some examples (cf. Section 9.3).

Figure 9.1.: Visualization of a geodesic distance field with respect to an *anisotropic metric* (shown by tensor ellipses on the left; field source on top). Next to a reference solution, we show the results and approximation errors of various adapted known algorithms applied for the distance computation. Our novel *Short-Term Vector Dijkstra* on the far right shows the best results.

Such applications typically operate in a discrete setting, e.g. on triangle meshes approximating manifolds. Numerous methods for approximate computation of (mainly isotropic) distances in such settings have been proposed. Their accuracy and efficiency typically depends on the quality of the triangulation: intuitively, the "rounder" the individual triangles, i.e. the closer to equilateral, the better. While working with rather nice triangulations is quite common in the digital geometry processing field (leveraged by powerful remeshing techniques), a problem emerges when anisotropic distances are dealt with: the notion of "roundness" is metric dependent! This means a perfectly round triangle which is equilateral in the standard metric can be a "cap" or "needle" far from round when viewed under another, anisotropic metric.

Unfortunately, when naïvely adapting traditional methods to non-standard metrics, they expect element roundness with respect to these metrics, while the meshes typically used in applications are optimized with respect to the standard metric – as often favored by the other processing steps. Hence, in practice anisotropic distance computations quickly arrive at inacceptably low accuracy with increasing degree of anisotropy, as illustrated in Figure 9.1 for $\gamma = 20$.

9.2. Related Work

Exact Distances Sophisticated methods using window propagation or sequence trees [MMP87, CH90, SSK*05, XW09] allow for the computation of exact geodesic distances on triangulated surfaces. Note that this exactness is with respect to the piecewise-linear surface specified by the mesh $\mathcal{M}$. In geometry processing scenarios where $\mathcal{M}$ itself is an approximation of a (piecewise) smooth manifold, the expense of employing such exact algorithms can be futile depending on the application. This holds even more when we turn our focus to non-standard metrics which are also specified only approximately, e.g. discretely per mesh element.

Graph Approximation Dijkstra's classical shortest path algorithm computes shortest paths and distances in graphs. By choosing an appropriate graph and suitable edge weights we can use the corresponding weighted graph distance to approximate distances on $\mathcal{M}$. In the simplest case this graph is the 1-skeleton, i.e. the edge graph, of the mesh $\mathcal{M}$, where the edges are weighted by their length. Higher accuracy, and actually an arbitrary balance between speed and accuracy, can be achieved by constructing a graph with additional Steiner vertices on $\mathcal{M}$'s edges and edges across $\mathcal{M}$'s faces [Lan99, LMS97, KS00]. The addition of edges between non-adjacent but nearby vertices [CBK12] allows for faster computations and (at comparable graph size) higher accuracy, but on the downside no arbitrary balancing is possible.

Consistent Approximation In contrast to these Dijkstra-based approaches, so-called *Fast Marching* methods compute and propagate distances not only along edges of a graph, but also "continuously" across the faces of a triangle mesh [KS98, SV00, Tsi95]. By appropriate choices of the per-face propagation rules the approximation can be made consistent – in the sense that the results could be driven towards the exact solution by refining the mesh. For this, $\mathcal{M}$ needs to be an acute triangulation, no obtuse inner angles are allowed. As this is hardly ever the case for unstructured meshes in practice, techniques that add additional virtual edges/triangles to be considered during the computation have been presented as a remedy [KS98, SV04, YSS*12]. Alternative non-linear propagation rules have been proposed [NK02, TWZZ07], which can typically increase accuracy in practice – although at the expense of losing consistency [WDB*08].

Non-Propagative A very different approach has been presented by Crane et al. [CWW13]. An approximation to the intrinsic distance field of a source is computed by means of solving two global linear systems instead of explicitly propagating distances from the source over the surface. An interesting property is leveraged by the information about sources appearing only on the right hand side of the system: after a pre-factorization, distance fields for different sources can be computed very efficiently (basically in linear time).

9.3. Anisotropic Metrics

We consider a discretized setting where, on a triangle mesh $\mathcal{M}$, an (anisotropic) norm $\|\cdot\|_g$ is specified in a sampled manner. The samples can be given per vertex, edge, or face of $\mathcal{M}$, denoted as $\|\cdot\|_{g_v}$, $\|\cdot\|_{g_e}$, or $\|\cdot\|_{g_f}$, respectively – where necessary, we can approximate one form from another via averaging/interpolation.

Looking at the application scenarios of anisotropic metrics which appeared in the literature so far, most often Riemannian metrics are dealt with. In such case, the corresponding norms can conveniently be expressed through a tensor field G as $\|v\|_{g_x} = \sqrt{v^T G_x v}$. Popular examples include:

- **Vector field tensor:** using the vectors of a tangent vector field as first eigenvector and given two (global) coefficients to be used as eigenvalues, tensors "aligned" with the field can be constructed. This is the scenario in the dual loops construction process (cf. Section 5.1.4).
- **Curvature tensor:** using the shape operator, tensors whose eigenvectors are aligned with directions of minimal and maximal curvature of $\mathcal{M}$ and whose eigenvalues are related to the magnitude of minimal and maximal curvature can be constructed. Figure 9.1 exemplarily visualizes such curvature-related tensors using ellipses.
- **Diffusion tensor:** the characteristics of the water diffusion process in biological tissue can be estimated from magnetic resonance imaging (MRI) acquisitions and be expressed as a tensor field. Note that this is typically applied in 3D volumes.

While we focus on 2-manifolds here, we show in Section 9.6.2 that our novel method is as well applicable to volumetric meshes or grids.

While such tensor based metrics are widely used, it bears noting that also more general metrics, based on non-elliptic norms, are of interest. Examples are terrain steepness profiles [LMS99], curvature (variation) minimizing metrics [YSS*12], or high angular resolution diffusion imaging (HARDI) metrics [PWT05]. We will hence keep the exposition general instead of restricting to Riemannian metrics.

9.4. Generic Adaptation

Before elaborating on the possibilities for individual adaptation of the available algorithms to a non-standard norm $\|\cdot\|_g$ in Section 9.5, we discuss a generic way (with certain limitations) in the following.

9.4.1. Discrete Metric

Typical implementations of the abovementioned distance computation algorithms use the vertex coordinates to derive metric dependent properties like edge lengths, angles, and areas. In this way computations implicitly rely on the standard metric induced by $\mathcal{M}$'s embedding in Euclidean space.

By not taking any extrinsic vertex coordinates into account, but instead relying on intrinsic edge lengths, computed according to $\|\cdot\|_g$ as $\ell_g(e) = \|\mathbf{e}\|_{g_e}$, most distance computation algorithms can directly be adapted to non-standard norms. To that end one, where required, computes angles and areas based on the intrinsic edge lengths. The intrinsic area A of a triangle with edge lengths a, b, c can be computed using Heron's formula

$$A = \frac{1}{4}\sqrt{(a+b+c)(-a+b+c)(a-b+c)(a+b-c)} \tag{9.1}$$

and the inner angle α opposing the edge with length a using the half-angle theorem

$$\tan\frac{\alpha}{2} = \sqrt{\frac{(a-b+c)(a+b-c)}{(a+b+c)(-a+b+c)}}. \tag{9.2}$$

While being extremely simple, this generic strategy has a few disadvantages:

Loss of fidelity The metric information is injected solely via the intrinsic edge lengths, which specifiy the so-called *discrete metric*[1] of the mesh. While such a discrete metric captures all the information of a (sampled) Riemannian metric, we inevitably lose fidelity when discretizing a more general (non-elliptic) norm in this way.

Violation of triangle inequality The computed intrinsic edge lengths might not fulfill the triangle inequality everywhere (strictly speaking, they do not form a discrete metric in this case). We found this to rather be the typical behavior than the exception, especially for high degrees of anisotropy, as also described by Kovacs et al. [KMZ11]. For some algorithms this can be unproblematic, for others (which need to derive angles or areas from these lengths) this is catastrophic. Edge lengths $\ell'(e)$ fulfilling all triangle inequalities closest to the desired lengths could in such cases be found via a suitable inequality-constrained least-squares system:

$$\sum_e (\ell'(e) - \ell(e))^2 \to \min \quad \text{s.t.} \quad \ell'(e_i) \leq \ell'(e_j) + \ell'(e_k) - \varepsilon \tag{9.3}$$

for all edge triples e_i, e_j, e_k incident to a common face, i.e. we have three inequality constraints per face.

Low triangulation quality The intrinsic roundness of the triangles is typically very bad, especially for higher degrees of anisotropy, i.e. there are angles close to 180° as well as angles close to 0°. For some algorithms this implies a low accuracy, for others a higher runtime. A remedy can be to retriangulate the mesh by means of an *intrinsic Delaunay triangulation* as detailed in the following.

9.4.2. Intrinsic Delaunay Triangulation

Using a Delaunay retriangulation algorithm which is based on an intrinsic discrete metric [FSSB07] we can adjust the connectivity of the mesh so as to obtain a mesh whose elements are nice with respect to this intrinsic metric (cf. Figure 9.2) – to the degree permitted by the given vertex set. A complete intrinsic remeshing, which not only adjusts

[1] In literature not related to meshes the term *discrete metric* by contrast homonymously refers to a metric which is 0 or 1 everywhere.

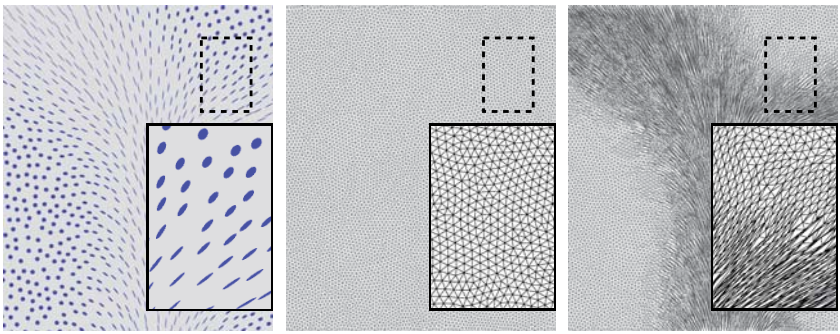


Figure 9.2.: Left: Anisotropic Riemannian metric, visualized through inverse tensor ellipses (intuitively showing the "speed profile" of the metric). Middle: Isotropic input mesh. Right: Corresponding intrinsic Delaunay retriangulation.

the connectivity but also redistributes vertices, is a theoretical option that could lead to even better results but would be rather problematic depending on the application.

While this intrinsic Delaunay Triangulation (iDT) can significantly improve the distance computation results of several algorithms, there are some drawbacks, too:

- Numerical inaccuracies can hinder the iDT's correct execution and termination, thus demand epsilon tweaking,
- The application and underlying data structures must support non-regular meshes [FSSB07] (or the algorithm must be relaxed to avoid such configurations),
- The input edge lengths must fulfill the triangle inequality everywhere,
- The worst case runtime complexity is quadratic in the mesh size.

9.5. Individual Adaptation

We now outline the algorithm specific effects when using the generic adaptation possibilities presented in the previous section and show options for improved, individual adaptation where possible.

Dijkstra

Clearly, the simplest solution to compute approximations to distances with respect to non-standard metrics is to apply Dijkstra's algorithm, taking the intrinsic edge lengths of the discrete metric into account. Note that, due to the graph nature of the algorithm, fulfillment of the triangle inequality is not required. Unfortunately, the (already in the isotropic case relatively high) average approximation error increases with increasing anisotropy, quickly leading to unacceptably low accuracy. Using the iDT, however, accuracy can be brought closer to the level of the isotropic case. Figure 9.3 illustrates this using the setup from Figure 9.2 with anisotropy $\gamma = 20$.

The Dijkstra-based method of Lanthier [Lan99] which considers additional vertices and edges across faces can be adapted to the anisotropic case as follows: instead of deriving the intrinsic lengths of the additional edges from the discrete metric, we calculate the length of an edge across face f directly using $\|\cdot\|_{g_f}$. This allows for higher fidelity in the case of non-Riemannian metrics. Figure 9.4 illustrates the behavior for increasing numbers of added vertices and edges.

Fast Marching

The Fast Marching approach [KS98] can be applied in the case of a non-standard metric by relying on the corresponding discrete metric (which must fulfill the triangle inequality everywhere as we need to derive, e.g., angles from it). The general problem is that in this metric the number of obtuse triangles is often enormous, requiring the consideration of a vast number of virtual edges. An iDT can be used to avoid this – but has its own abovementioned shortcomings. Figure 9.5 illustrates the qualities of these options. Another option is to deal with the inaccuracies due to obtuse angles using recursive improvement techniques [KSC*07]; however, this means losing consistency and convergence properties.

Sethian and Vladimirsky [SV04] proposed a more general Ordered Upwind method (OUM) which is already specifically designed for non-standard metrics. It is able to consistently deal with arbitrary anisotropy, i.e. one, in theory, has the possibility to arbitrarily increase accuracy through mesh refinement. In comparison to the other discussed algorithms, the implementation is probably the most complex one and runtime

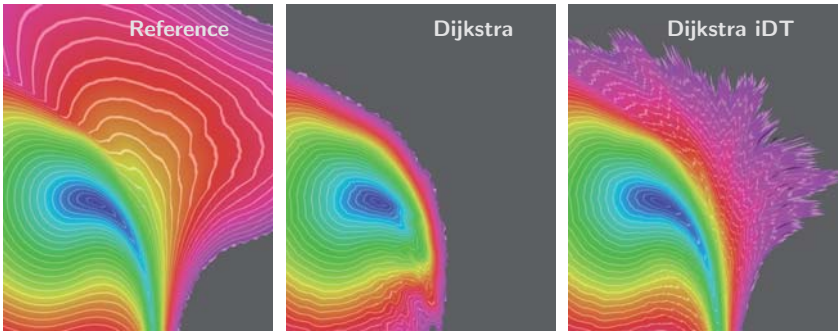


Figure 9.3.: Anisotropic distance field (up to a fixed maximum distance, so as to enhance clarity). Left: Highly accurate reference solution. Middle: Dijkstra result on an isotropic mesh. Right: Dijkstra result computed on the iDT.

