QuadMixer: Layout Preserving Blending of Quadrilateral Meshes

STEFANO NUVOLI, University of Cagliari, Italy
ALEX HERNANDEZ, Federal University of Rio de Janeiro, Brazil
CLAUDIO ESPERANÇA, Federal University of Rio de Janeiro, Brazil
RICCARDO SCATENI, University of Cagliari, Italy
PAOLO CIGNONI, CNR of Italy, Italy
NICO PIETRONI, University of Technology Sydney, Australia

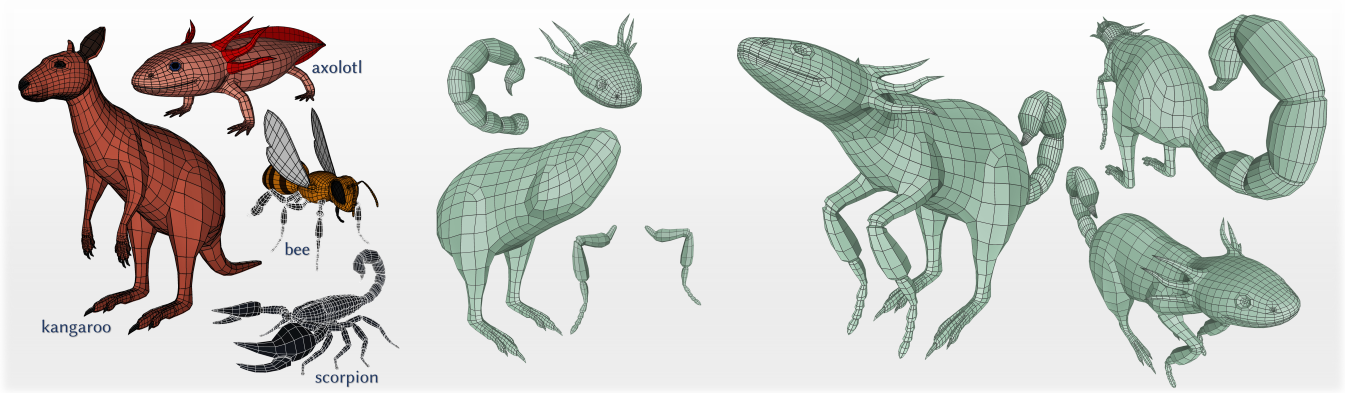


Fig. 1. With the proposed blending technique we can assemble pieces of different animals respecting the original quad meshing. In less than ten minutes we detached and combined these body pieces to automatically obtain the pure quad mesh shown on the right.

We propose QuadMixer, a novel interactive technique to compose quad mesh components preserving the majority of the original layouts. Quad Layout is a crucial property for many applications since it conveys important information that would otherwise be destroyed by techniques that aim only at preserving shape.

Our technique keeps untouched all the quads in the patches which are not involved in the blending. We first perform robust boolean operations on the corresponding triangle meshes. Then we use this result to identify and build new surface patches for small regions neighboring the intersection curves. These blending patches are carefully quadrangulated respecting boundary constraints and stitched back to the untouched parts of the original models. The resulting mesh preserves the designed edge flow that, by construction, is captured and incorporated to the new quads as much as possible. We present our technique in an interactive tool to show its usability and robustness.

CCS Concepts: • **Computing methodologies** → **Mesh models**.

Additional Key Words and Phrases: Mesh Modelling, Quadrangulation, Retopology

Authors' addresses: Stefano Nuvoli, Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy, s.nuvoli@studenti.unica.it; Alex Hernandez, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, ahernandezm@cos.ufrj.br; Claudio Esperança, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, esperanc@cos.ufrj.br; Riccardo Scateni, Dept. of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy, riccardo@unica.it; Paolo Cignoni, CNR of Italy, Pisa, Italy, paolo.cignoni@isti.cnr.it; Nico Pietroni, University of Technology Sydney, Sydney, Australia, nico.pietroni@uts.edu.au.

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, https://doi.org/10.1145/3355089.3356542.

ACM Reference Format:
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, Article 180 (November 2019), 13 pages. https://doi.org/10.1145/3355089.3356542

1 INTRODUCTION

The generation of high-quality 3D assets is a significant part of the production pipeline in the entertainment industry. Modeling complex shapes from scratch requires highly skilled artists with extensive professional training at a considerable cost. Moreover, these efforts are, in many cases, not exploitable multiple times. While for architectural and mechanical shapes, the high standardization of the basic elements allows the reuse of components, the creation of organic models with less structured shape often starts from scratch.

Various combining techniques since [Funkhouser et al. 2004] have been proposed to overcome this problem. Such approaches are quite intuitive and suitable for novice users. Their primary purpose is to directly combine parts from existing models to synthesize new models by allowing rapid assembling of complex 3D models from arbitrary input meshes.

This process of shape composition has recently gained much interest. These methods allow for cutting arbitrary surface parts from one model and automatically stitching them to existing holes into another [Schmidt and Singh 2010; Sharf et al. 2006]. Detail transfer techniques are used to copy high-frequency details [Biermann et al. 2002].

Recently, many efforts have concentrated on the other important task of suggesting or choosing what parts to combine. Modern techniques provide fully automated frameworks to indicate the widest choice of possible results. However, while this field of modeling-by-composition is quite active and it is generating promising results, all the proposed solutions cannot produce a fully quadrangulated mesh. Their output is always limited to triangulated surfaces.

In the industry, coarse quad layouts are often manually created by professional designers. They employ their semantic knowledge and experience to adjust the layout in the context of the particular application needs. Typical modeling systems used in the industry [Autodesk 2018; Pilgway 2017; Pixologic 1999] allows the user to draw vertices and edges on a surface manually. Since this manual procedure is time-consuming and error-prone, a series of sketch-based retopology approaches [Campen and Kobbelt 2014; Marcias et al. 2015; Takayama et al. 2014] have been proposed. These semi-automatic approaches automate a large part of the process while allowing the user to efficiently modify the topology of the layouts without having to start from scratch.

