Generalized Motorcycle Graphs for Imperfect Quad-Dominant Meshes

NICO SCHERTLER, TU Dresden
DANIELE PANOZZO, New York University
STEFAN GUMHOLD, TU Dresden
MARCO TARINI, Università degli Studi di Milano and ISTI - CNR

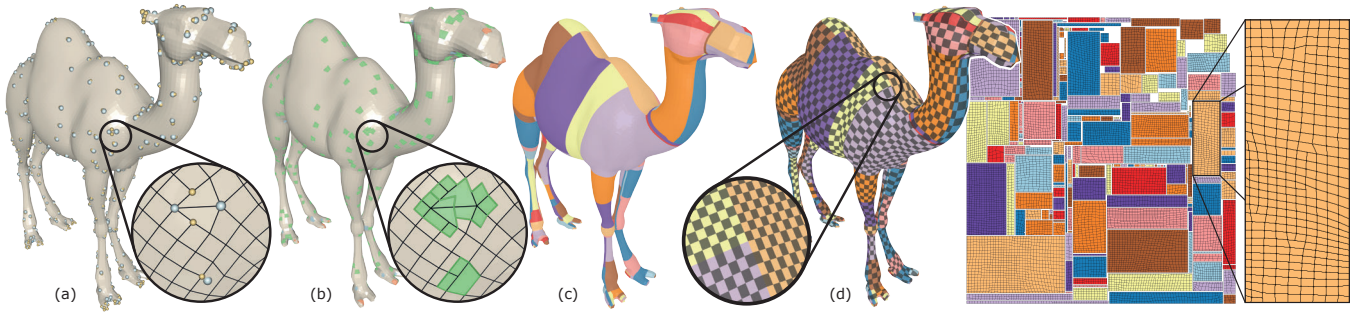


Fig. 1. Our method takes as input a semi-regular quad-dominant mesh (a – singularities marked with spheres) and produces a global parametrization. For that purpose, we evaluate the effect of singularities on the mesh topology with the help of *fenced regions* (b). We use this information to calculate a Generalized Motorcycle Graph (c), whose patches serve as rectangular domains for the parametrization (d). The parametrizations of adjacent patches are aligned to each other on a majority of cuts (see close-up), which allows to make them invisible in texturing applications. The rectangular shape of patches in the 2D parametric domain allows highly efficient packing of the texture.

We introduce a practical pipeline to create UV T-layouts for real-world quad dominant semi-regular meshes. Our algorithm creates large rectangular patches by relaxing the notion of motorcycle graphs and making it insensitive to local irregularities in the mesh structure such as non-quad elements, redundant irregular vertices, T-junctions, and others. Each surface patch, which can contain multiple singularities and/or polygonal elements, is mapped to an axis-aligned rectangle, leading to a simple and efficient UV layout, which is ideal for texture mapping (allowing for mipmapping and artifact-free bilinear interpolation). We demonstrate that our algorithm is an ideal solution for both recent semi-regular, quad-dominant meshing methods, and for the low-poly meshes typically used in games and movies.

Additional Key Words and Phrases: Texture Mapping, Motorcycle Graph, Parametrization

ACM Reference format:
Nico Schertler, Daniele Panozzo, Stefan Gumhold, and Marco Tarini. 2018. Generalized Motorcycle Graphs for Imperfect Quad-Dominant Meshes. *ACM Trans. Graph.* 37, 4, Article 155 (August 2018), 16 pages.
DOI: 10.1145/3197517.3201389

This work was partially supported by project 03ZZ0516A of the German Federal Ministry of Education and Research (BMBF), NSF CAREER award 1652515, MIUR project "DSURF" (PRIN 2015B8TRFM), a gift from Adobe Research, and a gift from NTopology.

DOI: 10.1145/3197517.3201389

1 INTRODUCTION

Quad-dominant semi-structured meshes, i.e. meshes that are predominantly composed of quadrilateral faces and regular vertices, are ubiquitous in computer graphics: they are the de-facto standard in the visual effects and computer animation industry and are also often used in most interactive applications. However, *clean* structured meshes are expensive to create. The number of irregular vertices is often too large and most pipelines do not support common small imperfections, requiring manual cleaning. This slows down existing content creation pipelines and prevents the direct usage of scanned models since reconstruction methods rarely create models adhering to these strict guidelines, even after an automatic cleanup.

We propose a practical way to make imperfect polygonal meshes more directly usable in downstream applications, by allowing the computation of a valid UV-layout robust to redundant irregular vertices as well as to quad mesh imperfections such as small holes, non-quadrilateral faces, T-junctions, small handles and tunnels, and non two-manifold configurations.

Our main contribution is the definition of a generalization of *Motorcycle Graphs* (MCG) [Eppstein et al. 2008]: our construction is identical on clean, highly regular quad meshes but gracefully handles imperfections and redundant irregular vertices (Section 1.1), always producing a valid segmentation into rectangular UV patches. The key idea is to identify a set of regions on the mesh which are to be considered equivalent to completely regular and clean grids for the purpose of the MCG algorithm, in spite of imperfections. Likewise, other regions will be treated as if containing one isolated irregular vertex, again disregarding the more complex actual configuration of the local meshing.

The induced patch decomposition is used to define a global parametrization targeted for texture mapping applications. The UV layouts are easy to pack, avoid interpolation (or MIP-mapping) artifacts at cuts, and are directly usable in real-time rendering pipelines without any manual cleanup.

We applied our algorithm to hundreds of scanned and hand-made models, demonstrating its robustness and practical applicability.

1.1 Motivations: Imperfect Quad-Dominant Meshes

Meshes that are **purely quadrilateral**, **extremely regular**, and **free from imperfections** are ideal, since they are maximally malleable to all kinds of processing. Unfortunately, they are also very challenging to create – either manually (by modelling artists) or by automatic approaches (e.g. remeshing range scanned data).

A much more common case is that of semi-regular, quad dominant meshes, often also coming with additional local imperfections. We will refer to this class of meshes as *imperfect (quad-dominant) meshes*. Specifically, these meshes feature one or more of:

Higher number of irregular vertices. In an ideal case, irregular (non valence 4) vertices are justified by the geometric shape of the surface, i.e., they are found in correspondence with high curvature regions. Imperfect meshes, in contrast, feature many more irregular vertices serving many other purposes, for example to control tessellation density or to orient edge along certain directions. Redundant irregular vertices are also introduced by suboptimal automatic quad-remeshing algorithms.

Non quadrilateral faces. Quad-dominant meshes present occasional triangular or pentagonal elements (and at times other polygons too). These elements are often introduced by re-meshing algorithms and modelling artists alike.

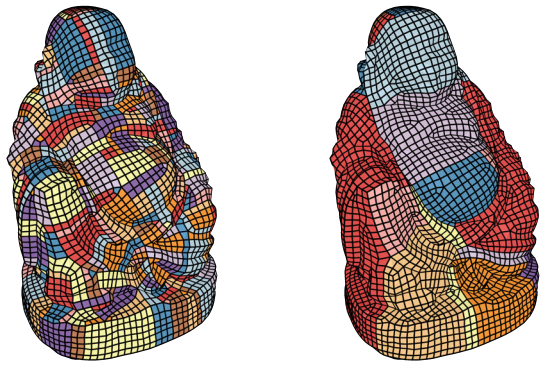
Other meshing imperfections. Small holes (missing data), topological noise (unwanted small handles/tunnels), jagged (rather than straight) boundaries, T-junctions, or local lack of two-manifoldness are common in range scanned surfaces, in procedural meshes, and in many manually modeled meshes.

Imperfect meshes can be directly captured [Schertler et al. 2017], produced by means of remeshing [Jakob et al. 2015], or directly modeled (compare e.g. [Denning et al. 2011]). They are the majority of the meshes available in online repositories (e.g. [TurboSquid 2018]). They represent an intermediate case between perfectly structured meshes, which are difficult to construct, and irregular structures, such as irregular triangle meshes or range scans, which are difficult to process. Our algorithm exploits their regularity to provide a high-quality output, but it reliably tolerates local imperfections.

1.2 Method Overview

Our method takes as input a semi-regular, but potentially imperfect, quad mesh M and produces a parametrization of M over a set of 2D *rectangular patches*, which are then packed tightly into one texture.

Objectives. Because we target texture mapping, we strive to limit the number of patches and therefore the amount of texture cuts, which are a source of rendering artifacts, memory overheads, and other complications in textures [Tarini et al. 2017]. In many scenarios, it is also desirable to preserve the original mesh connectivity, (e.g. to preserve geometry features, respect the modeller's choice of edge placement, etc.), although it is usually acceptable to refine a small number of faces locally, introducing a few additional edges, to represents cuts at these edges.



(a) Original Motorcycle Graph (b) Generalized Motorcycle Graph

Fig. 2. The original Motorcycle Graph (left) results in a heavy oversegmentation of the bouddha with 385 patches. Our Generalized formulation can extract a T-layout with only 24 patches.

We partition the mesh into a small number of rectangular patches and we parametrize each patch over an axis-aligned 2D rectangle. In other words, we extract a coarse T-layout from the original mesh (Section 3). Subsequently, we determine the parametric size of each 2D rectangle (Section 4), and construct a distortion-minimizing parametrization, i.e. a mapping from each patch into the corresponding 2D parametric rectangle. Finally, the 2D rectangles are packed into one unified texture domain (Section 5).

The coarse quad-layout construction is the core part of our approach, which is a generalization of Motorcycle Graphs. On a highly regular, imperfection-free, pure quad mesh, an MCG will automatically and robustly produce good, coarse quad-layouts. Unfortunately, MCGs are not applicable in the presence of imperfect or non pure quad meshes, producing excessively fine-grained partitions if the input is not highly regular everywhere. Our generalization (see Section 3) bypasses these limitations (see Figure 2).

2 RELATED WORK

Surface parametrization is the task of constructing an injective mapping between a given input surface S and a (typically) flat parametric domain D. *Global* surface parametrization is required when S is not topologically equivalent to a disk. In this case, cuts are introduced to split the surface into topological disks and to reduce distortion. Global surface parametrization has been extensively studied in the last three decades: we focus on the most closely related approaches, and we refer to [Floater and Hormann 2005] for an overview.

The intended application of our global parametrizations is to serve as UV-maps for *texture mapping*. This application imposes specific objectives, which are subtly different from those commonly considered for remeshing applications.

2.1 Parametrizations for Texture Mapping

In texture mapping, cuts cause rendering artifacts known as texture bleeding, which are due to bilinear interpolation and MIP-mapping. These artifacts can be greatly reduced by replicating texels; this, however, costs GPU memory and leaves residual artifacts due to a mismatch in the texel grid, which are especially notable under extreme magnification. Therefore, cuts can be tolerated but are undesirable. Construction methods seek a good trade-off between the amount of cuts and of distortion, either implicitly [Lévy et al. 2002; Smith and Schaefer 2015; Tarini 2016], or, in one recent case, by minimizing an energy explicitly accounting for both [Poranne et al. 2017]. Our approach offers a similar trade-off. In addition, our cuts are always axis-aligned in parametric space, miminizing the GPU memory overhead for texel replications. Optionally, most of them can be made *invisible*, in the following sense.

