COMPUTER GRAPHICS *Forum* 2014

Quad Layout Embedding via Aligned Parameterization

M. Campen and L. Kobbelt

RWTH Aachen University, Germany

Abstract

Quad layouting, i.e. the partitioning of a surface into a coarse network of quadrilateral patches, is a fundamental step in application scenarios ranging from animation and simulation to reverse engineering and meshing. This process involves determining the layout's combinatorial structure as well as its geometric embedding in the surface. We present a novel quad layout algorithm that focuses on the embedding optimization, thereby complementing recent methods focusing on the structure optimization aspect. It takes as input a description of the target layout structure and computes a complete embedding in form of a parameterization globally optimized for isometry and, in particular, principal direction alignment. Besides being suited for fully automatic workflows, our method can also incorporate user constraints and support the tedious but common procedure of manual layouting.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—

1. Introduction

In diverse scenarios in domains like animation, simulation, design, reverse engineering, or meshing, surfaces need to be partitioned into a coarse base mesh of conforming quadrilateral patches. From a technical point of view this process of *quad layouting* involves determining the layout's combinatorial structure as well as its geometric embedding in the surface. The embedding describes the locations of the layout's nodes and arcs as well as parameterizations of its patches.

Traditionally, quad layouting is often performed manually by skilled professionals through the construction of nets of surface curves. The typical goal is to convert digitized workpieces or virtually sculpted models to structured, higher-order representations, e.g. on the basis of NURBS patches or subdivision surfaces, for which a quad partition serves as domain. Another use case is the generation of semi-regular quad meshes (also known as *multiblock grids*), which "represent the most important class [of quad meshes] in terms of applications" [BLP*13], providing advantages like enabling the use of adaptive solvers for simulation. Recently, promising approaches to automatic quad layouting have been proposed [TPP*11, BLK11, CBK12]. These methods' main focus is on the structural or *topological aspect* of the problem.

We present a novel quad layout parameterization algorithm that focuses on the *geometric aspect*, i.e. the embedding optimization, and thus ideally complements these structure optimization methods. It takes as input a description of the desired layout structure together with a (possibly very rough) initial embedding of the layout's nodes and arcs (cf. Figure 1) and outputs an embedding in form of a global parameterization optimized in a shape-aware manner, possibly with respect to additional user guidance or constraints.

With this generic setup, our method cannot only be used in automatic scenarios, it can likewise support the process of manual layouting, which is still often inevitable in the industry, as outlined by Li et al. [LRL06]. Starting from an

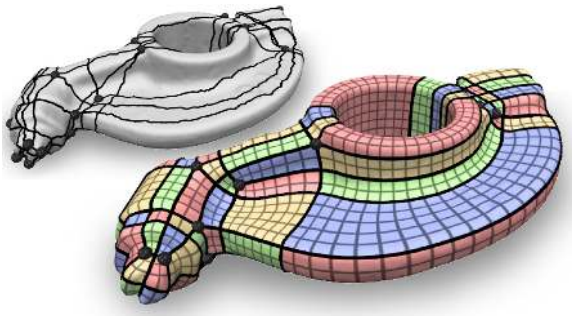


Figure 1: *Given a rough layout graph partitioning a surface into quadrilateral regions (top), our method creates a quad layout embedding and patch parameterization optimized for low distortion and alignment to principal directions.*

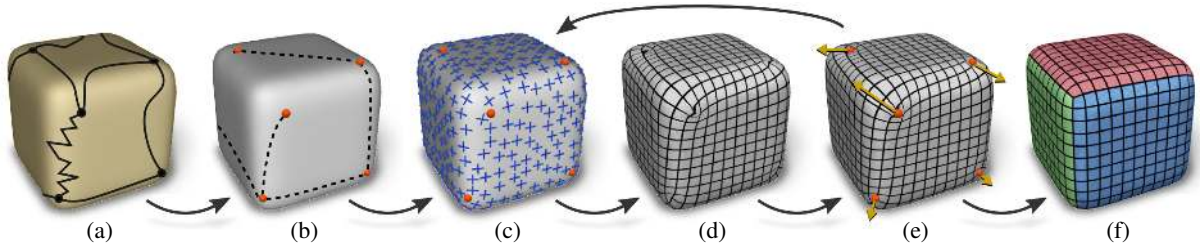
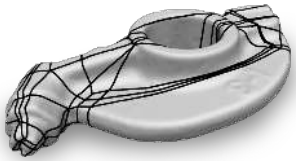