While automatic quadrangulation techniques can generate excellent results, the manual design of the quad flow over the meshes is still considered by content creators as part of the artistic process. For this reason, the preservation of the original quad meshes during the composition process is crucial to allow the effective use of modeling-by-composition in the field of quad meshes.

QuadMixer is a novel technique to compose quad meshes. Our modeling principles take inspiration from the classical boolean operations defined on triangle meshes, but with operators redesigned to work on quadrilateral meshes. *QuadMixer* mimics all the conventional boolean operations that are available for triangle meshes such as union, intersection, and difference. While it is generally a challenging task to define stable boolean operations, their result can be defined precisely in the context of triangle meshes [Zhou et al. 2016]. In contrast, this is not true for generic quad meshes. Concerning boolean operations, a first evident difference between a quad mesh and a triangle mesh is that the former does not admit a unique piecewise discretization. Hence, the single intersection between two quadrilateral elements cannot be unequivocally defined. In this light, we might refer to our operations as *blending* to distinguish them from classical boolean operations on triangle meshes.

We summarize the main contributions of this paper as follows:

- We propose a new technique to mimic boolean operations on quad meshes. Our system blends quad meshes preserving, as much as possible, the original quadrangulation.
- We define a new technique to robustly define a region of interface/blending between two intersecting surfaces whose boundary can be quadrangulated.
- We defined a strategy to ensure that the intersection between two quadrangulated models admits a valid quadrangulation.
- We integrated our technique in an interactive tool and demonstrated its practical use on modeling scenarios.

2 RELATED WORK

Robustly generating functional quad layouts and quad meshes is a well-studied research field. Extensive surveys on this topic are available [Bommes et al. 2013; Campen 2017a] as well as shorter and more introductory papers [Campen 2017b]. In the following, we will concentrate the discussion on the most closely related topics: the user-assisted generation of quad meshes and the meshing of polygonal patches.

Interactive Quadrangulation Tools. Many methods have been proposed to automatically design coarse quad layouts [Bommes et al. 2011, 2013; Campen et al. 2012; Tarini et al. 2011]. Most of these try to cope with specific needs in the production. For instance, [Marcias et al. 2013; Zhou et al. 2018] considers the deformation affecting a mesh during an animated sequence to generate a quad layout that remains good for all animation frames. While these automatic approaches are successful on several quantitative metrics (like singularity placement and coarseness), in production pipelines there is the need for a more art-controlled quad generation process. Due to the global nature of the problem, small changes in the user-provided initial constraints may completely alter the generated quad mesh. The combination of this global behavior with the non-interactive nature of these automatic approaches makes the tuning of parameters an unintuitive and time-consuming task.

Therefore, many approaches have considered the issue of helping this manual quadrangulation process leaving most of the control to the final user. Bessmeltsev et al. [Bessmeltsev et al. 2012] developed a technique for generating quad-dominant meshes starting from an input 3D curve network sketched by the user; with this approach, geometry and topology are defined based on regions identified by closed 3D paths.

Inspired quadrangulation [Tierny et al. 2011] can also be considered related to our approach. In this work, the authors transfer quadrangulations between surfaces on a per-partition basis (e.g., head, arm, torso) via cross-parameterization. Unfortunately, this approach does not provide precise local control over the mesh layout. Instead, our method enables the direct combination of portions of quad meshes.

Finally, connectivity editing operations have been developed to enable users to modify existing quad meshes by moving pairs of irregular vertices [Peng et al. 2011]. These methods provide lower-level local operators, and they can be integrated with practices of the previous class to fine-tune the mesh topology.

Quad-Meshing Patches. In our pipeline, we face the issue of completing a partial quad mesh. This task is an essential part of all sketch-based retopology techniques [Peng et al. 2014; Takayama et al. 2013, 2014]. In these semi-automatic approaches, a patch layout is first interactively sketched over the input surface. Then each patch side is subdivided into many edges as prescribed by the user [Nasri et al. 2009; Schaefer et al. 2004; Yasseen et al. 2013] and finally automatically quadrangulated. The present work also requires quadrangulating patches defined by their sides, which can be done using filling patterns generated procedurally as in [Peng et al. 2014]. In [Takayama et al. 2013, 2014], a set of manually designed patterns are expanded to tessellate arbitrary polygons with up to 6 sides.

More recently Marcias et al.[Marcias et al. 2015] proposed another approach for filling with quad patches a 2D $n-$sided patch by using a pattern-based algorithm that uses a trained database of quadrangular patches. As explained in Section 2, we expect to produce triangular patches that are relatively small with respect to the input meshes, and we propose using Takayama's method [Takayama et al. 2014]. While Marcias' method is generally better for controlling the edge flow, in our case, the edge flow is given by exploiting the existing field around the boundary of the patches.

Boolean and Composing Operations. As explained in the introduction, our approach leverages on the works on the modeling-by-composition paradigm [Schmidt and Singh 2010; Sharf et al. 2006; Zhang et al. 2010]. One fundamental step of these approaches is the detection and computation of the intersection between surfaces. A vast literature exists on how to solve this problem robustly and efficiently, and we exploited the results of [Jacobson et al. 2013a] for triangulated meshes. Moreover, the idea of trying to limit the modification on a mesh when performing boolean operations [Pavic et al. 2010], or repair the result [Bischoff and Kobbelt 2005], has been already explored. The approach of [Campen and Kobbelt 2010] can also perform boolean operations on generic input polygonal meshes. However, no solutions can smoothly composite quad meshes as the proposed approach. Given a pair of two-manifold watertight meshes composed only of quadrilateral elements, our method *blends* them into a new, closed, two-manifold, pure quadrilateral mesh.

As opposed to triangle meshes, performing blending on quadrilateral meshes poses extra challenges: while splitting triangles on a local basis will always result in a valid triangle mesh, quad meshes must be manipulated taking into account their entire connectivity. Indeed, the majority of applications, such as subdivision surfaces or finite element analysis, require the flow of the edges to be aligned with geometric features, imposing several conditions on the global structure and the placement of irregular vertices. Relying solely on local modifications [Tarini et al. 2010] cannot, in general, enforce such global characteristics.