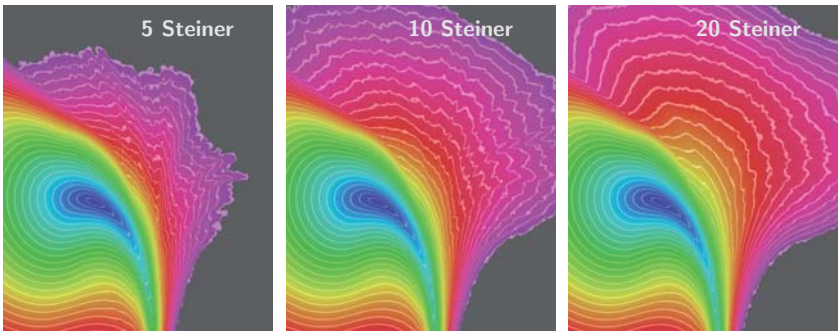


Figure 9.4.: Lanthier's method's distance field (on the iso-tropic mesh). Left: computed with 5 Steiner vertices per edge. Middle: 10 Steiner vertices. Right: 20 Steiner vertices.

and memory consumption is relatively high (cf. Section 9.7). A main factor is that, in addition to the actual anisotropic distance propagation, as an ingredient isotropic distances from every single vertex of the mesh to all of its neighbors within a radius of $\|e_{\max}\|\gamma$ need to be computed. Here $e_{\max}$ is the longest edge of the mesh and γ the anisotropy, i.e. the radius can become quite large for high anisotropies. While simple extrinsic distances can be used for efficiency, higher accuracy on non-planar meshes is achieved by computing intrinsic isotropic distances.

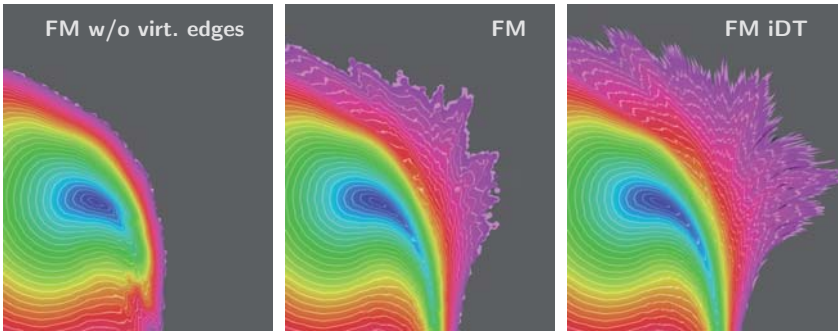
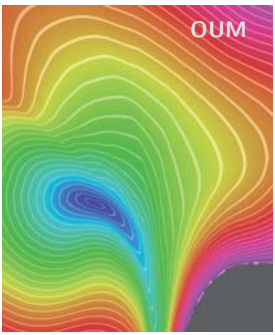


Figure 9.5.: Fast Marching distance field based on the discrete metric. Left: without virtual edges. Middle: with 13k virtual edges. Right: on the iDT, where only 6k virtual edges are necessary.

It is worth noting that, while other methods typically overestimate distances, OUM can also heavily underestimate distances, as can be seen when comparing the inset figure to the reference solution. This is due to the fact that distances are propagated over virtual triangles that span a long distance across the mesh. While the norm potentially varies on the mesh under these virtual triangles, the propagation across them is computed atomically using only the norm at one end point, easily allowing for too long as well as too short results.



Heat Method

Just like the other algorithms discussed so far, the heat method [CWW13] can be adapted to non-standard metrics by formulating it in terms of the discrete metric, i.e. based on the intrinsic edge lengths. This amounts to calculating the cotangent weights, element areas, and divergence values involved in the Laplacian and the Poisson system accordingly.

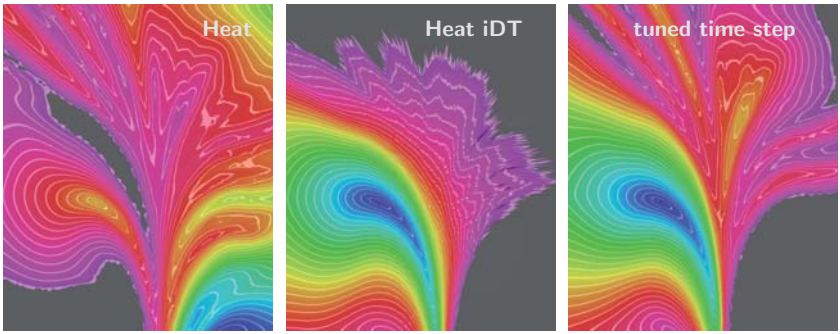


Figure 9.6.: Heat Method distance field based on the discrete metric. Left: computed on the isotropic mesh. Middle: computed on the iDT. Right: computed on the isotropic mesh but with individually tuned heat integration time step.

Unless the anisotropy is very moderate, the low intrinsic element roundness, however, does negatively affect robustness. The resulting distance fields then often show distortions and degeneracies like local minima. We observed that improved results can be achieved by individually tuning the heat integration time step instead of using the standard $c = 5$ proposed in the original publication, but found no general rule for a good automatic choice. Again, an iDT can be an option to remedy these problems (cf. Figure 9.6).

9.6. Short-Term Vector Dijkstra

We now present a novel method for the approximate computation of distances with respect to arbitrary metrics. In order to develop an intuitive understanding of the underlying principle, let us consider Dijkstra's classical algorithm again. Due to its graph nature, the distance approximations resulting from this algorithm are the lengths of edge paths that meander over the surface (cf. Figure 9.7 left) – not the lengths of true geodesic paths. They are thus rather inaccurate and also very triangulation dependent.

Instead of first measuring lengths of edges and then (scalarly) summing these, an interesting alternative is to first (vectorially) sum edges and then measure the length

of the sum. This *vector-valued Dijkstra* algorithm has been employed by Schmidt et al. [SGW06] to obtain geodesic distances for the purpose of local surface parameterization. Figure 9.7 illustrates the principle in the plane. In this planar case and with the standard metric the resulting distances are actually exact – as, independent of the path, the result is the Euclidean length of the vector pointing from source to target point. The concept can be transferred to 2-manifold meshes by performing vector addition after transferring the vectors into a common 2D reference system, which can be done in different ways [SGW06, Sch13]. In any case, a major problem is that this method is basically oblivious to holes, obstacles, and, when applied in a non-planar setting, geometric variations in the surface. Figure 9.7 right demonstrates this issue. Hence, while being adequate and efficient for computations in a local neighborhood, it is unsuitable for global distance computations on manifolds.

The idea underlying our *Short-Term Vector Dijkstra* (STVD) is to form a hybrid out of the classical scalar-valued variant and the vector-valued variant so as to combine the respective advantages. Conceptually, we equip the scalar-valued variant with a vector-valued short-term memory. In this way the meanders of the edge paths through the triangulation can locally be smoothed out without globally disregarding the surface geometry. The following pseudo code clarifies the details.

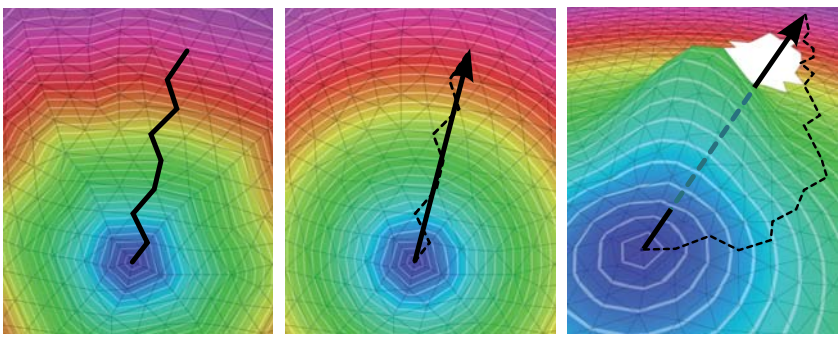


Figure 9.7.: Left: distance computation using Dijkstra's classical algorithm. Middle: computation using a vector-valued Dijkstra variant. Right: shortcomings of the vector-valued Dijkstra variant: it is oblivious to holes, obstacles, and geometric variations.

Algorithm: Short-Term Vector Dijkstra (STVD)
Input: polygon mesh, metric g, a vertex designated *source*
Output: vertex based field of geodesic distances to *source*

---

source.dist $\leftarrow 0$ all other distance values initially ∞
Q.insert(*source*) priority queue Q ordered by distance
while not Q.empty
  $v \leftarrow Q$.extract_minimum() get min. dist. vertex out of Q
  v.final $\leftarrow$ true
  for all w adjacent to v **where not** w.final
    if update_dist(v,w) $<$ w.dist
      w.dist $\leftarrow$ update_dist(v,w)
      w.pred $\leftarrow v$
      if not w in Q
        Q.insert(w)

This looks very much like the standard Dijkstra algorithm and indeed, if we use the following version of the update_dist function this is exactly what we get.

Function: update_dist(v,w) original Dijkstra version

---

return v.dist + $\ell_g(v, w)$ add length of edge (v,w) w.r.t. g

---

By instead exploiting a vector-valued short-term memory (a window of k preceding edge vectors), we obtain our STVD algorithm using the following variant of the distance update function:

Function: update_dist(v,w) our STVD version

---

tmp $\leftarrow w$.pred
w.pred $\leftarrow v$
dist $\leftarrow \min_{i=1}^{k} w.\text{pred}^i.\text{dist} + \ell_g\big(\sum_{j=1}^{i}(w.\text{pred}^{j-1}, w.\text{pred}^j)\big)$
w.pred $\leftarrow$ tmp
return dist

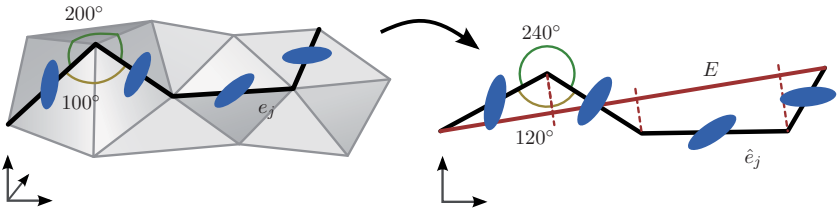
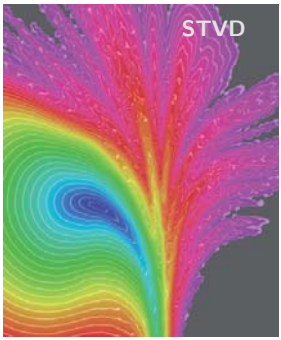


Figure 9.8.: Unfolding of edge chains to the plane. Edge lengths and relative 1-ring angles are preserved. The sum vector E (red) is then subdivided according to the ortho-projection of the individual edges. The resulting portions are measured by the respective norms $\|\cdot\|_{g_e}$ – here visualized as tensor ellipses (blue) – and their lengths summed to get ℓ_g.

where the predecessor relation is recursively defined as $w.\text{pred}^{i+1} = w.\text{pred}^{i}.\text{pred}$, with $w.\text{pred}^{0} = w$. It remains to be clarified how the edges $e_j = (w.\text{pred}^{j-1}, w.\text{pred}^{j})$ are summed vectorially and how the length ℓ_g of the sum with respect to g is measured:

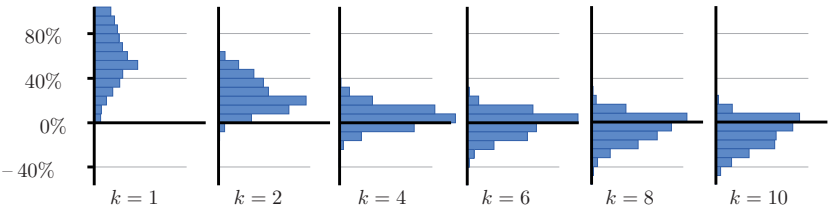
The norms $\|\cdot\|_{g_e}$ live in tangent planes, i.e. they allow to measure vector lengths in 2D. We thus unfold the chain of (directed) edges e_j into a common plane ($\mathbb{R}^2$) while preserving edge lengths and relative 1-ring angles. Figure 9.8 illustrates this. Note that no actual embedding of the mesh is required for this. The norms $\|\cdot\|_{g_e}$ (represented relative to their corresponding edges) now allow us to approximately measure the length of the sum $E = \sum_j \hat{e}_j$ of the unfolded edge vectors $\hat{e}_j$ in 2D. As these norms can differ, we decide how large portion of E is measured by which norm based on the edges' (signed) orthogonal projections onto E. Concretely, we use $\ell_g(\sum_j e_j) = \sum_j \hat{e}_j^\top \bar{E}\, \|\bar{E}\|_{g_{\hat{e}_j}}$ (clamped to $\mathbb{R}^{\geq 0}$), where $\bar{E} = E/\|E\|$, in the above distance update function.

It is worth noting that we can choose between two alternatives regarding the representation of the conceptual memory. It can be represented either explicitly by storing vectors of the last $k-1$ edges $(v.\text{pred}^{j}, v.\text{pred}^{j+1})$ at each front vertex v (in its local 2D coordinate system), or implicitly by gathering these vectors using a short back-trace via the predecessor relation when we need them in the distance update function.



With an appropriate choice of the short-term memory's depth k, the accuracy of the results is immensely improved. The inset illustrates this for $k = 10$ (detailed quantitative results are provided in Section 9.7). Despite the lack of smoothness in some regions, the result is closer to the reference than the alternatives from Section 9.5 (except Figure 9.4 middle, right).

Note that for $k = 1$ STVD is equivalent to Dijkstra's classical algorithm (generally overestimating distances), while for $k \to \infty$ it becomes an all vector-valued variant (which tends to underestimate unless the surface is developable and free of holes). This can also be observed in the following histograms which show the signed relative error distribution over all vertices of all examples from Section 9.7 (for $\gamma = 10$):



9.6.1. Speed vs. Accuracy

Some of the discussed algorithms allow to trade speed for accuracy at the user's discretion. By using a higher number of Steiner vertices in the edge-subdivision approach [Lan99], or by refining the input triangle mesh using some steps of 1-to-4 splits prior to the application of the Ordered Upwind method [SV04] (or, in the case of a Riemannian metric, also the Fast Marching adaptation) higher accuracy can be achieved – at the cost of quickly increasing runtime. Note that such property is not to be taken for granted: for instance the accuracy of distance computations using Dijkstra's algorithm does not generally increase under mesh refinement.

We empirically observed that such property can also be established for our STVD. To this end we need to achieve two seemingly contradicting goals: k needs to be increased so as to increase the angular resolution of the distance propagation, while the lengths of the used vector sums need to be decreased so as to reduce the approximation errors of the unfolding-based measurement. We can achieve both by refining the mesh using 1-to-4 splits (reducing edge lengths by a factor of 2) while increasing k by a factor < 2. The following table shows the decreasing mean error for increasing levels of refinement on three exemplary models ($\gamma = 20$):

Level	k	Elk	Gargoyle	RockerArm
0	7	9.8%	25.7%	15.3%
1	10	5.1%	13.4%	11.6%
2	14	2.5%	8.2%	6.5%
3	20	1.5%	4.3%	3.9%

More interesting than this possibility, however, we deem that, even in the case of strong anisotropy, STVD achieves reasonable, relatively good results already on unrefined meshes of resolutions typically encountered in the Computer Graphics and Geometry Processing field (cf. Section 9.7).

