Reliable Feature-Line Driven Quad-Remeshing

NICO PIETRONI, University of Technology Sydney, Australia
STEFANO NUVOLI, University of Cagliari, Italy
THOMAS ALDERIGHI, University of Pisa and ISTI-CNR, Italy
PAOLO CIGNONI, ISTI-CNR, Italy
MARCO TARINI, University of Milan, Italy

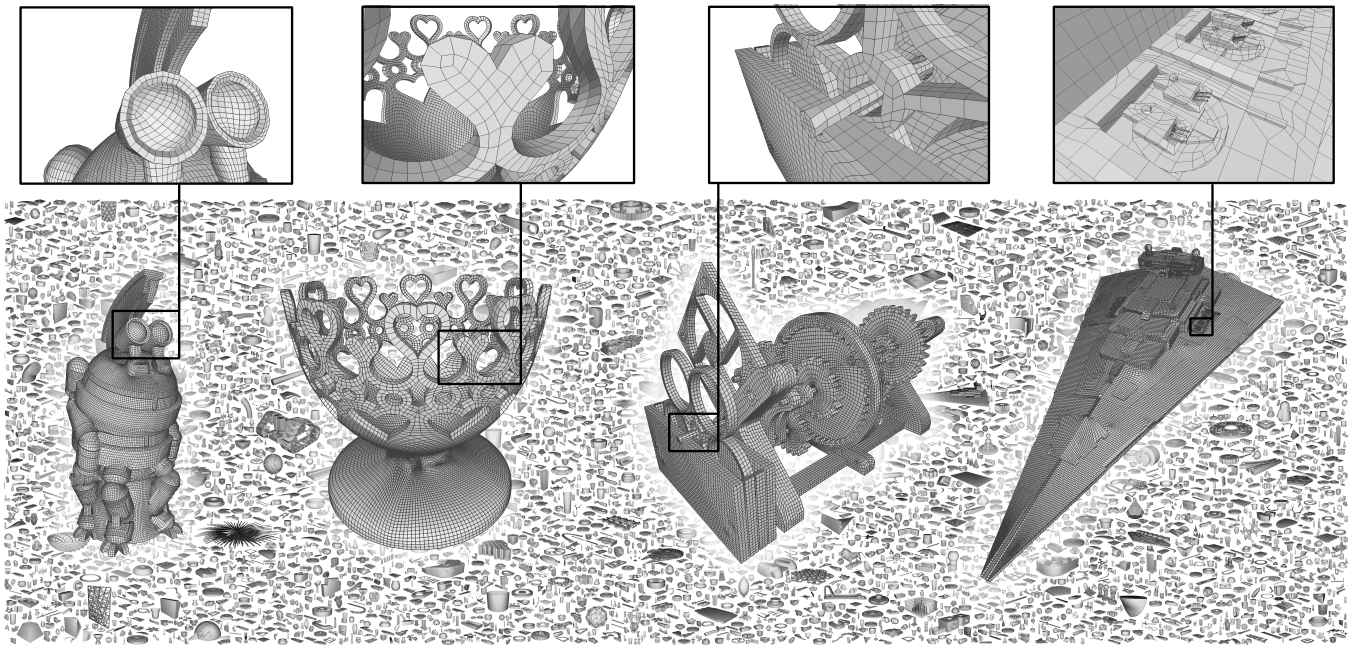


Fig. 1. A mosaic of the processed Thingi10K dataset. **The entire dataset is available at https://www.quadmesh.cloud**

We present a new algorithm for the semi-regular quadrangulation of an input surface, driven by its line features, such as sharp creases. We define a perfectly feature-aligned cross-field and a coarse layout of polygonal-shaped patches where we strictly ensure that all the feature-lines are represented as patch boundaries. To be able to consistently do so, we allow non-quadrilateral patches and T-junctions in the layout; the key is the ability to constrain the layout so that it still admits a globally consistent, T-junction-free, and pure-quad internal tessellation of its patches. This requires the insertion of additional irregular-vertices inside patches, but the regularity of the final-mesh is safeguarded by optimizing for both their number and for their reciprocal alignment. In total, our method guarantees the reproduction of feature-lines by construction, while still producing good quality, isometric, pure-quad, conforming meshes, making it an ideal candidate for CAD models. Moreover, the method is fully automatic, requiring no user intervention, and remarkably reliable, requiring little assumptions on the input mesh, as we demonstrate by batch processing the entire Thingi10K repository, with less than 0.5% of the attempted cases failing to produce a usable mesh.

CCS Concepts: • **Computing methodologies** → **Computer graphics**; **Shape modeling**; **Mesh geometry models**.

Additional Key Words and Phrases: modelling, geometry processing, quad-meshing

ACM Reference Format:
Nico Pietroni, Stefano Nuvoli, Thomas Alderighi, Paolo Cignoni, and Marco Tarini. 2021. Reliable Feature-Line Driven Quad-Remeshing. *ACM Trans. Graph.* 40, 4, Article 155 (August 2021), 17 pages. https://doi.org/10.1145/3450626.3459941

Authors' addresses: Nico Pietroni, University of Technology Sydney, Sydney, Australia, nico.pietroni@uts.edu.au; Stefano Nuvoli, Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy, s.nuvoli@studenti.unica.it; Thomas Alderighi, University of Pisa , ISTI-CNR, Pisa, Italy, thomas.alderighi@isti.cnr.it; Paolo Cignoni, ISTI-CNR, Pisa, Italy, paolo.cignoni@isti.cnr.it; Marco Tarini, University of Milan, Milan, Italy, marco.tarini@unimi.it.

0730-0301/2021/8-ART155 $15.00
https://doi.org/10.1145/3450626.3459941

1 INTRODUCTION

Automatic quad-remeshing, the task of constructing a good pure-quad meshing representation of a given surface, is a long-standing open problem, despite intensive research, a succession of progresses and breakthroughs, and an urgent need by the industry (where it is known as "retopology").

The problem is acknowledged as difficult, with a large number of conflicting and elusive objectives, such as isometry (use of controlled element size), well-shaped quadrangular polygons, regularity (the mesh should be mostly like a grid), and compact underlying structure (mesh should have only a limited number of well-aligned irregular vertices). Automatic methods still fall short of the results that are obtained by skilled human modelers. Pure-quad meshes are inherently a very constrained class of digital representation objects, with local changes having a long-reaching effect, making the problem impervious to any local approach; this is in stark contrast to triangle and quad-dominant meshes.

An often neglected objective is the ability to recreate *feature-lines* of the input shape, such as crease-angles (typical, for example, of CAD models), which, in the context of quad-models, impose that these lines are recreated by edges in the final meshing. This is difficult to reconcile with all the other objectives. Typical existing approaches either sacrifice this objective when it contrasts the others, or even ignore it, focus on the rest of the problem statement, and then try to recover the lines e.g. by snapping vertices, in the late stages of the process.

In this work, we devise an approach that, conversely, is designed around the need to preserve edges. It is a patch-based approach, where the surface is initially split into a coarse layout of polygonal patches, that are then individually tessellated in a globally consistent way. Differently from other patch-based approaches, we make all feature-lines by construction part of the patch-boundaries. To be able to do that, we relax considerably the requirements imposed on the layout: our layouts can present T-junctions and, most importantly, non-rectangular patches. These configurations will be dealt with later, in the tessellation phase, leveraging a better understanding of patch-quadrangulation afforded by recent advancements [Takayama et al. 2014; Tarini 2021].

To this end, we identify a minimal set of requirements that the layout must fulfill in order to admit a *valid* solution, and a strategy to construct layouts that embeds the feature-lines and meets these minimal conditions. We then identify a set of additional conditions, harder to fulfill, that promote the *quality* solution. Our layout construction strategy strives to fulfill this second set of conditions whenever possible. The result is that a final valid quadrangulation is virtually always produced, presenting very high-quality on average, at par with any state-of-the-art methods in this respect.

While previous layout-based methods may lack the flexibility to conform the layout to an arbitrary set of feature-lines, they are appealing because they naturally produce quadrangulations with an underlying coarse structure by construction (meaning that the few irregular vertices are aligned). In our case, extra irregular vertices that are required in the final patch tessellation can partially compromise this: however, we mitigate this substantially by explicitly optimizing for the reciprocal alignment of these vertices.

Reliability. Another issue of automatic quadrangulation methods is that they typically lack *reliability*, both because they necessitate several assumptions on the representation of the input shape and because several shapes are inherently difficult to quadrangulate. Our method, based on a principled analysis of minimal requirements, combined with a small number of *ad-hoc* measures counteracting commonly encountered problems in the input meshes (such as poor initial tessellation), makes the resulting method able to batch process very large collections of data with almost infallible success rate, with extremely good results both in terms of feature preservation and meshing quality.

2 RELATED WORK

The last two decades have seen drastic progress in the ability to automatically generate quad-meshes [Bommes et al. 2013b].

Our method falls in the category of coarse layout methods based on a coarse layout construction [Campen 2017]. Compared to literature, it advanced the state-of-the-art in terms of reliability, and in the ability to preserve feature-lines, without substantially sacrificing mesh-quality (in terms of quad shapes, number and distribution of irregular vertices, and so on).

2.1 Coarse-Layout based Methods

In this class of methods, a layout of patches is first constructed over the surface, either to serve as a domain for a globally smooth parametrization, or to tessellate each patch in the final mesh. In the former case, the final quad-mesh is then constructed by tracing isolines of the parametrization ([Kälberer et al. 2007; Ray et al. 2006; Schertler et al. 2018]).

Conforming layouts. In most direct applications of this idea, the layout is conforming (T-junction-free) and each patch is rectangular shaped (e.g. [Bommes et al. 2013a, 2011; Campen et al. 2012; Panozzo et al. 2011; Razafindrazaka and Polthier 2017; Tarini et al. 2011], among others), using a variety of approaches. This leads to a direct and easy final quadrangulation, but the constrained nature of the layout (or "base complex") makes its construction arduous.

Non-conforming layouts. For this reasons, many methods allow for T-junctions in their definition (e.g. [Myles et al. 2014; Myles and Zorin 2013; Schertler et al. 2018; Usai et al. 2016], among others). This eases the construction, providing more degrees of freedom, which can be exploited to optimize the domain in other respects, however this requires the solution of a non-trivial problem to produce a conforming final-mesh and to determine a globally consistent subdivision of the sides of the layout. To this purpose, similarly to us, it is common to cast this problem as a separate optimization problem, subject to a number of constraints. Unfortunately, it is known [Myles and Zorin 2013] that this problem is not necessarily feasible. Ad-hoc strategies have been proposed: for example, [Myles and Zorin 2013] changes the underling direction field, and [Usai et al. 2016] modifies the layout to avoid the difficult to solve configurations. During these necessary fixing operations, the pursue of any other objective is suspended. This is in contrast with our solution, which will be shown in Section 6.1, and which is the same used to deal with the following point.

Non-rectangular layouts. We go one step further and also drop the constraint of requiring the patch layout to be rectangular, endowing us more flexibility, which is necessary in our case to embed arbitrary features lines as patch boundaries. We are still able to use the resulting layout, thanks to the recent progresses in defining the conditions and the space of solutions of the quadrangulation of non-rectangular shaped regions, and in particular [Takayama et al. 2014; Tarini 2021]. To our knowledge this strategy in the context of quad-meshing has no predecessor. It was used in the similar problem of fusing together existing quad-meshes, redefining their structure only in the proximity of the connection [Nuvoli et al. 2019]. However, this approach would be unable to preserve the structure of the quadrangulation of an entire complex shape. [Takayama et al. 2014] employs non-rectangular patches in manually designed layouts. Neither approaches would be able to automatically construct an appropriate layout.