Figure 2: *Given a rough (manually or automatically generated) sketch of a layout with quadrilateral patches (a), the space of topologically compatible cross fields with suitable singularities is determined (b). Based on reliable principal curvature directions (and possibly feature information) a smooth, interpolating cross field is then created (c). Guided by this field, 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, together with smooth patch parameterizations (f).*

initial embedding roughly sketched by the user 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. 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 here on the layout from Figure 1 with geodesic arcs. Hence, our method takes an integrated, global approach.



The motivation for taking a two-step approach (1. structure determination, 2. embedding optimization) – where we provide a novel solution for step 2 – thus is two-fold: it allows to adhere to a user desired layout structure in semi-automatic workflows, and it allows to efficiently tackle the problem of automatic layout creation – whose hardness due to the complex interrelations between structure and geometry so far proves an ideal, simultaneous optimization elusive.

1.1. Principal Direction Alignment

The most important, essential difference of our method compared to previous approaches to embedding optimization is the fact that our formulation takes alignment (of isoparametric curves) to principal curvature directions into account (cf. Figures 1 and 7). This property's importance, beyond æsthetics, for prominent use cases of quad layouts, like semi-regular quad meshing (cf. Section 8), fitting of NURBS or other splines [EH96, LRL06], or subdivision fitting [LLS01] (which all build on parameterized quad layouts as starting point for optimization, refinement, etc.), 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 planarity [LXW*11], or achieving smooth curvature distribution (due to their tensor-product nature common spline surface representations are prone to ripples if aligned badly). Quad layouts and 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]. Hence, it is desirable to consider this alignment in the optimization.

1.2. Overview

Our method takes as input a quad layout graph with an initial embedding on a surface. Potential sources range from manual construction processes, like drawing a sketch on the surface, over assisted systems, to fully automatic layout structure generators (cf. Section 2). 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. The input layout is, however, assumed to be intended for a principal direction aligned embedding. While this is the case for modern automatic layout methods [CBK12, TPP*11, BLK11], a designer might arbitrarily (depending on the application intentionally) deviate from that. In such cases our method might not be the ideal choice as shown in Section 9.2.

Guiding Field (Section 4) In order to achieve alignment to principal curvature directions (where reliable), we build upon *field-guided parameterization*. 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 find a smooth cross field which aligns to specified directions.

Aligned Parameterization (Section 5) 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 can be extracted from the parameterization.

Node Optimization (Section 6) At the heart of our method, a meta-optimization strategy then optimizes 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 7) 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 2 these stages of our algorithm are illustrated. Our key contributions are:

- a description of how to, for the specific case of layouts, 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 (Sections 4 and 5).
- a novel concept to continuously optimize layout node positions, directly driven by embedding quality, based on an efficient meta-optimization strategy (Sections 6 and 7).

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 the recent [BCE*13]) and, even more important, the structure is a product of the algorithm itself. We strive for parameterization with a *prescribed*, 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 1.1. A comparison of alignment-oblivious and alignment-aware layout optimization is provided in Section 9.1. 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. Recent works in the area [CBK12, MZ12] state the lack of alignment-awareness as "fundamental limitation" whose addressing would require "substantial changes". 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.