9.6.2. Genericity

Polygonal Meshes It is worth noting that STVD does not rely on the property that $\mathcal{M}$ is a triangle mesh, i.e. it can also be applied to general polygonal meshes. Many other methods are designed specifically for triangle meshes and would need to be adapted – or the polygon mesh be triangulated.

3D Meshes While our focus here is on 2-manifold surface meshes, an interesting fact about STVD is that it can directly be applied also to volumetric meshes in $\mathbb{R}^3$ – whether they consist of tetrahedral, hexahedral, or general polyhedral cells. We simply skip the edge vector unfolding to the plane and sum the 3D edge vectors directly (cf. Figure 9.9). While also other methods can potentially be generalized to this setting, this is less trivial. Challenges lie in the correct determination of virtual simplices for the

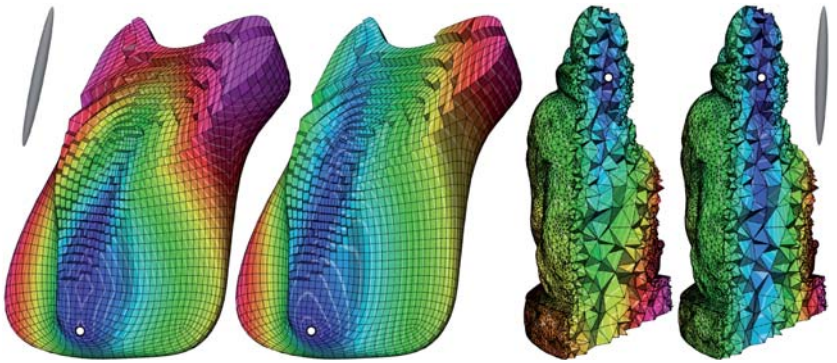


Figure 9.9.: Volumetric distance fields computed in a hexahedra and a tetrahedra mesh (sliced open to expose the interior). The constant Riemannian metrics used (depicted by ellipsoids next to the models) have an anisotropy of 12. Just like in the surface mesh case, Dijkstra's algorithm heavily overestimates distances, especially in vertical direction where true distances are shorter due to the anisotropy.

OUM or FMM, establishing an intrinsic Delaunay tetrahedralization or fixing violated tetrahedra inequalities, or in handling the large number of additional edges, which in the case of straightforward generalization of Lanthier's concept to 3D grows quartically with the number of Steiner vertices per edge.

9.7. Results

In the previous sections we have discussed the qualitative differences of the approaches that are available for the computation of anisotropic distances. In order to get a quantitative understanding for the accuracy and runtime performance of all these options, we implemented all variants and performed extensive experiments using various kinds of input meshes (cf. Figure 9.10, 9K-183K triangles), metrics (vector field based as well as curvature tensor based), anisotropies (uniform as well as varying $\gamma(x)$), and algorithm parameters.

The reference solutions we compare against have been computed using the edge-subdivision method of Lanthier [Lan99] with 200 Steiner vertices per edge (and runtimes of up to several hours for a single distance field). In this setting for every single triangle there are more than 120,000 virtual edges crossing it, resulting in a highly accurate approximation of the true distances. We show the results using error-over-runtime plots in Figure 9.11 (and using error histograms in the supplemental material). Intuitively, the closer a method lies to the bottom left, the better – as this means that a low error (high accuracy) is achieved in a short time. These plots further allow to quickly see what other options apart from the "best one" are available, e.g. how much more accuracy can be achieved by spending how much more time.

Regarding our STVD algorithm, we see that it lies in the bottom left region across the different levels of anisotropy, i.e. it achieves good results in short time when compared to the other options – especially for higher anisotropies. Good standard values for k can also be read from these plots: from around 5 for low anisotropy to around 10 for anisotropy 50. Note that such high anisotropy is not only of theoretical interest. In fact, an anisotropy of 30 is the standard we made use of in the dual loops construction process.

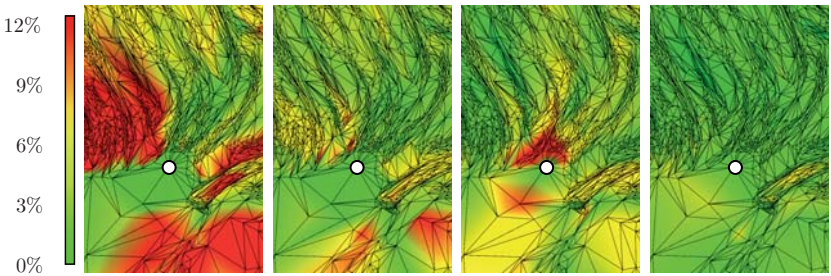
For lower anisotropies, the Fast Marching method applied to the intrinsic Delaunay triangulation of a subdivided version of the input mesh is very competitive. Note that this strategy requires several steps of preprocessing: 1) mesh subdivision, 2) discrete metric computation, 3) reestablishing triangle inequality fulfillment, and 4) iDT construction. This also implies the drawbacks discussed in Sections 9.4.1 and 9.4.2 and can take a considerable amount of (possibly amortizable) time. By contrast, STVD operates

Figure 9.10.: Models used for the evaluation, depicted with example fields.

directly on the input mesh without the need for any preprocessing and without implying additional complexity.

When a higher level of accuracy is required and more time is available, the edge subdivision method of Lanthier [Lan99] proves to consistently provide a good option.

In isotropic scenarios the advantage of STVD over, e.g., Fast Marching or the Heat Method diminishes. However, we observed that it is typically still very significant when dealing with meshes with badly shaped elements. This is illustrated here on an example mesh ($\gamma = 1$), where the error (w.r.t. exact geodesic distances [SSK*05]) is visualized:



9.8. Discussion

We have explored and discussed numerous options for the computation of distance fields with respect to general anisotropic metrics, based on generic as well as specific adaptations of known algorithms. We compared all these in terms of their accuracy and runtime performance and enriched this zoo of methods with our Short-Term Vector Dijkstra. Despite its simplicity, this method proved to provide a very interesting novel option, allowing to quickly compute results with practical accuracy without the need for complex optimization or costly preprocessing.

In the future ways to enhance the smoothness of the results could be explored. Possibilities could be the use of averaging schemes during propagation [Sch13] or of a kind of "gradually fading memory" instead of a hard limit k. The use of a locally varying k (based on local feature size, anisotropy, mesh density) is another interesting direction.

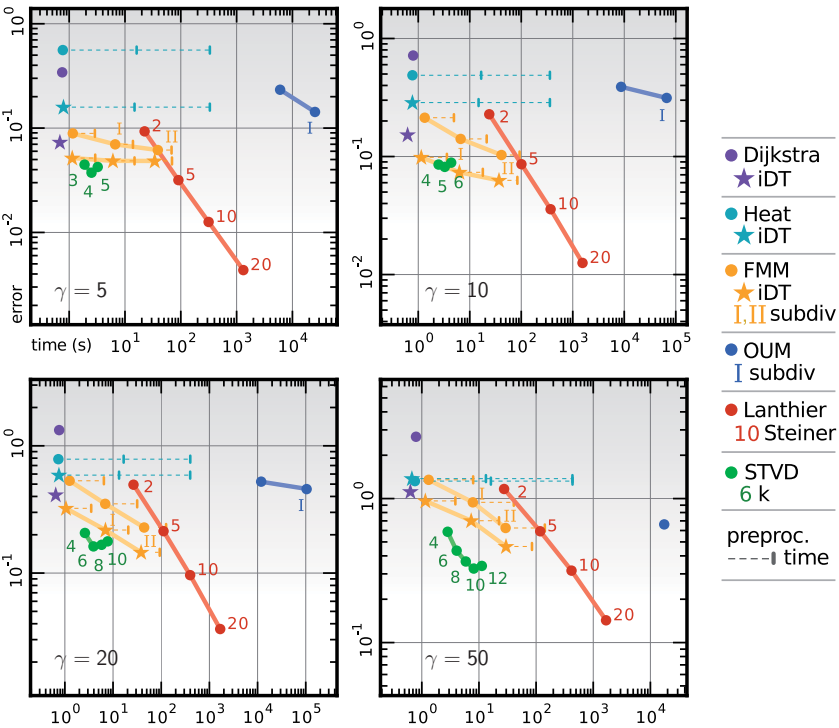


Figure 9.11.: Error-over-runtime plots of the results for various degrees of anisotropy (γ). Shown is the mean rel. abs. error (computed over all vertices) on the vertical axis, and the summed runtime (sec.) for the computation of a complete distance field per input test case (14 mesh/metric combinations). Mesh subdivision (1-to-4 split) levels are indicated by *I* and *II*; numbers indicate the STVD parameter k or the number of edge Steiner vertices, respectively. Preprocessing time (shown by dashed bars) includes triangle inequality fixing and (where applicable) subdivision and iDT. For the Heat method the first segment of the bar additionally indicates the time for setup and pre-factorization of the employed linear systems. Violated triangle inequalities have been fixed 1) for the heat method as described in Section 9.4.1 using IPOPT [WB06]; 2) for the FMM using a simpler (not least-squares optimal) strategy of iteratively adjusting individual triangles, as on the subdivided meshes the solver takes unacceptably long.

10. Meshless Geodesics

Note: *This chapter is based on [CK11].*

A basic assumption we made in all the presented algorithmic parts is that the input surface is given in form of a triangle mesh. When the ultimate goal is to generate a structured representation for digitized (e.g. scanned) surfaces, for instance in form of a quad mesh, multiblock grid, or spline network (cf. Chapter 1), one might ask whether it is reasonable or necessary to first construct a triangle mesh out of a point cloud or a set of range images. This question has already been asked for the case of quad mesh generation and solutions been presented [LLZ*11, PTSZ11] that circumvent the triangle mesh construction by directly operating on a point cloud or range images, i.e. geometry without complete mesh connectivity information.

While the generalization of the entire ensemble of our pipelines' algorithmic parts to such inputs is beyond the scope of this thesis, in the remainder of this chapter we show how geodesic distances and paths can be computed on point clouds or incomplete meshes as this is the next most important step towards the goal of being able to perform layout construction directly on such data: the most common building blocks in our algorithms are cross field generation, global parametrization, and geodesic computations – where adaptations of the former two have already been described [LLZ*11, PTSZ11].

Note that this generalization does not only leverage the use of scanned point cloud data: "real-world" triangle meshes, i.e. meshes that user's encounter in practice, do not always fulfill the basic assumptions we made about our methods' input meshes, namely that they are well-structured and manifold with complete connectivity information. Depending on their origin they may exhibit several kinds of defects – holes, gaps, (near-)degenerate polygons; or they might even be just a soup of polygons, completely lacking connectivity information. While a considerable number of methods that aim at repairing mesh defects have been presented, unfortunately there is no general solution.

As detailed in our recent survey [ACK13] the automatic repair problem is hard, ambiguous, and naturally ill-posed in most cases. Hence all repair approaches have certain advantages and disadvantages and often user interaction and tedious manual effort is required to obtain a clean mesh in the end. The fact that our adaptation does not rely on explicit connectivity information enables its application to the general class of "imperfect meshes" – of which point clouds and range image sets are basically special cases.

In detail, we present a computational framework that allows for the computation of meaningful approximate intrinsic distances and geodesic paths on meshes with all kinds of defects in a way tolerant to these defects (cf. Figure 10.1). We do this without explicitly repairing the mesh, circumventing the abovementioned problems. In the case of severe mesh defects, e.g. large missing parts, the computed distance fields might of course be inconsistent with those of the object that is actually *meant* to be represented by the partial data – in particular we do not propose new disambiguation or "defect hole" – "feature hole" distinction strategies.

Figure 10.1.: An intrinsic distance field and a geodesic path computed on an imperfect mesh in a defect-tolerant way.

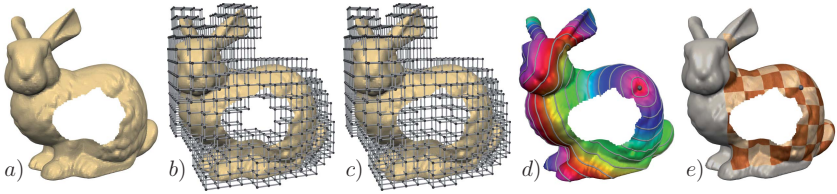


Figure 10.2.: Overview of our approach: a) Input mesh with defect (large artificial hole for demonstration), rendered with backface culling. b) Initial cubical complex constructed for this mesh (at a low resolution of 16^3 for illustration). c) Complex after applying topology-sensitive morphological operators; the hole is now bridged. d) Visualization of a geodesic distance field (with isolines) emanating from a point source, computed on the complex (at a resolution of 64^3), and mapped to the input mesh by interpolation. e) Application example: defect-tolerant decal textured onto the surface using a local geodesic parameterization.

The basic idea is to abstract from the mesh structure (and all its potential defects) and to perform all computations discretely in a crust volume tightly restricted to the spatial regions occupied by elements of the input. It has been proven that the extrinsic distance field in such crust volumes converges uniformly to the intrinsic distance field of the surface they bound with increasing tightness [MS01]. We show ways to perform the necessary computations in a memory-efficient manner such that tightness can be achieved by using high resolutions. The discrete structure readily allows for the application of topology-sensitive morphological operations [BK05] to make computations tolerant to gaps and holes. The 3D variant of the STVD algorithm described in Section 9.6.2 or the Fast Marching method [Set95] can then be applied to efficiently generate distance fields. The examples shown in the following have been generated using the Fast Marching method. Due to the abstraction from the input, applicability is not limited to polygon meshes; other representations like point sets, implicit functions, or NURBS patches can be handled as well.

10.1. Overview

The framework we present takes inconsistent raw polygon meshes (cf. Figure 10.2 a) as input and allows for the computation of (1) intrinsic distance fields with point or polygonal sources, (2) geodesic paths between surface points, and (3) various types of local surface parameterizations. Independent of which computations are to be performed for a specific application, the first step is to abstract from the mesh representation to a cubical complex, i.e. a Cartesian grid tightly restricted to the spatial regions occupied by the mesh elements (cf. Figure 10.2 b). To make further computations tolerant to gaps and holes we next apply topology-sensitive morphological operations [BK05] to this complex. This closes all gaps and holes of sizes up to a user-specified tolerance threshold and yields the final complex (cf. Figure 10.2 c). Details on this construction are presented in Section 10.2.

Source points or curves for the computation of distance fields and geodesic curves are mapped into the complex (cf. Section 10.3) to obtain initial conditions for the subsequent Fast Marching (cf. Section 10.3.1). Finally the mapping of the results back onto the input mesh is detailed in Section 10.3.3 (cf. Figure 10.2 d). In Section 10.4 we present several ways to construct local surface parameterizations in defect-tolerant ways using the introduced framework and show application to texturing of arbitrary meshes (cf. Figure 10.2 e). Further results and analyses are provided in Section 10.6.