Cut Invisibility. For a special class of cuts, which we call *invisible*, rendering artifacts are completely negated by texel replications. This was first explicitly observed in [Ray et al. 2010], but was also exploited in approaches like [Carr et al. 2006; Tarini et al. 2004]. In [Liu et al. 2017] a wider generalization of invisible cuts is offered, but this comes at the cost of limiting the assigned texel values. Invisible cuts also avoid artifacts introduced by MIP-mapping up to a prescribed level k. It is sufficient for the parametrization to be computed for the resolution of MIP-map level k, and then up-scaled to the highest resolution level 0.

Alternatives to Global Parametrizations. Although surface parametrizations are the standard approach to texture mapping, a long-lasting trend is to try to bypass its construction altogether, for example by endowing each mesh element with its own parametric domain [Burley and Lacewell 2008; Yuksel 2017]. Similarly, parametrizations are sometimes computed and stored volumetrically, bypassing the complications traditionally associated to cuts [Tarini 2016]. The reader is referred to [Tarini et al. 2017] for a gallery of other alternative approaches to texture mapping. These techniques, however, require changes of the standard real-time rendering pipeline, the asset production pipeline, or both.

2.2 Coarse Quad Layouts

Our parametric domain D is defined as the union of 2D rectangles, one for each patch. Global parametrizations in this class are enticing because 2D rectangles can be efficiently packed in a global texture sheet (in addition to the advantages given by axis-aligned patch boundaries).

The problem of producing a coarse quadrilateral layout over a surface has been extensively studied; we refer the reader to a survey [Bommes et al. 2013, Section 3.2] and a tutorial [Campen 2017].

Existing works are motivated by different purposes, such as regular quad-remeshing (each patch is subdivided into a regular quad grid e.g. [Campen et al. 2015]), construction of higher-order approximations (each patch represents one element of a quad control mesh for subdivision of parametric surfaces, e.g. [Panozzo et al. 2011]), detecting isomorphisms between meshes (isomorphic meshes share the same patch layout, e.g. [Eppstein et al. 2008]), or, like in our case, surface parametrization (each patch serves as one parametrization domain, e.g. [Bommes et al. 2009]). The objectives include topological correctness, domain coarseness, good patch shape, and alignment of patch boundaries to feature lines and/or curvature directions.

Existing solutions include drastically different approaches, for example based on Morse-Smale complexes [Ling et al. 2014], 3D morphing into piecewise axis-aligned surfaces [Fu et al. 2016], iterative coarsening of an initially densely tessellated quad mesh [Panozzo et al. 2011], following an internal skeleton [Usai et al. 2015], casting the problem as a coarse remeshing [Bommes et al. 2013], or tracing of boundary lines over the surface [Campen and Zorin 2017; Razafindrazaka and Polthier 2017].

Our approach falls in the latter category but has the following differences.

Input differences. Competing tracing-based approaches focus mainly on two types of input surface representation:

Irregular triangular meshes with an accompanying cross field [Campen et al. 2015; Pietroni et al. 2016; Ray and Sokolov 2014]. In this case, the challenge is to robustly trace straight lines over a piecewise linear, irregularly sampled surface. This requires extreme care during the implementation, often with sophisticated algorithms devoted to the sub-problems. Also, an accompanying cross field is required, which is not always available. E.g., manually edited meshes do not have one. Finally, the quality of the cross field is crucial: in particular, the amount of singularities will heavily affect layout coarseness. In [Campen and Kobbelt 2014], an initial parametrization and user-drawn sketches mimic a cross field and act as a guidance for tracing lines, sharing similar challenges.

Highly regular pure quad meshes. This class of meshes simplifies processing; tracing becomes simply a straight traversal of mesh edges across regular vertices, as exploited in [Eppstein et al. 2008; Tarini et al. 2011], allowing for simple, efficient, and robust implementations. Unfortunately, this class of meshes is rare and difficult to automatically generate. The methods are very sensitive by construction to redundant irregular vertices and other potential problems (Section 1.1). In contrast, our method has fewer assumptions on the input and is able to process imperfect input surfaces (Sec. 1.1).

Output differences. Existing tracing-based approaches can be categorized into two groups according to the desired output:

Conforming quad layouts, i.e. layouts that are free from T-junctions [Campen et al. 2012; Fu et al. 2016; Usai et al. 2015]. In this case, 2D rectangles have side-to-side adjacency relationships, which is challenging to achieve. The resulting layout tends to be much less coarse [Daniels et al. 2008]. On the other hand, this provides advantages for remeshing purposes [Bommes et al. 2013]. To simplify their computation, field alignment is often sacrificed in exchange for this property [Bommes et al. 2011; Campen et al. 2012; Razafindrazaka et al. 2015; Tarini et al. 2011].

T-layouts, i.e. layouts with T-junctions [Campen et al. 2015; Campen and Zorin 2017; Eppstein et al. 2008; Myles et al. 2010; Pietroni et al. 2016], where two rectangular domains may share only a part of an edge. The ability to insert T-junctions enlarges the solution space, allowing for coarser layouts with milder distortion.

Our approach, being a generalization of [Eppstein et al. 2008], targets T-layouts. This choice is justified by our application context. Other common applications for T-layouts include T-splines and T-NURCCS [Campen and Zorin 2017; Myles et al. 2010; Pietroni et al. 2016].

T-layout with scaling. We further increase the flexibility of our layout by allowing the *transition functions* (i.e. the functions mapping the two sides of a cut to each other in parametric space) to include small integer scaling factors ≥ 1 (Section 4). This change dramatically increases the solution space, while not introducing any downside for texture mapping (Section 2.1). Our formulation for parametric sizes is thus analogous to [Campen and Zorin 2017], but our context does not require to enforce loop conditions around vertices. This allows for non-degenerate solutions even in otherwise unsolvable configuration such as the one shown in [Campen and Zorin 2017, Fig.6] or [Karciauskas et al. 2017, Fig.2].

2.3 Relationship to Field-Aligned Remeshing Approaches

A class of field-guided methods construct a semi-regular quad mesh driven by two related but distinct fields defined on the surface, which are often computed in cascading order [Jakob et al. 2015; Ray et al. 2006; Schertler et al. 2017]. In [Jakob et al. 2015], they are termed *RoSy* and *PoSy* field. The *RoSy* field is a tangent vector field that determines the local orientations of the edges, whereas the *PoSy* field is a position field that determines the positions of elements in parametric space. Each field comes with its own set of singularities: a cross-field singularity will produce a single irregular vertex in the final quad mesh. Conversely, a positional-field singularity will be translated in either a small configuration of irregular vertices, non quadrilateral elements, or a T-junction. Our approach can be understood as a way to classify these cases solely analyzing the final mesh. The latter cases will be embedded inside irregular fenced regions, and the former in regular ones. It could be argued that this observation implies that a better solution would be to base the analysis on the fields that produced the input mesh. Our motivation for relying solely on the final mesh instead is based on a better generality and a wider applicability, also considering that *imperfect* quad-dominant meshes have different origins (see Sec. 1.1).

2.4 Relationship to Mesh Optimization Approaches

Our method is reminiscent of mesh optimization approaches, which change the connectivity of an input quad-mesh striving to reduce the number of its irregular vertices (among other objectives). For example, [Peng et al. 2011] presents a set of local connectivity operators to relocate configurations of irregular vertices, which can be combined to bring closer and then cancel pairs of irregular vertices of opposite valence excess, such as a valence 3 with a valence 5. In [Verma and Suresh 2015, 2016], local patches containing irregular vertices are identified and individually remeshed more regularly; these patches resemble our proposed fenced region. With respect to any approach in this category, important differences stem from our targeted application: in our case, we do not to change the original mesh but only define a parametrization with controlled resolution jumps for it. Local changes of the connectivity of a quad mesh require to respect scrupulous conditions to limit domino effects, which would otherwise propagate over the entire structure (see [Daniels et al. 2008]), whereas we can afford to be more aggressive. For example, our algorithm will consider one of our fenced region containing a single 3-5 pair of irregular vertices as regular, which does not complexify the T-layout, whereas any quad-mesh optimization approach cannot simplify this configuration in isolation. Informally speaking, our *valence cancellation* effect is, therefore, more similar to the singular-point cancellation experienced when smoothing a cross-field (e.g. [Jakob et al. 2015]).

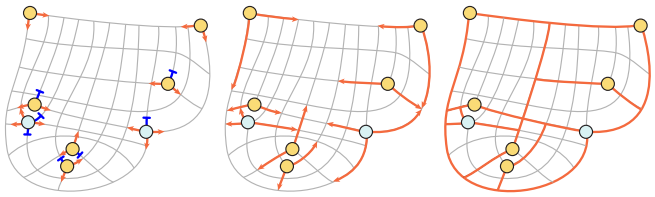


Fig. 3. A Motorcycle Graph is calculated by spawning motorcycles at edges of singularities (left) and tracing them (middle) until they all collide (right).

3 GENERALIZED MOTORCYCLE GRAPHS

For completeness, we first recap the original Motorcycle Graph (MCG) algorithm, first proposed in [Eppstein et al. 2008]. MCG is originally motivated by the task of providing a canonical partitioning of quad meshes with shared connectivity, toward the goal of finding an isomorphisms between them. However, we will be using it to construct a parametrization intended for texture mapping.

3.1 The Original Motorcycle Graph (MCG)

The idea of Motorcycle Graphs is to trace particles (called *motorcycles*) along the edges of a two-manifold, pure-quad mesh until they collide with another motorcycle or the trail thereof (see Figure 3). A motorcycle is spawned at each edge around each irregular vertex (i.e. non valence 4 internal vertex) going outward, and traced across edges, going straight in a topological sense (i.e., it always proceeds to the opposite edge). Motorcycles are traced in parallel, interleaving advancement steps of each motorcycle over an edge, while marking traversed edges and vertices. A motorcycle terminates as soon as it reaches a vertex already traversed by any motorcycle. When all motorcycles are terminated, the set of all traversed edges partitions the mesh into regions (or patches) which are guaranteed to be topologically rectangular and to be fully regular internally.

There are two kinds of collisions: *head-on* collisions (which are rare) between two motorcycles coming from (topologically) opposite directions; and *lateral* collisions, where one motorcycle hits the trail of a second motorcycle traveling in an (topologically) orthogonal direction. In head-on collisions, both motorcycles are terminated; in lateral collisions, only the first motorcycle is terminated, and a T-junction is formed in the final layout.