3. Preliminaries

Let G be a multigraph, called *layout graph*, which may contain dangling arcs which are incident to only one node. Consider an embedding of G in an oriented manifold surface $\mathcal{S}$ (with or without boundary $\partial\mathcal{S}$) such that arcs only intersect at their endpoints and dangling arcs end on $\partial\mathcal{S}$. This partitions the surface into regions (called *patches*) bounded by embedded arcs, nodes, and possibly $\partial\mathcal{S}$ – such patches bounded partially by $\partial\mathcal{S}$ are called *trimmed*, all others *full*. If all patches are homeomorphic to disks, the embedding is a *2-cell embedding (with boundary)*.

A *quad layout* $\mathcal{L}$ is a layout graph G with a 2-cell embedding in surface $\mathcal{S}$ where each full patch is of valence 4, i.e. there are 4 (not necessarily distinct) nodes along the patch border, and each trimmed patch is of valence < 4. Figure 2 (a) shows a simple example of such layouts. Let g denote the genus and b the number of boundaries of $\mathcal{S}$. Further, s is the number of irregular (valence $\neq 4$) nodes in G.

Discrete Representation We consider a triangulation $\mathcal{M}$ of $\mathcal{S}$, assumed to be free of degeneracies and noise since this is required by the basic techniques we build upon (guiding field and parameterization optimization). The embedding of G in $\mathcal{M}$ is such that nodes are mapped to vertices. Let $\mathsf{v}(h)$ denote the vertex which node h is mapped to. For an oriented embedded arc a let $\boldsymbol{\gamma}'(a)$ denote the sequence of edges crossed when traveling along a from start to end, $\boldsymbol{\gamma}(a)$ the sequence of faces crossed. $\boldsymbol{\gamma}(a)_{\vdash}$ denotes the first face in the sequence, $\boldsymbol{\gamma}(a)_{\dashv}$ the last one. Details on how to obtain this representation from a given layout are given in Appendix A.

Rotation System A compact representation of the topology of the quad layout $\mathcal{L}$ is its *rotation system*: a pair (σ,θ) of permutations acting on the set of half-arcs (for each arc there exist two 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 an half-arc to the next one in the clockwise (with respect to the orientation of $\mathcal{S}$) cyclic order of half-arcs incident to the same node.

4. Guiding Field

In order to obtain a guiding field for the subsequent parameterization step, we construct a smooth tangent 4-symmetry direction field (*cross field*) C on $\mathcal{S}$ that topologically conforms with $\mathcal{L}$ (Section 4.1) and geometrically follows principal directions and sharp features of $\mathcal{S}$ (Section 4.2).

4.1. 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 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.

For an irregular node its (clockwise) turning number t is determined from its valence v as $t=-\frac{1}{4}v$. For a homology generator or boundary cycle c the turning number needs to be fixed to $t=\pm\frac{1}{4}(n_n-n_a)$, where n_a is the number of arcs crossed by c and n_n the number of nodes in the closest homotopic arc cycle c'. The sign is determined by the relative orientation of c and c' – for details we refer to Appendix B.

In the discrete setting $\mathcal{C}$'s topology can 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 equivalent alternative (which, however, only works for complete layouts, not partial ones, cf. Section 10).

4.2. 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. 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.

5. Aligned Parameterization

Now we formulate a global parameterization problem to simultaneously optimize the embedding of arcs and patch interiors guided by $\mathcal{C}$. 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

$$E=\sum_{t\in T}\left(||\nabla_t u-\mathbf{u}_t||^2+||\nabla_t v-\mathbf{v}_t||^2\right)A_t\ \rightarrow\ \min \qquad (1)$$

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

$$(u,v)_t=R(r_{st})(u,v)_s+(j_{st},k_{st}) \qquad (2)$$

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 (2) with fixed r and variables (j,k) are then incorporated as linear constraints into (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 3). 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 (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 3 right). This ultimately allows us to derive an embedding for $\mathcal{L}$ from $\mathcal{P}$.

5.1. Node Connection Constraints

We have to ensure for each arc that the two incident nodes lie on a common iso-parametric curve of $\mathcal{P}$. 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

$$\left[\tau(a)(u,v)_s\right]_{\lambda(a)} = \left[(u,v)_t\right]_{\lambda(a)} \tag{3}$$

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 (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.