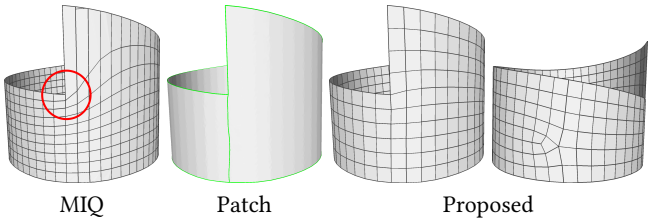


Fig. 2. A simple strip glued to itself with boundary-conforming cross-field: the mixed-integer parametrization (MIQ) [Bommes et al. 2009] cannot obtain a valid quadrangulation because of huge distortions and foldovers and in the intermediate parametrization (circled in red). In our method, instead, irregular vertices are added inside the path, in appropriate position, resulting in a valid and high-quality semi-regular tessellation (as shown in the rear view, right-most image).

2.2 Field-aligned methods

A fruitful trend in quad-remeshing strategies (including ours) is to rely on an auxiliary tangent space direction field to control the orientation of the edges, and, by implication, the location of irregular vertices (at field singular points). Several methods specialized in the design of cross-field with appropriate characteristics, e.g. curvature alignment (see Course [Vaxman et al. 2017]). In the context of coarse domain layout, this concept translates in the idea of constructing the layout tracing lines which are aligned to the field, as we do. The biggest difference with the State of the Art is that we trace lines avoiding field singularities, rather that to connect them [Bommes et al. 2011; Campen et al. 2013; Pietroni et al. 2015; Tarini et al. 2011], because we let the feature line dictate the location of the corner of the patches, and we are able to deal with the resulting non-rectangular patches. This is similar to what it is done in the first phase of [Campen et al. 2012]. Robustly tracing lines on a discretized surface is an interesting problem that has been solved before, e.g. [Pietroni et al. 2016; Razafindrazaka et al. 2015]. Our solution, shown in Section 5.3 adapts a similar concept.

The parametrization method of Myles et al. [2014] resembles our method, in that the mesh is split into a set of quadrilateral patches that allows for T-junctions, by tracing field-aligned lines, although its objective is to derive a bijective parametrization rather than a global quadrangulation. Our objective imposes additional consistency constraints, such as to tessellate boundaries shared by two patches consistently; this is a central part of our method, and can require the addition of extra irregular vertices even inside quadrangular patches (e.g. see Figure 2). Among other key differences, in [Myles et al. 2014] the lines are traced from field singularities, disallowing patches to contain them, whereas we allow for irregular vertices inside (non necessarily quadrilateral) patches and explicitly optimize their placement; also, we allow the traced lines to drift from the direction field in order to optimize the layout structure and to avoid "spiralling".

2.3 Feature-edges preservation

In earlier approaches, feature lines are preserved by snapping [Tarini et al. 2010] vertices of a quad-meshes edges into feature lines during the process. In a similar spirit, parametrization-based methods force the parametric position of the feature-edges on the closest integer parametric line, e.g. [Tarini et al. 2011]. Such approaches can get stuck in local minima or oscillate, as one snap operation can undo another. In a similar spirit, other remeshing methods foster feature-lines preservation by adding soft forces pushing or keeping vertices on feature lines [Bommes et al. 2011, 2013b, 2009; Fang et al. 2018]. This is not necessarily effective on all the targeted features. Also, it is difficult to ensure that this does not impact other requirements, such as the absence of fold-overs in the parametrization. Methods such as [Bommes et al. 2013a; Campen et al. 2015] introduce constraints to avoid fold-overs, while another class of solutions goes one step back in the pipeline [Diamanti et al. 2015; Myles and Zorin 2013; Zhang et al. 2020] and re-optimize the cross-field and provide better input for the following parameterization step. Instant Meshes [Jakob et al. 2015] formulates an "extrinsic" parametrization approach that "naturally" exhibit a form of sharp-feature alignment. The resulting method behaves similarly to a feature-preserving force, leading to similar problems (although no soft constraint is explicitly added, and feature lines are not even explicitly detected).

A more recent technique proposed by Fang and colleagues [2018] combines global parametrization with a local Morse-based approach around the singularities. Such a method allows for managing complex, sharp-features. Unfortunately, it tends to introduce unexpected irregularities in the final quadrangulation and then break the overall flow.

A new approach to preserve sharp-features has been recently proposed by LoopyCuts [Livesu et al. 2020]. This technique (originally designed to create hex-meshes) distributes loops that run nearby sharp features and then snap them at the end of the process. However, this approach is not guaranteed to handle complex feature configurations, which our method is able to process.

An orthogonal but closely related problem is how to automatically *detect* features to be preserved (see e.g. [Matveev et al. 2020] for a recent technique). We do not claim any advancement in this area, and methods more sophisticated than the trivial dihedral angle thresholding we employed can improve our results.

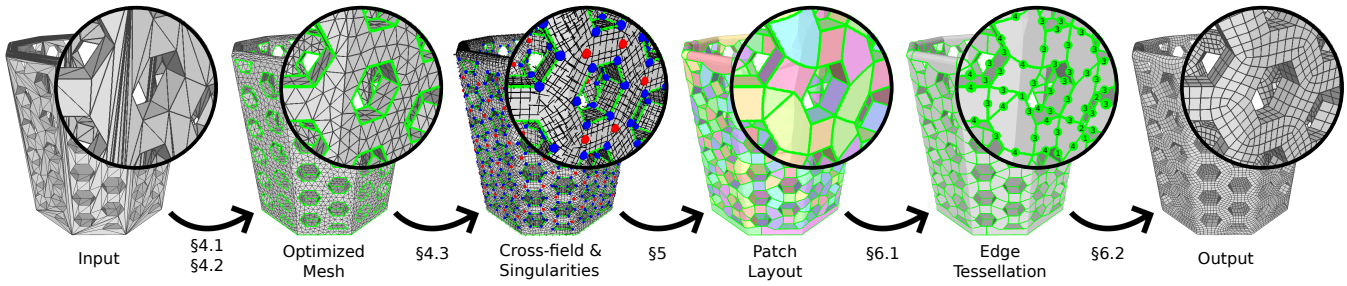


Fig. 3. The phases of our remeshing algorithm. Refer to Section 3.2 for an overview, and the referred Sections for the specific phases.

3 OVERVIEW

Our processing pipeline takes as input a triangle mesh, with a given set of feature-lines, and produces as output a good-quality quadrilateral mesh reproducing all the feature-lines as edges. Our method falls in the category of coarse-layout based methods: first, we partition the mesh into a layout of coarse regions (*patches*), then we consistently quadrangulate each patch to obtain the final quad-mesh.

Although any set of the input mesh-edges can be selected as a feature-line, in our experimental setup, we automatically identify feature-lines as geometric creases.

The main parameter of the method is the target edge length, which will be only roughly matched by the final mesh. In our experiment, we set it automatically as the side of a square having $1/K$ the area of the mesh, with $K = 10^4$ (unless otherwise specified).

3.1 Objectives

Our method is designed primarily around the need to accurately preserve feature-lines. We achieve this goal by ensuring that feature-edges are preserved, or reproduced, by construction during each phase of the method. The central part is that we tailor the patch-layout around the feature-lines, ensuring that they are all represented as patch boundaries.

Naturally, we also strive to maximize the quality of the output meshing, meaning that the resulting quad meshes should have approximately rectangular and flat-shaped faces, reasonable uniform edge-sizes, and, most importantly, only the necessary irregular vertices. Specifically, in our method, meshing quality is achieved primarily by producing coarse-layouts where most or all patches allow for predictably good-quality internal quadrangulations.

As an additional quality requirement, irregular vertices should also be reciprocally aligned, whenever possible; this is known to be beneficial [Bommes et al. 2011; Tarini et al. 2010] as it leads to simpler separatrix graphs (e.g. see Figure 25).

Finally, we want our method to be reliable and fully automatic, i.e., to be able to automatically produce good results on the largest possible set of input meshes, with minimal requirements on their initial meshing quality or consistency. We empirically verify the fulfillment of this objective by successfully batch-processing thousands of models from the entire database Thingi10k [Zhou and Jacobson 2016].

These objectives make our method ideal for automatic processing of CAD models.

3.2 Steps breakdown

Figure 3 shows an overview of our pipeline.

Input-mesh optimization. As a preliminary step, we perform an automatic triangle-mesh optimization of the input (Section 4.2, and Figure 3.b), by performing a sequence of local operations aimed at improving the triangle-shapes and vertex distribution of the input mesh. During this process, we simply disallow any operation that would disrupt a feature-edge. This step confers to our system the ability to be used on meshes with very unevenly sized or ill-shaped triangles, such as most CAD models.

Cross-field construction. Then, we compute a *cross-field*, i.e., a smooth, 4-rotational-symmetric tangent-vector field, on the surface (see Section 4.3, and Figure 3.c). The field is aligned to the feature-lines by construction and propagated over the rest of the surface. In absence of feature-lines, the cross-field is aligned to main curvature directions. The cross-field is sampled at mesh faces and defines a set of singularities at mesh vertices.

Layout construction. Next, we trace a number of *paths* across the surface, partitioning the surface into the layout *patches* to be quadrangulated (see Section 5, and Figure 3.d). All feature-lines are automatically considered part of the paths, and therefore they appear as patch boundaries by construction. Additional paths are traced along the directions specified by the cross-field. This is the central step of the pipeline; its objective is to produce patches that are easily *"quadrangulable"*, that is, which have a geometric shape and a topology that strongly favors the existence of a valid, good-quality semi-regular internal quadrangulation. We define a set of topological and geometric criteria that allow us to quickly estimate whether this is the case for a given patch. Our criteria descend from the patch-quadrangulation strategy that will be used by the final step. Armed with this, we adopt a greedy algorithm to identify a minimal set of paths such that the criteria are met for all patches.

Tessellation of paths. In the next step, we tessellate the sides of each patch in a number of final mesh-edges (see Section 6.1). We determine the exact number (for each side) by solving a global Integer Linear Program (ILP), driven by an objective function which balances conflicting objectives such as constant edge length, and regularity of the final meshing (see Figure 3.e). Specifically, the main component of the objective function favors solutions that allow, in most or all patches, a regular quadrangulation in the last step.

Patch quadrangulation. Once the number of edges around the sides of each patch is determined, the final stage of the pipeline consists of quadrangulating the interior of each patch (see Section 6, and Figure 3.f). This task is carried independently for each patch and results in the final conforming (T-junction free) pure quad-meshing. In contrast to many previous patch-based methods, the internal quadrangulation of a patch can feature irregular vertices. In each patch, this task can be solved using one of two possible strategies. The *"simple strategy"*, used whenever applicable, consists of the insertion of at most a single internal irregular vertex (and none in rectangular patches), as studied in [Tarini 2021]. Only in the few patches where this solution is not applicable, we resort, as a fallback, to the *"general strategy"*, i.e. the general patch tessellation algorithm described in [Takayama et al. 2014]; this always finds a valid solution, by the use of multiple internal singularities, although at the price of a diminished local meshing quality, edge alignment, and regularity. The benefit of the *simple strategy* is not only that it consistently produces good quality, maximally regular meshing, but, more importantly, that the conditions for its applicability are expressed in closed-form, as a function of the number of edges around the patches [Tarini 2021], and this can be targeted in the other stages of the pipeline; specifically, we use this to derive the characterization of quadrangulable patches (for the layout-construction phase), and the conditions pursued by the ILP in the path tessellation phase.

Irregular vertices and their reciprocal alignment. In our method, irregular vertices of the final quad-mesh stem from two different sources. They appear at corners of the patches, and internally in patches. The former ones tend to be reciprocally aligned, by virtue of the coarse structure of the layout (this is the premise of any layout-based approach). To also favor the reciprocal alignment of the latter vertices, we add an apposite term in the energy minimized by the ILP.

The following Sections detail each phase of the above pipeline.

4 INPUT PREPARATION

As a preliminary step, we make the input triangle-mesh ready for the subsequent phases.

4.1 Feature-edge identification

First of all, we mark a selection of mesh-edges as feature-edges, which will be maintained (by construction) throughout all the rest of the method, and thus will be reproduced in the final quad-mesh. In all our experiments, we automatically select feature-edges as creases, thresholding the dihedral angles, which works well for CAD models, but other strategies can be devised.

When the input-mesh is not closed, we also choose to mark boundary edges as feature-edges, so that they will also be preserved.