In a variation of this algorithm, also introduced in [Eppstein et al. 2008], fewer than v motorbikes are spawned around an irregular vertex of valence v, as long as at least one is spawned at each two consecutive edges around that vertex. Therefore, only two motorcycles may be emanated from a valence-3 vertex, and only three from a valence-5 or valence-6 vertex. In Figure 3, the motorcycles which are prevented by this variation are indicated by blue marks. This

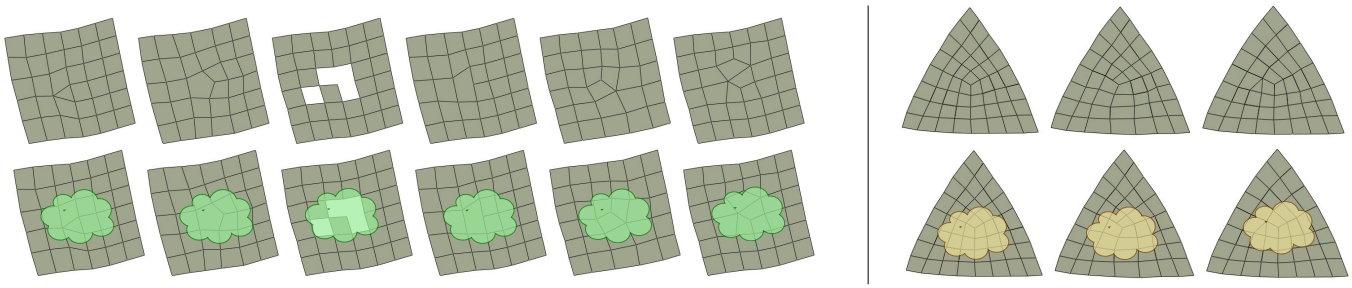


Fig. 4. The intuition behind our generalization. **Left, first row**: several examples of imperfect connectivities that are commonly encountered in semi-regular quad-dominant meshes: a triangular element, a pentagonal element, a hole (note also the non-two-manifold vertex in the middle), a T-junction, a pair of irregular vertices (valences 3 and 5), a configuration of four irregular vertices (with valences 3,3,5, and 5). **Left, second row**: concealing the imperfections visually reveals how none of the the imperfections affect the overall regularity of the mesh away from them. For the purpose of our algorithm, these areas will be considered regular to all effects. **Right, first row**: a similar situation arises when a irregular valence 3 vertex, which changes the flow of a surrounding areas (left), is accompanied by a few connectivity defects, such as a T-junction (middle) or a triangular element. **Right, second row**: concealing the zones shows how the three configurations have indistinguishable effects on the edge directions away from them. For the purposes of our algorithm, the three cases will be treated identically. The concealed parts represent the *fenced regions*.

results in fewer patches being produced (i.e., in a coarser layout), while maintaining the guarantees on the rectangular shape and internal regularity of the patches. This variation is directly applicable also to our generalization of MCG, so we adopt it.

The algorithm is easily extensible to open meshes. We adopt this formulation, which is equivalent to the original one: a boundary vertex is considered irregular when its edge-valence is not equal to 3; we allow only motorbikes spawned on boundary edges to travel over boundary vertices; other motorbikes are terminated just before reaching any boundary vertex. This will cause the entire boundary of the mesh to be eventually traced by motorbikes.

Benefits. In spite of its simplicity, the original Motorcycle Graph algorithm has many desirable properties when applied to clean meshes: it is fully automatic and reliable; it produces coarse quad layouts, which are useful in many contexts, such as serving as a parametrization domain for low-distortion, artifact-free, efficiently packed texture mapping. In other words, an MCG is a straightforward way to exploit the high regularity of a pure quad mesh. It exemplifies the ease of parametrization of such meshes, which is among the main motivations making these kind of meshes sought after.

Limited Applicability. In spite of all its benefits, the concept of Motorcycle Graphs will not work well, or at all, when applied to the commonly encountered quad-dominant semi-structured meshes (see Section 1.1), even when they present few and sparse *imperfections*. There are several reasons for this.

First, a large number of irregular vertices results in an explosion of the number of patches.

Second, MCG only targets pure-quad meshes and breaks in presence of even a single pentagonal or triangular element. This problem can be addressed by one iteration of topological Catmull-Clark subdivision, which turns every polygonal mesh into a pure quad mesh. However, this introduces many additional irregular vertices, exacerbating the former problem, and increases the complexity of the model by an average factor of four.

Lastly, MCG does not allow for other imperfections either. MCG will treat small holes as legitimate mesh boundaries (rather than just incomplete data), causing additional irregular vertices, again exacerbating the first problem. Similarly, small handles/tunnels are considered as legitimate surface features (instead of meshing artifacts), resulting in a large number of irregular vertices. An open mesh with jagged (rather than straight) boundaries will also be treated as having a large number of irregular vertices. The presence of T-junctions or of non two-manifold vertices are not dealt with by MCG.

The effect of these problems propagates across the mesh. This means that the final quad layout, even if it can be constructed, will lack coarseness also in the clean parts.

3.2 Generalizing MCG: Main Intuition

The basic idea of MCG is to spawn motorcycles at irregular areas and propagate them across regular areas. This concept can be applied in spite of the above listed local defects of the quad mesh connectivity.

For the purpose of the algorithm, an area can be considered regular as soon as it is assimilable to a regular grid, that is, if it does not disrupt the regular 2D grid pattern away from it. This can be the case even if the area is not tessellated as a completely regular grid. See Figure 4 (left) for examples. No motorcycle needs be spawned in such areas, and other motorcycles will traverse this area *as if* it was regular.

The only potential effect outside an area of this kind is a change of grid density around it. Many applications, such as texture mapping, are fairly tolerant to the small variations of densities, which result in only moderate parametrization distortions. The combined effect of multiple such cases can either cancel out or accumulate beyond the final application tolerance. We deal with the latter case in a subsequent phase by splitting the final rectangular regions (trading distortion for a marginal decrease of the coarseness of the layout).

Similarly, the locations which spawn off motorcycles are not identified by the immediate 1-star around a vertex but by regions that

have a large-scale effect on the edge orientations of their surroundings. See Figure 4 (right) for examples.

We turn this intuition into an algorithm by introducing the concept of *fenced regions*.

3.3 Fenced Regions

Metaphorically, we fence-in any problematic configuration breaking mesh regularity and cleanness, such as irregular vertices, holes, T-junctions, et cetera. A fenced region is defined as a contiguous region of a mesh (a collection of faces) that hosts one or multiple such configurations but is nonetheless to be treated either as entirely regular (*regular* fenced regions) or as containing a single irregular vertex (*irregular* fenced regions).

Classification of Fenced Regions. The *fence*, i.e. the boundary of a fenced region, is a collection of mesh edges and vertices. A fenced region is *valid* if all the vertices on the fence are regular; we will be only using valid fenced regions. A valid fenced region can be classified as regular or irregular solely according to its boundary, regardless of its interior, as follows.

Each vertex on a fence can be classified according to the number of its outward edges (edges that are *outside* the fenced region): convex vertices have two, straight vertices have one, and concave have none. We then define the *valence* v of the fenced region as the sum of turns over all boundary vertices, i.e. $+1$ for convex vertices, ± 0 for straight vertices, and -1 for concave vertices. Equivalently, the valence of a fenced region is given by

$$v = n_e - n_v,$$

where n_e is the total count of outward edges and n_v is the number of vertices on the fence, but not on the mesh boundary. Analogously to mesh vertices, a fenced region is *regular* if $v = 4$, and *irregular* otherwise.

We also define degenerate fenced regions, having no faces, zero area, and consisting of a single irregular vertex. A degenerate fenced region is always valid, and its valence is defined as the one of that vertex.

3.4 Generalizing MCG: Overall Algorithm

Our generalized algorithm produces a patch layout as follows.

First, we determine all the fenced regions (Sec. 3.6). Then, we perform the analogue of the standard MCG algorithm:

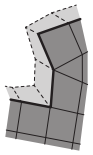
(1) spawn motorcycles at irregular fenced regions (Sec. 3.7);
(2) trace the motorcycles in parallel across the mesh (including across regular and irregular fenced regions), until each is terminated by a collision (Sec. 3.8);
(3) extract patches of the resulting graph (Sec. 3.9).

Finally, we post-process patches by splitting a few in order to alleviate excessive distortion. To ensure global consistency (i.e. to produce a pure rectangular layout) we need to enforce topological consistency conditions within steps 1 and 2 as described below. One merit of our approach is that these sub-problems are *local* and the size of their instances is limited, allowing for easy solutions.

Before the algorithm is run, we pre-process the input mesh (Section 3.5). This simplifies the formulation (and implementation) by reducing the number of cases which must be accounted for.

3.5 Preprocessing

Virtual refinement. We consider any T-junction as one extra corner of a polygon (e.g. a quad with a T-junction as a pentagon), and we perform one global iteration of topological Catmull-Clark subdivision. This subdivision is only temporary: edges introduced by the subdivision are tagged as "CC", and dissolved after the T-layout has been extracted, except the few ones that have been traversed by motorcycles. All subsequent phases of the algorithm strive to avoid routing motorcycles across CC edges, therefore the subdivision is almost completely reverted in practical cases; (in our experiments, the final mesh has an increased edge count by less than 1% in average). While not strictly necessary, this step offers practical benefits: it turns non-quadrilateral polygons and non conforming vertices into irregular vertices, reducing the number of cases which need to be dealt with, thus simplifying the implementation; it increases the number of regular vertices so that more valid fenced regions can be identified; and it provides more edges for the navigation of motorcycles (although they are used only as a last resort). Unless the input mesh is already conforming (T-junction free) and pure-quad, we always performed this step in our examples.



Virtual boundary expansion. According to our definition, an irregular boundary vertex cannot be part of any valid fenced region (except degenerate ones) because only regular vertices are allowed on the fences. Instead of modifying the definition to include boundary cases, we *virtually* pad the boundary with one layer of faces (dotted lines in the inset) by adding an outward edge to each vertex on the boundary (thick line). The new boundary is completely regular and original boundary irregular vertices are pushed into the interior. This padding is kept entirely implicit and the mesh is not actually modified. See Fig. 5 for an example of the effect of this on the overall algorithm.

3.6 Identification of Fenced Regions

In this step, we identify fenced regions that encapsulate all irregular points and defects while ensuring that each area is smaller than a maximal size T_{max}. T_{max} is the only parameter of our method and has an intuitive interpretation: it represents the area size of the largest feature which is to be ignored by the layout; higher values trade layout coarseness for parametric distortions. We always used $T_{max} = 20$ times the average face area.

Within these requirements, we would ideally like the number of irregular fenced regions to be minimized as this results in coarser layouts (see Figure 4). We design a heuristic to seek this objective, which we describe below. Crucially, this always produces a valid solution (at worst, the initialization). Figure 7 shows an example of the output.

Initialization. Initially, we encapsulate all irregular vertices in a minimal set of non-degenerate fenced regions (and, if necessary, a few degenerate ones). To do so, we initialize a fenced region for each irregular vertex v_i from its one-ring. Figure 6 (a) shows this starting point for a single irregular vertex. If another irregular vertex v_j lies on its boundary, then the fenced region is invalid. In this case, we expand the region by including the one-ring of v_j. This is repeated

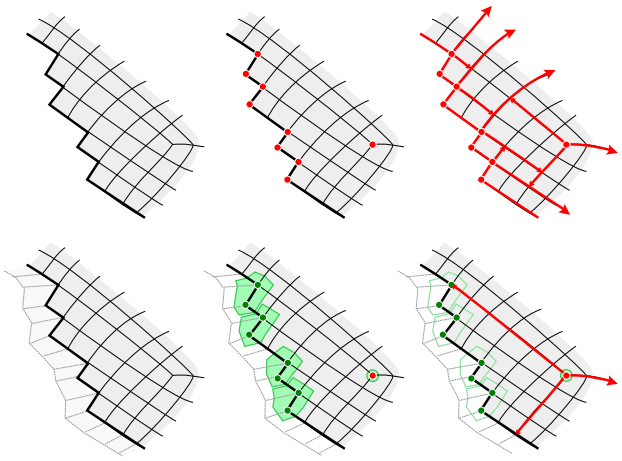


Fig. 5. Above: with the original MCG algorithm, many small patches are in presence of a jugged boundary, as in this example. Below: an example of the application of the Generalized MCG for the same input mesh. The boundary is first (virtually) expanded (left); five fenced regions (four regular, plus one degenerate of valence 3) are identified (middle), and a single patch with an open boundary is created covering the boundary (right). CC edges are not shown.