5.1.1. 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.

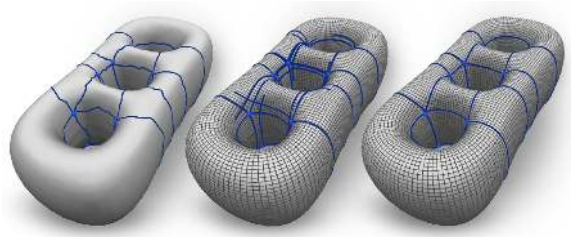


Figure 3: *Left: input layout. Middle: intermediate state of tracing iso-parameter curves from the singularities in an unconstrained parameterization to obtain the base complex. Right: base complex of a parameterization with node connection constraints; now structurally equivalent to the input.*

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 $\mathcal{L}$:

$$\bar{\lambda}(\sigma(a)) = \tau_\sigma(a)\,\bar{\lambda}(a) \tag{4}$$

$$\bar{\lambda}(\theta(a)) = \mathrm{Rot}(\pi/2)\,\tau_\theta(a)\,\bar{\lambda}(a). \tag{5}$$

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 (4) and (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 (4) and (5).

With constraints (3) in effect, we then compute $\mathcal{P}$ (1) whose base complex is now structurally equivalent to $\mathcal{L}$ (cf. Figure 3 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 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].

5.2. Embedding Extraction

A global parameterization $\mathcal{P}$ with a base complex structurally equivalent to $\mathcal{L}$ naturally induces an embedding for

$\mathcal{L}$ together with coherent patch parameterizations (cf. Figure 3 right). The arcs' embedding is found as iso-parametric curves starting from the nodes until another node is reached.

An individual parameterization for each patch over a rectangular domain $[0,w] \times [0,h]$ 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 curves emanating from the corner nodes is parameterized over some rectangle $[a,b] \times [c,d]$ by $\mathcal{P}$. If transitions are involved, we need to express all parameter values with respect to a common chart (cf. Appendix D for implementation details). As each patch is disc-homeomorphic and contains no singularities in its interior, this is possible without ambiguity.

5.3. Optional Extensions

We would like to point out that the functional (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. 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, it further gives rise to large distortions or even local non-injectivities due to fold-overs (cf. Figures 4 and 5). 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.

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 (1) – higher overall distortion is traded for local improvements (cf. Section 6.3). 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 4 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 2.

Technically speaking, we are going to optimize (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.

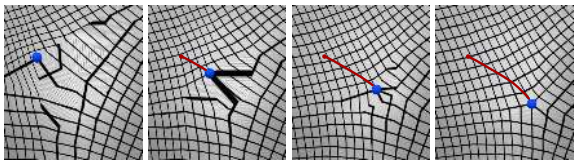


Figure 4: *Visualization of the trajectory (red) of a node (blue) as it moves in negative gradient direction* $\mathbf{d}$*. In the beginning severe distortions and inversions are present which successively vanish in the course of the movement.*

6.1. Gradient Descent

In order to locally move a node h in such a way that the residual of (1) (which we now just call E for brevity) decreases, we need to move this node in direction

$$\mathbf{d}(h) = -\left(\frac{\partial}{\partial x} E(h_x, h_y),\ \frac{\partial}{\partial y} E(h_x, h_y) \right) \tag{6}$$

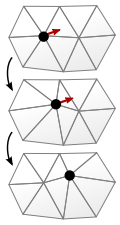
i.e. in a gradient descent manner. Here (x,y) are 2D coordinates in some local coordinate chart of $\mathcal{S}$ around h and (h_x, h_y) expresses the current position of node h accordingly. Note that E depends on x and y because nodes are embedded in vertices, i.e. node positions are vertex positions. In Section 7 we address the computation of $\mathbf{d}(h)$ as well as of a suitable corresponding descent step length $l(h)$ in detail.