10.2. Mesh Abstraction

Given an input mesh $\mathcal{M} = (F, E, V)$ consisting of sets F, E, V of faces, edges, and vertices respectively. We do not want to make any assumptions about the integrity of $\mathcal{M}$, i.e. it may contain holes, gaps, singularities, degeneracies, or missing connectivity information. Hence, we first abstract from this mesh to a cubical complex representation. The idea of using some kind of discrete abstraction of polygonal geometry has been applied for the same reason in various fields, from simplification [ABA02] over vectorization [MZL*09] to model repair [BPK05], to name only a few.

10.2.1. Initial Complex Construction

The cubical complex we use is essentially a cut-out of a three-dimensional Cartesian grid such that all elements of F, E, and V are contained in the union of its cells. It should be minimal, i.e. restricted to the regions occupied by the mesh elements as tightly as possible. Hence, the construction of this initial minimal complex $\mathcal{C}_\mathcal{I}$ basically corresponds to the three-dimensional rasterization (or "voxelization") of the mesh elements, since the obtained voxels directly correspond to the 3-cells of the desired complex $\mathcal{C}_\mathcal{I}$. Details on efficient implementation are postponed to Section 10.5. We base the further description on the cubical complex notation since also the 0-cells and 1-cells of the complex are involved in the computations. In the following we will refer to the 0-cells of the complex as *nodes*, to the 1-cells as *arcs*, to the 2-cells as *walls*, and to the 3-cells simply as *cells*. Further, by $N(c)$, c being a cell of $\mathcal{C}$, we denote the set of nodes incident to c, and $N(C) = \bigcup_{c \in C} N(c)$ for a set C of cells.

Some computations can benefit from normal information at the nodes. For each node we average the normal vectors of the faces (or vertices) intersecting the eight incident cells to obtain an estimated normal vector. Since this is only meaningful if the set of normal vectors is coherent at least to some extent, we do only store such an estimated normal vector at nodes where the cone spanned by the set of normal vectors has a non-reflex opening angle – which can conservatively be estimated by all pairs of these vectors spanning angles smaller than $2/3\,\pi$ for simplicity. For the estimation process the normal vectors of the input have to be oriented consistently. If this is not the case it can for instance be enforced by the method of Borodin et al. [BZK04].

10.2.2. Morphological Operations

Bischoff et al. [BK05, BPK05] successfully applied morphological operations to voxel representations in the context of 3D model repair. By applying the discrete morphological dilation operation to our initial complex $\mathcal{C}_\mathcal{I}$ we can easily close holes and bridge gaps up to a user-specified width. In principle holes of any size can be bridged in this way, however, note that intentional feature holes, constrictions, or tunnels up to the chosen size are also closed. If these can safely be distinguished from defect holes in a specific

scenario, dilation can of course be restricted to defect hole boundaries, e.g. as done in [BPK05], to alleviate this behavior.

Performing distance computations on the resulting dilated complex $\mathcal{C}_\mathcal{D}$ would, however, result in significantly lowered accuracy [MS01, MS05]. Hence we apply a morphological erosion operation on $\mathcal{C}_\mathcal{D}$ to obtain the final complex $\mathcal{C}$. To prevent this operation from tearing closed holes and gaps open again, a topology-preserving variant [BK05] is employed. This operation removes cubes only if this does not change the digital topology (defined via wall-adjacency, or *6-neighborhood*) of the complex thereby leaving a minimal sheet of cubes in holes and gaps. The entire process is illustrated in Figure 10.3, implementation details are given in Section 10.5.

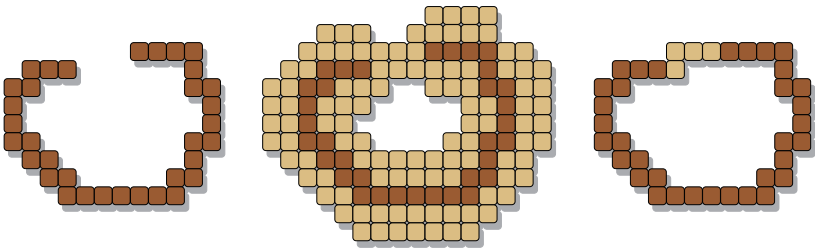


Figure 10.3.: 2D schematic example of the employed morphological operators *dilation* (middle) and *topology-preserving erosion* (right), filling holes up to a specified size.

10.3. Geodesic Computations

Having obtained the computational domain in form of a cubical complex $\mathcal{C}$ as described in the last section, we can now perform approximate geodesic distance computations by Fast Marching (FM) [Set95, DC00]. This method, applied to the complex, will compute a distance field $d : N(\mathcal{C}) \to \mathbb{R}^+$.

A distance field d usually emanates from (a set of) source points or curves on the input surface, i.e. on $\mathcal{M}$ (but sources in free space can be handled as well). To set initial values for the front propagation of the FM method this information needs to be transferred into $\mathcal{C}$.

For the set S of all specified point and curve sources of d we determine the set C_S of cells of $\mathcal{C}$ they intersect and initialize the distance values $d(n)$ for all $n \in N(C_S)$. We set $d(n) = \min_{s \in S} dist(n, s)$. If an averaged normal vector is not available we choose $dist(\cdot, \cdot)$ to be the Euclidean distance. Otherwise we can enhance accuracy by calculating $dist(n, s)$ as the Euclidean distance between n and the orthogonal projection of s onto the tangent plane T_n at n. This suppresses the surface-orthogonal distance component merely introduced by the nodes not lying directly on the surface. The distance value of all other nodes is yet undefined, i.e. we initially set $d(n) = \infty$ for all $n \notin N(C_S)$.

10.3.1. Fast Marching

Starting from the initialized nodes $N(C_S)$ the FM method [Set95, DC00] can now be applied to perform a front propagation over all other nodes of $\mathcal{C}$ in order to determine distance values $d(n)$ for all nodes n that closely approximate their defect-tolerant geodesic distance to the set S of sources. The FM method keeps the nodes that are part of the current front in a priority queue, sorted by $d(\cdot)$, and always removes the node with lowest distance while updating the distance values of its adjacent nodes and adding them (back) to the queue. Since this propagation is done in upwind direction, distance values of cells removed from the queue can justly be considered final.

In the FM front propagation process a distance value update for a node is computed from the distance values at up to three adjacent nodes by a form of extrapolation. Originally, a gradient-based first-order update rule was proposed. Higher-order [Set99, HF07] rules can be employed to increase accuracy. Especially for the case of circular distance fields of point sources the first-order rules overestimate distances as pointed out by Novotni and Klein [NK02]. They show that higher accuracy is achieved by rules that take the particular circular nature of the front into account. We next extend these rules to our three-dimensional setting.

On a triangle mesh, given two vertices of a triangle with known intrinsic distances to the source, [NK02] determine a virtual point source in the plane of the triangle that maintains these distances. The distance value of the third vertex can then be updated to its Euclidean distance to this virtual source. In our three-dimensional domain, to update the distance value $d(n)$ of a node n from three adjacent nodes n_0, n_1, n_2 with already computed distance values, we determine a virtual point source s in space by

trilateration: the points s_1 and s_2 are found as the two points satisfying the three sphere equations $d(n_i)^2 = ||x - n_i||^2$, $0 \leq i \leq 2$. Since propagation proceeds in upwind direction, the one with larger distance has to be chosen, i.e. we apply the update $d(n) = \max(||n - s_1||, ||n - s_2||)$. When less than three adjacent nodes have distances available we fall back to lower dimensional lateration.

10.3.2. Polar Angle Propagation

Additionally to computing a distance field d of a point source an angular coordinate field θ can be constructed to obtain a (local) polar surface parameterization (d, θ). Schmidt et al. [SGW06] construct both fields on meshes approximately by a modified version of Dijkstra's shortest path algorithm, essentially unwrapping the surface into the tangent plane at the source. The accuracy of the radial coordinate computed in this way, however, usually cannot compete with that of the FM method. We incorporate the general idea into the FM method to simultaneously construct angular coordinates θ. For this construction normal vector information must be available (cf. Section 10.2.1). Nodes that lack normal information simply inherit the normals from their predecessors in the front propagation process. This proved to be sufficient in our experiments and compared to more sophisticated global normal diffusion methods does not hinder a sweeping implementation (cf. Section 10.5.2).

Let T_s be the tangent plane of the surface at the source point s and a the polar axis in that plane, defining the orientation of angle 0. Furthermore let $\mathrm{uv}(d, \theta) = (d\cos\theta, d\sin\theta)$ denote the 2D Cartesian vector defined by d and θ in T_s and $\mathrm{angle}((u, v)) = \arctan(v/u)$ the angular coordinate of the vector (u, v) in the polar system defined by T_s and a.

During an update step of the FM method, updating node n based on the distance values of one, two, or three adjacent nodes $n_0, \ldots, n_m$ as described in the last section, $\theta(n)$ is set as follows: we first choose one of the updating nodes n_i and take its so-called inverse exponential map vector $\exp_s^{-1}(n_i) = \mathrm{uv}(d(n_i), \theta(n_i))$. The ideal choice is $n_{min} = \mathrm{argmin}_{i \in \{0,\ldots,m\}} d(n_i)$ since this can be expected to have lowest accumulated error in its d and θ values. Then we "unwrap" the vector $n - n_{min}$ into the tangent plane T_s by first projecting it orthogonally onto $T_{n_{min}}$ (to get rid of the orthogonal component merely introduced by n not lying directly on the surface), then rotating it into T_s around the axis orthogonal to the normals at s and n_{min} (*hinge map*), and finally

transforming it into a 2D vector in the coordinate system of T_s. By adding these two vectors we obtain an approximation for the angular component of $\exp_s^{-1}(n)$, i.e. we set $\theta(n) = \text{angle}(\exp_s^{-1}(n_{min}) + TRP(n - n_{min}))$, where T, R, and P are the transformation, rotation, and projection operations. $d(n)$ is computed by the FM update rules as before for accuracy. The adjacent figure shows a visualization of such an angular field on a curved surface, including isolines in radial and axial directions.

10.3.3. Interpolation

After distance values and possibly angular values have been computed at the nodes of $\mathcal{C}$ we want to transfer this information back onto the input mesh $\mathcal{M}$. Let P be the set of points on $\mathcal{M}$ at which these values shall be made available (in most applications this is simply the set V of vertices). While it can be expected that most accurate results are achieved by integrating the points of P into $\mathcal{C}$ as virtual nodes to compute the values by the described FM update rules, this proved to result in slight discontinuities between points that are nearby but fall into different cells. By contrast, the application of trilinear interpolation leads to smooth results due to its very nature. Hence, we interpolate the values at a point p of P from the eight nodes incident to the cell of $\mathcal{C}$ that includes p.

In case a Cartesian (u, v) parameterization is to be constructed from a computed polar parameterization, it is beneficial to perform the transformation already at the nodes and then interpolate the (u, v) coordinates. This avoids special case handling near the singularity at the field's pole where trilinear interpolation is unsuited for angles and distances.

10.3.4. Geodesic Paths

Shortest geodesic paths between two points p and q on the surface can be constructed using a gradient descent procedure. First the distance field for source p is computed. Then, starting from q, a piecewise linear path through $\mathcal{C}$ can be constructed by pro-

ceeding stepwise in direction of the negative gradient of this field. The constructed path lies in $\mathcal{C}$, i.e. in the surrounding space of $\mathcal{M}$; if a path *on* $\mathcal{M}$ is desired, we perform a projection where possible – in hole regions the path simply remains in the cell sheet that survived erosion.

10.4. Parameterization

Geodesic computations have proven to be a valuable tool to constructively generate local surface parameterizations [ZG04, GGGZ05, SGW06, BMBZ02]. We now present various methods to construct such parameterizations as they allow us to comprehensibly visualize the strengths of our method in Section 10.6. Figure 10.4 shows examples for comparison.

Center Point Parameterization A parameterization around a point can be constructed by computing a radial and angular coordinate field for this source. The resulting polar parameterization can be transformed into a Cartesian (u, v) parameterization with a user-specified orientation and scale, e.g. for applying decals onto curved surfaces in an intuitive way [SGW06]. Due to the defect-tolerance of our approach such decal application can be performed across non-connected mesh components, bridging gaps and holes, thus on a much broader range of meshes. The distortion – necessarily introduced on surfaces with non-zero Gaussian curvature – is minimal in the center and typically grows with increasing distance depending on the curvature distribution.

Boundary Curve Parameterization More flexibility and a less center-biased distortion distribution is achieved by specifying four curves u_0, v_0, u_1, v_1 forming a quadrilateral on the surface and for each computing the distance field emanating from it. The distance fields d_{u_0} and d_{u_1} of opposite curves u_0, u_1 can then be blended into d_u by $d_u = d_{u_0}/(d_{u_0} + d_{u_1})$, analogously for d_v, and these two resulting fields d_u and d_v be taken as u and v coordinates of a parameterization that is aligned with the surface quadrilateral, mapping this region to the domain $[0, 1]^2 \subset \mathbb{R}^2$.

Corner Point Parameterization Specifying desired parameterization boundary curves on a surface can be tedious. Furthermore, in order to reduce distortions in the parameterization generated as described in the last section these curves should be geodesic paths between their endpoints. We can simplify the specification task to choos-

ing the four corner points of the desired quadrilateral and then automatically determine geodesic paths between the points (cf. Section 10.3.4) to construct a geodesic quadrilateral as basis for the boundary curve parameterization.

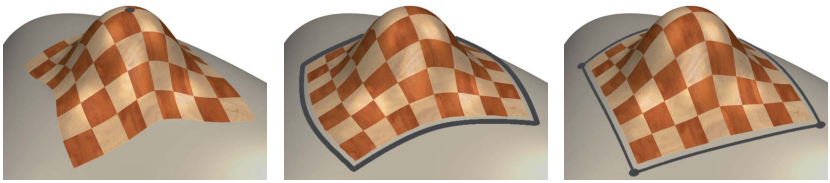


Figure 10.4.: Parameterization examples: center point parameterization (left), boundary curve parameterization from user-specified quadrilateral (middle), and corner point parameterization from automatically determined geodesics (right).

10.5. Implementation Details

We now provide some details on the efficient implementation of the computations described in the last sections. In order to enable broad applicability an implementation variant optimized for performance as well as one optimized for memory efficiency is presented.

10.5.1. Cubical Complex Construction

To allow for high resolutions without excessive memory requirements we employ an octree $\mathcal{O}$ that is adaptive in multiple ways. This further allows for the efficient establishment of correspondences between elements of $\mathcal{M}$ and $\mathcal{C}$, required to transfer information between the two representations.