4.2 Preliminary input-mesh optimization

Most of the CAD models present an irregular tessellation with a big disparity in triangle sizes, in edge lengths, and often extremely long and thin triangles. This hinders subsequent phases, such as cross-field definition, or path tracing.

To counter this, we perform a simple preliminary re-meshing step on the input model, following the well-known approach of [Hoppe et al. 1993]. A sequence of local operations like edge-flips, edge-collapses, and edge-splits, are performed in order to make edge lengths close to a given target value. We simply cycle over all available operations in an arbitrary order and perform any that has a beneficial effect, evaluated on the lengths of all affected edges. In our case, during the entire process, we preserve feature-lines by simply disallowing any operation that would disrupt them.

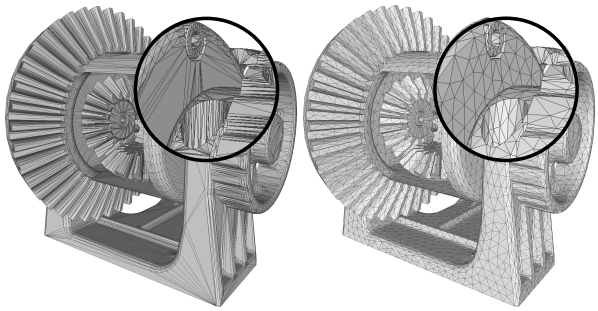


Fig. 4. The effect of adaptive remeshing on one mechanical object from the Thingi10k [Zhou and Jacobson 2016] repository.

Our preliminary remeshing procedure comprehends two passes. We perform a first cycle of operations targeting a uniform target edge-length, set at half the edge-length requested in the final quad mesh. Normally, [Hoppe et al. 1993] produces a triangle mesh with fairly regularly-shaped and equally-sized faces; in our case, feature-lines preservation can result in badly-shaped and smaller triangles near conglomerations of feature-lines. We detect this occurrence and interpret it as an indication of the necessity for a higher local resolution (because the tangent-field will typically require high-frequency features around these areas). For this reason, we perform a second pass of operations where we target an adaptive edge-length, defined locally as a function of the aspect ratio of the triangles (measured as the ratio between inner and outer circle) at the end of the first pass. We clamp the 10% percentile of faces with the lower aspect ratio, then linearly interpolate the target edge-length between 0.3 and 3 times the original requested length. This results in a triangle mesh with a local resolution that is roughly adaptive to the complexity of the input set of feature-lines (see Figure 4).

4.3 Cross-field definition

Next, we construct a cross-field (a 4-RoSy tangent-vector field [Vaxman et al. 2017]) on the input surface, aligned to feature-lines, by assigning a tangent direction in each triangle.

Initially, we split any input mesh face adjacent to multiple feature-edges until each face is adjacent to at most one (see Figure 5). We assign to any face adjacent to a feature-edge the field values matching the direction of that edge. Finally, we diffuse the cross-field values to every other triangle. During the propagation, we add soft constraints to align the field to the main curvature directions, similarly to [Panozzo et al. 2012]. This lets us define a cross-field even in absence of any feature-line. We adopt the field propagation method described in [Diamanti et al. 2014], but other similar solutions, e.g. [Jakob et al. 2015; Zhang et al. 2020] could be used interchangeably. The per-face cross-field determines irregular points at vertices.

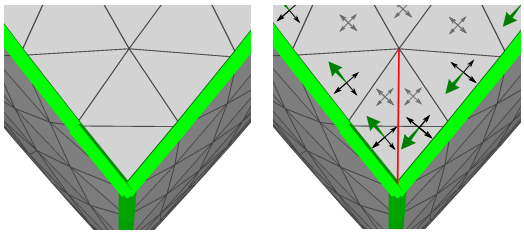


Fig. 5. Faces that are adjacent to multiple features edges are split, adding new edges (red lines), and each face is assigned to the field direction aligned to its only adjacent feature edge (green arrows). A cross field is then propagated over other triangles (gray crosses).

5 PATCH LAYOUT DECOMPOSITION

In this step of the pipeline, we trace a set of field-aligned *paths* along the field, which partition the surface into a layout of *patches*. Patches need not be rectangular. Paths are allowed to intersect, as long as they follow orthogonal, rather than parallel, directions of the cross-field at the crossing. Paths can be closed loops, or have end-points at mesh boundaries, or on other paths, possibly creating T-junctions.

As a starting point, we convert all feature-lines to paths, simply by marking all feature-edges as path-edges. The rest of the layout is to be constructed by adding additional paths, striving to obtain patches with a valid topology and a favorable shape. The new paths never pass through singular points of the cross-field. New paths are drawn as chains of input-mesh edges.

First, we cut the mesh open along feature-edges: each original feature-edge generates two *directed* boundary-edges, one on each side. Affected vertices are duplicated as required, and all copies become boundary-vertices (we keep a list, for each vertex, of its duplicates). If a duplicated vertex corresponds to a singular value of the cross-field, it loses that status, and a value for the field is updated by averaging the connected faces.

Then, each boundary-edge is labeled with an *orientation*, i.e. an index specifying one of the four cross-field directions of its (unique) adjacent face (cross-field directions are arbitrary numbered, clockwise, from 0 to 3). Observe that one of these directions is always, by construction, perfectly aligned to the considered edge.

Classification of boundary vertices. Boundary-edges are necessarily grouped in directed loops (surrounding a patch, or internal to a patch). Inside each loop, we classify each traversed vertex according to the number k of 90° clockwise turns that are required to make the orientation of its previous and next edge match in the field. This can be determined by a traversal of the fan of triangles around that vertex. When $k = 0$, the boundary-vertex is said to be *straight*, when $k = 1$, (equivalently, -3) it is a *right-turn*, when $k = -1$ (equivalently, $+3$), a *left-turn*, and when it is $k = \pm 2$, it is a *U-turn* (see Figure 6).

Observe that an original mesh-vertex on a feature-line is split into boundary vertices receiving in general different labels and that this labeling is also applied, unmodified, to copies of vertices that were originally classified as field-singularities (see Figure 7).

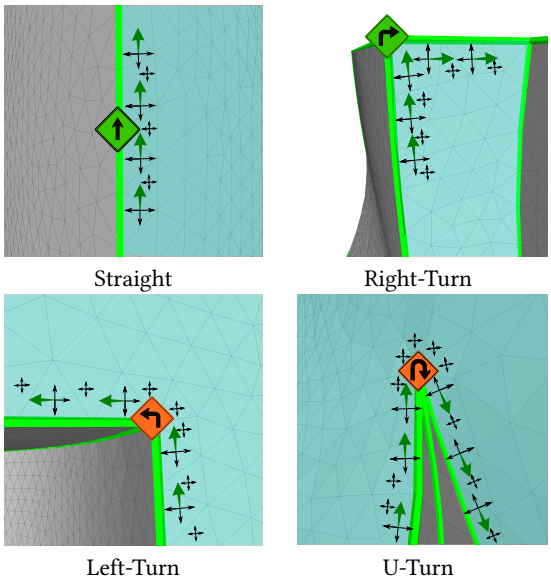


Fig. 6. The four different classes of boundary-vertices of the bluish patch. The classification is defined by the orientation of the green boundary-edges (green arrows) propagated across the cross field (black crosses).

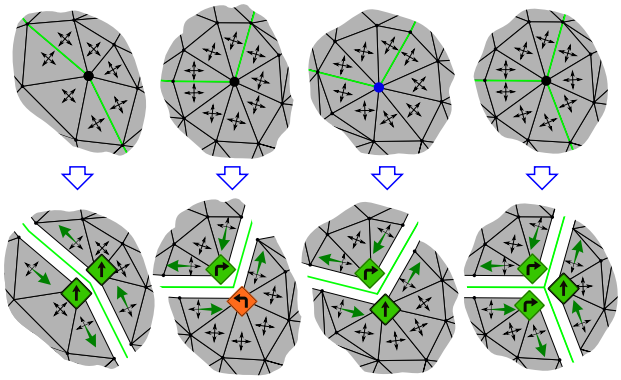


Fig. 7. Examples of duplicating vertices along a path, and the labeling of the boundary vertices derived by the field directions (green and orange boxes, see Fig.6). The blue vertex is a field singularity (observe the field).

5.1 Conditions for patches

As mentioned, we seek the patches that allow for a valid and high-quality internal quadrangulation. At a bare minimum, we need them to be a valid input for our "general" quadrangulation strategy (which is the algorithm of [Takayama et al. 2014]).

This is guaranteed by the following three *"validity conditions"*:

Topological constraint The patch must be homeomorphic to a disk, meaning that it must be adjacent to a single loop of border-edges.

Valence constraint The valence of the patch must be between 3 and 6.

Convexity Only straight vertices and right-turns are allowed on the boundary loop.

The *valence* of a (disk-homeomorphic) patch, used by the second condition, is defined as the number of turns in its boundary loop, and can be understood as its number of topological sides (a side of a patch is the part of the loop from one turn to the next). The upper limit to the value 6 reflects a limitation of [Takayama et al. 2014].

The third condition disallows left-turns, as they would result in quad angles larger than 180°, and U-turns, as the resulting angle would be close to 360°(in Figures 6,7,14, disallowed boundary vertices are colored in orange, and allowed ones in green).

This set of conditions, while weak and easy to enforce, suffice to guarantee the applicability of our fallback "general" strategy. Yet, there is no guarantee on the quality of the resulting meshing; experiments confirm that this can result in the introduction of an excessive constellation of new internal irregular vertices (disrupting field alignment), and impact negatively the isometry and quad-shapes of the final mesh.

Conversely, our "simple" quadrangulation strategy, when applicable, results in maximal regularity and tends to create good quality meshing, well aligned to the smooth cross-field. Its applicability, as studied in [Tarini 2021], depends on the fulfillment of simple equations or inequalities defined, in closed-form, on the integer number of edges e_i on each side i of a patch. We report these conditions in full, for the cases that are relevant to us:

$$\begin{aligned}
&\text{in a 3-sided patch:} && \forall i,\ e_i \le e_{i+1} + e_{i+2} \\
&\text{in a 4-sided patch:} && \forall i,\ e_i = e_{i+2} \\
&\text{in a 5-sided patch:} && \forall i,\ e_i + e_{i+1} + e_{i+4} \ge e_{i+2} + e_{i+3} \\
&\text{in a 6-sided patch:} && \forall i,\ e_i \le e_{i+2} + e_{i+4}
\end{aligned} \tag{1}$$

(the indices are intended to be modulo the valency of the patch).

The integer edge numbers e_i will only be determined in a subsequent phase. However, because we are targeting a constant edge size, we can expect these numbers to be roughly proportional to the (geodesic) lengths of the respective patch side. Therefore, we can predict that the conditions above will be more likely to be met whenever the same equation or inequality approximately holds on the scalar lengths of the patch sides.

The conditions in Equation 1 can be understood as a generalization to non-rectangular patches of the well-known condition, sought in many coarse-layout based remeshing approaches, to attain opposite sizes of rectangular patches of a matching length ([Campen et al. 2012; Pietroni et al. 2015; Schertler et al. 2018; Tarini et al. 2011], among others).

With this set of motivations in mind, we want layout patches to also fulfill the two following *"quality conditions"*:

Geometric Condition The lengths of the patches sides must approximately fulfill the inequality or equality above (1). We add a tolerance equal to the length of the shortest side.

Valence Match Rectangular (4-sided) patches should contain no field-singular vertex, and patches with a single internal field-singular vertex should only contain a valency matching that of the singularity; no patches should contain more than one field singularity.

The latter condition is useful as it favors the alignment of final quad-mesh edges with the smooth cross-field, resulting in a higher-quality meshing. This is because the simple strategy, when applied on a non-rectangular patch, produces a single irregular vertex with a valency corresponding to that of the patch, located somewhere in the interior of the patch, and, when applied to a rectangular patch, produces no irregular vertex.

Differently from the validity conditions, which are *necessary* to guarantee a valid solution, adherence to the quality conditions is only *desirable* because it favors the final meshing quality.

5.2 Layout Construction Procedure

The initial layout generated by the sole feature-lines typically infringes the conditions stated above. We complete this layout by adding new paths, to enforce the fulfillment of the validity conditions on all patches, and maximizing the number of patches that also meet the quality conditions.