6.1.1. Node Movement

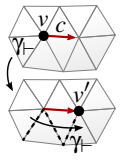
Assume node h is currently at position p. To determine the new position p' on $\mathcal{M}$ we trace a straightest geodesic g of length $l(h)$ starting at p in direction $\mathbf{d}(h)$ [PS98] (stopping if a boundary is reached). If h lies on a feature curve, we first project $\mathbf{d}(h)$ onto it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



6.1.2. 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.

6.2. 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.

6.3. 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 in-between. 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.

	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 change caused by moving a singularity. 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 (1). In this context singularity locations are typically determined a priori [KNP07, BZK09, MZ12], with only limited awareness of implications for the parameterization. The advantage of relocating them (directly driven by parameterization quality) over the alternative of using stiffening is demonstrated in the following example (using the MIQ approach [BZK09]):

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

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 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.

7. Gradient Computation

The gradient (6) of (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 $\mathbf{v}(h)$ 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 (cf. Appendix C for pseudo code).

7.1. Efficient Estimator

Let's consider the energy functional (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.

This approximation 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 5 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.

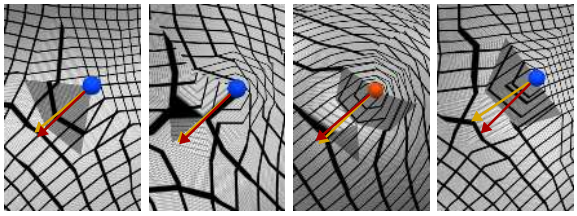
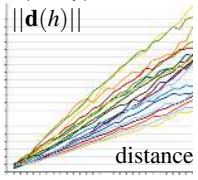


Figure 5: *Visualization of descent directions: negative gradient* $\mathbf{d}$ *(yellow) and our estimator* $\tilde{\mathbf{d}}$ *(red). The angular deviation is typically very low, even in complicated regions with severe distortions and inversions (shaded darker).*

7.2. 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 $\mathcal{M}$.

8. Quad Mesh Generation

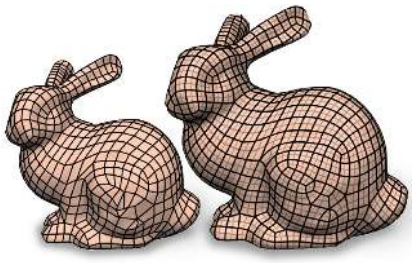
One particular use case of quad layouts is the generation of quad meshes via subsequent refinement, as the block structure of the resulting meshes provides specific advantages [BLP*13]. One option is *a posteriori* subdivision of each patch parameterization into an $m \times n$ grid, where m and n comply 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 (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].

9. Results

We applied the proposed method (using the efficient gradient estimator) to input layouts sketched manually as well as to layouts constructed by automatic methods [BLK11, CBK12]. Figure 6 shows the input and output layouts, Table 1 the corresponding statistics. The patch parameterizations are visualized using $m \times n$ grid textures (as described in Section 8). The accompanying video demonstrates the optimization process on several examples.

We used CHOLMOD [CDHR08] for the equation systems and IPOPT [WB06] to handle the orientation constraints. Runtime is dominated by the repeated systems solving – the complexity of the layout has only little influence (cf. Table 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.

9.1. 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.

The following table shows 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 $E_{\text{planar}} = \frac{1}{\sqrt{A}}\int_{\mathcal{M}} \text{II}\left(\nabla u/||\nabla u||, \nabla v/||\nabla v||\right)$ which, based on $\mathcal{M}$'s second fundamental form II [LXW*11], quantifies how non-planar the (infinitesimal) elements of a quad mesh generated from the parameterization would be.

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 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

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 principal direction field prevents $E^\circ = 0$ in general). In combination with good conformality (no large difference between both methods) our method also generally achieved better E_{planar} values. Figure 7 illustrates the differences.

9.2. Limitations

There are types of input which are problematic for the proposed method, as outlined in the following.