We start by defining the cubical root cell of $\mathcal{O}$ to include the bounding box of $\mathcal{M}$. The elements of $\mathcal{M}$ are then "inserted" into $\mathcal{O}$, intersected cells are refined up to a user-specified maximum level l and marked as *solid*. The set of these solid cells then forms the voxel representation of $\mathcal{M}$. The vertices of V (or the set P of points for which

distance field values shall be computed, cf. Section 10.3.3) can optionally be recorded in the containing leaf cells, allowing for direct access during interpolation. Methods for the efficient traversal of octrees have been presented [FP02, Sam89], and by installing so-called *ropes* to explicitly link neighboring cells [MB90], the computational cost of cell navigation can be reduced, trading memory requirements for efficiency.

Efficient dilation operators for octrees with on-demand refinement of cells to level l are presented by Bischoff et al. [BPK05]. Let λ denote the width of a cell on level l. Given that holes and gaps up to a width of ρ shall be considered unwanted, we need to determine how many layers of cells need to be added to close these. Due to the discrete setting the dilation process has a directional bias. Slowest growth happens in space-diagonal direction where γ dilation steps bridge gaps of widths up to $2\,\gamma\,\lambda/\sqrt{3}$. Hence, to ensure closing of all holes of widths up to ρ we choose $\gamma = \rho/(2\,\lambda/\sqrt{3})$.

To now remove all dilated cells except for thin hole and gap bridging sheets we perform a topology-preserving erosion [BK05]. In contrast to the original description we do not only apply γ erosion steps but keep eroding until no more dilated cells can be removed without changing the digital topology of the voxel set specified by solid and dilated cells. This eliminates the "*closing*" character of the operations. Afterwards the set of solid and remaining dilated cells corresponds to the desired complex $\mathcal{C}$.

Instead of extracting the desired cubical complex $\mathcal{C}$ from $\mathcal{O}$ to represent it by a separate data structure we directly represent it by the octree. Unfortunately the FM front propagation operates on the graph of nodes and arcs of $\mathcal{C}$ – which are not explicitly represented in the octree data structure. However, we can establish a graph isomorphism between the nodes and a certain set of octree cells. This isomorphism identifies an octree cell with the node in its upper-right-back corner. The set of octree cells that is required for this purpose contains the solid cells, the dilated cells, and all cells which are incident to the lower-left-front corner (any other pair of opposite corners could have been chosen as well) of a solid or dilated cell (refined to level l if not yet the case). The 6-neighborhood graph of these cells, directly represented by the ropes, can then be used instead of the nodes-arcs graph.

10.5.2. Memory Efficiency

The memory requirements of the entire procedure are mainly determined by the number of cells that are constructed and hence heavily depends on the resolution chosen for processing. High resolutions can be desirable since then the cell set bounds the mesh elements tighter and resolves finer features. The number of cells of the final octree of course depends on the geometry of the input, but in our experiments we observed that on average roughly 50 million cells are generated at a resolution of 4096^3. Since about 20-50 bytes need to be stored per cell (for index pointers to the parent cell and one child, location codes [FP02], state codes, a distance value, optionally a normal vector, an angle value, and ropes) this allows us to rasterize models at resolutions up to 4096^3 without exceeding today's PCs' main memories. However, the application of morphological operations results in a higher peak cell count. For instance, performing a dilation at the abovementioned resolution of 4096^3 to close holes and gaps up to a size of 3% of the bounding box diagonal increases the number of cells from 50 million to over 1000 million, clearly constraining applicability to lower resolutions in such cases.

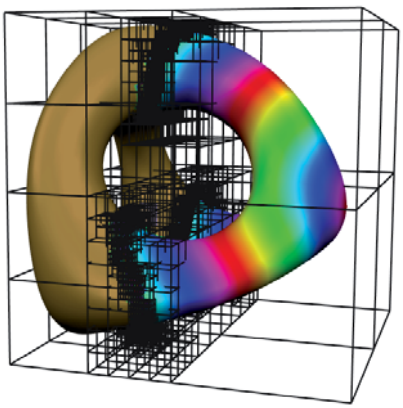
Tiling

To avoid this high memory peak we can perform the dilation and erosion process in a *tiled* fashion. The dilation and erosion operators are local in the sense that they only affect a local neighborhood of cells they are applied to, i.e. when applying γ dilation steps followed by γ erosion steps, the final state of a cell does not depend on cells farther away than 2γ in the 6-neighborhood graph of the cell set. Hence, when we apply these operations to one tile (the cell set clipped to an axis-aligned rectangular box) the resulting state of all cells except those in the outer 2γ cell layers of the tile is not affected by this clipping. By covering the bounding box with tiles overlapping by 4γ cells layers, the correct cell states can be obtained for all cells by applying the morphological operations to each tile separately. As pointed out in the last section restricting the erosion to γ steps results in a morphological closing of cavities in addition to gaps and holes, lowering the accuracy of distance computations. By applying $\gamma + \delta$ erosion steps (and choosing an overlap of $4\gamma + 2\delta$) this can be alleviated to any desired degree. In our experiments $\delta = 3\,\gamma$ was the maximum encountered that was necessary to achieve the same results as with unlimited non-tiled erosion.

Since in each tile cells can be collapsed again after erosion, the peak cell count can be reduced significantly. Due to the required overlap this reduction is bounded depending on γ (and δ), but since the presence of large holes (requiring large γ) introduces significant uncertainties, the appropriateness of using very high resolutions to achieve high accuracy seems to be questionable in such cases anyway.

Sweeping

Despite the tiling approach the final cell set still has to fit into memory entirely. To allow for even higher resolutions we introduce a *sweeping* variant of our method. In this implementation variant cells are dynamically created by octree refinement when they are reached by the FM front propagation process and deleted by collapsing when the front has passed them. For this purpose we separate the morphological hole filing from the distance propagation process. We perform the morphological operations in a tiled fashion as described above but discard the cells of each tile after its construction – we only record the center point of each non-solid cell that survived the erosion. This yields a set of points which effectively "fill" gaps and holes of $\mathcal{M}$ up to voxel resolution.



The sweeping can then be performed without complex in-line morphological operations by considering the union of $\mathcal{M}$ and this point set as input. The adjacent figure shows this sweeping in action: the octree is visualized at an intermediate state – only at the current front of propagation cells are at the finest level, away from it they are as coarse as possible.

In the following description of this sweeping, by the term "(octree) cell" we also refer to the node that is identified with the respective cell by the underlying graph isomorphism. In order to directly obtain the set of octree cells that are required to establish the isomorphism we do not consider cells whose volume is intersected by mesh elements (or hole-filling points) as solid, but cells whose volume extended by cell size λ in upper, right, and back direction is intersected. This avoids the subsequent additional

refinement of cells incident to the lower-left-front corner – which would be cumbersome to manage in this sweeping variant. Cells record contained vertices as described in the previous section.

Initially only cells containing sources are refined to level l, initialized, and inserted into the front propagation queue. During the propagation, whenever a value is to be propagated into a solid neighboring cell (resp. node) this cell is refined to level l. When a cell c is removed from the queue (i.e. its distance and angle values are final) interpolation has to be performed (cf. Section 10.3.3). Due to the extended virtual cell size exactly those vertices whose interpolated values depend on the node corresponding to c are recorded at c. Hence we can easily add the values computed for c, multiplied by the trilinear interpolation factors, to the (zero-initialized) values of these vertices ("transposed interpolation").

Afterwards, since it is not needed for interpolation anymore, c can be marked *collapsible* – unless there is a neighboring solid cell that is not yet finalized and hence might need the values of c for an FM update triggered by another cell. To cover this case we also call the collapsibility check for neighbor cells of c that are finalized but could not get marked collapsible so far. Whenever a cell gets marked collapsible its parent cell checks whether all children are collapsible and performs the collapse by deleting them. The parent cell is then marked collapsible and this process is invoked recursively to always obtain a maximally sparse octree.

10.6. Results

We now present some results generated with our implementation including runtime measurements. Experiments have been performed on a PC with 2.8 GHz Intel Core i7 CPU.

Figure 10.5 shows distance fields, geodesic paths, and parameterizations computed on a scanned model (358K polygons) which contains holes (partly non-simple, with *islands*). By choosing the dilation distance such that these holes are bridged the computed intrinsic distance approximations tolerate these defects. Computation times are presented in Table 10.1.

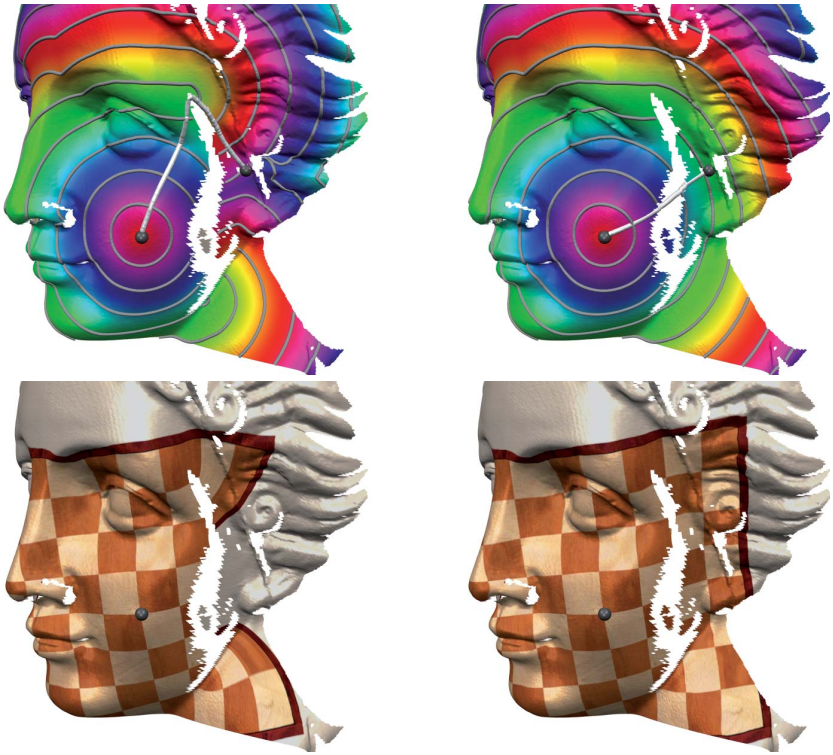


Figure 10.5.: Defect-tolerant computations on a raw scanned mesh FACE containing holes due to occlusion effects. Left: without morphological operations. Right: with morphological operations for hole bridging. The results of computations at resolution 256^3 are depicted.

Figure 10.6 depicts another mesh as it commonly appears in practice. This mesh was created by a commercial CAD tool, by inconsistent tessellation of NURBS patches. It consists of nearly 1 million polygons in more than 11,000 non-aligned connected components (see the close-up). Converting such models into manifold one-component meshes usually requires computationally intensive global repair methods. Using our framework the model can be handled directly. Timings are presented in Table 10.2. Since the

Resolution	64^3	128^3	256^3	512^3	1024^3
γ	2	4	8	16	32
Voxelize	1.12	1.40	2.0	3.3	6.1
Dilate	0.01	0.07	0.5	4.0	31.4*
Erode	0.02	0.09	1.0	9.4	80.2*
FM	0.02	0.09	0.3	1.4	5.6

Table 10.1.: Timings (in seconds) for distance field computation on model FACE (cf. Figure 10.5). *) Here *tiling* has been used (with $\delta = 0$; for $\delta = 1.2\gamma$, which sufficed to remove every unnecessary dilated cell, morphology took about 20% longer).

Resolution	256^3	512^3	1024^3	2048^3	4096^3	8192^3
Voxelize	4.3	5.8	8.8	18.4	-	-
FM	0.2	1.0	4.7	22.3	-	-
Sweeping	8.6	12.9	27.1	82.2	314.1	1370

Table 10.2.: Timings (in seconds) for one distance field computation on model CAR (cf. Figure 10.6). With the sweeping implementation higher resolutions can be handled (the peak cell count at 8192^3 is only 305K).

processing is blind to gaps and holes below leaf cell size the application of morphological operators was unnecessary in this case. Note that once rasterization, dilation, and erosion have been performed the obtained cubical complex (resp. octree) can be used for the quick computation of multiple distance fields etc. – it does not have to be rebuilt each time.

Figure 10.7 exemplifies the behavior on a polygon soup and shows how the morphological operators handle large holes/gaps.

Due to the finite resolution distances computed by our method of course usually deviate from actual intrinsic distances (on consistent models). Figure 10.8 illustrates how the deviation decreases with increasing resolution.

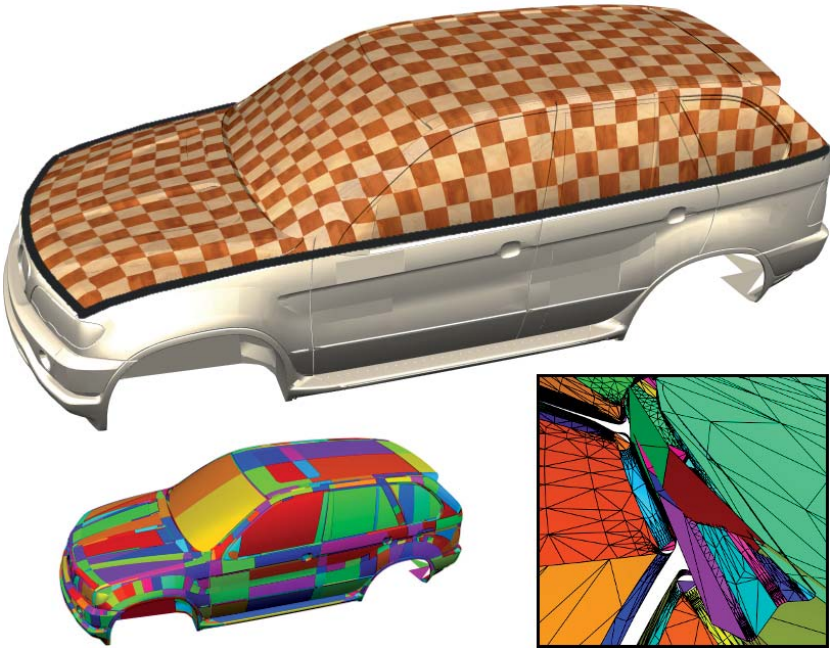


Figure 10.6.: Defect-tolerant texturing of an inconsistently tessellated NURBS model CAR by a boundary curve parameterization. The insets depict the color-coded individual connected components. As exemplified in the close-up, lots of non-trivial gaps, double-walls, and complex self-intersections are contained, essentially disqualifying boundary-snapping based algorithms for easy repair. The result of computation with resolution 128^3 is depicted.

10.7. Discussion

We presented a method that abstracts from the topological structure of a given input mesh, bridges gaps and holes up to a user-specified width, and thereby allows for the computation of plausible intrinsic distances and geodesic paths on meshes with all kinds of defects.