A simple way to tackle this problem might be to transform the quad mesh into triangles first, perform the boolean operation, and finally use a quad-meshing algorithm based on global parametrization [Bommes et al. 2009; Jakob et al. 2015] or cross-field tracing [Myles et al. 2014; Pietroni et al. 2016] to obtain a sound quadrilateral mesh. However, this approach will inevitably modify the entire mesh. This result is far from ideal from a modeling point of view, as an artist would more likely prefer to preserve as much as possible the connectivity he has designed. As illustrated in Figure 2, a complete quadrilateral re-meshing using the approach of [Jakob et al. 2015] on the result of the boolean operation will inevitably cause the loss of most of the features designed by the artist, like the eyes of the pig or the armadillo.

Among commercial software packages, to the best of our knowledge, only the Modo suite [Visionmongers 2018] provides a tool, called *MeshFusion*, that can combine quad-based mesh representations with boolean operations while partially preserving their structure. Here, the user authors a tree of boolean operations with coarse quad meshes as the leaves; during editing, the system silently computes and displays a low-level representation, consisting of

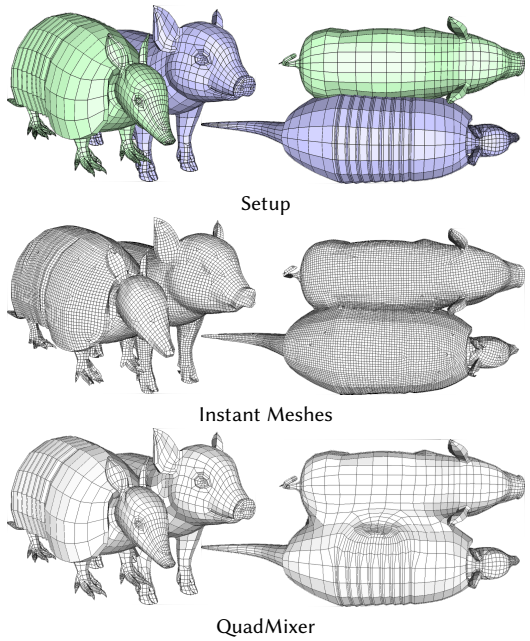


Fig. 2. Current solutions [Jakob et al. 2015] require to re-mesh entirely the results of the boolean operation and original connectivity is lost. Our system efficiently preserves the connectivity and is capable of blending two different connectivities

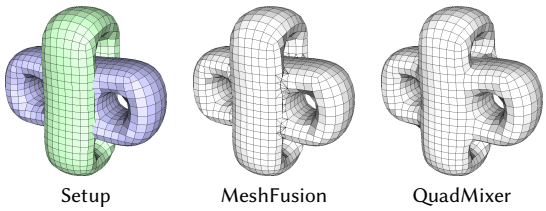


Fig. 3. A comparison of QuadMixer and MeshFusion ([Visionmongers 2018]): the difference in the intersection portion of the mesh is noticeable.

a quad-dominant mesh, obtained by subdividing the coarse quad representations, and performing the boolean operation over the subdivided results (considered as triangular meshes in the intersections). Therefore, differently from our case, a quad-pure representation of the result is never explicitly computed, and the results include triangulated regions around the intersection lines. Figure 3 shows a comparison between the quad-dominant representation obtained by MeshFusion and the result of our method.

A fundamental task is to preserve as much as possible the connectivity of the original quadrangulated models and change the elements only in regions modified by the boolean operation. Figure 4 shows an overview of our entire pipeline.

- We first compute a patch decomposition of the quadrilateral meshes by using a simple *motorcycle graph* [Eppstein et al. 2008] tracing algorithm (see Figure 4.a) or solely emanating separatrices from irregular vertices.
- We split each quad into two triangles, and we perform the

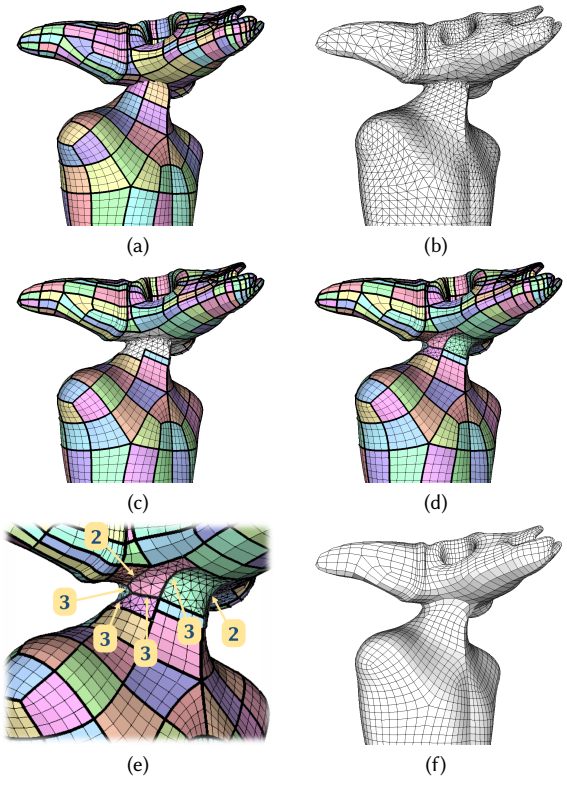


Fig. 4. An overview of our processing pipeline: (a) Given two separate quadrilateral meshes we compute an initial patch layout for each; (b) then quads are split to form triangular meshes, which are then combined. (c) We update the patch layout for the patches that are affected by the boolean operation. (d) We split the triangulated portion of the surface into sub patches. (e) We derive the optimal subdivision for each side. (f) We perform the final quadrangulation.

boolean operation using the implementation of [Zhou et al. 2016] (see Figure 4.b).