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 8a is surrounded by a distorted but still valid parameterization, there are fold-overs around the valence 12

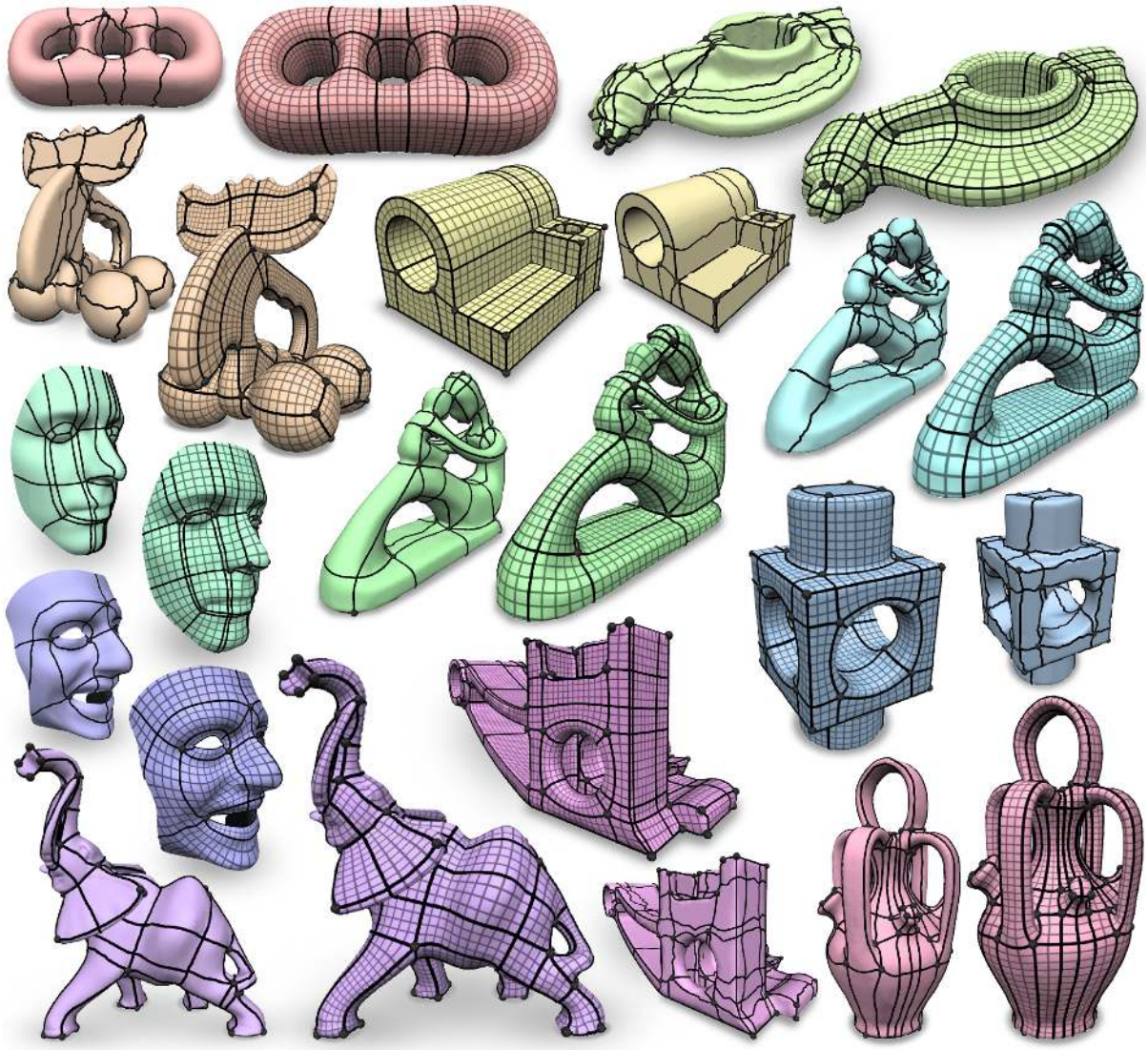


Figure 6: *Input and result layouts. Feature alignment has been used for models* JOINT *(yellow) and* LEVER *(pink), boundary alignment on eyes and mouth of the* MASK *model (blue). Irregular nodes are displayed as little spheres for visual orientation.*

node in Figure 8b – fold-overs which (in the 1-ring) 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 parameterization all inner angles are less than π. Note that this is a general problem of PL triangle parameterizations [NP09].