Our heuristic consists of a succession of "*rounds*", in each of which we insert a certain number of new paths, targeting one specific problem. The sequence of rounds is described below in Section 5.4.

All rounds work in the same fashion: we first trace a large number of *candidate paths* on the mesh (Section 5.3), we sort them to get a good spatial distribution and, in that order, we decide whether or not to use them for the patch layout. Any candidate path that does not intersect *tangentially* previously inserted paths is chosen if either: (1) it contributes directly toward the fulfillment of any one objective (for example by separating two singularities, or removing a left-turn) or (2) it splits any patch that still does not meet all the objectives. The latter helps because smaller patches make easier for the objectives to be fulfilled.

After all rounds, we check if any patches still fail to meet the validity conditions. In this case, we repeat a sequence of rounds concentrated only on that patch, this time with only the candidate-paths inside it and disallowing them to exit it (any new candidate-path is stopped as soon as it reaches the patch boundary, creating a T-junction). We recursively repeat this step until all patches are solved.

This strategy allows the construction of shorter candidate paths, with fewer chances to be prevented by invalid intersections with existing paths; these new paths can, in turn, serve as a stopping point in subsequent recursive steps, and so on until all constraints are met (see Figure 8).

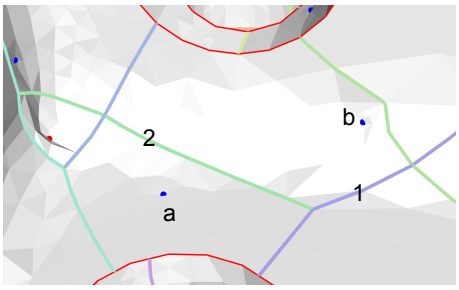


Fig. 8. An example of the recursive strategy fixing a constraint: the path 1 creates a T-junction, but, in the next recursive step, it allows path 2 to terminate early (with another T-junction), thus allowing field-singularities a and b to end up in two different patches, meeting the valency constraint.

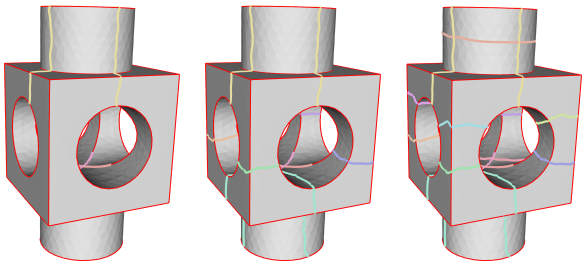


Fig. 9. Examples of paths inserted in succession (in the Looped paths round).

Candidate Sorting. The order in which potential paths are considered for insertion drastically affects the set of paths that are picked. We want to avoid conglomerations of close paths, so (similarly to LoopyCuts [Livesu et al. 2020]) we always analyze the potential path that presents the largest distance from the closest edge in any picked path (including initial ones). Figure 9 exemplifies the effect of this choice.

Clean-up of redundant paths. Once the rounds of path insertion have completed we have a layout that meets the validity conditions on all patches, and the quality conditions on most patches (if not necessarily all). During its construction, however, there is no easy way to predict whether a path being inserted will end up being useful to the fulfillment of any constraint because multiple paths can contribute to meet the same constraints. Therefore, at the end of the insertion process, we iterate over all paths again and dissolve the ones whenever we determine that its removal does not infringe any condition that is already met. Once again, the ordering of testing is relevant; empirically, we found that processing paths in reverse of their insertion works best. Figure 10 shows the result of this processing step.

Next, we illustrate how a single candidate path is constructed.

5.3 Tracing a Candidate Path

We cast the problem of tracing a new candidate path as a shortest-path search between a given *starting point* and a set of potential *ending points*. This is solved by executing the Dijkstra algorithm over an auxiliary *directed weighted graph* G which is constructed from the mesh for this purposes.

Auxiliary graph construction. First, we assign a cross-field value at each vertex, except for field-singularities, by averaging the values

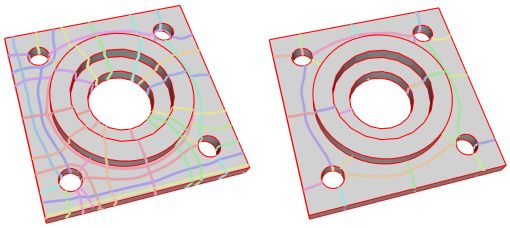


Fig. 10. The effect of Clean-up of redundant paths.

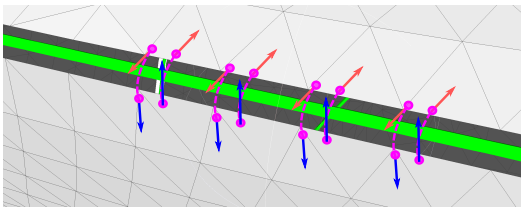


Fig. 11. Nodes of graph G from both sides of a split edges are connected. In the image, positions of nodes has been offset to illustrate the connectivity of the graph.

of the surrounding faces. While averaging cross-field values, we take into account the 90° rotational symmetry of the field.

Similarly to [Campen et al. 2012] (which follows *M4* mesh stratification introduced by [Kälberer et al. 2007]), we populate G with four nodes for each mesh-vertex, one for each tangent direction of the field at that vertex, except for field-singularities. Singular vertices, which do not have an associated field value, are not represented in G (so, no path will be traced across them).

Because each node in G represents both a vertex and a field direction, by selecting the starting node and the (potential) ending nodes of the path search, we are prescribing not only the starting and ending positions, but also the outgoing and incoming cross-field directions of the candidate path being traced.

We connect two nodes in G whenever (1) they reside at two vertices v_i and v_j connected by a mesh-edge, (2) they correspond to a matching direction of the cross-field, and (3) the angle between $(v_j - v_i)$ and their (averaged) field direction f is smaller than 45°. Each connection is weighted by the modulus of its *drift* value, defined, as in [Tarini et al. 2011], as the quantity

$$d_{ij} = (v_i - v_j) \cdot f^{\perp}$$

where $f^{\perp}$ is the tangent direction orthogonal to f, found as the cross product with the surface normal ($f^{\perp} = f \times n_v$); this value measures how much a path traversing that connection deviates from the field.

We also add a zero-weighted connection between any two nodes located at straight vertices that are duplicates of an original mesh vertex, having a matching tangent direction, but not on either of the two opposite field directions aligned to the path (see Figure 11). This allows a candidate path to be traced across an existing path, but never tangentially to it.

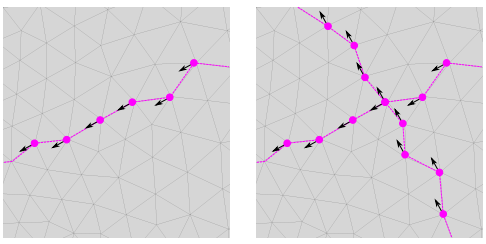


Fig. 12. Paths are drawn over mesh edges, so in general they will be jagged. Right: intersection occur at vertices, only if the followed direction fields (black arrows) are orthogonal.

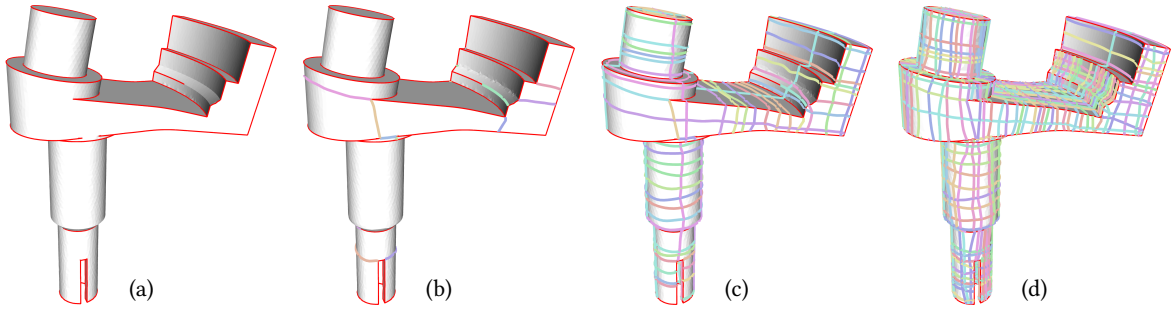


Fig. 13. Different rounds of the tracing process: initial sharp-features (a); convexity-enforcement (b); looped-paths (c); border-to-border paths round (d).

Due to its connectivity, graph G automatically allows only for candidate paths that go topologically straight over a cross-field direction, meaning that traces path always follow a consistent direction of the cross-field. The weighting favors the alignment of the path to the cross-fields.

The tracing of a candidate path can fail, when there are not enough edges in the mesh.

Jagged paths. Because new candidate paths are drawn over mesh-edges, they will in general be jagged (see Figure 12). We use the modulus of the drift value as weigh because Dijkstra algorithm requires weights to be non-negative. This has the drawback of penalizing zigzagging paths during the search, because their drifts in opposite directions are summed up instead of canceling. To mitigate this problem, we augment the connectivity of the G by inserting a virtual connection between every two nodes originally separated by two connections, setting its cost to be the sum of the two (signed) drift values. However, whenever a virtual connection is traversed, we also visit the intermediate node, in order to be able to detect (tangential) crossings with any other candidate path.

Tangential crossing detection. Given two candidate paths, it is trivial to check whether or not they intersect tangentially, by testing if they pass through two nodes that reside at the same mesh vertex, associated to two cross-field directions that are not orthogonal.

Arch-length estimation. To evaluate the fulfillment of the Geometric Condition (Section 5.1), we need to measure the lengths of the sides of a patch. Summing all mesh edges forming the path would overestimate its length of jagged paths. Instead, we only sum the distance spanned by each edge along the cross-field associated to that edge (which is the dot-product of the edge-vector with the field-direction).

5.4 Rounds of insertion of paths

In each round of insertions, we identify a set of potential starting nodes S, and of ending nodes E. Then, we attempt to trace a new path from each node in S, toward any node in E. All successfully traced paths are added to the pool of candidate paths for that round.

The performed rounds are as follow:

Convexity enforcement round. We construct candidate paths explicitly targeted at fixing left-turn and U-turns. As starting nodes S for each left-turn vertex, we add two nodes, one going on straight, and one turning right (see Figure 14,top); for each U-turn vertex, we add three nodes in S, going straight or turning either direction (see Figure 14,bottom). As ending nodes E, we choose the ones residing on the vertices of S with opposite direction: this attempts the creation of paths that fix two different turns, one at each end. This round is repeated a second time, if there are still non-convex patches, using, as E all the straight nodes; paths traced in this way will end on a T-junctions (or on the mesh boundary). Figure 13.b shows the result of this round.

Looped-paths round. In this phase, we populate the pool of potential paths with a set of loop paths. We select a uniformly distributed subset of internal vertices (using the approach of [Corsini et al. 2012]), and we attempt to trace two potential looping paths. A potential looping path is traced by using the same node as the only element of S and E. Figure 13.c shows the result of this step.

Border-to-border paths round. The last phase of our tracing process consists of joining *straight* boundary-nodes with other *straight* boundary-nodes. As starting nodes S, we select a sub-sampling of all *straight* boundary nodes. This creates small paths that can be useful to solve residual problems. Figure 13.d and 9 shows the result of this round.

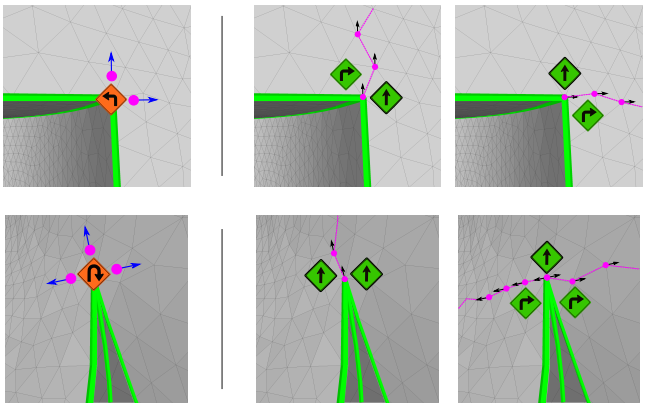


Fig. 14. The *convexity enforcement round*. Left: the starting of nodes added in S for candidate-paths designed to address a left-turn (Top) and a U-turn (Bottom) boundary vertex. Right two images: alternative examples of candidate-paths that can be inserted to meet convexity constraints.