- We select the patches that have not been modified by the boolean operation. We retract the sides of the patches that are partially affected by the boolean operations. Those patches are the ones that contain only a subset of the original set of quads (see Figure 4.c). At this stage the mesh can be divided into two sets: a quadrilateral mesh Q^0 and a triangle mesh $\mathcal{T}$ (see Figure 4.c) which share a common boundary.
- We smooth these internal patches to generate a fair geometry surface, more straightforward to be nicely quadrangulated. The user can control the amount of introduced smoothing.
- We trace a set of internal patches $\mathcal{P}$ on the triangulated mesh $\mathcal{T}$ (see Figure 4.d). This step uses the definition of a cross-field [Diamanti et al. 2014] and applying a tracing algorithm [Campen et al. 2012].
- We solve an Integer Quadratic Program with linear constraints to derive the optimal subdivision for each side of the patches $\mathcal{P}$. The energy formulation balances regularity of the patches (to avoid inserting unnecessary irregular vertices) with the global uniformity of edge sizes (see Figure 4.e). The number of subdivisions along the border sides of $\mathcal{P}$ are constrained to match the corresponding subdivisions on Q^0.
- We quadrangulate each patch using the data-driven approach proposed in [Takayama et al. 2014] (see Figure 4.f) obtaining a new quadrangulated mesh Q^1. The union of Q^1 and Q^0 provides the final result.

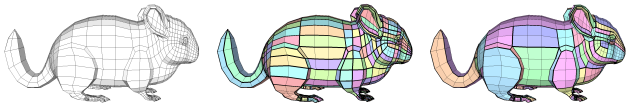


Fig. 5. On the left: the input quad layout; in the center: the patch layout with separatrices; on the right: the patch layout typical of motorcycle graph.

3 OUR METHOD

The input to our method consists of pure quadrilateral meshes. They can be the result of automatic methods or, more usually, models produced by a digital artist. The first step of our pipeline extracts a quad patch layout. This step is not difficult and consists of tracing all the separatrices stemming from irregular vertices (see Figure 5). Each separatrix is defined as a sequence of edges starting at a singular vertex (i.e., a vertex of valence different from four) and ending at another singular one. When applied to manually-modeled meshes, this process typically produces well-structured and compact patch decompositions, since the tools used by the artists tend to align the singularities naturally. As an alternative, we can contemporanily propagate all the separatrices and stop tracing each separatrix as soon as it crosses another one. Literature usually describes this procedure as tracing *motorcycle graphs* [Eppstein et al. 2008]. While this tends to create fewer patches, it can easily introduce t-junctions in the patch layout. We do not need to make any particular assumption on the structure or the alignment of the separatrices. Hence, both approaches are valid as they produce quadrilateral patches. Any algorithm capable of improving the regularity of the patch layout (such as [Bommes et al. 2011; Tarini et al. 2011]) is not useful in this context since it might modify the original edge flow designed by the artist.

3.1 Optimal Patch Retraction

Given two pure quadrilateral meshes Q^A and Q^B, along with their original quad patch layouts (Figure 6.a), we start by splitting each quad element along its smaller diagonal to transform Q^A and Q^B into two triangle meshes $\mathcal{T}^A$ and $\mathcal{T}^B$. Next, we perform the boolean operation between $\mathcal{T}^A$ and $\mathcal{T}^B$ using [Zhou et al. 2016]. The result is the new triangle mesh $\mathcal{T}^{\text{bool}}$. Notice that the triangles in $\mathcal{T}^{bool}$ are of two kinds: (i) the triangles from the input quads; (ii) the triangles modeling the intersection region between the two meshes. The exact implementation of the boolean operations guarantees that the vertices of the new triangles lie on the input mesh.

We now need to rebuild the quad patch layout. We start by preserving the quads of Q^A and Q^B that do not contain triangles not changed in the boolean operation (Figure 6.b). Then we consider, as

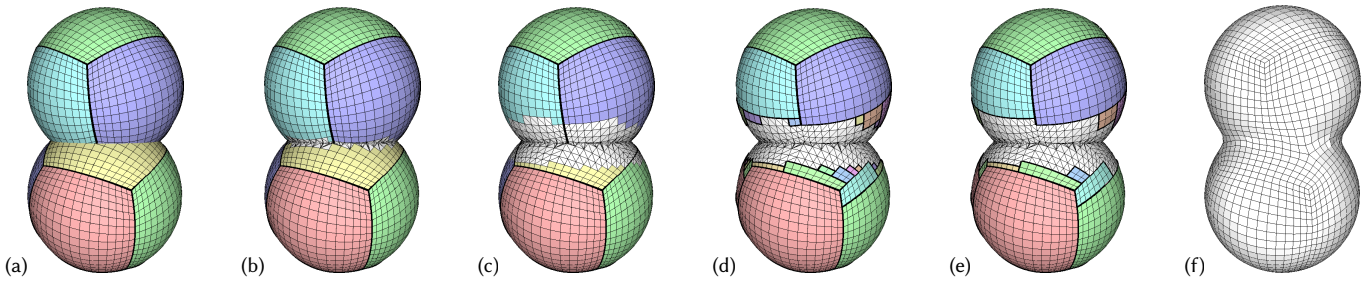


Fig. 6. Patch layout retraction: (a) The initial patch layout of the two meshes. (b) The two quad meshes are split into triangle meshes to perform the boolean operation, and then the triangles are clustered to recompose the original quads when possible. (c) Original patches are retracted to maintain a certain geodesic distance from intersection line. (d) The new patches are extracted by repeatedly finding the largest rectangles composed of quads in the partially preserved patches. (e) Small patches are pruned. (f) The final quadrangulation.

part of a blending area that will be remeshed, also the quads that are close to the intersection curve to provide sufficient space to blend between quadrilateral layouts smoothly. For this area, we consider the quads on each side where their geodesic distance to the intersection line is below a given threshold of δ_r which is proportional to the average edge of the retracted patches (Figure 6.c). At this point, we have a collection of un-organized quads that we need to assemble into patches by building a new layout. The idea is to prefer the formation of large, compact rectangular patches with a regular and straight boundary with the remaining triangulated surface. For this purpose, we repeatedly search, in the set of un-organized quads, the largest inscribed rectangle composed only by quads not associated with any entirely preserved patch (Figure 6.d). Specifically we use the *largest rectangle in a histogram* (see [Morgan 1994], chapter 21) algorithm to generate this new set of rectangular patches.

Finally, we perform a pruning step that eliminates all the newly created patches having a number of forming quads below a given threshold (Figure 6.e). We set this threshold as a fraction of the average area of the current patches.