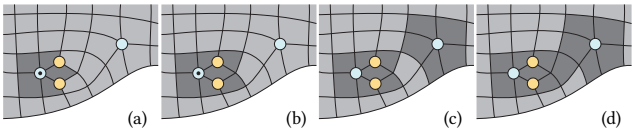


Fig. 6. Four steps in the iterative fenced region identification process. (a) an invalid fenced region is seeded from the one-ring of the marked irregular vertex; (b) the fenced region is expanded until it becomes valid; (c-d) two valid irregular fenced regions are merged into a single regular one.

until either the fenced region is valid or its area exceeds T_{max}. In the latter case, we dissolve the entire region and create a degenerate fenced region around each irregular vertex inside it. The result of this iterative expansion is shown in Figure 6 (b).

Merging by expansion. Next, we try to merge irregular fenced regions into fewer, regular ones. This procedure consists in progressively expanding irregular fenced regions in parallel and merging the ones which come into contact with each other (if possible). More specifically, we perform a sequence of atomic growing operations, each consisting in the expansion of one irregular, (non-degenerate) fenced region over one neighboring face, such that the new area does not exceed T_{max}. At every iteration, we pick the available operation where the face is geometrically closest to the starting point of fenced region. If the selected face already belongs to a different fenced region, then, instead of expanding the area, we test if the two areas can be merged. The merge is only performed if the summed area does not exceed T_{max}. Note that the merged region can then be regular and, if so, it is never expanded again. Faces surrounding degenerate fenced regions are never considered for expansion. Once there are no available operations left, this phase is over. Figure 6 (d) shows the result of merging the two fenced regions from Figure 6 (c).

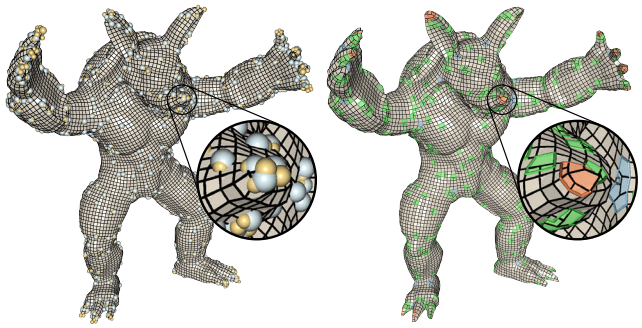


Fig. 7. One example of the results of fenced region identification phase. All singularities (left) are covered by fenced regions. Green overlays correspond to regular fenced regions, orange ones to fenced regions with valence < 4, and blue ones to fenced regions with valence > 4 (CC edges are not shown).

Shrinking back. Finally, we undo all the expansions that did not result into merging of fenced regions. Each fenced region is shrunk by iteratively testing and removing faces adjacent to the boundary. A removal is rejected if it causes loss of validity (that is, if an irregular vertex lies on the new border) or if it changes the disk-topology of the region.

3.7 Spawning Motorcycles

For every irregular fenced region of valence v, we spawn v motorcycles and immediately trace each of them to an exit position on the boundary of the containing fenced region (in reality, a subset of motorcycles are omitted, so as to coarsen the resulting layout, see Section 3; for exposition purposes, we ignore this detail here).

For a degenerate fenced region consisting of a single irregular vertex, this is done trivially: a motorcycle is spawned along each of the v edges stemming out of that vertex.

For non-degenerate fenced regions, we need to pick one common internal *starting position*, exit positions for each motorcycle, and routes from the former to the latter.

To do so in a valid way, we first partition the fence edges according to their topological *orientation*, which is an index from 0 to $v-1$. We pick an arbitrary edge and assign its orientation to 0. Then we navigate counter-clockwise around the fence, accumulating the turns (modulo v), and assigning to each traversed edges the accumulated orientation (see Figure 8 for an example). This algorithm is well-defined because vertices on a fence are always regular.

Consistency Conditions. One consistency requirement is that each motorcycle leaves the fenced region traversing an edge with a different orientation. This condition is not sufficient (cf. Figure 8-left): an additional requirement is that the motorcycles are spawned around the starting position in the same counter-clockwise order as their exit orientation (see Figure 8-right).

Searching for a solution. We now need to pick the starting and exit position in accordance to the above constraints. Because the fenced region is small, we can quickly consider all the possibilities, as follows. We trace paths backward, from every potential exit position towards the inside (when an irregular internal vertex is reached, the

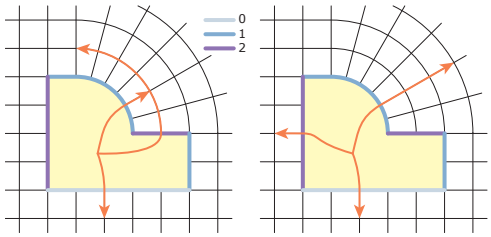


Fig. 8. A valence 3 fenced region with two alternative initializations for motorcycles. Fence edges are colored according to their topological orientation (from 0 to 2). Left: breaking the ordering requirement results in an invalid, non quadrilateral layout. Right: an initialization respecting it.

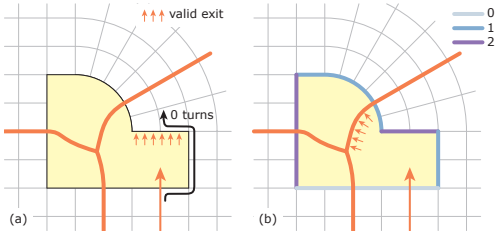


Fig. 9. Determination of valid exit points for a motorcycle entering a fenced region according to our consistency conditions. (a) The motorcycle can leave the fenced region if the path between entry and exit along the fence has a total of zero turns. (b) The motorcycle entering through an edge with orientation 0 can collide with a motorcycle that entered through an edge with orientation 1.

path tracing direction is not determined and all alternatives are explored). In this way, we quickly identify internal vertices capable of reaching a valid set of exit positions. Among the available solutions, we choose the one maximizing an ad hoc *geometric fitness*, defined as a measure combining straightness of paths; see the supplemental material for details.

Fallback strategy. In rare cases, no valid initialization exists (e.g. this happens with fenced regions with extreme valences lacking an internal vertex with sufficiently high valence or for fenced regions with many missing internal elements). If this happens, we simply dissolve the fenced region and create a degenerate fenced region for each contained singularity (which are valid unconditionally).

3.8 Tracing Motorcycles

A motorcycle travels straight until a collision occurs. A collision is detected when a motorcycle reaches a vertex that has been traversed by any motorcycle before. Outside fenced regions, traversed vertices are always regular, and motorcycles are routed normally (as in original MCG).

Inside fenced regions, the path can visit irregular vertices, making routing non-unique. Our solution is to consider all potential paths from the entry point, and pick an *admissible* one. As soon as a motorcycle steps inside a fenced region, it is routed until it either traverses across the fenced region and exits it again, or it collides somewhere inside it. Consistency conditions are defined differently for these two cases, as follows.

Consistency conditions for traversals. Conceptually, in this case the path must travel topologically straight across the patch. If the motorcycle enters the fenced region at vertex v_a and exits it at vertex v_b, then the route starting at v_a, traveling along the fence, and finally leaving at v_b, must have a total *turn count* of 0 (see Fig. 9 (a)). For regular fenced regions, both possible routes (either clockwise or counter-clockwise) leading to valid exit positions will have the same turn count. For irregular fenced regions, we pick the route along the fence that does not intersect with any of the exit positions of the initial motorcycles (see Section 3.7). This condition is well defined because vertices on the fences are always regular by construction.

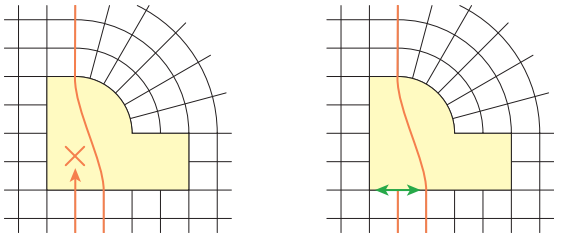
Conditions for collisions. Inside any fenced region, collisions must fulfill the following consistency requirement: a motorcycle entering through an edge with orientation i can only collide with a motorcycle that entered through an edge with orientation $i + 1$ or $i - 1$ (modulo the valence v of the fenced region, see Fig. 9 (b)). Note that collisions of this kind are always considered *lateral* (terminating only the motorcycle being moved). *Head-on* collisions can never happen in the interior of fenced regions.

Enumerating and selecting potential paths. Inside fenced regions, a path is followed straight (in the topological sense) when passing through regular vertices, and forks among all possibilities otherwise. A path reaching a boundary edge inside a fenced region proceeds over all possible vertices along the boundary of that hole. We use Dijkstra's algorithm to trace a path from the entering position to any admissible end (collision or exit). The per-edge cost function penalizes traversal of CC edges, which are usually entirely avoided. Secondarily, it favors a geometric measure of path straightness.

Fallback Strategies. In rare cases, no consistent path can be found. When this happens, we attempt a number of fallback strategies in cascade. The first is to cancel the offending motorcycle completely, remaking the arbitrary choice of which motorcycle to spawn around its spawning vertex. This cannot be done if it would cause the respawning of a motorcycle that was already canceled before. A second strategy consists in spawning two new stopping motorcycles at the entry point, going in the two directions orthogonal to the original motorcycle such that the latter immediately collides and never enters the fenced region (see Figure 10). It is necessary, however, that the two stopping motorcycles have a valid route until their eventual termination. 10 If non of the above strategies are possible, we dissolve the entire fenced region as a last infallible strategy. Doing so, we substitute it with one degenerate fenced region for each irregular vertex and restart construction of the graph. This is necessary only very rarely for highly irregular meshes.

3.9 Patch Extraction from the Motorcyle Graph

To extract the quadrilateral patches of the Motorcycle Graph, we trace the contour of each quadrilateral patch traversing all half-edges bounding each patch. Then we partition the mesh-faces into patches by a simultaneous flood-fill of all mesh faces, seeded at the bounding half-edges.



No valid route. Stopping by a pair of new motorcycles (green)

Fig. 10. One fallback strategy for a motorcycle that cannot be routed across a fenced region: We prevent its entry by spawning orthogonal stopping motorcycles.

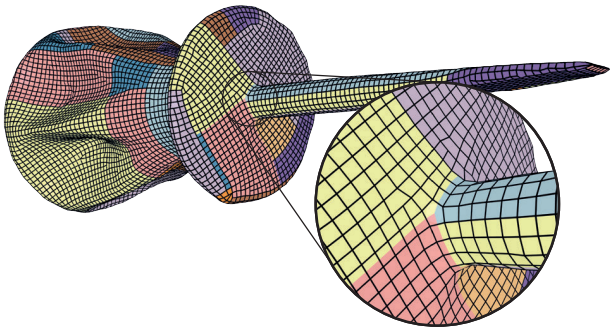


Fig. 11. After the T-layout is extracted, the Catmull-Clark subdivision performed as a prepossessing (see Sec. 3.5) is almost completely reverted by redissolving the introduced CC edges (not shown in this wireframe). CC edges which are part of patch boundaries (see zoom-in) cannot be removed but are rare by design.

This algorithm also deals correctly with rare cases in which a small bridge or tunnel connects two distinct quadrilateral patches (see inset). This can happen when a motorcycle traversed a fenced region featuring internal topological noise, which was purposely considered equivalent to a regular region. In our experimentation, this occurred with one dataset (the David head), out of several hundreds we tested.