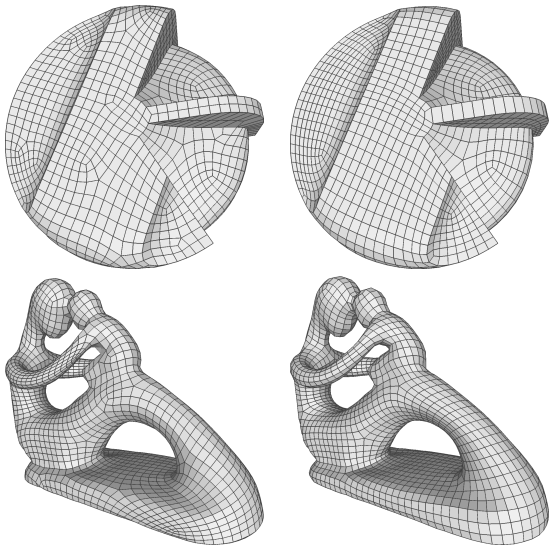


Fig. 15. The improvements induced by the regularity term for non-rectangular patches.

6 FINAL QUADRANGULATION

After the layout is constructed, we need to quadrangulate each patch. The quadrangulation must match at patch boundary, to ensure that the final quadrangulation is conforming. We address this problem by, first, determining the number of edges along each side.

6.1 Patch-side tessellation

We cast this problem a global Integer Linear Program. We split the paths of the layout at every turn vertex and intersection, obtaining a set of *arches*. For each arch, we represent the number of edges contained an integer variable $s_i \geq 1$. The number of edges found along each side of a patch is given by the sum of a given set of s_i (see the two rightmost images in Figure 3).

The ILP minimizes an objective function defined over the s_i. This is given by the sum of several weighted terms, subject to the following constraint.

Parity constraints. It is well known that a patch can be quadrangulated only if it is bounded by an even number of edges. Hence, for each set of arches P surrounding a patch, we add a separated auxiliary linear variable n and include a linear constraint:

$$\sum s_i = 2n, \forall i \in P \tag{2}$$

Isometry term. We define an *isometry* term of the objective function that penalizes the (squared) discrepancy of every s_i from its ideal value $\hat{s_i}$, given by the length of the arch (computed as seen in Section 5.3) divided by the targeted quad-mesh edge:

$$\sum (s_i - \hat{s_i})^2$$

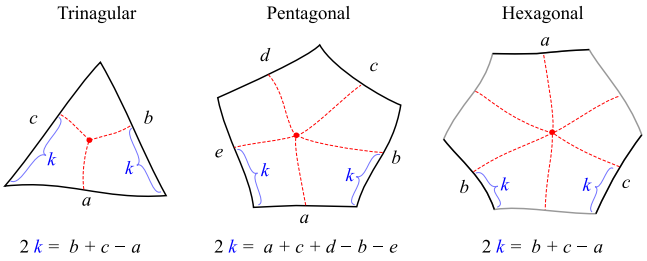


Fig. 16. In a triangular, pentagonal, or hexagonal shaped patch, with a single internal singularity (red dot), the number of edges on each side (a, b, c, d, e) determine the positions k (expressed as a number of edges) at which the separatrix line (red dotted lines) meet the shown side. Formulas for k values, derived from [Tarini 2021], are shown for two of the sides; the others can be determined by rotating the figure.

Regularity term (rectangular patches). A rectangular (4-sided) patch, with e_0, e_1, e_2, e_3 edges on each side, can be tessellated regularly whenever opposite side match, i.e. when $e_0 = e_2$ and $e_1 = e_3$ (which are two linear constraints, because each e_i is by the sum of a small subset of the variable s_i). It is known that this condition cannot always be globally enforced. A simple counterexample is depicted in Figure 2. In other patch-based approaches (e.g. [Usai et al. 2016]), this problem is addressed by detecting when the system is unfeasible, and changing the layout accordingly, or modifying the underlying guiding field [Myles and Zorin 2013]. In our system, we only strongly favor, rather than impose, the equalities. The occasional patches where the value mismatch be addressed, in the next phase, by the "fallback" quadrangulation strategy.

Regularity term (non-rectangular patches). In our approach, patches can also have $n = 3$, 5, or 6 sides. For these patches, the most regular internal quadrangulation possible is the one featuring a single irregular vertex of valency n, which is our *simple* quadrangulation strategy. After [Tarini 2021], a necessary condition for this strategy to be viable is that the inequalities (Equation 1) hold for each side of the patch $i \in [0, n)$ (note that the inequalities are not strict, the equality case just resulting in the irregular vertex to be located along the boundary of the patch). Similarly to the case for rectangular patches, we include these conditions as hard-constraints could make the system unfeasible; instead, we discourage their infringement by penalizing them with a term in the objective function (see Figure 15). Specifically, whenever we need that $a \leq b$, with a and b two quantities linear with the variables, we include a new auxiliary variable in the system K, and we impose the hard linear constraints $a \leq b + K$ and $K \geq 0$. Then, K is added to the objective function. Considering that different patches will have a different number of inequalities, we normalize these energy terms for each patch.

For $n = 3, 4, 5$, the conditions above are sufficient to ensure the viability of the *simple* strategy. For hexagonal patches, an additional condition is required [Tarini 2021]: the three even and the three odd sides of the patch sum up to an even number of edges. To enforce this in a soft sense, we add two auxiliary integer variable k and h, and we constrain both $k \geq 0$ and $a = 2h + k$, with a being the sum of all arches found on any even sides of the patch (and likewise, for the odd sides).

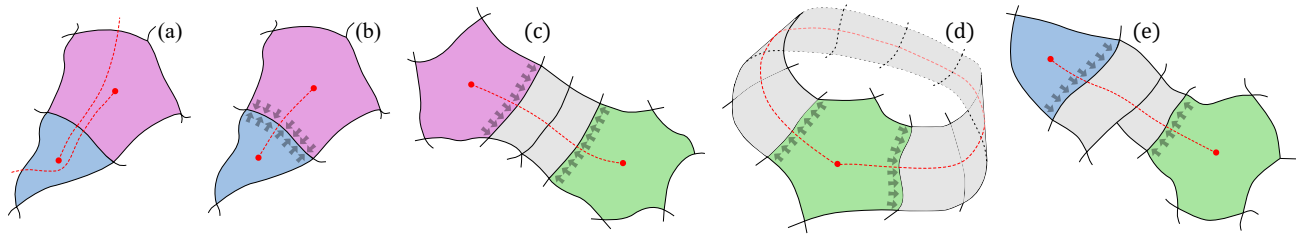


Fig. 17. The irregular vertices added internally to the patches will in general be misaligned (a); (b-e) this is countered by adding a Singularity alignment term for each pair of matching sides of non-rectangular patches, pointed by arrows (see text).

Singularity alignment term. Assuming that the "simple" strategy is applied inside a non-rectangular patch, exactly one stream of edges stemming from the irregular point (often called a separatrix, e.g. [Tarini et al. 2011]) will exit from each side of the patch (red lines in Figure 16), splitting the side into two "sub-sides". The number of edges in the two is given by a linear equation of the number of edges on the sides of the patch [Tarini 2021]. For completeness, in Figure 16 we list the linear equations for all the cases that are relevant in our scenario.

We can exploit this information to promote the reciprocal alignment of the internal singularities in two adjacent patches to be reciprocally aligned. We proceed as follows. First, we identify pairs of non-rectangular patches sides that are either adjacent (Figure 17.b), or separated by a sequence of rectangular patches (Figure 17.c). Note that pair of matching sides can also be found between different sides of the same patch (Figure 17.d). For each such pair, we impose the equality of the two matching sub-sides, in a soft sense (that is, we impose an objective function term, consisting of the absolute difference between the sub-sides that must be matching). For simplicity of implementation, our prototype neglects to add this term for pairs of charts separated by T-junctions (Figure 17.e).

The effectiveness of this term is exemplified in Figure 18.

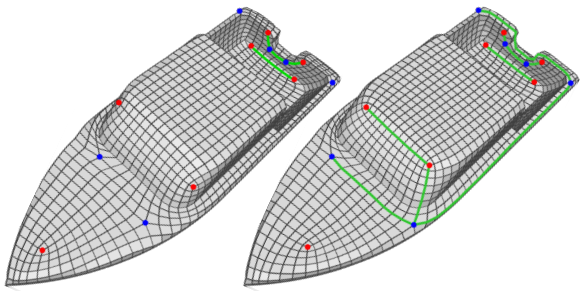


Fig. 18. The effect of the singularity alignment constraints on the final result (left not aligned, right aligned).

Balancing objectives. The four objective function terms, for isometry, regularity (for rectangular and non-rectangular patches), and singularity-alignment, are weighted by parameters that can be used to strike a well-balanced trade-off between these potentially conflicting objectives. For the examples shown in Figures 24, 27 and 28, we used 1 for isometry, 99 and 91 for the two regularity terms, and 9 for singularity-alignment. However, the overall system allows for any variations of these parameters.

When the conditions reflected by regularity and singularity-alignment terms are not met *perfectly*, we incur in a small deterioration of the final quality (for example, extra singular vertices are included, or existing singularity get misaligned); these penalties do not necessarily aggravate for larger infringements of the conditions, which, conversely, are penalized proportionally more by our objective functions. To counter this, we adopt a simple strategy where the ILP is solved twice in succession, the second time dropping the singularity alignment terms that were not met in the first run.

Implementation details. In some examples, the extreme complexity of the shape generates a large number of variables, disproportionately affecting the performance of this phase. As a simple optimization, we group the patches into multiple subsets of m adjacent patches (we used $m = 300$) and separately solve each subsystem fixing the number of edges in the arches along its boundaries.

6.2 Per-patch tessellation

As the last phase, each patch is individually quadrangulated, using either the *simple strategy* [Tarini 2021] whenever applicable (fulfillement of Equation 1), or the *general strategy* otherwise [Takayama et al. 2014]. The constraints we imposed in precedent phases ensure that, at worst, the latter case can be used, and a valid final mesh is generated. As a final phase, we also apply a step of tangent space smoothing [Pietroni et al. 2015] to improve the shape of the quads while constraining vertices on the feature-lines.

7 RESULTS

We initially tested our method on a dataset composed of 308 triangle meshes, which includes all the models used in [Myles et al. 2014] and [Fang et al. 2018]. On a consumer-level laptop (MacBook Pro, 2.9 GHz Intel Core i7, 16GB RAM), the processing time for the entire dataset was about 5 hours (a minute per mesh), distributed as follows: 2h and a half for the initial meshing optimization, 50m for the layout construction, and 2h for the final quadrangulation. We defined features by thresholding crease edges dihedral angles at 45° followed by a erode-dilate procedure to remove small and noise edges (if more than two sharp features meet at a single vertex, they are not eroded). As the target edge-size, we used twice the average edge-size of the input model after mesh optimization. Our prototype is single-threaded and not optimized; it has been implemented using the VCG Library [CNR 2013], CG3Lib [Muntoni and Nuvoli 2021],

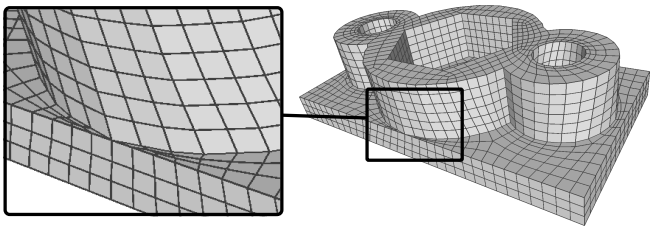


Fig. 19. Feature lines are correctly preserved even when this necessarily implies the creation of poorly shaped quads, with extremely acute angles.