At the end of this step, we obtain a new quad-only mesh Q^0 and a triangulated surface $\mathcal{T}^0$. Notice that because of the robust and precise implementation of the boolean operations, the triangle and the quad meshes will necessarily share the same set of boundary edges, that is, the boundary of Q^0 coincides precisely with the boundary of $\mathcal{T}^0$.

3.2 Patch Subdivision

We now need to transform the triangulated portion of the surface $\mathcal{T}^0$ into a quad mesh Q^1 that, once attached to the preserved quadrangulation Q^0, will become the final quad mesh of the mixed shape. To be able to join the two portions correctly, we must match, for each portion of the boundary of Q^1, the number of subdivisions of Q^0 along the common border. Since the number of edges along the boundaries is unchangeable, the problem becomes untractable with methods that derive quadrangulations from field-aligned global parameterizations [Bommes et al. 2009]. To the best of our knowledge, none of these methods can guarantee to produce valid quadrangulations for an arbitrary subdivision of the boundaries.

Hence, we rely on procedural methods that are explicitly designed to produce valid quadrangulations for a given input boundary subdivision. These methods automatically insert singularities in the interior of the patch to accommodate for the changes in the resolution needed to match the prescribed boundary subdivisions. However, these methods work only on input patches homeomorphic to a disk, and with a given maximum number of sides. In particular, the method by Takayama et al. [Takayama et al. 2014] requires the number of sides of the input patch to be between three and six. As a consequence, to use this method in our pipeline, we need to split the triangulated surface $\mathcal{T}^0$ into patches respecting these requirements.

3.2.1 Field-aligned Patch Tracing. We produce a valid decomposition via a two-step process: (i) we first derive a smooth cross-field; (ii) we iteratively trace polylines along the cross-field to create a proper patch decomposition. Every trace starts from a border vertex and ends on another border vertex, following the flow of the underlying field. To ensure that the patch layout will blend smoothly with the existing quadrangulation Q^0 at the border, we align the cross-field along the boundaries, and we smooth it in the interior. We can also include in the field computation any additional conditions on prescribed curvature directions. However, given the small areas covered by the boundaries, these conditions are usually not taken into account.

The following are the steps of the subdivision pipeline:

- We perform a re-meshing of the initial surface $\mathcal{T}^0$ keeping fixed all triangles that have an edge on the boundary. We use this step to remove badly shaped triangles appearing along the intersection lines, blending the tessellation with the boundary constraints. This pre-processing step increases the robustness of the overall approach. We use the iterative approach included in the Meshlab framework [Cignoni et al. 2008].
- We compute a smooth cross field that conforms to the boundary using the poly-vector field smoothing [Diamanti et al. 2014]. The cross-field is computed for each face and then re-interpolated for each vertex considering the invariance to $\frac{\pi}{2}$ rotations.
- We split all concave corners by tracing polylines with an end-point in the corner using [Campen et al. 2012].
- We iteratively re-apply this step for any sub-patch not yet

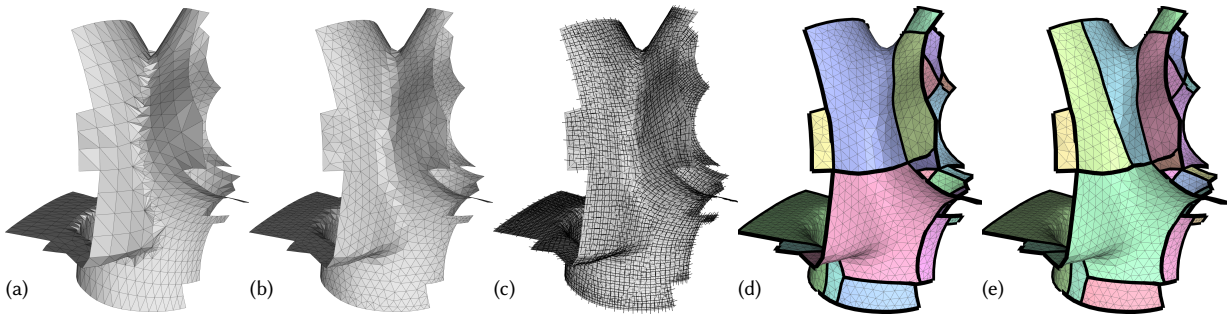


Fig. 7. Patch tracing procedure: (a) The Initial patch; (b) The re-meshing step; (c) Cross field computation; (d) Concave corner tracing; (e) Final splitting of the patches to match the requirements. This patch is the one shown in the accompanying video when displaying the two intersecting fertility models.

respecting the conditions imposed by the constrained quadrangulation algorithm.

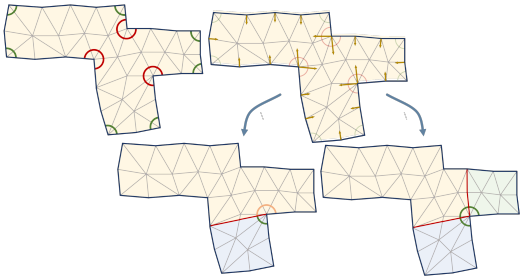


Fig. 8. On the top left the initial corner classification: we mark each angle either as convex or as concave; on the top right we show the directions stemming from concave vertices that we use for splitting the patch in quads; on the bottom we show how we can split the concave corner into a flat and a convex corner using a single trace (left), or into three convex corners using two traces at the same time.

Patch configuration. After computing a cross-field on the triangle mesh $\mathcal{T}^0$ (for a more exhaustive description of cross-fields see [Vaxman et al. 2017]), we classify each border vertex of the triangle mesh $\mathcal{T}^0$ as *convex* (if less than $\pi - \pi/8$), *concave* (if greater than $\pi + \pi/8$), or *flat* elsewhere. The classification of a vertex v_i (see Figure 7 for an example) is based upon the angle between the two edges of $\mathcal{T}^0$ incident on v_i. We iteratively trace polylines until each patch has a number of sides between three and six.