3.10 Subdivision Reversal

After extracting the patches, we can revert the initial Catmull-Clark subdivision in all places by simply dissolving CC edges that are not traversed by any motorcycle. Because our algorithm avoids traversal of CC whenever possible, most CC edges can be dissolved.

Figure 11 shows the result for an example mesh, where almost the entire subdivision can be reverted.

4 CHOOSING PARAMETRIC SIZES

Once the layout is constructed, the next task is to determine the sizes of each rectangle in parametric space.

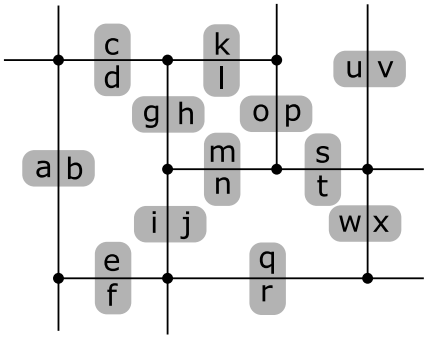


Fig. 12. An example of a T-layout. Letters from a to w: variables assigned to half-arc lengths.

In the original MCG for clean pure-quad meshes, this step simply consists in counting the number of edges on the boundary of the rectangular regions (relying on the assumptions that edges approximately share the same length). By construction, the opposite sides of each rectangle will amount to the same edge count.

In our setup, this is not the case as regular fenced regions can affect the tessellation density inside a the patch (e.g. via T-junctions or configurations of irregular vertices). We determine the parametric lengths as follows.

4.1 Problem Formulation

We consider a graph where nodes are vertices of the final layout, including T-junctions. Each side of each rectangle is made up of one or multiple consecutive *arcs* of this graph (see Fig. 12).

Similarly to [Campen and Zorin 2017], we formulate the problem by assigning one strictly positive *length* variable to each *half-arc* and solving for them by optimizing an objective function that measures isometry under consistency conditions that ensure rectangular patches in parameter domain as well as invisible cuts along patch boundaries. In our case, the variables are lengths expressed in number of texels and must thus be integer.

Arc Length Estimation. In a first step, we estimate the actual size of each side of the rectangular patches in 3D. This is simply done by summing the edge lengths across all edges. For a more accurate estimation, we trace additional internal paths parallel to the measured side inside the half-arc's patch (as described in Section 3.8) and average their estimated lengths. As a result, the target length of a half-arc is the average height or width of the according patch. When the lengths of the sides or the internal motorcycles are drastically different, we trigger a horizontal or vertical split of the patch in order to reduce distortions (see 4.5). A *target length* estimation is assigned to each arc by distributing the estimated length of each side proportionally to the extent of the arc.

Objective Function. Our objective function is simply a measure of isometry, i.e. the preservation of lengths along the arcs. We compute a target length for each half-arc (see above) and we enforce the actual parametric length to match it in the least square sense. Target lengths are multiplied by a global scaling factor given by $\sqrt{N_T/A}$, where A is the total area of the input mesh and N_T is the approximate

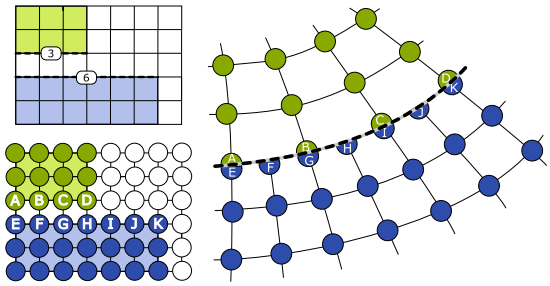


Fig. 13. A toy example illustrating an *invisible seam* construction in our framework. Top left: a rectangular domain 3×2 and one 6×2; the parametric lengths of the two matching sides (dotted lines) were made to match with an integer factor of $\times2$. Bottom left: the two domains are sampled by texels over a regular grid of 4 and 7×3 respectively (sizes are increased by one, to account for the borders), and then packed side to side in one texture, with no additional space necessary in between. Right: the distribution of texels in 3D space is consistent with bilinear interpolation, preventing any visual discontinuity artifact to appear along the cut. In this example, values of texels (A,E), (B,G), (C,I) and (D,K) are made to match, while values of texels F, H, J are set as the average of (E,G), (G,I), (I,J) respectively; in total, there are four *distinct*, fully unconstrained texel values along the border, on a total of 11 stored texels. No other texel is required in order to avoid bleeding or discontinuity artifact. See Figure 20 for an example of an actual rendering.

requested number of texels (e.g. $1.6\cdot10^7$ for a full resolution $4k\times4k$ texture).

4.2 Consistency Constraints

For Rectangles. As the patches should be rectangles in parametric space, the sum of assigned half-arc lengths on the opposite sides of each rectangle must be equal. For example, for the top rectangle in Figure 12, we impose $b = g + i$ and $d = e$.

For Arcs. If the lengths of all pairs of matching half-arcs (for example, $a = b$, $c = d$ etc. in Figure 12) are equal, the parametrization is *seamless* in the classical sense of [Bommes et al. 2009]. With the replication of a few texel values, interpolation artifacts can be avoided completely, making cuts *invisible* (see Figure 20). Unfortunately, as observed in [Campen and Zorin 2017], enforcing equal lengths of matching half-arcs together with rectangle constraints can result in a global system which admits only very few and heavily distorted solutions or even none at all (for strictly positive variables). In our scenario, we can relax these constrains in two different ways as described below.

4.3 Relaxing the System

Invisible Cuts. We borrow from [Ray et al. 2010] the observation that if a transition function at a cut includes an integer scaling, then the cut is still invisible (see Fig. 13). In practice, we found that we can limit the integer multiplier to $\times1$ and $\times2$. In other words, we allow an occasional $\times2$ resolution jump across a cut, so to relax the global system. Such jumps are necessary only for a tiny minority of the cuts.

Visible Cuts. Allowing for integer jumps already increases the degrees of freedom drastically. We noticed that disregarding only a few arc consistency constraints allows the system to reach a lower distortion everywhere (and thus a smaller energy). This trade-off is convenient, because interpolation artifacts at cuts are minor and local. In most scenarios, a few visible seams can be tolerated, whereas the gain in reduced distortion is global (in the industry standard, cuts are very rarely invisible, see Sec. 2.1).

4.4 Solving the System

We need to solve for half-arc lengths, integer multipliers, and disabled arc constraints. This makes for a mixed integer non-linear problem, which, however, can be solved using a simple heuristic.

Multipliers. The patch graph comprises two types of cuts: Those that are generated during patch splitting and those that are generated from the Generalized Motorcycle Graph. We observed that the latter kind is almost never required to have multipliers other than 1 because they usually trigger a patch split otherwise. Therefore, we set those multipliers to one and derive the remaining multipliers from the patch splitting phase (see next section).

Arc Lengths. Given the pre-determined multipliers, the aforementioned objective function becomes a linearly constrained quadratic function, which we solve with a commercial IQP solver [Gurobi Optimization 2016]. When the solver determines the model to be infeasible, we use its feasibility relaxation to remove the arc consistency constraints with minimal total length that make the system feasible. Similarly, if a solution produces an unfavorable ratio of half-arc length and its target length (we use a threshold of 2), we remove all consistency constraints for the arcs on the same side in the respective patch. This relaxes the system more, allowing solutions with less distortion.

4.5 Patch Splitting

Whenever the ratio between the internal paths that are used for patch size determination are too extreme, we split the corresponding patch. For this, we consider each of these internal paths as cut candidates and choose the subset that results in the least distortion as actual cuts. During this procedure, we maintain integer multipliers on the cuts to make it invisible. We optimize for the distortion-minimizing cuts with an incremental rounding approach of the respective multipliers. For details, refer to the supplemental material.

5 PARAMETRIZATION AND PACKING

In the final step, we compute the mapping from each patch on the mesh to the corresponding axis-aligned quadrilateral domain in texture space. This is constructed in the standard form of a UV assignment to each mesh vertex (replicating vertices at patch boundaries). Any patch distortion minimization parametrization method could be used for this purpose. We adopt Scaffold Map [Jiang et al. 2017] (initialized from a harmonic map), because it provides a good balance between area and angle preservation, while dealing well with open boundaries by avoiding global overlaps. The latter property is needed for patches which include open edges. For patches

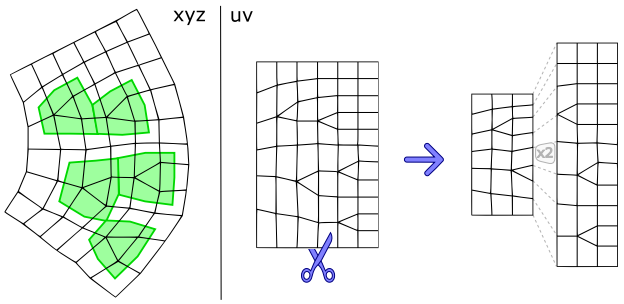


Fig. 14. Left: due to the combined effect of five regular fenced regions (green, left), a topologically rectangular patch ends up having fairly different geodesic length on east and west sides, resulting in a heavily distorted map (middle, note the anisotropic quad shapes). Right: by splitting the patch vertically, two less distorted patches can be obtained, connected by an *invisible* cut associated to a resolution multiplier of 2 (cf. Figure 13).

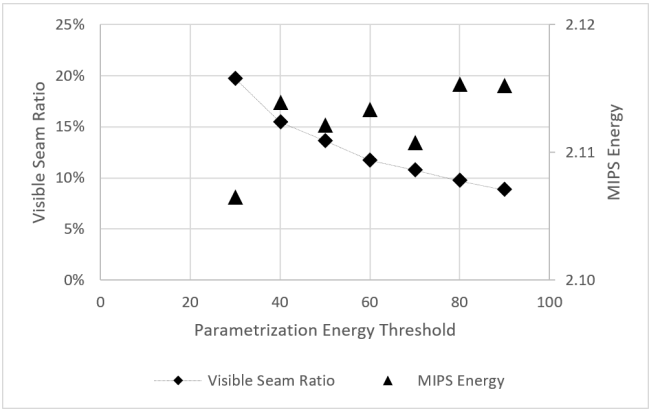


Fig. 15. Dependency of the amount of visible seams (w.r.t. the total amount of seams) and the distortion of the parametrization (measured by the average MIPS energy per triangle; the minimum MIPS energy is 2.0) on the user-definable parametrization energy threshold. The presented data are medians over all models from the [Jakob et al. 2015] data set (see Table 1). Increasing the threshold reduces the amount of visible seams and introduces slightly more distortion. The latter series exhibits some outliers due to the discrete nature of the underlying optimization problem.

that cannot be embedded without overlap (e.g. due to small handles or non-manifold configurations), we employ a least-squares conformal map [Lévy et al. 2002].

Boundary conditions are set as follows: (1) vertices in the interior of patches are unconstrained, and (2) vertices mapped on a side of a rectangle are constrained to never leave that side, but are allowed to slide along it. This is trivially achieved by constraining either their u or v coordinate to a constant value (the four corners, belonging to two sides, always have both coordinates fixed).