Libigl [Jacobson et al. 2013], and Gurobi [Gurobi Optimization 2018] to solve the ILP problem of Section 6.1.

Figure 24 shows a subset of the dataset; Figure 27 shows a challenging mechanical examples in this dataset. Our method successfully preserves all the sharp features specified in the original mesh and performs well even with smoother organic meshes with no or reduced sharp features, like the ones shown in Figure 28. Table 1 reports statistics of the processed dataset.

We performed a larger scale test on the Thingi10k dataset [Zhou and Jacobson 2016], comprising ten thousand meshes. Our method successfully quadrangulated 9877 models, after only very basic automatic clean-up. Another 3 input models are point clouds or empty meshes; 66 could not be open by the OpenSource software Meshlab [Cignoni et al. 2013]. To perform this test, we automatically split each any existing non-manifold edge, and we remove any resulting non-orientable connected-component (this occurred 36 times). Some of the results are shown in Figure 1 and Figure 29; 90% of the meshes required less than 2.5 minutes, and the 99% of them within 12 minutes.

The entire set of results of both tests can be inspected at **www.quadmesh.cloud**.

Figure 19 shows some challenging examples of sharp feature preservation that would be hard to handle with quadrangulation methods based on global parametrization. Our method can be used to construct mesh at any prescribed, as exemplified in Figure 20; the lower limit of the resolution is tied to the shortest side in the patch layout, which is linked to the size of details with sharp features. Our approach is resilient with a number of inconsistencies of the input mesh, graceful degrading the result and delivering an output reflecting the input problems, but usable in other areas (Figure 23).

Comparisons. We performed a direct comparisons against competing quad-remeshng methods. Figure 21 reports a visual comparison with the quadrangulation technique of Fang et al. [2018]. Figure 26 shows a qualitative and quantitative comparison with Instant-Meshes [Jakob et al. 2015] and Quadriflow [Huang et al. 2018]; faces are color-coded according to a measure of shape quality for quad faces, the Scaled Jacobian [Stimpson et al. 2007], and their distributions are shown by means of histograms. In addition to the aforementioned better preservation of feature lines, the comparison is favourable to our method, both in terms of required number of irregular vertices and shape quality, especially in proximity of the feature lines.

Table 1. Measurements on the result meshes shown in various Figures. We report: the number of vertices (#V), faces (#F), and irregular vertices (#I), the average angle deviation from 90° (AD), the edge-length deviation from the average edge-size (ED), and the average Scaled Jacobian (SJ) [Stimpson et al. 2007]. For Fig 26, we compare with the meshes obtained with Instant Meshing (IM) [Jakob et al. 2015] and Quadriflow (QF) [Huang et al. 2018].

Fig	Model	#V	#F	#I	AD (°)	ED (%)	SJ
27	Spiral	2389	2387	28	19.76	17.40	0.92
	Nasty cheese	15311	15574	2116	24.11	22.31	0.88
	Sydney	3298	3188	247	17.47	15.66	0.93
	Metatron	4647	4667	106	22.46	14.68	0.90
	Transmission	13231	13247	169	7.83	11.06	0.97
	Mazewheel	17682	17712	577	22.33	15.78	0.88
	Twirl	3708	3701	61	22.72	16.31	0.92
	Fusee Lp	14788	14822	275	19.89	13.07	0.90
	Gearbox	67680	67834	3314	22.77	20.08	0.87
	Plate Lp	13114	13130	238	13.25	11.75	0.95
28	Guy	4125	4123	163	16.38	16.92	0.94
	Chinese lion	20919	20915	303	14.53	12.29	0.95
	Octopus	4339	4337	68	12.86	21.16	0.95
	Camel	4479	4479	143	18.67	29.30	0.92
	David	8111	8109	381	16.26	16.93	0.93
	Gargoyle2	31071	31069	480	15.14	15.77	0.94
	Pegaso	18453	18457	336	14.23	15.33	0.95
	Vase lion	25127	25125	357	15.46	15.10	0.94
	Thai statue	28117	28121	538	14.79	13.66	0.94
	Filigree	26067	26171	844	16.31	16.04	0.93
29	N.44503	18937	19130	829	0.28	13.11	0.95
	N.128001	56309	56162	3917	14.44	19.14	0.93
	N.58012	113799	113879	7977	15.49	12.28	0.93
	N.61394	38038	37730	1685	20.82	19.22	0.87
	N.67792	16615	16735	490	14.55	13.78	0.95
	N.69975	22629	22563	938	16.44	15.99	0.92
	N.81877	35185	35305	1835	19.62	15.99	0.92
	N.94884	47116	47466	3479	21.85	20.07	0.89
	N.230923	45410	45398	2108	22.66	30.57	0.87
	N.364256	70060	68905	6654	18.04	16.14	0.91
	N.423069	60804	60624	4123	23.79	41.29	0.85
	N.498974	376509	380068	32241	19.86	16.02	0.89
26	Ujoint *(ours)*	2674	2676	**8**	**4.11**	11.97	**0.99**
	↪ *with QF*	2847	2849	32	7.45	7.89	0.97
	↪ *with IM*	2764	2766	84	5.77	**4.65**	0.98
	Sculpt *(ours)*	2148	2150	**16**	15.76	16.92	0.95
	↪ *with QF*	1848	1850	52	17.52	15.42	0.92
	↪ *with IM*	2214	2216	149	**10.32**	**9.79**	**0.96**
	Bolt *(ours)*	2045	2045	24	7.78	**5.23**	**0.98**
	↪ *with QF*	1914	1914	**22**	10.32	8.42	0.97
	↪ *with IM*	2104	2104	88	**7.60**	7.24	0.97
	Gear *(ours)*	2752	2750	88	**5.27**	**7.25**	**0.99**
	↪ *with QF*	2639	2637	**86**	8.69	7.46	0.97
	↪ *with IM*	2742	2740	129	5.98	7.38	0.98
	Sharp S. *(ours)*	3484	3482	**34**	24.28	18.47	0.87
	↪ *with QF*	3539	3537	98	13.22	**9.63**	**0.95**
	↪ *with IM*	3268	3260	267	**12.25**	12.94	0.95

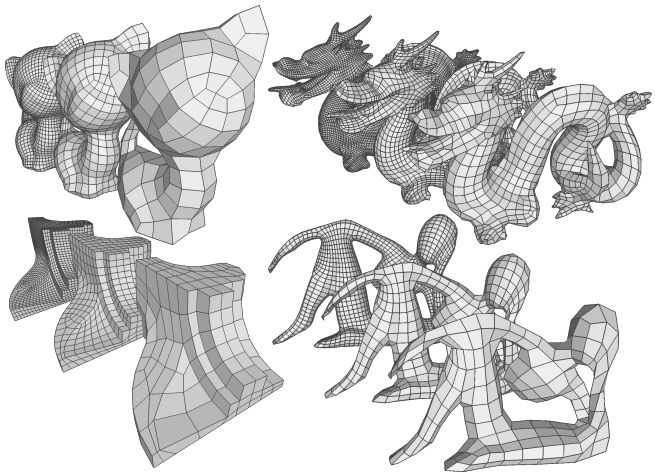


Fig. 20. Examples of quadrangulations created at different resolution.

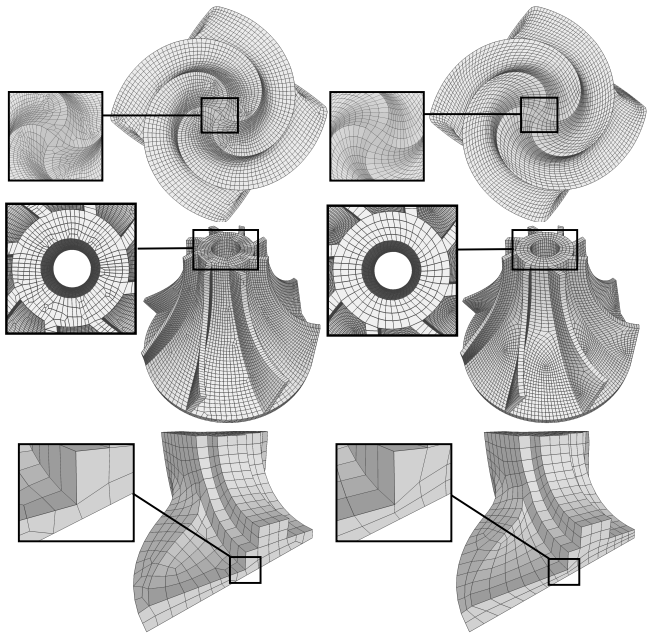
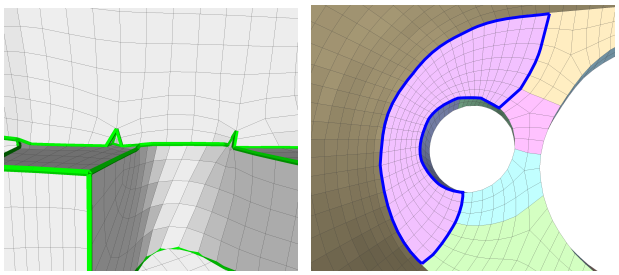


Fig. 21. Comparison of our method (right) with [Fang et al. 2018] (left).

8 DISCUSSION

Our novel remeshing approach is designed around the requirement to preserve feature-lines, and produces good quality, isometric, pure-quad, and conforming meshes. Its reliability is confirmed by extensive experiments where a large number of models are automatically quadrangulated.

One innovative element of our method is that irregular vertices are introduced by two separated approaches: at corners of patches (during layout construction), and internally to patches (during patch quadrangulation). We are encouraged to consider this as a strongly beneficial addition in the panorama of automatic quadrangulators, as we observe that it closely reflects the practice of hand-designed quadrangulation by human modelers (which is still able to produce results of unparalleled quality). The first class of irregular vertices is directly dictated by the shape features, such as the corners of a cubic shape; the second class of irregular vertices is indirectly imposed by the constraints of quad meshing, such as a valency 5 vertex in the center of a pentagonal panel, or the irregular vertices needed to adjust the local tessellation density.



(a) (b)

Fig. 22. (a) Miscategorized feature-edges might influence the final quadrangulation; (b) A squeezed patch considered almost perfectly quadrangulable according to the side-length based criteria.

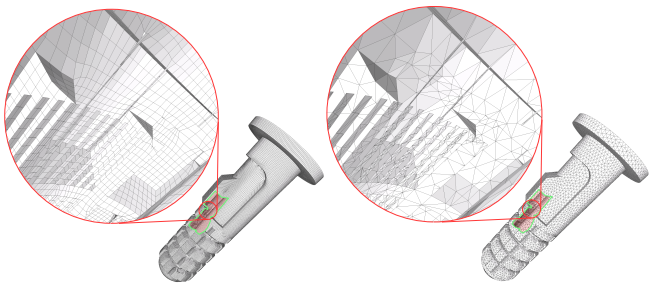


Fig. 23. Resilience: even when the input presents local inconsistencies such as self-intersections, topological noise, double surfaces, and CSG leftovers, our approach succeeds to build a quad tessellation, that is correct in areas where the input is correct.

8.1 Limitations

Lack of strict guarantees. We do not possess a demonstration that the minimal required conditions (during layout construction of Section 5, and edge assignment, Section 6.1) can always be met through our layout construction procedure, which in part relies on heuristics. However, our system never failed to produce a result in almost ten thousand models over real datasets, except in the 0.5% of the cases (which mostly corresponds to non-orientable mesh, see Section 7).

Problems affecting quality. The strict preservation of feature-lines makes the system susceptible to miscategorized feature-edges, or bad-quality (for example, noisy) ones (see Figure 22.a). With CAD models, automatic labeling based on dihedral angles works well, but with other categories of models, such as scanned meshes, this requires more care and it is not trivial (see [Matveev et al. 2020]).