Each vertex has a different number of possible directions for tracing the splitting pipelines (yellow arrows in Figure 8): the concave vertices have two tracing directions, the flat vertices have one tracing direction, and the convex vertices have no tracing directions. To reduce the number of sides in the patch, we repeatedly trace the subdividing polylines (the red lines in Figure 8) following the tracing directions. Note that each split of a patch reduces the number of sides in the two resulting parts by, at least, one.

Each split changes also the classification of the associated boundary vertex as follows (see Figure 7 for examples of this reclassification):

- A single trace splits a concave corner into a convex and a flat corner.
- Two traces stemming from the same concave corner split it into three convex corners.
- A trace splits a flat corner into two convex corners.
- Two orthogonally intersecting traces originate four convex corners.
- A trace can never split a convex corner.

Notice that the two corners resulting from a split belong to different sub-patches.

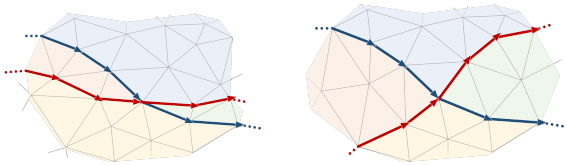
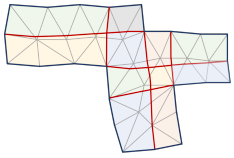


Fig. 9. Tangential (left) and orthogonal (right) path intersections.

Cross-field generation. To associate a cross-field to the triangle mesh $\mathcal{T}^0$, we use the anisotropic field tracing strategy as proposed in [Campen et al. 2012]. We recap the tracing method in Appendix A. In our experiments, we always have accurate results and, thus, we do not use more sophisticated trace strategies (e.g., the ones described in [Myles et al. 2014] or [Pietroni et al. 2016]). These methods usually require a more complex pre-processing step, and this might affect the interactivity of the modeling process negatively.

Once we have linked the cross-field to the mesh, every vertex has an associated tracing direction in the $\mathcal{M}4$ domain, as described in [Campen et al. 2012]. When two traces intersect, we classify their intersection either as *tangential* or as *orthogonal* by looking at the index of the field in $\mathcal{M}2$. We want to avoid inserting any tangential crossing in the final layout, as they tend to create poorly shaped, elongated patches. Orthogonal intersections, instead, create a well-shaped patch layout that efficiently captures the structure of the underlying field. Figure 9 shows the difference between the two types of intersections.

Concave vertex tracing. We trace the polylines that split the patches, choosing first the ones that have an end-point in a concave vertex. When



multiple alternative traces are available, we follow these selection heuristics: (i) we never add a new trace if it introduces tangential intersections; (ii) we favor traces starting from concave vertices not yet split; (ii) we prefer shorter traces to longer ones.

As we repeat the process in the sub-patches still having concave vertices, we dramatically reduce the probability of tangential collisions. At the end of the process, all concave vertices should vanish, and we have a correct convex patch layout. This condition is necessary to make the procedural quad mesh generation of [Takayama et al. 2014] to generate high-quality quadrilateral elements. Although we have no theoretical guarantee to be able to split all concave corners, we never encountered any failure case in our editing session, even for complex configurations.

Patch splitting. At this point, we must still check that each generated patch has no more than six sides, where every side of the patch consists of a sequence of edges between convex corners, and that it is homeomorphic to a disk. If we find an unsuitable patch, we first initialize a dense set of candidate traces that do not intersect tangentially. This set is obtained by tracing from all flat border vertices, and iteratively removing traces having tangential intersections, favoring the shortest ones. The result of this final step is the patch layout for $\mathcal{T}^0$.

Discussion. Another possible strategy to procedurally quadrangulate a disk-like 3D surface has been proposed by Peng et al. [Peng et al. 2014]. This approach requires first to derive a bijective parametrization of the triangular patch; then the final patch layout is produced by tracing straight lines in the bidimensional parametric space. While we share some ideas with that approach (like the classification between concave and convex corners and the emanating directions), we do not require any bijective parametrization, and this is a significant advantage. The decision to trace paths directly on the surface is crucial to make the method general and reliable. Constructing a low distortion bijective parametrization can be difficult and time-consuming for the general case and even particularly tricky for the thin regions which can result from the boolean operations. Moreover, designing a cross-field that aligns to boundary constraints leads the traced subdivision to blend the flow among the existing quadrangulations smoothly.

3.3 Subdivision Optimization

Once we derive a proper patch layout, we have to devise the optimal integer subdivision for each edge. We set up a global Integer Program with multiple quadratic objective functions, and linear constraints that contribute to obtaining a proper tessellation:

(i) A patch admits a quadrangulation only if the sum of boundary subdivisions is even [Takayama et al. 2014].
(ii) The subdivisions on the boundaries are constrained to match the preserved quadrangulated mesh.
(iii) To increase the *isometry* of the tessellation, we penalize the discrepancy of each side with respect to its ideal subdivision. We compute the ideal subdivision for each patch that has a border by averaging the edge size of adjacent quads. Then we propagate that value on internal patches, and we smooth between adjacent patches in order to discourage abrupt changes of resolution.
(iv) To increase the *regularity* of the tessellation, we favor the equality of opposite edges for every 4-sided patches.

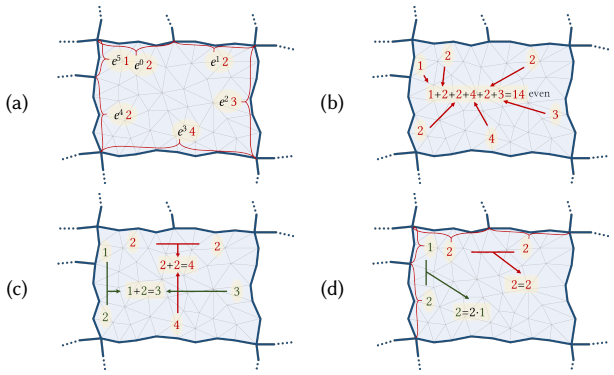


Fig. 10. An example, with actual values, of the optimization procedure inside a patch. In (a) each side is divided into sub-sides, considering all incident t-junctions of adjacent patches (numerical values in red); in (b) we show that the sum of all the edge sizes must be even; in (c) we show how regularity pushes equal subdivisions on the opposite sides of the quad patch; in (d) we show how isometry encourages each sub-side to match geometrically sound values.