Principal direction conflict Figure 8c/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.

Overly coarse structure The layout in Figure 8e 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.

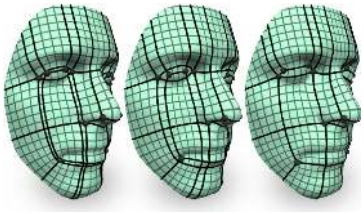
Notice that in all these cases it is questionable whether a method aiming for principal direction alignment is the right choice. It seems unlikely that such layouts are actually intended to be aligned, or that there even exists a solution with reasonable alignment. Layout optimization methods which do not aim for alignment (cf. Section 2) are then preferable.

10. Outlook & Future Work

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, we experimented with node spacing constraints – similar to the node connection constraints (3). 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 $\left[\tau(a)(u,v)_s\right]_i = \left[(u,v)_t\right]_i \pm \varepsilon$ 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. If such structure modification is desired we can constrain the parametric distance to zero when it drops below ε, effectively enforcing a *poly-chord collapse*. These two options are illustrated here next to the standard solution (left). We leave in-depth exploration of such possibilities to future work.



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 layout system with interactive feedback.

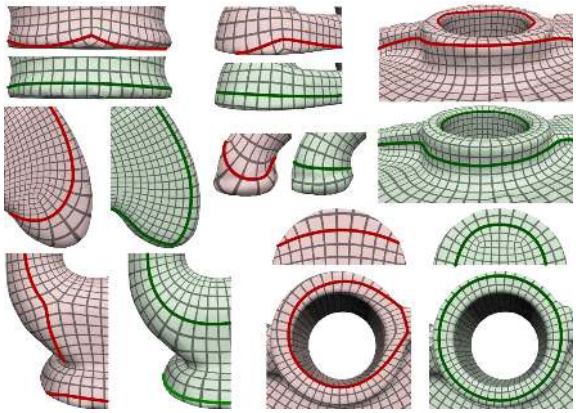


Figure 7: *Zoom-ins comparing TPP (red) and our method (green). Some isocurves are highlighted to ease visual alignment quality comparison. (cf. Appendix E for full models.)*

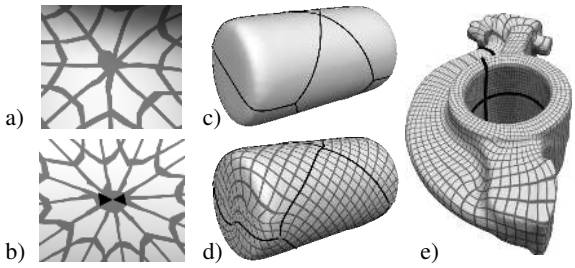


Figure 8: *Problematic input configurations: high valence nodes, principal direction conflicts, overly coarse structure.*

11. Conclusion

We have presented a method for the simultaneous 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. Its applicability extends to related areas like quadrangular remeshing. With these capabilities our method complements recent approaches to automatic quad layout structure generation and offers support to the process of manual quad layouting.

Acknowledgements

Thanks go to David Bommes and Henrik Zimmer for numerous helpful discussions and to Jan Möbius for support with the OpenFlipper framework. Models have been obtained from the AIM@SHAPE repository, the Stanford 3D Scanning Repository, and the Image-based 3D Models Archive, Télécom Paris. This research project was funded by the DFG Cluster of Excellence *UMIC* (DFG EXC 89).

References

[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.

[ACSD*03] ALLIEZ P., COHEN-STEINER D., DEVILLERS O., LÉVY B., DESBRUN M.: Anisotropic polygonal remeshing. *ACM Trans. Graph. 22*, 3 (2003), 485–493.