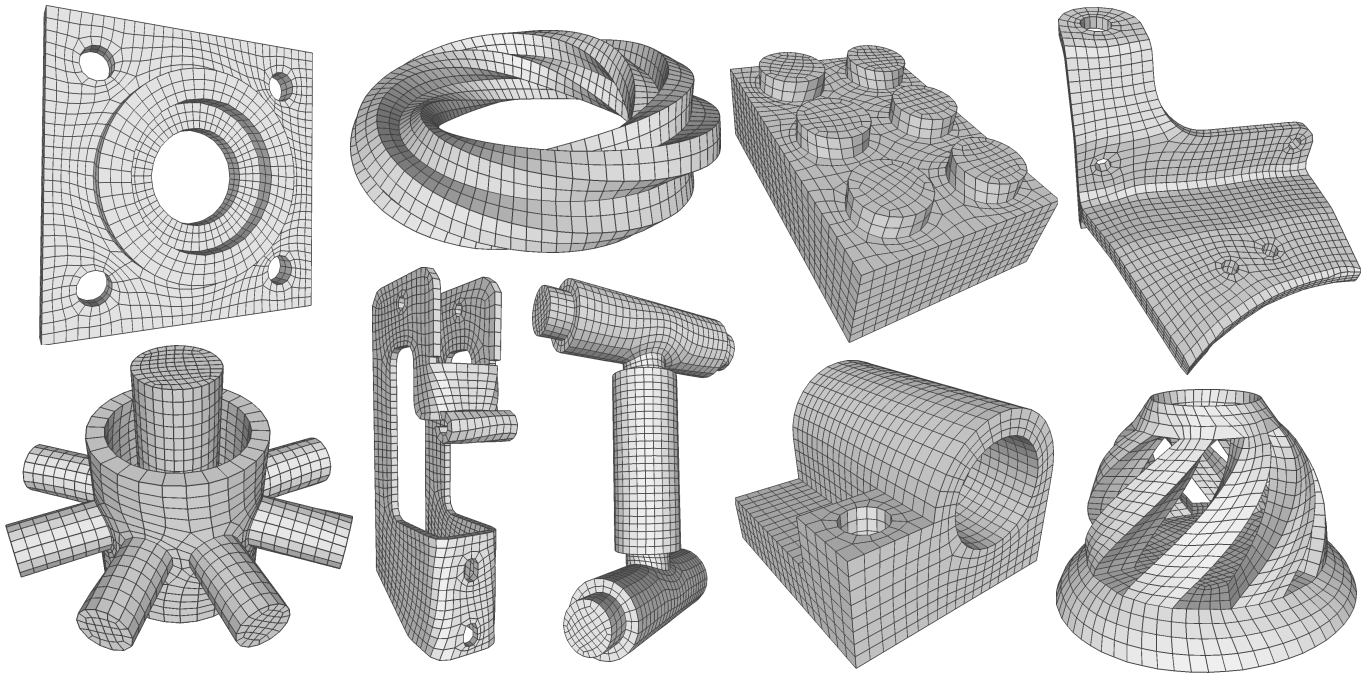


Fig. 24. Examples of quadrangulation of mechanical meshes with simple shapes; the connectivity closely resemble manully modelled meshing.

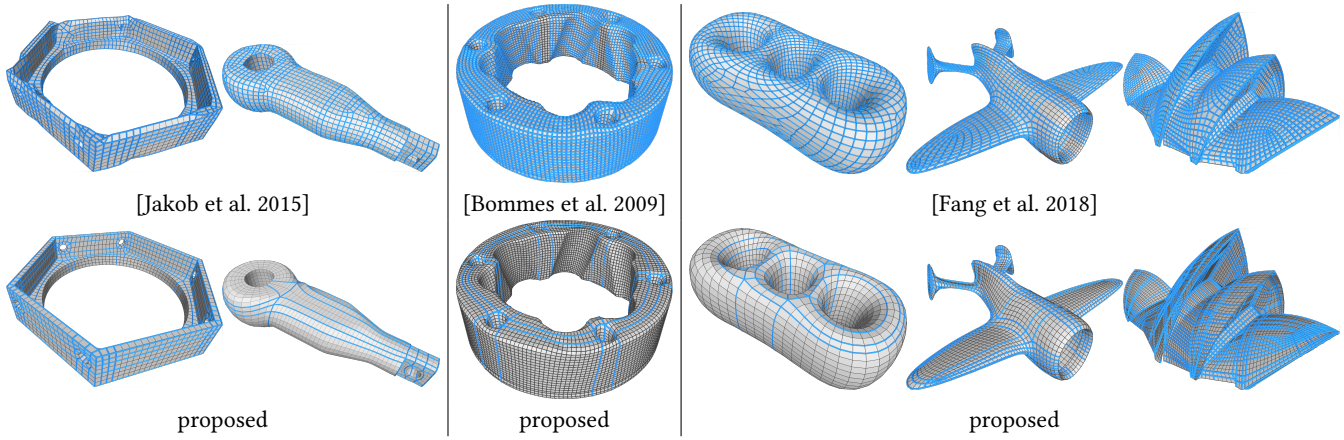


Fig. 25. Because it strives to align close irregular vertices inside patches, our method tends to produce meshes in which the graph of separatrices emanated from irregular vertices (shown in blue) is simpler, compared to competing methods, with fewer edges traversed by separatrices.

A potential weakness of our method is that choosing the internal quadrangulation of patches based only on the boundary of the patch can in theory result in loss of edge isometry. Note that this problem cannot arise when the interior of the patch presents significant geometric features, that would induce additional field singularities, and thus trigger a split of the patch. Our experiments show that this phenomenon is rare, but also that it results in substantially shorter edges when it occurs (see Figure 22.b). Settling this question requires more investigation.

Alignment of irregular vertices . Our method aligns close irregular vertices to each other, when this is compatible with the other objectives. As a result, the graph of separatrices constructed from our meshes tends to be simpler than the one produced by competing semi-regular quad-remeshing methods (Figure 25), but not, in many cases, to the point of resulting in a coarse-quad layout. This calls for more aggressive strategies to enforce alignment; in alternative, separatrix-graph simplification methods such as [Bommes et al. 2011; Tarini et al. 2011] could be employed as a post-processing step.

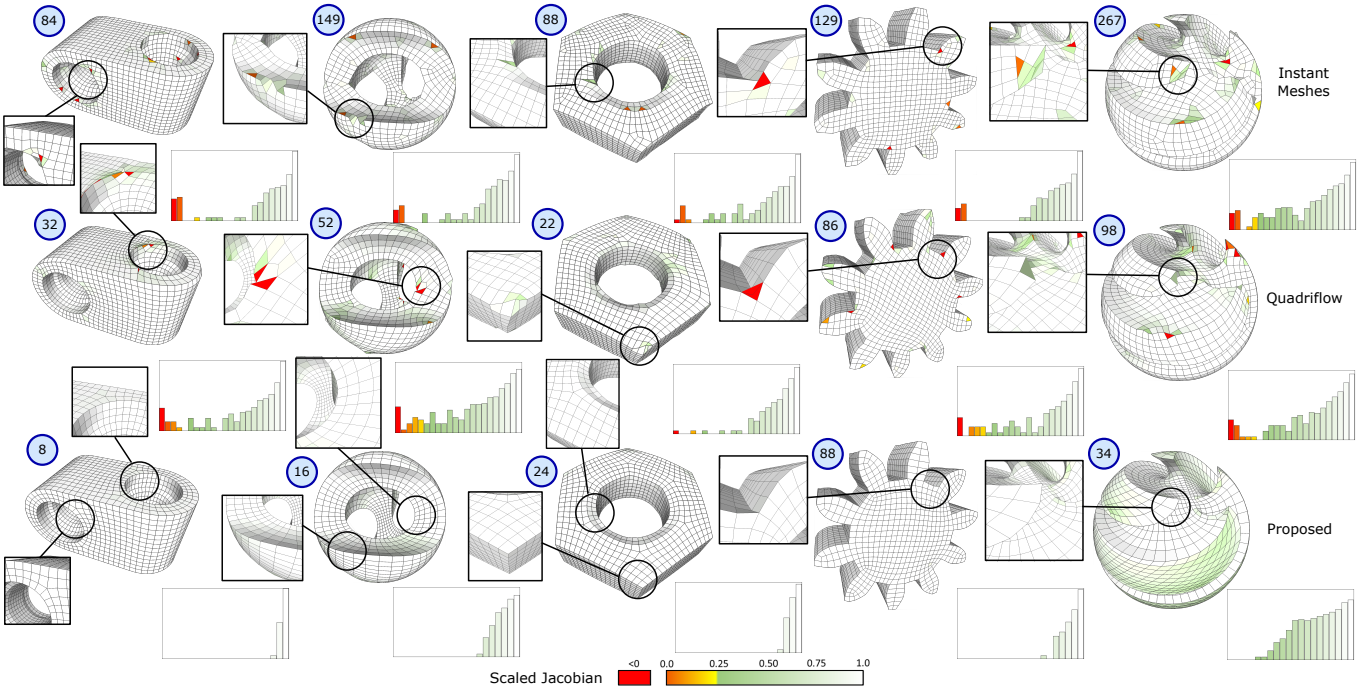


Fig. 26. A comparison between Instant Meshing [Jakob et al. 2015] (top), Quadriflow [Huang et al. 2018] (middle) and the proposed method (bottom). The faces are color-coded by shape quality, measured as their Scaled Jacobian [Stimpson et al. 2007]. In each model, we report the number of singularities in the light blue circle, and the histogram of quality, in logarithmic scale – negative values, corresponding to concave faces, are represented in the red bin).

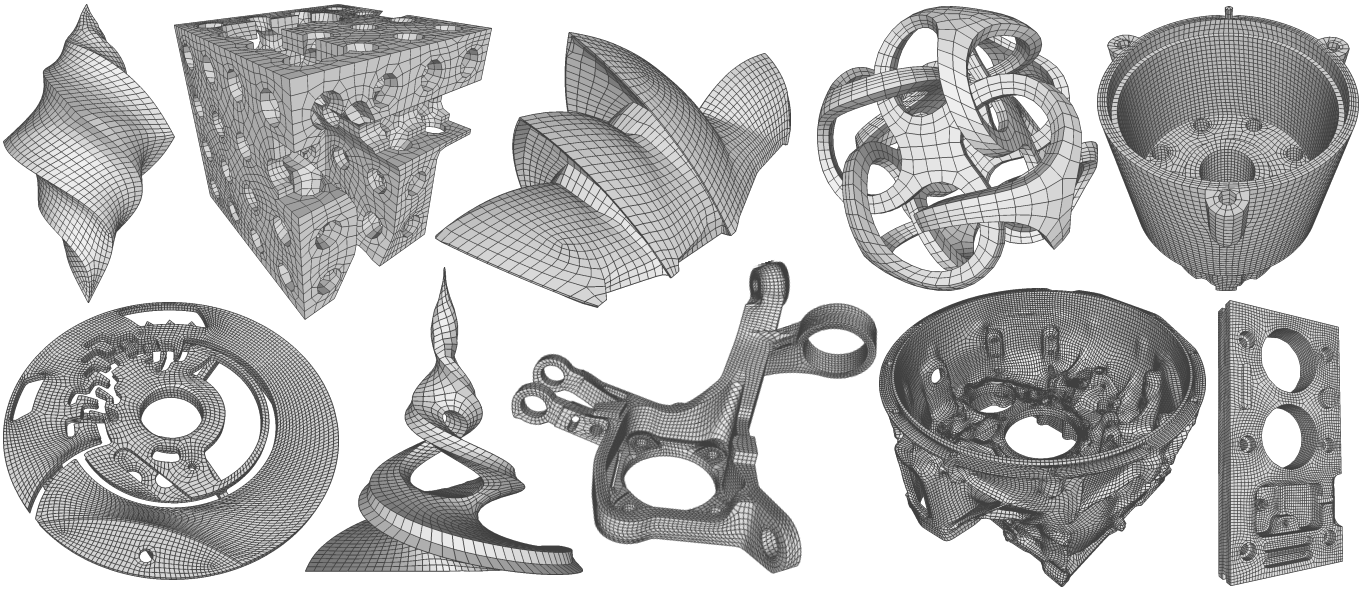


Fig. 27. Examples of quadrangulation of mechanical models with complex, challenging shapes.