Objectives (i) and (ii) are *hard constraints* that have to be satisfied to admit a valid quadrangulation, while (iii) and (iv) are energy terms that favor the formation of nicely shaped quad layouts.

We define an integer positive variable e_i for each sub-side of an edge. Because our patch layouts admit the formation of *T-junctions*, we must split each side of a patch into sub-sides by considering all of the subdivision with respect to the adjacent patches. An example of edge variable definition is shown in Figure 10.a.

We first define a least squares *isometry* energy term that penalizes the discrepancy of every e_i from its ideal size $\hat{e_i}$:

$$\begin{array}{ll} \min & \sum (e_i - \hat{e_i})^2 \\ \text{s.t.} & e_i \geq 1 \end{array} \tag{1}$$

Every subdivision e_i should be at least 1. For each sub-side on the border B, we also add a linear constraint to force its value to match the one defined by the quadrilateral mesh.

$$e_i = q_i \quad \forall e_i \in B \tag{2}$$

As previously stated, a quadrangulation is only possible if the sum of its subdivision is an even number. Hence, for each patch P_k, we include an additional linear constraint:

$$\begin{array}{l} \sum e_j = 2n \quad \forall e_j \in P_k \\ n \geq 1 \end{array} \tag{3}$$

For a quadrilateral patch, we want to favor the formation of a regular grid tessellation. Hence, for each quadrilateral patch, we add an energy term that favors the equality of opposite sides:

$$\min \quad (\sum_{e_u \in S_0} e_u - \sum_{e_w \in S_2} e_w)^2 \quad + \quad (\sum_{e_u \in S_1} e_u - \sum_{e_w \in S_3} e_w)^2 \tag{4}$$

where S_0, S_1,S_2 and S_3 are the four sides of the quadrilateral patch. We use a parameter $0 < \alpha < 1$ to blend the two energy terms expressed by equation 1 and 4. The effect of this parameter on the final quadrangulation is quite intuitive (see Figure 11). We verified in our experiments that a value of $\alpha = 0.5$ is a good compromise between regularity and isometry. For complex quad layouts, we can use norm one minimization in equation 1 to speed up the entire process, in such case, both equation 1 and 4 are modified such that the least-squares energy term is substituted with an absolute difference. This modification can speed the solving up to a factor of 10 when using $\sim$ 150 subdivision variables. While this approximation might accumulate the error on single variables (rather than distributed in as the least-squares minimization), we experimented that it works pretty well in practice.

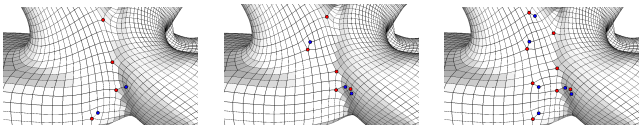


Fig. 11. The regularization term governs the distribution of singularities and the isometry of the tessellation.

3.4 On the existence of a valid solution

In order to derive a proper subdivision assignment, it is necessary that every connected component of the triangulated patch decomposition has an *even* number of subdivisions at the boundary. Indeed, in case this pre-condition is not verified, it is not possible to obtain a quadrangulation of such a patch: regardless of the internal patch subdivision, an odd subdivision at the boundary will necessarily introduce an unsolvable set of constraints.

Luckily, this happens only in particular cases. In the general case, when blending two genus-zero quadrilateral meshes, the intersection curves will always define over each mesh disk-like regions that, by construction, will be bounded by an even number of quads. However, when higher genus meshes are involved, it can happen that the intersection curves cut out, over one of the meshes, regions with a more complex topology that are bounded by multiple boundaries. In this case, it still holds that the overall sum of the subdivisions of all the boundaries is even, but single boundaries can have an *odd* number of subdivisions. Figure 12 shows such an example: the intersection between a torus with a pentagonal section and a box. In this case, every single boundary over the torus will have five sides, and the connected component identified by the intersection curves over the torus (with a non disk-like topology) has an overall even number of sides (10). However, the two blending regions that we need to quadrangulate will have an odd number of sides, making their direct quadrangulation not possible. To solve this problem, we could refine the whole connected component with non disk-like topology in order to make these boundaries even. However, to make this modification minimal, we search for the shortest polychord, connecting two odd boundaries and we refine only this strip of quads [Daniels et al. 2008].

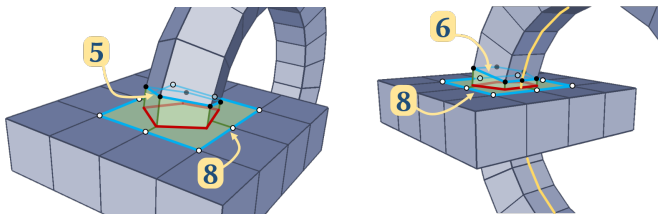


Fig. 12. A torus with a pentagonal section intersects a block (left). The patch surrounding the intersection curve has a boundary with an odd number of sides (5 on the torus and 8 on the block) and therefore cannot be quadrangulated. Refining a polychord makes the boundary even (6 and 8 sides) and the patch become quadrangulable.

Note that it is always possible to find such a connecting polychord. First consider that the overall number of the edges on the open boundary is even (by construction: they have been generated by removing quads from a closed mesh). Without loss of generality assume we have just two open boundaries A, B both with an odd edges number. Consider all the polychords exiting from the edges of A; by contradiction suppose that there is no polychord going from A to B, then all the polychords start and end on A covering an even number of edges, that is wrong because we assumed A has an odd number of edges. To handle the more generic case where you have more than just two open odd boundaries, first, consider that all the even boundaries could be *virtually* closed by quads in any way, leaving only an even number of odd boundaries; it is then evident that these remaining boundaries can be handled, pair by pair, as explained.

3.5 Final quadrangulation

At this point, we have a set of disk-like triangle patches whose sides are between the limits imposed by the quadrangulation algorithm. Also, every patch has a single boundary. We then map the boundary of the patch onto the borders of a regular polygon and use this mapping as a constraint to parameterize the interior using least-squares conformal maps [Lévy et al. 2002]. We compute the quadrangulation in parametric 2D space using [Takayama et al. 2014], and then we interpolate the 3D positions of the vertices in parametric space.