Over all sides that are *invisible* cuts (see 4.2), the two copies are constrained to slide in sync with respect of each other by enforcing the two vertices to have the same *barycentric* position (in [0,1]) over the segment (a linear constraint in the texture coordinates). The same constraint is applied to T-nodes, effectively fixing both coordinates of the respective vertex. At this point, the system could be solved globally. However, this requires the optimization of two energy types at the same time (scaffold map and LSCM). We experimentally observed that the global solution can be approximated well by a simpler, local approximation. We fix both coordinates of boundary vertices, decoupling the optimization between patches and solving smaller systems. To overcome the inevitable distortion introduced by fixing the positions on the boundary, we evaluate the parametrization's energy and make the longest patch side a visible cut if it exceeds a user-definable threshold. Figure 15 shows how increasing this threshold leads to less visible seams with marginally higher parametrization distortion.

Finally, we pack all rectangular patches into a unified texture. For each patch of size $m \times n$, we allocate $(m + 1 \times n + 1)$ texels (because texels are sampled at the boundary of the rectangles – cf. Fig 13). Therefore, no empty texels need to be left unused between the patches. For packing, we employ the maximal rectangles algorithm [Jylänki 2010] (patches with open boundary are represented by their axis aligned bounding rectangle). For the typical sizes of the problem, this is very efficient, producing extremely tight packings (< 5% wasted space) within tenths of milliseconds. This is one of the benefits of a rectangular based domain.

To allow MIP-mapping up to a user-specified MIP-map level l, we perform all previous steps for the given level. I.e., the arcs' target lengths are scaled by an additional factor of 2^{-l}. The final parametrization is found by scaling texture coordinates back to the finest level using a factor of 2^l. Note that this will introduce small gaps between patches on finer levels. We share this inherent cost with any other standard texture-mapping technique. In our setup, the issue of MIP-mapping is alleviated, simplified, but not bypassed, by the axis-alignment of cuts.

The resulting UV-mapped mesh can be saved and used in any standard downstream application.

6 RESULTS

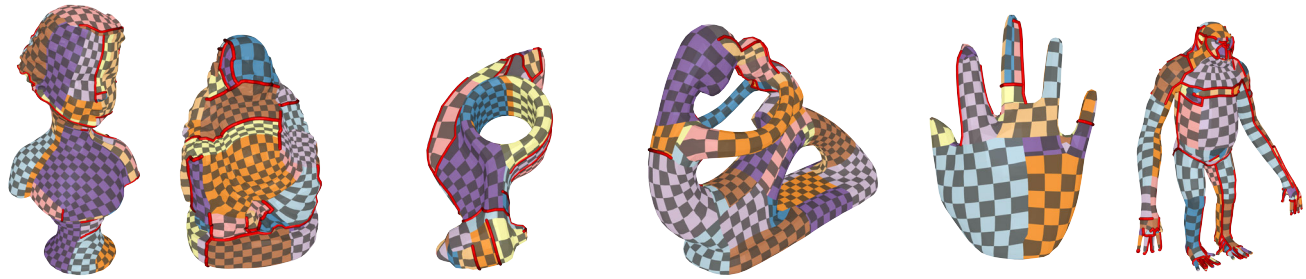
An important feature of our algorithm is that it always produces a valid T-layout while producing a good (coarse) layout when the input regularity can be exploited.

We verified this by successfully testing on models from six different sources (see Table 1) without encountering a single failure case out of more than one hundred cases. We produced the first dataset by feeding the triangular meshes used by [Myles et al. 2014] into the Instant Meshes algorithm [Jakob et al. 2015]. The resulting quad-dominant meshes are characterized by a large number of irregular vertices and occasionally holes. Other four datasets are provided by the authors of the respective papers. The "handmade" dataset consists of semi-regular quad-dominant meshes hand-modeled by professional artists, downloaded from the repository [TurboSquid 2018]. Each dataset presents a different degree of regularity and different types of problems. Figure 16 shows one sample from each of datasets and more are shown in Figure 22.

We compared the performance of our method with existing solutions on a challenging case (Figure 17). Compared to the UV layout produced by a commercial software (Autodesk Maya), the Generalized Motorcycle Graph layout is more GPU memory efficient, because of the superior packing efficiency, and secondarily because

Data Set	Models	$\lvert S\rvert/\lvert V\rvert$	s. red.	vis. s.	$R_{v\,10}$	$R_{v\,90}$	M_v	$R_{i\,10}$	$R_{i\,90}$	M_i
[Jakob et al. 2015]	115	1.8%	95.2%	12.0%	0.69	1.37	2.00	0.65	1.47	2.12
[Ebke et al. 2016]	4	7.1%	70.8%	21.8%	0.38	1.96	2.04	0.42	1.98	2.73
[Marinov and Kobbelt 2006]	4	7.5%	83.1%	31.4%	0.39	1.62	2.09	0.42	1.54	2.33
[Bommes et al. 2009]	7	1.4%	42.9%	7.1%	0.58	1.47	2.06	0.49	1.48	2.22
[Ray et al. 2006]	4	5.8%	61.7%	18.0%	0.68	1.65	2.04	0.64	1.84	2.16
Hand-made	10	2.7%	61.1%	19.6%	0.46	2.11	2.11	0.45	2.24	3.10

Table 1. Result statistics for models of different types. We report the number of models in the dataset, the number of singularities $|S|$ w.r.t. the number of mesh vertices $|V|$, and the relative singularity reduction due to fenced regions. Parametrization statistics are reported by the percentage of visible cuts (w.r.t. the length of all cuts), area ratios R of the parametrization with respect to the model surface as a distortion measure , and the average MIPS energy M. We report the distortion measures for visible seam parametrization (R_v, M_v) and invisible seam parametrization (R_i, M_i). We present the area ratios as the 10-th and 90-th percentile as robust substitutions for minimum and maximum. All values are medians over all models in the data set. Inivisible seam parametrization is performed with a medium parametrization energy threshold (50, cf. Fig. 15)



[Jakob et al. 2015] [Ebke et al. 2016] [Marinov and Kobbelt 2006] [Bommes et al. 2009] [Ray et al. 2006] Hand-made

Fig. 16. Examples from the data sets presented in Table 1. Visible cuts are marked with red lines.

fewer domains with straighter boundaries require fewer texel replication. Compared to the plain motorcycle graph, our method produces a layout with about $\frac{1}{20}$ of the patches (Figure 2).

Our reference implementation and selected data sets are available at https://github.com/NSchertler/GeneralizedMotorcycleGraph.

6.1 Applications

3D Scanning. In Figure 18, we show how Generalized Motorcycle Graphs can close the last gap of the Online Surface Reconstruction scanning pipeline proposed by [Schertler et al. 2017], which uses a specialized surface format based on Mesh Colors [Yuksel et al. 2010] to store surface information. Consequently, the model cannot be used directly in standard applications. By rendering the colors into a rectangular texture, which has been laid out with a Generalized Motorcycle Graph, we can make the data available to all applications that support textured meshes.

Figure 18 also shows that Generalized Motorcycle Graphs are insensitive against small imperfections in the input data. The highlighted area on the mesh shows a non-manifold configuration in combination with holes. While other T-layout generation methods cannot handle these cases, Generalized Motorcycle Graphs robustly integrate these imperfections within a large texture patch. The non-manifold configuration prohibits the existence of an overlap-free embedding. Since a bijective parametrization does not exist, the parametrization produced by Generalized Motorcycle Graphs includes a small area of overlap in the problematic region and leaves the other parts of the texture unaffected.

Other kinds of surface information can be stored in textures laid out by Generalized Motorcycle Graphs as well. Figure 19 shows an application to light baking, where the ambient occlusion term calculated by a preprocess has been baked into a texture and applied to a low-resolution model.

Cuts Invisibility. An additional benefit is the ability to make most cuts *invisible* (see Section 4.2). Figure 20 shows a rendering example.

Texture Reduction. Thanks to its packing efficiency, Motorcycle Graphs can also be beneficial to reduce the size of textures used in video games or movies. After asset creation, textures can be automatically resampled into a tighter texture, saving GPU memory while reducing the occurrences of seam artifacts (Figure 21).

Remeshing. Our approach is tailored for texture mapping applications, but under certain circumstances it can be employed for remeshing as well: we can resample each produced domain with a regular grid (an example is shown in Figure 23). This, however, implies a resampling of the vertices, which can potentially introduce errors at sharp features. Furthermore, T-junctions are produced along any cut for which the parametric lengths do not match (Sec. 4.2); for scenarios where T-junctions can be tolerated (e.g. in hidden parts), this application can be seen as a convenient way to sweep out different sources of irregularity scattered across the mesh and concentrate them at the cuts in form of T-junctions. The resulting

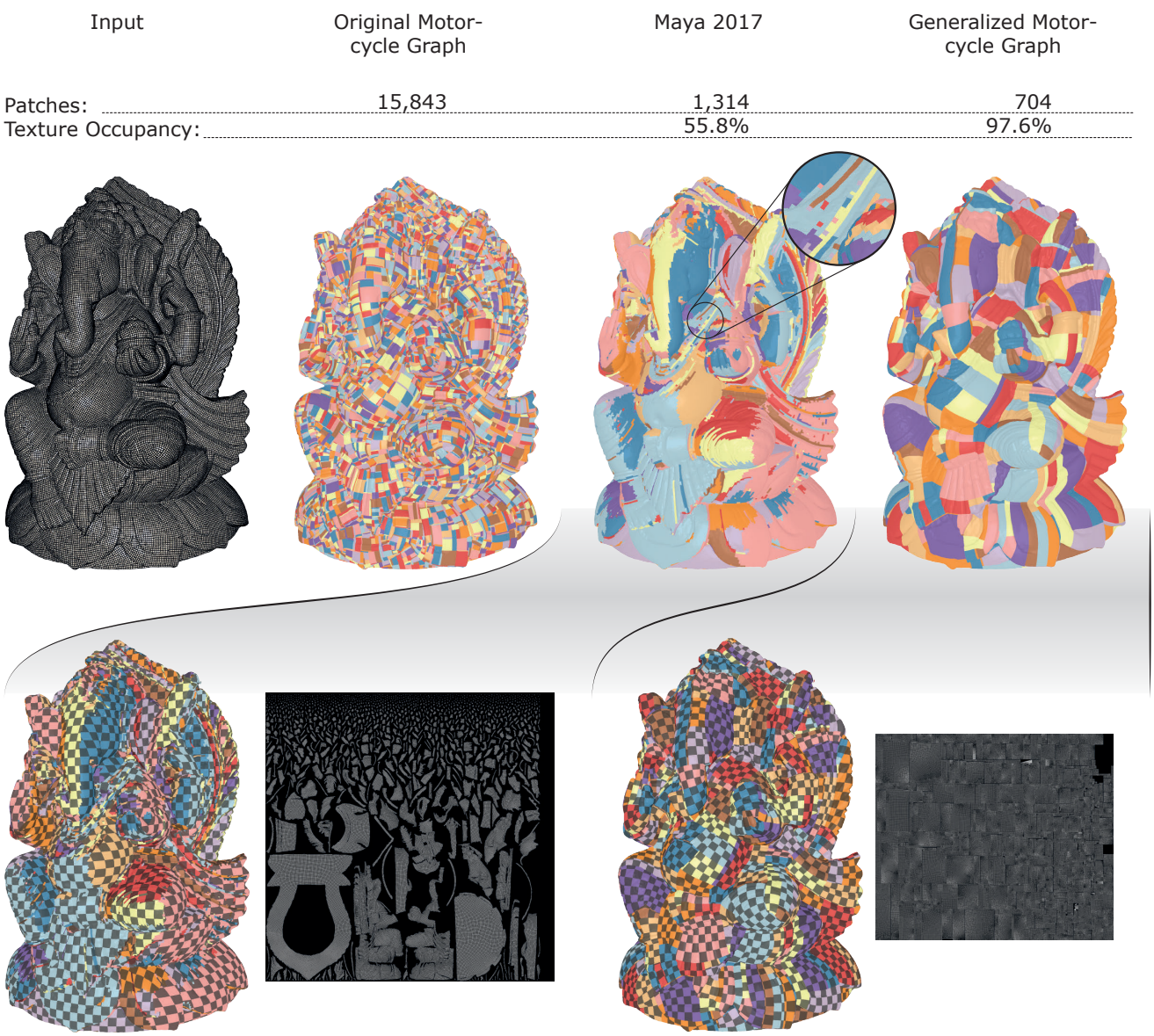


Fig. 17. Comparison of UV-atlases automatically generated for a complex model. From left to right: input quad-dominant mesh, with 138, 290 vertices and 10, 919 singularities; partition produced by the original Motorcycle Graph (after one step of Catmull-Clark); atlas parametrization produced by Autodesk Maya's automatic UV-mapping tool (optimized for the number of patches); and the result of our own approach. Below: the UV-layouts of the latter two models. Our method results in almost half the patches, and a drastically superior texture packing efficiency, with only 2.4% unused texels.

remeshings are typically much more regular, which greatly helps for example compression (see [Sander et al. 2003]).