Symmetries. A visual (qualitative) inspection of our results reveals in many cases a quality reminiscent of what is typically associated with manual modeling by digital artists. One notable remaining problem, however, is that symmetries are not always respected, including the recurring case where similar features in an object (e.g. in a decorative pattern) are quadrangulated slightly differently.

Fig. 28. Examples of quadrangulation of organic-looking model. Even if our method is designed around the need of preserving feature-lines, it also works well on organic smooth mesh with no feature-lines.

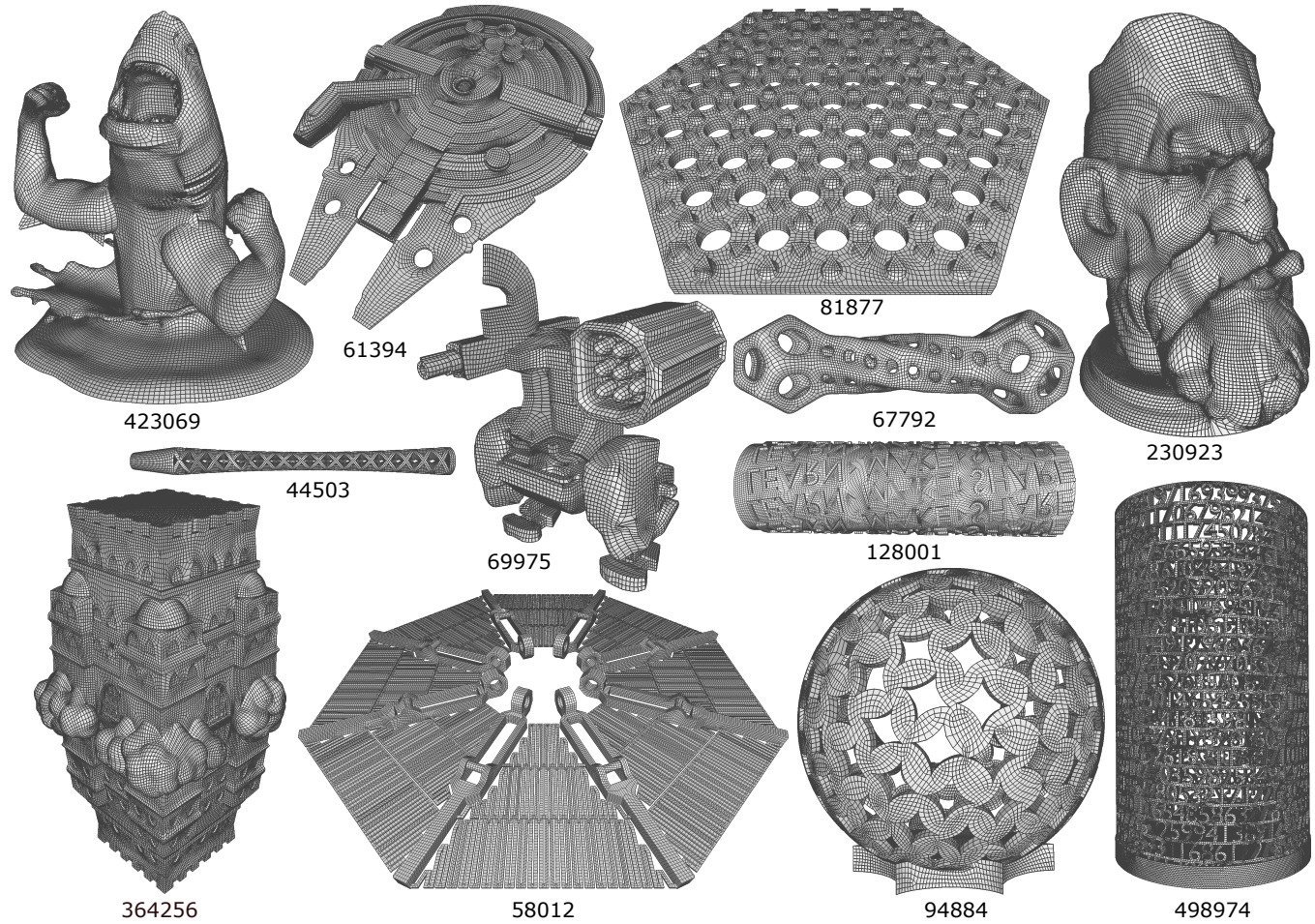


Fig. 29. Examples of hi-res quadrangulated model from the Thingi10k repository.

ACKNOWLEDGMENTS

The authors thank Luigi Malomo for helping with triangular remeshing. The models are courtesy of the AIM@SHAPE Shape Repository, Stanford 3D Scanning Repository and Thingi10K Repository.

REFERENCES

David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt. 2013a. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph* 32, 4 (2013), 98:1–98:12.

David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Comput. Graph. Forum* 30, 2 (2011), 375–384.

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Cláudio T. Silva, Marco Tarini, and Denis Zorin. 2013b. Quad-Mesh Generation and Processing: A Survey. *Comput. Graph. Forum* 32, 6 (2013), 51–76.

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

Marcel Campen. 2017. Partitioning Surfaces Into Quadrilateral Patches: A Survey. *Comput. Graph. Forum* 36, 8 (2017), 567–588.

Marcel Campen, David Bommes, and Leif Kobbelt. 2012. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph.* 31, 4 (2012), 110:1–110:11.

Marcel Campen, David Bommes, and Leif Kobbelt. 2015. Quantized global parametrization. *ACM Trans. Graph* 34, 6 (2015), 192:1–192:12.

Marcel Campen, Martin Heistermann, and Leif Kobbelt. 2013. Practical Anisotropic Geodesy. *Comput. Graph. Forum* 32, 5 (2013), 63–71.

Paolo Cignoni, Guido Ranzuglia, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Marco Di Benedetto, Fabio Ganovelli, Giorgio Marcias, Gianpaolo Palma, Nico Pietroni, Federico Ponchio, Luigi Malomo, Marco Tarini, and Roberto Scopigno. 2013. MeshLab: an Open-Source Mesh Processing Tool. http://www.meshlab.net.

CNR. 2013. The Visualization and Computer Graphics Library. http://vcg.isti.cnr.it/vcglib/.

Massimiliano Corsini, Paolo Cignoni, and Roberto Scopigno. 2012. Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes. *IEEE Trans. Vis. Comput. Graph* 18, 6 (2012), 914–924. http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.34

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Designing N-PolyVector Fields with Complex Polynomials. *Comput. Graph. Forum* 33, 5 (2014), 1–11.

Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Integrable PolyVector fields. *ACM Trans. Graph* 34, 4 (2015), 38:1–38:12.

Xianzhong Fang, Hujun Bao, Yiying Tong, Mathieu Desbrun, and Jin Huang. 2018. Quadrangulation through Morse-Parameterization Hybridization. *ACM Trans. Graph.* 37, 4, Article 92 (July 2018), 15 pages. https://doi.org/10.1145/3197517.3201354

LLC Gurobi Optimization. 2018. Gurobi Optimizer Reference Manual. http://www.gurobi.com

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh Optimization. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques* (Anaheim, CA) *(SIGGRAPH '93)*. Association for Computing Machinery, New York, NY, USA, 19–26. https://doi.org/10.1145/166117.166119

Jingwei Huang, Yichao Zhou, Matthias Niessner, Jonathan Richard Shewchuk, and Leonidas J. Guibas. 2018. QuadriFlow: A Scalable and Robust Method for Quadrangulation. *Computer Graphics Forum* 37, 5 (2018), 147–160. https://doi.org/10.1111/cgf.13498 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13498

Alec Jacobson, Daniele Panozzo, et al. 2013. libigl: A simple C++ geometry processing library. http://igl.ethz.ch/projects/libigl/.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Instant field-aligned meshes. *ACM Trans. Graph.* 34, 6 (2015), 189:1–189:15.

Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover - Surface Parameterization using Branched Coverings. *Comput. Graph. Forum* 26, 3 (2007), 375–384.

Marco Livesu, Nico Pietroni, Enrico Puppo, Alla Sheffer, and Paolo Cignoni. 2020. LoopyCuts: practical feature-preserving block decomposition for strongly hex-dominant meshing. *ACM Trans. Graph* 39, 4 (2020), 121.

Albert Matveev, Alexey Artemov, Ruslan Rakhimov, Gleb Bobrovskikh, Daniele Panozzo, Denis Zorin, and Evgeny Burnaev. 2020. DEF: Deep Estimation of Sharp Geometric Features in 3D Shapes. arXiv:2011.15081 [cs.CV]

Alessandro Muntoni and Stefano Nuvoli. 2021. CG3Lib: A C++ geometry processing library. https://doi.org/10.5281/zenodo.4431777

Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust field-aligned global parametrization. *ACM Trans. Graph.* 33, 4 (2014), 135:1–135:14.

Ashish Myles and Denis Zorin. 2013. Controlled-distortion constrained global parametrization. *ACM Trans. Graph.* 32, 4, Article 105 (July 2013), 14 pages. https://doi.org/10.1145/2461912.2461970

Stefano Nuvoli, Alex Hernandez, Claudio Esperança, Riccardo Scateni, Paolo Cignoni, and Nico Pietroni. 2019. QuadMixer: layout preserving blending of quadrilateral meshes. *ACM Trans. Graph* 38, 6 (2019), 180:1–180:13.

Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. 2012. Fields on symmetric surfaces. *ACM Trans. Graph* 31, 4 (2012), 111:1–111:12.

Daniele Panozzo, Enrico Puppo, Marco Tarini, Nico Pietroni, and Paolo Cignoni. 2011. Automatic Construction of Quad-Based Subdivision Surfaces using Fitmaps. *IEEE Transaction on Visualization and Computer Graphics* 17, 10 (october 2011), 1510–1520. https://doi.org/10.1109/TVCG.2011.28

Nico Pietroni, Enrico Puppo, Giorgio Marcias, Roberto Scopigno, and Paolo Cignoni. 2016. Tracing Field-Coherent Quad Layouts. *Comput. Graph. Forum* 35, 7 (2016), 485–496.

Nico Pietroni, Davide Tonelli, Enrico Puppo, Maurizio Froli, Roberto Scopigno, and Paolo Cignoni. 2015. Statics Aware Grid Shells. *Comput. Graph. Forum* 34, 2 (2015), 627–641.

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

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

Faniry H. Razafindrazaka, Ulrich Reitebuch, and Konrad Polthier. 2015. Perfect Matching Quad Layouts for Manifold Meshes. *Comput. Graph. Forum* 34, 5 (2015), 219–228.

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 (July 2018), 16 pages. https://doi.org/10.1145/3197517.3201389

CJ Stimpson, CD Ernst, P Knupp, PP Pébay, and D Thompson. 2007. The Verdict library reference manual.

Kenshi Takayama, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Pattern-Based Quadrangulation for N-Sided Patches. *Comput. Graph. Forum* 33, 5 (2014), 177–184.

Marco Tarini. 2021. Closed-form Quadrangulation of N-Sided Patches. arXiv:2101.11569 [cs.GR]

Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. 2010. Practical quad mesh simplification. *Comput. Graph. Forum* 29, 2 (2010), 407–418.

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 (2011), 142:1–142:12.

Francesco Usai, Marco Livesu, Enrico Puppo, Marco Tarini, and Riccardo Scateni. 2016. Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Trans. Graph.* 35, 1, Article 6 (Dec. 2016), 13 pages. https://doi.org/10.1145/2809785

Amir Vaxman, Marcel Campen, Olga Diamanti, David Bommes, Klaus Hildebrandt, Mirela Ben-Chen Technion, and Daniele Panozzo. 2017. Directional Field Synthesis, Design, and Processing. In *ACM SIGGRAPH 2017 Courses* (Los Angeles, California) *(SIGGRAPH '17)*. Association for Computing Machinery, New York, NY, USA, Article 12, 30 pages. https://doi.org/10.1145/3084873.3084921

Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. 2020. Octahedral Frames for Feature-Aligned Cross Fields. *ACM Trans. Graph.* 39, 3 (2020), 25:1–25:13. https://doi.org/10.1145/3374209

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:1605.04797 [cs.GR]