Figure 10.7.: Inconsistent polygon soup with a slice cut out. Dilated cells that survived erosion, a distance field, and two geodesic paths are visualized. Increasing the gap width at some point leads to the morphological operators closing the two holes instead of bridging the gap, as depicted on the right.

Based on this, the most fundamental operation of our stage 2, the dual loops construction, could be performed also on imperfect meshes. Together with techniques for the computation of cross fields and parameterization on such meshes, it is imaginable to extend our whole quad layout generation and optimization pipeline to imperfect meshes. But also many further existing methods and applications that rely on geodesic distance computations can be extended to deal with such meshes by employing the presented framework.

Due to the automaticity and generality of the method, it is naturally not able to resolve ambiguities that are inherent in the input due to large missing parts. Hence, the computed distance fields might be inconsistent with those of the object that is actually meant to be represented by input. Additional knowledge about the represented object

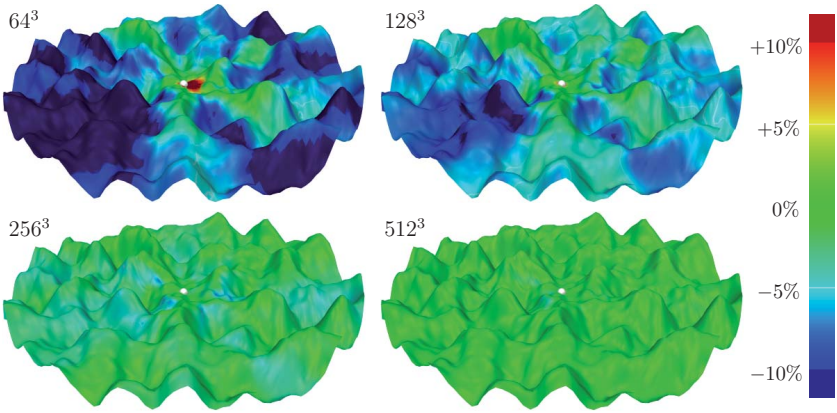


Figure 10.8.: Visualization of the error to ground truth distances on an exemplary surface, clearly decreasing with increasing resolution.

or manual interaction would be required to more plausibly handle the hole bridging in such cases.

11. Conclusion

To conclude we summarize our findings and contributions and provide an outlook on open problems and potential future research directions in the field of quad layouting.

Summary

In this thesis we investigated the problem of fully automatic or semi-automatic interactive quad layout generation. After an analysis of the problem structure and the involved quality criteria, revealing continuous, discrete, and topological degrees of freedom, we designed strategies for the subsequent optimization of these different aspects of a quad layout.

In Part I, which was devoted to the automatic pipeline, we specifically contributed a novel framework to describe and process structurally valid and geometrically faithful duals of quad layouts. We developed efficient methods for the geometry-aware construction of embedded loops that are the key components of the dual layouts in this framework. Using a practical greedy algorithm based on the topological concept of singularity separation the quad layout structure could automatically and efficiently be generated. We then showed how the embedding of nodes, arcs, and patches can be optimized in an integrated, global manner using an efficient alternating optimization strategy that separates linear and non-linear aspects of the optimization problem such that they can be handled using appropriate techniques each. To this end we extended established optimization techniques for cross fields and parameterizations by structural constraints, such that they can be used to optimize quad layout embeddings, with a particular focus on alignment to principal directions. In this context we furthermore introduced a novel concept for the quasi-continuous layout node position optimization, whose applicability extends to related fields such as global parametrization and quad meshing.

In Part II, where the interactive pipeline was described, we proposed a new quad layout design operator: the construction of dual strips. To enable its efficient use we described a novel, fast method for the construction of constrained elastica on surfaces, without any restrictions on topology or homotopy. Based on this we have been able to provide intuitive interaction metaphors to the user, granting full design flexibility, while the system automatically ensures structural consistency and promotes geometric quality by evaluating the intermediate design states and visualizing the assessment to guide the user. Furthermore, we developed means to further reduce the user's interaction efforts, for instance by automatically ensuring symmetric results on symmetric surfaces.

Finally, in Part III we introduced novel techniques for the efficient computation of geodesic distances and geodesic paths/loops. Specifically, we described a novel method for the computation of distance fields with respect to anisotropic metrics as required by the dual loops construction. We extensively compared its performance to that of existing options and demonstrated its efficiency in terms of run time and accuracy. Finally, towards the adaptation of the described algorithms to raw point cloud data or imperfect meshes, we elaborated on a concept for meshless geodesic computations.

In summary we have developed two pipelines, one for the automatic, one for the interactive, assisted generation of quad layouts on surfaces. All algorithmic parts are built directly on the three generic layout quality objectives (structural simplicity, geometric fidelity, principal direction alignment) and we demonstrated that in this way high-quality quad layouts can be created, while additional or deviating intents can be infused using the efficient interaction techniques. The advantages over the few previous approaches have been discussed and been demonstrated in comparisons. In this sense our work provides important contributions to the field of quad structures and has been able to advance the state-of-the-art in this relatively young field.

Outlook

Despite these advances there are still a lot of opportunities for future research in this field – a field to which, apart from a few early approaches, only very recently deeper investigation has been devoted. We outline our thoughts about the most promising and most valuable directions in the following.

Specialization

The automatic pipeline is based on generic quality criteria assumptions. The specialization to specific use cases would of course be very valuable in practice. To this end a first important step would be the investigation and formulation of concrete quality measures. For instance, if the target application is NURBS fitting, how does one rate the quality of the layout? It is most probably a function of fitting accuracy (surface approximation error, normal approximation error) as well as of layout complexity. But the concrete nature of this function is not obvious and the trade-off between accuracy and complexity surely dependent on the specific application scenario. The availability of such concrete measures would furthermore be of great benefit for the quantitative comparison of results of future approaches to the quad layout problem.

Sub-Problem Coupling

When splitting the quad layout problem into sub-problems and developing optimization strategies for these, we followed the paradigms of forward anticipation and backward modification. It would be interesting to explore further possibilities in this regard, e.g. enabling a broader range of modification operators (not only merging nodes, but also splitting them, not only collapsing poly-chords but more general quad decimation or refinement).

Regarding the forward anticipation, there is a particular lack between stage 1 and 2 of the automatic pipeline – the node generation stage is not aware of the nodes' suitability for a good connectivity. It is actually hard to imagine how this could be estimated at all in the first stage. For this reason it could be of benefit for the final layout quality if the first two stages were not separated but integrated. In fact, just this is the case in our interactive pipeline: nodes and connectivity emerge from the same process. It would hence be very interesting to investigate whether it would be possible to design an automatic algorithm based on elastica loops instead of cross field guided loops. The singularity separation concept that drives the loops selection strategy in stage 2 could potentially be adapted to rely on the singularities of the dynamically changing cross field that is used for the strip region computation.

Alignment

In our methods we favored alignment to principal curvature directions. The numerous advantages of that were outlined in Section 2.2. Depending on the application it can be interesting to investigate the use of alternative alignment objectives. The alignment to Langer's lines (capturing the fiber directions of the human skin) could be of benefit for character animation purposes, the alignment to (estimates of) characteristic lines of differential equations could provide advantages for simulation accuracy, and the alignment to principal directions of stress or strain tensor fields is of interest for architectural applications.

Hex Layouts

In this thesis we focussed on quad layouts for *surfaces*. While the surface indeed is the most important feature of an object, some applications require a three-dimensional (piecewise) representation of its solid interior volume – for instance realistic simulations that involve an object's stability, rigidity, or conductivity. The representations discussed in Chapter 1 (quad mesh, multiblock grid, spline net) naturally extend to this volumetric setting. In this case, we are dealing with *hex(ahedral) layouts* of volumes, where each *cell* of the layout maps to a three-dimensional cuboid. A generalization of our automatic and interactive pipelines to this hex layout case would be of high value. Unfortunately, while the final stage (embedding optimization) could be adapted quite easily based on three-dimensional field-guided parameterization [NRP11], the other stages turn out to require a different approach – straightforward generalization is not possible.

The first problem is the generation of three-dimensional analogues of cross fields, sometimes termed *frame fields* [NRP11] or *3D cross-frame fields* [HTWB11]. While cross fields have isolated point singularities, such fields have an entire network of interconnected point and line singularities. Just like in the cross field case, these singularities can be represented using a generalization of the period jump concept [LVRL06]. Unfortunately, validity is no longer inherent and additional constraints need to be met such that a valid singularity network is implied. So far thus manual singularity specification [NRP11] has been used, or attempts been made to repair invalid configurations in a post-process [LLX*12, JHW*14]. However, to cite Jiang et al. [JHW*14], "it remains an open

problem to give a sufficient condition on singularity graphs compatible to non-degenerate hexahedral meshing".

The concepts of dual loops and dual strips extend to the three-dimensional case in the form of dual sheets and dual layers [MBBM97]. The efficient generation of candidate dual sheets for the automatic pipeline and of dual layers for the interactive pipeline is another unsolved problem. While minimal loops can be generated using efficient Dijkstra-type techniques (cf. Section 5.2.1, Section 8.1, Chapter 9), minimal surfaces (as natural model for good dual sheets) are not amenable to such efficient generation [Gra10], especially when only a seed point is given as constraint. A particular challenge is that dual sheets of unknown topology (arbitrary genus, arbitrary number of boundaries) must be dealt with, while dual loops are always simple curves.

Another, though less elegant and powerful, option for generalization to the hexahedral layout case is the use of a technique along the lines of [MH98, KBLK13]. In these works surface quad meshes are taken as input and extended to volume hex meshes. A similar approach could be taken to extend surface quad layouts to hex layouts. These techniques are, however, only applicable to a certain restricted class of quad layouts [Eri13] e.g. without self-crossing dual loops. The investigation of possibilities to adapt our methods to such restricted classes of connectivity that are of interest in certain applications is another interesting avenue for future work.

Bibliography

[AAB*88] Andersson E., Andersson R., Boman M., Elmroth T., Dahlberg B., Johansson B.: Automatic construction of surfaces with prescribed shape. *Comput. Aided Des. 20*, 6, 1988, 317–324.

[ABA02] Andújar C., Brunet P., Ayala D.: Topology-reducing surface simplification using a discrete solid representation. *ACM Trans. Graph. 21*, 2, 2002, 88–105.

[ACK13] Attene M., Campen M., Kobbelt L.: Polygon mesh repairing: An application perspective. *ACM Comput. Surv. 45*, 2, 2013, 15.

[ACSD*03] Alliez P., Cohen-Steiner D., Devillers O., Lévy B., Desbrun M.: Anisotropic polygonal remeshing. *ACM Trans. Graph. 22*, 3, 2003, 485–493.

[AG03] Alliez P., Gotsman C.: Recent advances in compression of 3D meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling*, 2003, pp. 3–26.

[AUGA08] Alliez P., Ucelli G., Gotsman C., Attene M.: Recent advances in remeshing of surfaces. In *Shape Analysis and Structuring, Mathematics and Visualization*, 2008, Springer, pp. 53–82.

[BC11] Benmansour F., Cohen L. D.: Tubular Structure Segmentation Based on Minimal Path Method and Anisotropic Enhancement. *Int. J. of Computer Vision 92*, 2, 2011, 192–210.

[BCE*13] Bommes D., Campen M., Ebke H.-C., Alliez P., Kobbelt L.: Integer-Grid Maps for Reliable Quad Meshing. In *ACM Trans. Graph.*, 2013, vol. 32, pp. 98:1–98:12.

[BCLC09] Benmansour F., Cohen L. D., Law M. W. K., Chung A. C. S.: Tubular anisotropy for 2D vessel segmentation. In *CVPR*, 2009, pp. 2286–2293.

[BDL10] Brakhage K.-H., Dahmen W., Lamby P.: A Unified Approach to the Modeling of Airplane Wings and Numerical Grid Generation Using B-Spline Representations. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design 109*, 2010, 239–263.

[Bei68] Beineke L. W.: Derived graphs of digraphs. *Beiträge zur Graphentheorie*, 1968, 17–33.

[BK01] Botsch M., Kobbelt L.: Resampling Feature Regions in Polygonal Meshes for Surface Anti-Aliasing. *Comput. Graph. Forum 20*, 3, 2001, 402–410.

[BK05] Bischoff S., Kobbelt L.: Structure Preserving CAD Model Repair. *Comput. Graph. Forum 24*, 3, 2005, 527–536.

[BLK11] Bommes D., Lempfer T., Kobbelt L.: Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum 30*, 2, 2011, 375–384.

[BLP*13] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D.: Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum 32*, 2013.

[BMBZ02] Biermann H., Martin I. M., Bernardini F., Zorin D.: Cut-and-paste editing of multiresolution surfaces. In *SIGGRAPH '02*, 2002, pp. 312–321.

[BMRJ04] Boier-Martin I. M., Rushmeier H. E., Jin J.: Parameterization of Triangle Meshes over Quadrilateral Domains. In *Proc. SGP '04*, 2004, pp. 197–208.

[BNR01] Bruckstein A., Netravali A., Richardson T.: Epi-Convergence of Discrete Elastica. *Applicable Analysis 79*, 2001, 137–171.

[Bom12] Bommes D.: *Quadrilateral Surface Mesh Generation for Animation and Simulation*. PhD thesis, RWTH Aachen University, 2012.

[BPC08] Bougleux S., Peyré G., Cohen L. D.: Anisotropic Geodesics for Perceptual Grouping and Domain Meshing. In *ECCV '08*, 2008, pp. 129–142.

[BPK05] Bischoff S., Pavic D., Kobbelt L.: Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4, 2005, 1332–1352.

[BVK08] Bommes D., Vossemer T., Kobbelt L.: Quadrangular parameterization for reverse engineering. *Mathematical Methods for Curves and Surfaces*, 2008, 55–69.

[BWK05] Bischoff S., Weyand T., Kobbelt L.: Snakes on Triangle Meshes. In *Bildverarbeitung für die Medizin*, 2005, pp. 208–212.

[BZK04] Borodin P., Zachmann G., Klein R.: Consistent Normal Orientation for Polygonal Meshes. In *CGI '04: Proc. Computer Graphics International*, 2004, pp. 18–25.

[BZK09] Bommes D., Zimmer H., Kobbelt L.: Mixed-Integer Quadrangulation. In *ACM Trans. Graph.*, 2009, vol. 28, pp. 77:1–77:10.

[BZK12] Bommes D., Zimmer H., Kobbelt L.: Practical Mixed-integer Optimization for Geometry Processing. In *Proc. Curves and Surfaces*, 2012, pp. 193–206.