7 LIMITATIONS AND CONCLUDING REMARKS

Our work allows automatic and high-quality UV mapping of a class of meshes in between fully structured and unstructured meshes, which, we argue, has high practical importance, both in manually modeled and automatically remeshed models. Fully automatic pipelines that produce semi-regular meshes from captured data can benefit from Generalized Motorcycle Graphs since it extends the automatic workflow to texturing. Manually modeled semi-regular meshes can also take advantage of the ability to automatically construct GPU memory-efficient, artifact free, high-quality UV maps.

Limitations. The main limitations of our method stem directly from our starting choices: we target T-layouts only; our parametrization is not globally seamless (in the sense of [Bommes et al. 2009]), nor globally conformal (in the sense of [Campen and Zorin

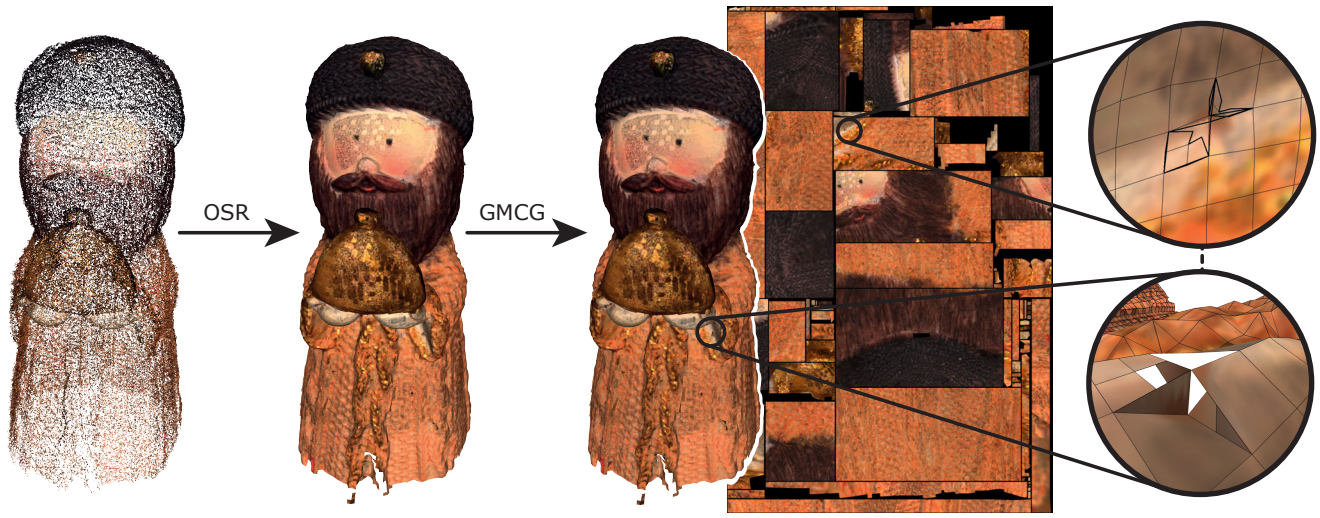


Fig. 18. Parametrization by Generalized Motorcycle Graphs close the gap between meshing approaches like Online Surface Reconstruction (OSR) and many applications that use standard textured meshes as input. OSR takes several point clouds as input (left) and produces a semi-regular mesh in a specialized format (middle). Natural Motorcycle Graphs can generate a standard textured mesh (right) from this format. To support this application, it is imperative that Natural Motorcycle Graphs handle imperfections in the input data (highlighted areas) and produce a practically usable texture map.

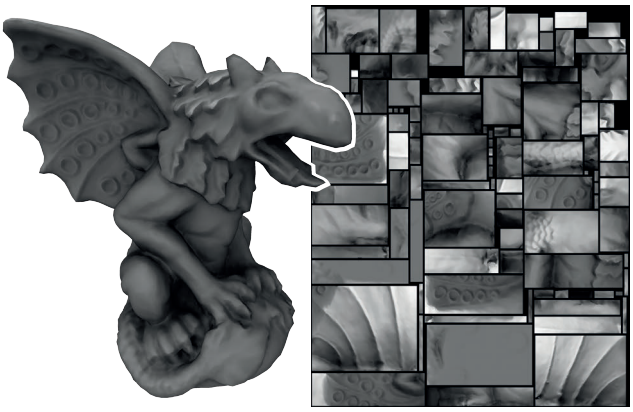


Fig. 19. Generalized Motorcycle Graphs can be used to store various surface attributes in a texture. In this example, the ambient occlusion term of a high-resolution surface was baked into a texture (right) for use with a low-resolution model (left).

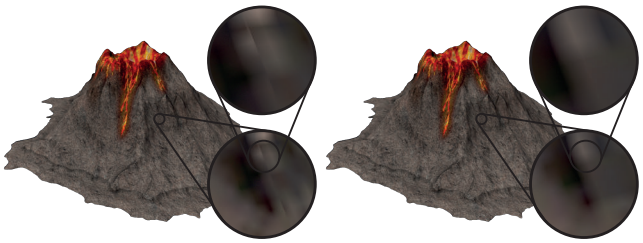


Fig. 20. Left: without per-arc consistency constraints, bilinear texture interpolation produces artifacts, which are visible under extreme magnification and reveal the presence of the texture cut. Right: enforcing per-arc consistency constraints (4.2), no artifacts appear regardless of the zoom factor.

Fig. 21. Generalized Motorcycle Graphs can be used to resample and repack textures of a hand-made model (left) to a denser texture (right). In this example, texture occupancy increased from 64.3% to 97.9%.

2017]); we need to allow for integer jumps. Also, outside the targeted class of meshes, our method is either superfluous (for very regular inputs, where the original MCG can be directly employed) or out-performed by standard global parametrization methods (for very irregular inputs); targeting full automatism, our method is not designed to be steerable by the user (although it could be expanded in this direction). These choices are, however, justified by our objectives.

Reliability. Reliability is one of the main motivations behind this work and our major strength: our approach will always produce a valid and usable UV mapping despite the imperfections of a given semi-structured mesh, allowing its usage in a graphics pipeline without a manual cleanup. This is achieved by enforcing (local) consistency constraints and employing infallible (local) fall-back strategies. The assumptions on the input, i.e. quad-dominance and

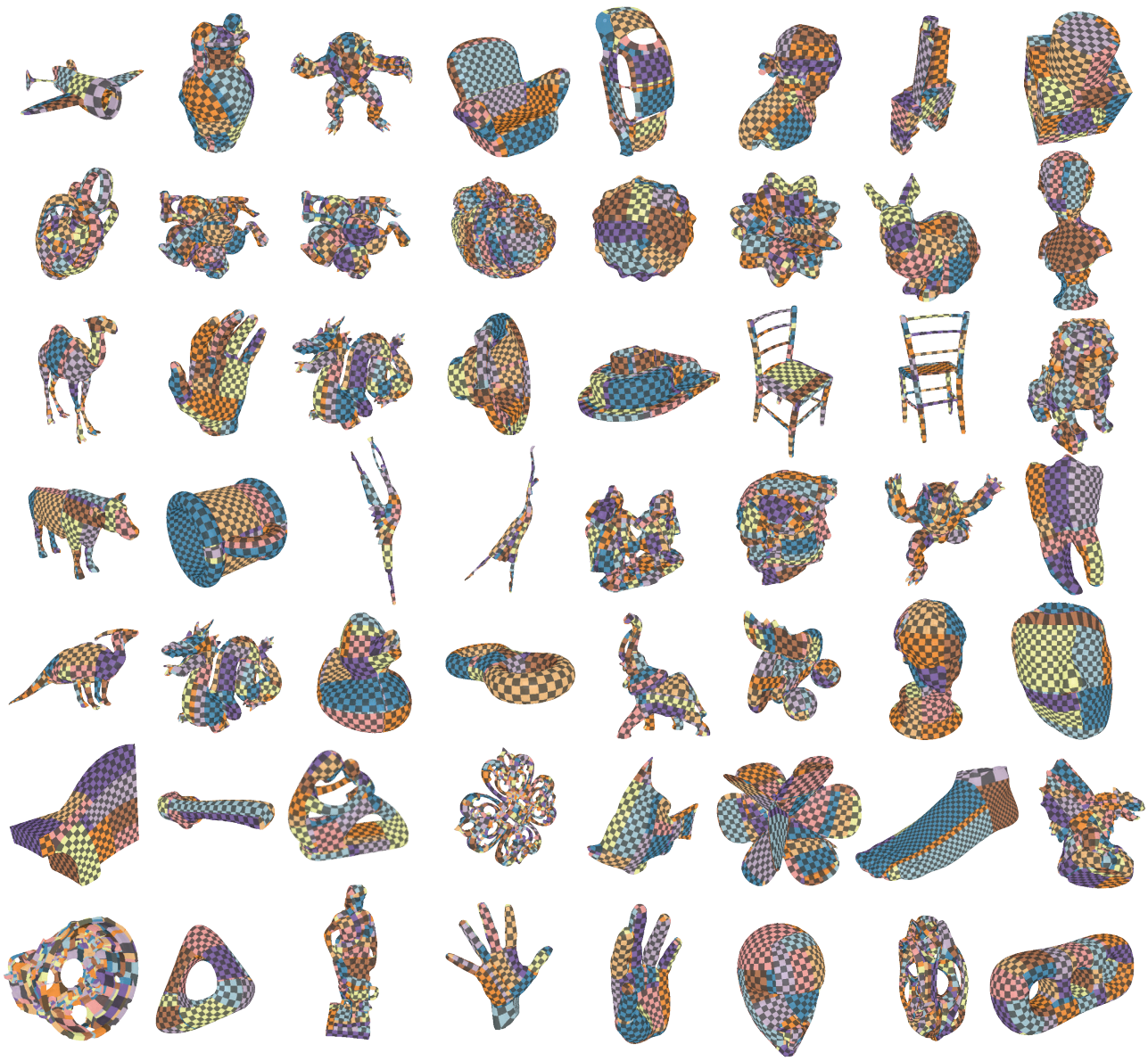


Fig. 22. The first 56 models produced by [Jakob et al. 2015] used to empirically validate our method.

preponderant regularity, are exploited whenever they are fulfilled; conversely, when they are broken, the output just gracefully downgrades its quality (in terms of layout coarseness and distortions). This observation also holds at both extremes of the regularity spectrum: for a completely clean, pure-quad, and regular mesh, our approach is equivalent to the original motorcycle graph (hence, it is a proper generalization of it); for a generic irregular triangle mesh the output will still be a valid (though very dense) quad partitioning of the mesh. At the worst, our algorithm will produce a PTex-styled parametrization [Burley and Lacewell 2008] over a Catmull-Clark subdivision of the input mesh. The strength of our method lies in-between these two extrema.