[BCE*13] BOMMES D., CAMPEN M., EBKE H.-C., ALLIEZ P., KOBBELT L.: Integer-grid maps for reliable quad meshing. In *Proc. SIGGRAPH 2013* (2013), pp. 98:1–98:12.

[BK01] BOTSCH M., KOBBELT L.: Resampling feature regions in polygonal meshes for surface anti-aliasing. *Comput. Graph. Forum 20*, 3 (2001), 402–410.

[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).

[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.

[BVK08] BOMMES D., VOSSEMER T., KOBBELT L.: Quadrangular parameterization for reverse engineering. *Mathematical Methods for Curves and Surfaces* (2008), 55–69.

[BZK09] BOMMES D., ZIMMER H., KOBBELT L.: Mixed-integer quadrangulation. In *Proc. SIGGRAPH 2009* (2009), pp. 1–10.

[CBK12] CAMPEN M., BOMMES D., KOBBELT L.: Dual loops meshing: quality quad layouts on manifolds. In *Proc. SIGGRAPH 2012* (2012), p. 110.

[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 (SGP) 29*, 5 (2010), 1525–1533.

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. In *Proc. SIGGRAPH 2004* (2004), pp. 905–914.

[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.

[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.

[EBCK13] EBKE H.-C., BOMMES D., CAMPEN M., KOBBELT L.: Qex: Robust quad mesh extraction. In *Proc. SIGGRAPH Asia 2013* (2013), pp. 168:1–168:10.

[EH96] ECK M., HOPPE H.: Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *Proc. SIGGRAPH 96* (1996), pp. 325–334.

[FH05] FLOATER M. S., HORMANN K.: Surface parameterization: a tutorial and survey. In *Advances in Multiresolution for Geometric Modelling*. Springer, 2005, pp. 157–186.

[GVSS00] GUSKOV I., VIDIMCE K., SWELDENS W., SCHRÖDER P.: Normal meshes. In *Proc. SIGGRAPH 2000* (2000), pp. 95–102.

[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.

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384.

[KS04] KRAEVOY V., SHEFFER A.: Cross-parameterization and compatible remeshing of 3d models. In *Proc. SIGGRAPH 2004* (2004), pp. 861–869.

[Lip12] LIPMAN Y.: Bounded distortion mapping spaces for triangular meshes. In *Proc. SIGGRAPH 2012* (2012), pp. 108:1–108:13.

[LLS01] LITKE N., LEVIN A., SCHRÖDER P.: Fitting subdivision surfaces. In *IEEE Visualization 2001* (2001), pp. 319–324.

[LRL06] LI W.-C., RAY N., LÉVY B.: Automatic and interactive mesh to t-spline conversion. In *Proc. SGP '06* (2006), pp. 191–200.

[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.

[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.

[MPKZ10] MYLES A., PIETRONI N., KOVACS D., ZORIN D.: Feature-aligned T-meshes. In *Proc. SIGGRAPH 2010* (2010), pp. 117:1–117:11.

[MZ12] MYLES A., ZORIN D.: Global parametrization by incremental flattening. In *Proc. SIGGRAPH* (2012), pp. 109:1–109:11.

[Nie12] NIESER M.: *Parameterization and Tiling of Polyhedral Surfaces*. PhD thesis, Freie Universität Berlin, 2012.

[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.

[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), 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.

[RLL*06] RAY N., LI W. C., LÉVY B., SHEFFER A., ALLIEZ P.: Periodic global parameterization. *ACM Trans. Graph. 25* (2006), 1460–1485.

[RVAL09] RAY N., VALLET B., ALONSO L., LEVY 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.

[SAPH04] SCHREINER J., ASIRVATHAM A., PRAUN E., HOPPE H.: Inter-surface mapping. In *SIGGRAPH* (2004), pp. 870–877.

[TACSD06] TONG Y., ALLIEZ P., COHEN-STEINER D., DESBRUN M.: Designing quadrangulations with discrete harmonic forms. In *Proc. SGP '06* (2006), pp. 201–210.

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. In *Proc. SIGGRAPH* (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.

[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.