[CBK12] Campen M., Bommes D., Kobbelt L.: Dual Loops Meshing: Quality Quad Layouts on Manifolds. *ACM Transactions on Graphics 31*, 4, 2012, 110:1–110:11.

[CC78] Catmull E., Clark J.: Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes. *Computer-Aided Design 10*, 6, 1978, 350–355.

[CDHR08] Chen Y., Davis T. A., Hager W. W., Rajamanickam S.: Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *ACM Trans. Math. Softw. 35*, 3, 2008, 22:1–22:14.

[CDS10] Crane K., Desbrun M., Schröder P.: Trivial Connections on Discrete Surfaces. *Computer Graphics Forum 29*, 5, 2010, 1525–1533.

[CH90] Chen J., Han Y.: Shortest Paths on a Polyhedron. In *Proc. Symp. Comp. Geom.*, 1990, pp. 360–369.

[CHK13] Campen M., Heistermann M., Kobbelt L.: Practical Anisotropic Geodesy. In *Computer Graphics Forum*, 2013, vol. 32, pp. 63–71.

[CK11] Campen M., Kobbelt L.: Walking On Broken Mesh: Defect-Tolerant Geodesic Distances and Parameterizations. In *Computer Graphics Forum*, 2011, vol. 30, pp. 623–632.

[CK14a] Campen M., Kobbelt L.: Dual Strip Weaving: Interactive Design of Quad Layouts using Elastica Strips. In *ACM Trans. Graph.*, 2014, vol. 33, pp. 183:1–183:10.

[CK14b] Campen M., Kobbelt L.: Quad Layout Embedding via Aligned Parameterization. *Computer Graphics Forum 33*, 2014, 69–81.

[CSAD04] Cohen-Steiner D., Alliez P., Desbrun M.: Variational shape approximation. In *Proc. SIGGRAPH 2004*, 2004, pp. 905–914.

[CSM03] Cohen-Steiner D., Morvan J.-M.: Restricted Delaunay Triangulations and Normal Cycle. In *Proc. Symp. Comp. Geom.*, 2003, SCG '03, pp. 312–321.

[CWW13] Crane K., Weischedel C., Wardetzky M.: Geodesics in Heat. *ACM Trans. Graph. 32*, 2013.

[D'A00] D'Azevedo E. F.: Are Bilinear Quadrilaterals Better Than Linear Triangles? *J. Sci. Comput. 22*, 1, 2000, 198–217.

[DBG*06] Dong S., Bremer P.-T., Garland M., Pascucci V., Hart J. C.: Spectral surface quadrangulation. In *Proc. SIGGRAPH 2006*, 2006, pp. 1057–1066.

[dC76] do Carmo M. P.: *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[DC00] Deschamps T., Cohen L. D.: Minimal Paths in 3D Images and Application to Virtual Endoscopy. In *ECCV*, 2000, pp. 543–557.

[DHM09] Dahmen W., Hovhannisyan N., Mller S.: *Adaptive Multiscale Methods for Flow Problems: Recent Developments, IGPM Report #293*. Tech. rep., RWTH Aachen, 2009.

[DS78] Doo D., Sabin M.: Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design 10*, 6, 1978, 356–360.

[DSC09] Daniels J., Silva C. T., Cohen E.: Semi-regular Quadrilateral-only Remeshing from Simplified Base Domains. *Comput. Graph. Forum 28*, 5, 2009, 1427–1435.

[DSSC08] Daniels J., Silva C. T., Shepherd J., Cohen E.: Quadrilateral mesh simplification. *ACM Trans. Graph. 27*, 5, 2008, 148.

[DVPSH14] Diamanti O., Vaxman A., Panozzo D., Sorkine-Hornung O.: Designing N-PolyVector Fields with Complex Polynomials. *Computer Graphics Forum 33*, 5, 2014.

[EBCK13] Ebke H.-C., Bommes D., Campen M., Kobbelt L.: QEx: Robust Quad Mesh Extraction. In *ACM Trans. Graph.*, 2013, vol. 32, pp. 168:1–168:10.

[ECBK14] Ebke H.-C., Campen M., Bommes D., Kobbelt L.: Level-of-Detail Quad Meshing. In *ACM Trans. Graph.*, 2014, vol. 33, pp. 184:1–184:11.

[EGKT08] Eppstein D., Goodrich M. T., Kim E., Tamstorf R.: Motorcycle Graphs: Canonical Quad Mesh Partitioning. *Computer Graphics Forum 27*, 5, 2008, 1477–1486.

[EH96] Eck M., Hoppe H.: Automatic reconstruction of B-spline surfaces of arbitrary topological type. In *Proc. SIGGRAPH 96*, 1996, pp. 325–334.

[EHP02] Erickson J., Har-Peled S.: Optimally cutting a surface into a disk. In *Proc. Symp. on Computational Geometry*, 2002, pp. 244–253.

[Eri13] Erickson J.: Efficiently Hex-Meshing Things with Topology. In *Proc. Symp. Comp. Geom. (SoCG ' 13)*, 2013, pp. 37–46.

[EW05] Erickson J., Whittlesey K.: Greedy optimal homotopy and homology generators. In *Proc. 16th Ann. ACM-SIAM Symp. Discrete Algo-*

rithms, 2005, pp. 1038–1046.

[Far02] Farin G.: *Curves and Surfaces for CAGD. A Practical Guide*. Morgan-Kaufmann, 2002.

[FH05] Floater M. S., Hormann K.: Surface Parameterization: a Tutorial and Survey. In *Advances in Multiresolution for Geometric Modelling*. Springer, 2005, pp. 157–186.

[FP02] Frisken S. F., Perry R. N.: Simple and Efficient Traversal Methods for Quadtrees and Octrees. *Journal of Graphics Tools 7*, 3, 2002, 2003.

[FSSB07] Fisher M., Springborn B., Schröder P., Bobenko A. I.: An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing. *Computing 81*, 2-3, 2007, 199–213.

[GGGZ05] Gatzke T., Grimm C., Garland M., Zelinka S.: Curvature Maps for Local Shape Comparison. In *Shape Modeling International*, 2005, pp. 244–256.

[Gra10] Grady L.: Minimal Surfaces Extend Shortest Path Segmentation Methods to 3D. *IEEE Trans. Pattern Anal. Mach. Intell. 32*, 2, 2010, 321–334.

[GVSS00] Guskov I., Vidimce K., Sweldens W., Schröder P.: Normal meshes. In *Proc. SIGGRAPH 2000*, 2000, pp. 95–102.

[Hat02] Hatcher A.: *Algebraic Topology*. Cambridge University Press, 2002.

[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 Mechaniccs and Engineering 194*, 2005, 4135–4195.

[HF07] Hassouna M. S., Farag A. A.: MultiStencils Fast Marching Methods: A Highly Accurate Solution to the Eikonal Equation on Cartesian Domains. *IEEE Trans. Pattern Analysis and Machine Intelligence 29*, 2007, 1563–1574.

[HJS*14] Huang J., Jiang T., Shi Z., Tong Y., Bao H., Desbrun M.: ℓ_1-Based Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph. 33*, 3, 2014, 25:1–25:11.

[HP04] Hofer M., Pottmann H.: Energy-minimizing splines in manifolds. *ACM Trans. Graph. 23*, 3, 2004, 284–293.

[HTWB11] Huang J., Tong Y., Wei H., Bao H.: Boundary Aligned Smooth 3D Cross-frame Field. *ACM Trans. Graph. 30*, 6, 2011, 143:1–143:8.

[HWW*06] He Y., Wang K., Wang H., Gu X., Qin H.: Manifold T-Spline. In *Proc. Geometric Modeling and Processing*, 2006, pp. 409–422.

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. In *Proc. SIGGRAPH 2000*, 2000, 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, 147.

[JHW*14] Jiang T., Huang J., Wang Y., Tong Y., Bao H.: Frame Field Singularity Correction for Automatic Hexahedralization. *Transactions on Visualization and Computer Graphics, 20*, 8, 2014, 1189–1199.

[JLW10] Ji Z., Liu L., Wang Y.: B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. In *Proc. Pacific Graphics '10*, 2010, pp. 2169–2178.

[JT73] Jucovič E., Trenkler M.: A theorem on the structure of cell-decompositions of orientable 2-manifolds. *Mathematika 20*, 1973, 63–82.

[KBLK13] Kremer M., Bommes D., Lim I., Kobbelt L.: Advanced Automatic Hexahedral Mesh Generation from Surface Quad Meshes. In *Proceedings of the 22nd International Meshing Roundtable*, 2013, pp. 147–164.

[KCPS13] Knöppel F., Crane K., Pinkall U., Schröder P.: Globally optimal direction fields. *ACM Trans. Graph. 32*, 4, 2013, 59.

[KL96] Krishnamurthy V., Levoy M.: Fitting Smooth Surfaces to Dense Polygon Meshes. In *Proc. SIGGRAPH 96*, 1996, pp. 313–324.

[KLS03] Khodakovsky A., Litke N., Schröder P.: Globally smooth parameterizations with low distortion. *ACM Trans. Graph. 22*, 3, 2003, 350–357.

[KMZ11] Kovacs D., Myles A., Zorin D.: Anisotropic quadrangulation. *Comp. Aided Geom. Design 28*, 8, 2011, 449–462.

[KNP07] Kälberer F., Nieser M., Polthier K.: QuadCover – Surface Parameterization using Branched Coverings. *Computer Graphics Forum 26*, 3, 2007, 375–384.

[KS98] Kimmel R., Sethian J. A.: Computing geodesic paths on manifolds. *Proc. Natl. Acad. Sci. 95*, 15, 1998, 8431–8435.

[KS00] Kanai T., Suzuki H.: Approximate Shortest Path on Polyhedral Surface Based on Selective Refinement of the Discrete Graph and its Applications. In *Geometric Modeling and Processing*, 2000, pp. 241–250.

[KS04] Kraevoy V., Sheffer A.: Cross-parameterization and compatible remeshing of 3D models. In *Proc. SIGGRAPH 2004*, 2004, pp. 861–869.

[KSC*07] Konukoglu E., Sermesant M., Clatz O., Peyrat J.-M., Delingette H., Ayache N.: A Recursive Anisotropic Fast Marching Approach to Reaction Diffusion Equation: Application to Tumor Growth Modeling. In *IPMI*, 2007, pp. 687–699.

[Lan99] Lanthier M.: *Shortest Path Problems on Polyhedral Surfaces*. PhD thesis, School of Computer Science, Carleton University, 1999.

[Lip12] Lipman Y.: Bounded Distortion mapping spaces for triangular meshes. In *Proc. SIGGRAPH 2012*, 2012, pp. 108:1–108:13.

[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. 16*, 1, 2010, 95–108.

[LKH08] Lai Y.-K., Kobbelt L., Hu S.-M.: An Incremental Approach to Feature Aligned Quad Dominant Remeshing. In *Proc. Symp. Solid and Physical Modeling 2008*, 2008, pp. 137–145.

[LL02] Lee Y., Lee S.: Geometric Snakes for Triangular Meshes. *Comput. Graph. Forum 21*, 3, 2002, 229–238.

[LLS01] Litke N., Levin A., Schröder P.: Fitting Subdivision Surfaces. In *IEEE Visualization 2001*, 2001, pp. 319–324.

[LLX*12] Li Y., Liu Y., Xu W., Wang W., Guo B.: All-hex Meshing Using Singularity-restricted Field. *ACM Trans. Graph. 31*, 6, 2012, 177:1–177:11.

[LLZ*11] Li E., Lévy B., Zhang X., Che W., Dong W., Paul J.-C.: Meshless quadrangulation by global parameterization. *Computers & Graphics*, 2011, 992–1000.

[LMS97] Lanthier M., Maheshwari A., Sack J.-R.: Approximating weighted shortest paths on polyhedral surfaces. *Proc. Symp. Comp. Geom. (SCG '97)*, 1997, 274–283.

[LMS99] Lanthier M., Maheshwari A., Sack J.-R.: Shortest Anisotropic Paths on Terrains. *ICAL 26*, 1999, 523–533.

[LRL06] Li W.-C., Ray N., Lévy B.: Automatic and interactive mesh to T-spline conversion. In *Proc. SGP '06*, 2006, pp. 191–200.

[LSS*98] Lee A. W. F., Sweldens W., Schröder P., Cowsar L., Dobkin D.: MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *Proc. SIGGRAPH '98*, 1998, pp. 95–104.

[LVRL06] Li W. C., Vallet B., Ray N., Lévy B.: Representing Higher-Order Singularities in Vector Fields on Piecewise Linear Surfaces. *IEEE TVCG 12*, 5, 2006, 1315–1322.

[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. *ACM Trans. Graph. 30*, 6, 2011, 140:1–140:10.

[MB90] MacDonald D. J., Booth K. S.: Heuristics for ray tracing using space subdivision. *Vis. Comput. 6*, 3, 1990, 153–166.

[MBBM97] Murdoch P., Benzley S., Blacker T., Mitchell S. A.: The spatial twist continuum: a connectivity based method for representing

all-hexahedral finite element meshes. *Finite Elem. Anal. Des. 28*, 1997, 137–149.

[MBVW95] Milroy M. J., Bradley C., Vickers G. W., Weir D. J.: G1 continuity of B-spline surface patches in reverse engineering. *Computer-Aided Design 27*, 6, 1995, 471–478.

[MH98] Müller-Hannemann M.: Hexahedral Mesh generation by Successive Dual Cycle Elimination. In *Int. Meshing Roundtable 98*, 1998, pp. 379–393.

[Mit00] Mitchell S. A.: High Fidelity Interval Assignment. *Int. J. Comput. Geometry Appl. 10*, 4, 2000, 399–415.

[MK95] Ma W., Kruth J.-P.: Parametrization of randomly measured points for least squares fitting of B-spline curves and surfaces. *Computer-Aided Design 27*, 1995, 663–675.

[MK04] Marinov M., Kobbelt L.: Direct Anisotropic Quad-Dominant Remeshing. In *Proc. Pacific Graphics '04*, 2004, pp. 207–216.

[MK05] Marinov M., Kobbelt L.: Automatic Generation of Structure Preserving Multiresolution Models. *Computer Graphics Forum 24*, 3, 2005, 479–486.

[MMP87] Mitchell J. S., Mount D. M., Papadimitriou C. H.: The Discrete Geodesic Problem. *SIAM Journal on Computing 16*, 4, 1987, 647–668.

[MPKZ10] Myles A., Pietroni N., Kovacs D., Zorin D.: Feature-aligned T-meshes. In *Proc. SIGGRAPH 2010*, 2010, pp. 117:1–117:11.

[MPWC13] Mitra N. J., Pauly M., Wand M., Ceylan D.: Symmetry in 3D Geometry: Extraction and Applications. *Comput. Graph. Forum 32*, 6, 2013, 1–23.