ACKNOWLEDGMENTS

We thank Xifeng Gao for his help in preparing our test data sets and Zhongshi Jiang for adapting his scaffold map code to our needs.

REFERENCES

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013. Integer-grid Maps for Reliable Quad Meshing. *ACM Trans. Graph.* 32, 4, Article 98 (July 2013), 12 pages. DOI: https://doi.org/10.1145/2461912.2462014

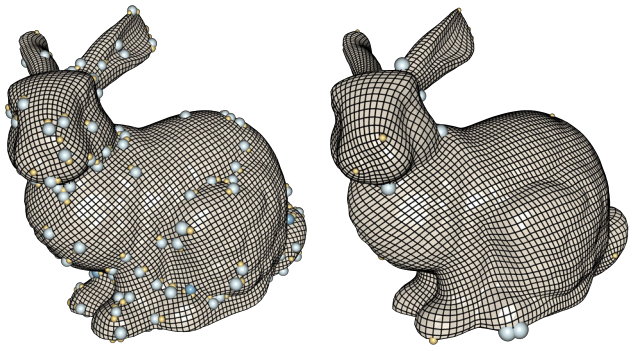


Fig. 23. A remeshing with a Generalized Motorcycle Graph. In this example, the number of singularities (depicted as spheres) is reduced from 351 (left) to just 32 in the output (right).

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30 (2011), 375–384. Issue 2. DOI: https://doi.org/10.1111/j.1467-8659.2011.01868.x

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 51–76. Issue 6.

David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Trans. Graph.* 28, 3 (2009), 77.

Brent Burley and Dylan Lacewell. 2008. Ptex: Per-Face Texture Mapping for Production Rendering. In *Eurographics Symposium on Rendering 2008*. 1155–1164.

Marcel Campen. 2017. Partitioning Surfaces into Quad Patches. In *EG 2017 - Tutorials*, Adrien Bousseau and Diego Gutierrez (Eds.). The Eurographics Association. DOI: https://doi.org/10.2312/egt.20171033

Marcel Campen, David Bommes, and Leif Kobbelt. 2012. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph.* 31, 4 (2012), 110.

Marcel Campen, David Bommes, and Leif Kobbelt. 2015. Quantized Global Parametrization. *ACM Trans. Graph.* 34, 6, Article 192 (Oct. 2015), 12 pages. DOI: https://doi.org/10.1145/2816795.2818140

M. Campen and L. Kobbelt. 2014. Quad Layout Embedding via Aligned Parameterization. *Computer Graphics Forum* 33, 8 (2014), 69–81. DOI: https://doi.org/10.1111/cgf.12401

Marcel Campen and Denis Zorin. 2017. Similarity Maps and Field-Guided T-Splines: a Perfect Couple. *ACM Trans. Graph* 36, 4 (2017).

Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. 2006. Rectangular multi-chart geometry images. In *Proc. of the 4th Eurographics symp. on Geom. proc.*

Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. 2008. Quadrilateral Mesh Simplification. In *ACM SIGGRAPH Asia 2008 Papers (SIGGRAPH Asia '08)*. Article 148, 9 pages. DOI: https://doi.org/10.1145/1457515.1409101

Jonathan D. Denning, William B. Kerr, and Fabio Pellacini. 2011. MeshFlow: Interactive Visualization of Mesh Construction Sequences. *ACM Trans. Graph.* 30, 4, Article 66 (July 2011), 8 pages. DOI: https://doi.org/10.1145/2010324.1964961

Hans-Christian Ebke, Patrick Schmidt, Marcel Campen, and Leif Kobbelt. 2016. Interactively Controlled Quad Remeshing of High Resolution 3D Models. *ACM Trans. Graph.* 35, 6, Article 218 (Nov. 2016), 13 pages. DOI: https://doi.org/10.1145/2980179.2982413

David Eppstein, Michael T Goodrich, Ethan Kim, and Rasmus Tamstorf. 2008. Motorcycle graphs: canonical quad mesh partitioning. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1477–1486. Issue 5.

Michael S. Floater and Kai Hormann. 2005. Surface Parameterization: a Tutorial and Survey. In *Advances in Multiresolution for Geometric Modelling*, Neil A. Dodgson, Michael S. Floater, and Malcolm A. Sabin (Eds.). Springer, 157–186.

Xiao-Ming Fu, Chong-Yang Bai, and Yang Liu. 2016. Efficient Volumetric PolyCube-Map Construction. *Computer Graphics Forum* 35, 7 (2016), 97–106.

Inc. Gurobi Optimization. 2016. Gurobi Optimizer Reference Manual. (2016). http://www.gurobi.com

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant field-aligned meshes. *ACM Trans. Graph.* 34, 6 (2015), 189.

Zhongshi Jiang, Scott Schaefer, and Daniele Panozzo. 2017. Simplicial complex augmentation framework for bijective maps. *ACM Trans. Graph.* 36, 6 (2017), 186.

Jukka Jylänki. 2010. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. *retrived from http://clb. demon. fi/files/RectangleBinPack. pdf* (2010).

Kestutis Karciauskas, Daniele Panozzo, and Jörg Peters. 2017. T-junctions in spline surfaces. *ACM Trans. Graph.* 36, 5 (2017).

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. *ACM Trans. Graph.* 21, 3 (July 2002), 362–371. DOI: https://doi.org/10.1145/566654.566590

Ruotian Ling, Jin Huang, Bert Jüttler, Feng Sun, Hujun Bao, and Wenping Wang. 2014. Spectral Quadrangulation with Feature Curve Alignment and Element Size Control. *ACM Trans. Graph.* 34, 1, Article 11 (Dec. 2014), 11 pages.

Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. 2017. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.* 36, 6, Article 216 (Nov. 2017), 15 pages.

Martin Marinov and Leif Kobbelt. 2006. A Robust Two-Step Procedure for Quad-Dominant Remeshing. In *Computer Graphics Forum*, Vol. 25. Wiley Online Library, 537–546. Issue 3.

Ashish Myles, Nico Pietroni, Denis Kovacs, and Denis Zorin. 2010. Feature-aligned T-meshes. *ACM Trans. Graph.* 29, 4, Article 117 (July 2010), 11 pages. DOI: https://doi.org/10.1145/1778765.1778854

Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust field-aligned global parametrization. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 135.

D. Panozzo, E. Puppo, M. Tarini, N. Pietroni, and P. Cignoni. 2011. Automatic Construction of Quad-Based Subdivision Surfaces Using Fitmaps. *IEEE Transactions on Visualization and Computer Graphics* 17, 10 (Oct 2011), 1510–1520.

Chi-Han Peng, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2011. Connectivity Editing for Quadrilateral Meshes. *ACM Trans. Graph.* 30, 6, Article 141 (Dec. 2011), 12 pages. DOI: https://doi.org/10.1145/2070781.2024175

Nico Pietroni, Enrico Puppo, Giorgio Marcias, Roberto Roberto, and Paolo Cignoni. 2016. Tracing Field-Coherent Quad Layouts. In *Comp. Graph. F.*, Vol. 35. Wiley Online Library, 485–496. Issue 7.

Roi Poranne, Marco Tarini, Sandro Huber, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Autocuts: simultaneous distortion and cut optimization for UV mapping. *ACM Trans. Graph.* 36, 6 (2017), 215.

Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. 2006. Periodic Global Parameterization. *ACM Trans. Graph.* 25 (Oct. 2006), 1460–1485. Issue 4. DOI: https://doi.org/10.1145/1183287.1183297

Nicolas Ray, Vincent Nivoliers, Sylvain Lefebvre, and Bruno Levy. 2010. Invisible Seams. *Computer Graphics Forum* 29 (2010). Issue 4. DOI: https://doi.org/10.1111/j.1467-8659.2010.01746.x

Nicolas Ray and Dmitry Sokolov. 2014. Robust Polylines Tracing for N-Symmetry Direction Field on Triangulated Surfaces. *ACM Trans. Graph.* 33, 3, Article 30 (June 2014), 11 pages. DOI: https://doi.org/10.1145/2602145

Faniry H. Razafindrazaka and Konrad Polthier. 2017. Optimal base complexes for quadrilateral meshes. *Computer Aided Geometric Design* 52-53 (2017), 63 – 74. DOI: https://doi.org/10.1016/j.cagd.2017.02.012 Proc. GMP.

Faniry H Razafindrazaka, Ulrich Reitebuch, and Konrad Polthier. 2015. Perfect matching quad layouts for manifold meshes. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 219–228. Issue 5.

P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. 2003. Multi-chart Geometry Images. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (SGP '03)*. Eurographics Association, 146–155.

Nico Schertler, Marco Tarini, Wenzel Jakob, Misha Kazhdan, Stefan Gumhold, and Daniele Panozzo. 2017. Field-aligned online surface reconstruction. *ACM Trans. Graph.* 36, 4 (2017), 77.

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (July 2015), 9 pages.

Marco Tarini. 2016. Volume-encoded UV-maps. *ACM Trans. Graph.* 35, 4, Article 107 (July 2016), 13 pages. DOI: https://doi.org/10.1145/2897824.2925898

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-Maps. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 853–860. DOI: https://doi.org/10.1145/1015706.1015810

Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. 2011. Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Trans. Graph.* 30, 6, Article 142 (Dec. 2011), 12 pages. DOI: https://doi.org/10.1145/2070781.2024176

Marco Tarini, Cem Yuksel, and Sylvain Lefebvre. 2017. Rethinking Texture Mapping. In *ACM SIGGRAPH 2017 Courses (SIGGRAPH '17)*. Article 11, 139 pages.

TurboSquid. 2018. 3D Models for professionals. (2018). https://www.turbosquid.com [Online; accessed 23-January-2018].

Francesco Usai, Marco Livesu, Enrico Puppo, Marco Tarini, and Riccardo Scateni. 2015. Extraction of the quad layout of a triangle mesh guided by its curve skeleton. *ACM Trans. Graph.* 35, 1 (2015), 6.

Chaman Singh Verma and Krishnan Suresh. 2015. A robust combinatorial approach to reduce singularities in quadrilateral meshes. *Procedia Engineering* 124 (2015), 252–264.

Chaman Singh Verma and Krishnan Suresh. 2016. αMST: A Robust Unified Algorithm for Quadrilateral Mesh Adaptation. *Procedia Engineering* 163 (2016), 238 – 250. 25th International Meshing Roundtable.

Cem Yuksel. 2017. Mesh Color Textures. In *High-Performance Graphics (HPG 2017)*. 11.

Cem Yuksel, John Keyser, and Donald H House. 2010. Mesh colors. *ACM Trans. Graph.* 29, 2 (2010), 15.