[MPZ14] Myles A., Pietroni N., Zorin D.: Robust Field-aligned Global Parametrization. In *Proc. SIGGRAPH 2014*, 2014, pp. 135:1–135:14.

[MS01] Mémoli F., Sapiro G.: Fast Computation of Weighted Distance Functions and Geodesics on Implicit Hyper-surfaces. *J. Comput. Phys. 173*, 2, 2001, 730–764.

[MS05] Mémoli F., Sapiro G.: Distance Functions and Geodesics on Submanifolds of R^d and Point Clouds. *SIAM Journal of Applied Mathematics 65*, 4, 2005, 1227–1260.

[MZ12] Myles A., Zorin D.: Global parametrization by incremental flattening. In *Proc. SIGGRAPH 2012*, 2012, pp. 109:1–109:11.

[MZ13] Myles A., Zorin D.: Controlled-distortion Constrained Global Parametrization. *ACM Trans. Graph. 32*, 4, 2013, 105:1–105:14.

[MZL*09] Mehra R., Zhou Q., Long J., Sheffer A., Gooch A., Mitra N. J.: Abstraction of Man-Made Shapes. *ACM Transactions on Graphics 28*, 5, 2009, 137:1–137:10.

[Nie12] Nieser M.: *Parameterization and Tiling of Polyhedral Surfaces*. PhD thesis, Freie Universität Berlin, 2012.

[NK02] Novotni M., Klein R.: Computing Geodesic Paths on Triangular Meshes. In *WSCG*, 2002, pp. 341–348.

[NL12] Naumann U., Lotz J.: Algorithmic differentiation of numerical methods: tangent-linear and adjoint direct solvers for systems of linear equations. *Tech. Report AIB-2012-10, RWTH Aachen*, 2012.

[NP09] Nieser M., Polthier K.: Parameterizing Singularities of Positive Integral Index. In *13th IMA Int. Conf. on Mathematics of Surfaces*, 2009, pp. 265–277.

[NRP11] Nieser M., Reitebuch U., Polthier K.: CubeCover – Parameterization of 3D Volumes. *Comput. Graph. Forum 30*, 5, 2011, 1397–1406.

[NSP10] Nieser M., Schulz C., Polthier K.: Patch layout from feature graphs. *Computer-Aided Design 42*, 3, 2010, 213–220.

[PBDSH13] Panozzo D., Baran I., Diamanti O., Sorkine-Hornung O.: Weighted Averages on Surfaces. *ACM Trans. Graph. 32*, 4, July 2013, 60:1–60:12.

[PLPZ12] Panozzo D., Lipman Y., Puppo E., Zorin D.: Fields on symmetric surfaces. *ACM Trans. Graph. 31*, 4, 2012, 111.

[PPT*11] Panozzo D., Puppo E., Tarini M., Pietroni N., Cignoni P.: Automatic Construction of Quad-Based Subdivision Surfaces Using Fitmaps. *IEEE Trans. Vis. Comput. Graph. 17*, 10, 2011, 1510–1520.

[PPTSH14] Panozzo D., Puppo E., Tarini M., Sorkine-Hornung O.: Frame Fields: Anisotropic and Non-Orthogonal Cross Fields. *ACM Transactions on Graphics 33*, 4, 2014, 134.

[PS98] Polthier K., Schmies M.: Straightest Geodesics on Polyhedral Surfaces. In *Vis. and Math. '97*. Springer, 1998, pp. 135–150.

[PSS01] Praun E., Sweldens W., Schröder P.: Consistent mesh parameterizations. In *SIGGRAPH 2001*, 2001, pp. 179–184.

[PTC10] Pietroni N., Tarini M., Cignoni P.: Almost Isometric Mesh Parameterization through Abstract Domains. *IEEE Trans. Vis. Comput. Graph. 16*, 4, 2010, 621–635.

[PTSZ11] Pietroni N., Tarini M., Sorkine O., Zorin D.: Global Parametrization of Range Image Sets. *ACM Trans. Graph. 30*, 6, 2011, 149:1–149:10.

[PWKB02] Parker G. J. M., Wheeler-Kingshott C. A. M., Barker G. J.: Estimating Distributed Anatomical Brain Connectivity Using Fast Marching Methods and Diffusion Tensor Imaging. *IEEE Trans. Med. Imaging 21*, 5, 2002, 505–512.

[PWT05] Pichon E., Westin C.-F., Tannenbaum A. R.: A Hamilton-Jacobi-Bellman approach to high angular resolution diffusion tractography. *Medical image computing and computer-assisted intervention 8*, Pt 1, Jan. 2005, 180–7.

[PZ07] Palacios J., Zhang E.: Rotational symmetry field design on surfaces. In *Proc. SIGGRAPH 2007*, 2007, pp. 55:1–55:10.

[PZKW11] Peng C.-H., Zhang E., Kobayashi Y., Wonka P.: Connectivity editing for quadrilateral meshes. *ACM Trans. Graph. 30*, 6, 2011, 141.

[Rei95] Reif U.: A Unified Approach to Subdivision Algorithms Near Extraordinary Points. *Comput. Aided. Geom. Des. 12*, 1995.

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

[RNLL10] Ray N., Nivoliers V., Lefebvre S., Lévy B.: Invisible Seams. In *Proc. Eurographics Symposium on Rendering*, 2010.

[RR90] Rowe N. C., Ross R. S.: Optimal grid-free path planning across arbitrarily-contoured terrain with anisotropic friction and gravity effects. *IEEE Trans. Robot. Autom*, 1990.

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

[RVLL08] Ray N., Vallet B., Li W. C., Lévy B.: N-symmetry direction field design. *ACM Trans. Graph. 27*, 2008, 10:1–10:13.

[Sam89] Samet H.: Neighbour finding in images represented by octrees. In *Vision, Graphics & Image Proc.*, 1989, pp. 367–386.

[SAPH04] Schreiner J., Asirvatham A., Praun E., Hoppe H.: Inter-surface mapping. In *SIGGRAPH 2004*, 2004, pp. 870–877.

[SB96] Spekreijse S., Boerstoel J. W.: Multiblock Grid Generation. Part 2: Multiblock Aspects. In *VKI Lecture Series 1996-06*. von Karman Institute for Fluid Dynamics, 1996, pp. 1–39.

[SC07] Schoenemann T., Cremers D.: Introducing Curvature into Globally Optimal Image Segmentation: Minimum Ratio Cycles on Product Graphs. In *ICCV*, 2007, IEEE, pp. 1–6.

[SCF*04] Sederberg T. W., Cardon D. L., Finnigan G. T., North N. S., Zheng J., Lyche T.: T-spline Simplification and Local Refinement. *ACM Trans. Graph. 23*, 3, 2004, 276–283.

[Sch13] Schmidt R.: Stroke Parameterization. *Comp. Graph. Forum 32*, 2, 2013.

[Set95] Sethian J. A.: A Fast Marching Level Set Method for Monotonically Advancing Fronts. In *Proc. Nat. Acad. Sci*, 1995, pp. 1591–1595.

[Set99] Sethian J. A.: Fast Marching Methods. *SIAM Review 41*, 1999, 199–235.

[SGW06] Schmidt R., Grimm C., Wyvill B.: Interactive decal compositing with discrete exponential maps. In *Proc. SIGGRAPH 2006*, 2006, pp. 605–613.

[SJC09] Seong J.-K., Jeong W.-K., Cohen E.: Curvature-based anisotropic geodesic distance computation for parametric and implicit surfaces. *The Visual Computer 25*, 8, 2009, 743–755.

[SMC11] Schoenemann T., Masnou S., Cremers D.: The Elastic Ratio: Introducing Curvature Into Ratio-Based Image Segmentation. *IEEE Trans. Img. Proc. 20*, 9, 2011, 2565–2581.

[SSK*05] Surazhsky V., Surazhsky T., Kirsanov D., Gortler S. J., Hoppe H.: Fast exact and approximate geodesics on meshes. In *Proc. SIGGRAPH 2005*, 2005, pp. 553–560.

[SV00] Sethian J. A., Vladimirsky A.: Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes. *Proc. Nat. Acad. Sci. 97*, 11, 2000, 5699–5703.

[SV04] Sethian J. A., Vladimirsky A.: Ordered Upwind Methods for Static Hamilton-Jacobi Equations: Theory and Algorithms. *SIAM J. Num. Anal. 41*, 1, 2004, 325–363.

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

[TA93] Tam T. K. H., Armstrong C. G.: Finite element mesh control by integer programming. *Int. J. Numerical Methods in Engineering 36*, 1993, 2581–2605.

[TACSD06] Tong Y., Alliez P., Cohen-Steiner D., Desbrun M.: Designing quadrangulations with discrete harmonic forms. In *Proc. SGP '06*, 2006, pp. 201–210.

[TDN*12] Tierny J., Daniels J., Nonato L. G., Pascucci V., Silva C.: Interactive quadrangulation with Reeb atlases and connectivity textures. *IEEE TVCG 18*, 2012, 1650–1663.

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: PolyCube-Maps. In *Proc. SIGGRAPH 2004*, 2004, pp. 853–860.

[TPP*11] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P.: Simple Quad Domains for Field Aligned Mesh Parametrization. *Proc. SIGGRAPH Asia 2011 30*, 6, 2011.

[TPSHSH13] Takayama K., Panozzo D., Sorkine-Hornung A., Sorkine-Hornung O.: Sketch-Based Generation and Editing of Quad Meshes. In *Proc. SIGGRAPH 2013*, 2013, pp. 97:1–97:8.

[Tsi95] Tsitsiklis J. N.: Globally Optimal Trajectories. *IEEE Transactions on Automatic Control 40*, 9, 1995, 1528–1538.

[TWZZ07] Tang J., Wu G.-S., Zhang F.-Y., Zhang M.-M.: Fast approximate geodesic paths on triangle mesh. *International Journal of Automation and Computing 4*, 1, 2007, 8–13.

[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*, 1, 2006, 25–57.

[WDB*08] Weber O., Devir Y. S., Bronstein A. M., Bronstein M. M., Kimmel R.: Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Transactions on Graphics 27*, 4, 2008, 104:1–104:16.

[WW94] Welch W., Witkin A. P.: Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH '94*, 1994, pp. 247–256.

[WZXH12] Wang W., Zhang Y., Xu G., Hughes T.: Converting an unstructured quadrilateral/hexahedral mesh to a rational T-spline. *Computational Mechanics 50*, 1, 2012, 65–84.

[XW09] Xin S.-Q., Wang G.-J.: Improving Chen and Han's algorithm on the discrete geodesic problem. *ACM Trans. Graph. 28*, 4, 2009, 104:1–104:8.

[YLY*12] Yu H., Lee T.-Y., Yeh I.-C., Yang X., Li W., Zhang J.: An RBF-Based Reparameterization Method for Constrained Texture Mapping. *IEEE TVCG 18*, 7, 2012, 1115 –1124.

[YSS*12] Yoo S. W., Seong J.-K., Sung M.-H., Shin S. Y., Cohen E.: A Triangulation-Invariant Method for Anisotropic Geodesic Map Computation on Surface Meshes. *IEEE Trans. Vis. Comput. Graph. 18*, 10, 2012, 1664–1677.

[ZG04] Zelinka S., Garland M.: Similarity-based surface modelling using geodesic fans. In *Proc. Symposium on Geometry Processing*, 2004, pp. 204–213.

[ZHLB10] Zhang M., Huang J., Liu X., Bao H.: A wave-based anisotropic quadrangulation method. In *Proc. SIGGRAPH 2010*, 2010, pp. 118:1–118:8.

Model Sources

The models used herein to demonstrate our techniques have been obtained from the AIM@SHAPE repository, the Stanford 3D Scanning Repository, and the Image-based 3D Models Archive, Télécom Paris. The Guy model was initially created using Cosmic Blobs® by Dassault Systèmes Solidworks Corp.

Publications

of Marcel Campen, 2009-2014

Marcel Campen, Leif Kobbelt: Dual Strip Weaving: Interactive Design of Quad Layouts using Elastica Strips. In *Proc. SIGGRAPH Asia*, 2014.

Hans-Christian Ebke, Marcel Campen, David Bommes, Leif Kobbelt: Level-of-Detail Quad Meshing. In *Proc. SIGGRAPH Asia*, 2014.

Marcel Campen, Leif Kobbelt: Quad Layout Embedding via Aligned Parameterization. In *Computer Graphics Forum*, 33(8), 2014.

Marcel Campen, Martin Heistermann, Leif Kobbelt: Practical Anisotropic Geodesy. In *Proc. Eurographics Symposium on Geometry Processing*, 2013.

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, Leif Kobbelt: Integer-Grid Maps for Reliable Quad Meshing. In *Proc. SIGGRAPH*, 2013.

Hans-Christian Ebke, David Bommes, Marcel Campen, Leif Kobbelt: QEx: Robust Quad Mesh Extraction. In *Proc. SIGGRAPH Asia*, 2013.

Henrik Zimmer, Marcel Campen, Leif Kobbelt: Efficient Computation of Shortest Path-Concavity for 3D Meshes. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

Marco Attene, Marcel Campen, Leif Kobbelt: Polygon Mesh Repairing: An Application Perspective. In *ACM Computing Surveys*, 45(2), 2013.

Marcel Campen, David Bommes, Leif Kobbelt: Dual Loops Meshing: Quality Quad Layouts on Manifolds. In *Proc. SIGGRAPH*, 2012.

Henrik Zimmer, Marcel Campen, David Bommes, Leif Kobbelt: Rationalization of Triangle-Based Point-Folding Structures. In *Proc. Eurographics*, 2012.

Henrik Zimmer, Marcel Campen, Ralf Herkrath, Leif Kobbelt: Variational Tangent Plane Intersection for Planar Polygonal Meshing. In *Proc. Advances in Architectural Geometry*, 2012.

Marcel Campen, Leif Kobbelt: Walking On Broken Mesh: Defect-Tolerant Geodesic Distances and Parameterizations. In *Proc. Eurographics*, 2011.

Marcel Campen, Leif Kobbelt: Polygonal Boundary Evaluation of Minkowski Sums and Swept Volumes. In *Proc. Eurographics Symposium on Geometry Processing*, 2010.

Marcel Campen, Leif Kobbelt: Exact and Robust (Self-)Intersections for Polygonal Meshes. In *Proc. Eurographics*, 2010.

Darko Pavic, Marcel Campen, Leif Kobbelt: Hybrid Booleans. In *Computer Graphics Forum*, 29(1), 2010.

Marcel Campen: Ein Framework für Geometrieverarbeitung basierend auf hybriden Oberflächendarstellungen. In *Informatik Spektrum*, 33(1), 2010.

Marcel Campen: A Framework for Geometry Processing based on Hybrid Surface Representations. In *GI Lecture Notes in Informatics*, Volume S-8, 2009.