4 IMPLEMENTATION DETAILS

To test the proposed layout preserving blending technique we have implemented a small interactive system that allow to detach portions of meshes and freely combine them controlling the degree of smoothness.

4.1 The detaching tool

To detach components we followed an approach similar to the one proposed in [Ji et al. 2006]: the user has to select pairs of opposite points on the surface that defines a smooth polyline loop that splits

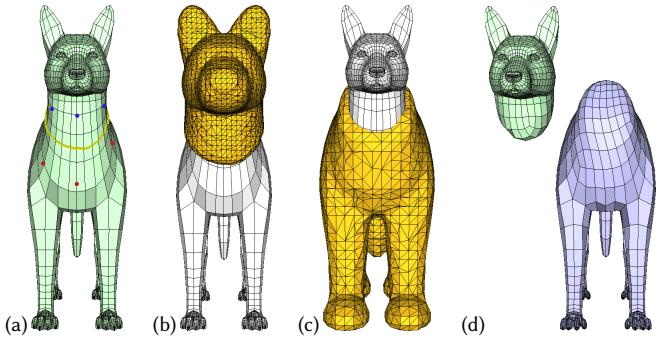


Fig. 13. The detaching tool: The user select a sequence of opposite points on the model (a); Two offset surfaces are created (b) and (c); Two pieces are obtained using the intersection with the offset surfaces.

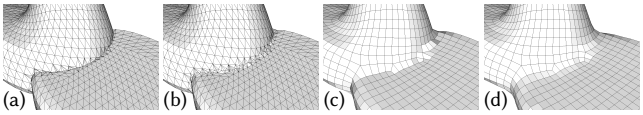


Fig. 14. The effect of the smoothing of the intersection curve: (a)The result of the boolean operation; (b) the first smooth steps on the triangulated mesh; (c) The quadrangulation step with tangent space smoothing; (d) the final result after the last step of Laplacian smooth.

the mesh into two separate components. Once we have divided the mesh into two distinct parts, we close the holes, and we create an offset surface that is used to detach the portion of the mesh using an intersection boolean operation. Figure 13 shows this pipeline.

4.2 Smoothing the surface nearby the intersection curve

Given two meshes, the user can smooth along the intersection of the two meshes providing a more attractive organic look to the final result. However, the smoothing should not be too invasive and let that part of the original mesh remain as close to the original as possible.

We apply this initial smoothing step on the triangulated mesh resulting from the first boolean operation. We first select the intersection curve, then we propagate a geodesic from the intersection curve toward the interior, and we choose the subset of vertices whose geodesic distances are below a certain threshold (we use a 5% of the diagonal of the bounding box). Then we perform a Laplacian smoothing on this subset of vertices. Intuitively, the vertices that are close to the intersection lines should move more than the ones that are far away. To obtain this effect, we linearly weight the effect of the smoothing w.r.t. to the geodesic distance from the intersection line (see Figures 14.a and 14.b). Once the final quadrangulated mesh is obtained, we perform an additional smooth step in tangent space that successfully redistributes the total distortion. This step is localized to a neighborhood of the new created surface. Finally, we perform a Laplacian smooth close to the intersection line. Figures 14.c and 14.d show how these smooth operations can significantly improve the final tessellation. As with any other blending tool, we let the user exert control over the smoothing steps and on the area of influence.

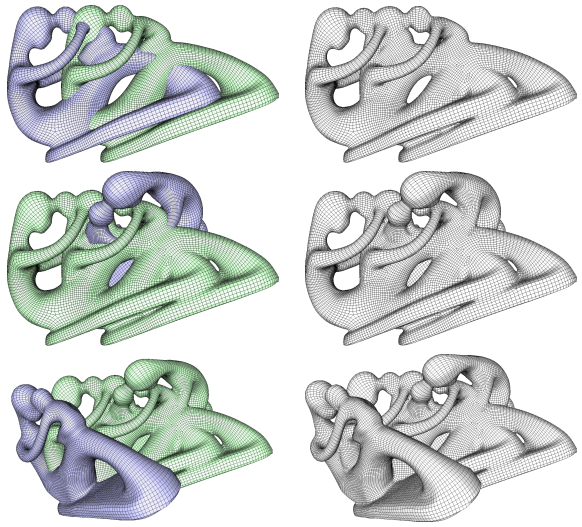


Fig. 15. Iteratively merging the fertility model to check the robustness of the proposed approach.

5 RESULTS

We performed our test on a desktop computer with an Intel i7-8750H processor with 16GB of RAM. We used Gurobi [Gurobi Optimization 2018] to solve the minimization of Section 3.3. All the code is single-threaded and not highly optimized; it has been implemented using the VCG Library [CNR 2013], CG3Lib [Muntoni et al. 2019], libigl [Jacobson et al. 2013b], and CGAL [The CGAL Project 2019]. To test the robustness of the proposed approach, we iteratively added merging operations on rotated versions of the fertility model. Our technique always produced a two-manifold closed quadrilateral surface. Figure 15 shows some of the first steps of the test. In Figure 16, we show a full set of combinations among six meshes having different topologies, complex connectivity, or intricate geometric details.

We tested our method on a collection of professionally designed quadrangulated animals. Some of the results are shown in Figure 17. The presented models have been produced through interactive editing sessions by the authors. We composed models by simply detaching portions of the body from one animal and combining them with the body of another animal. Our method always succeeded in producing a two-manifold quadrilateral surface. Most of the original quad layout has always been successfully preserved, and our method was able to create a smooth flow in the blended portions of the meshes. The operations have always been performed within 1 second for meshes in the animal dataset. The merging operations of Figure 15 required up to 15 seconds for some of the blending operations. As expected, the running time is proportional to the number of triangles in the blending area that in this specific test case can involve most of the original surface. Timings are summarized in Table 1.

Figure 18 reports the distribution of distortion in individual elements relative to the experiment shown in Figure 2. Distortion has been computed by using the distance with respect to the ideal quad

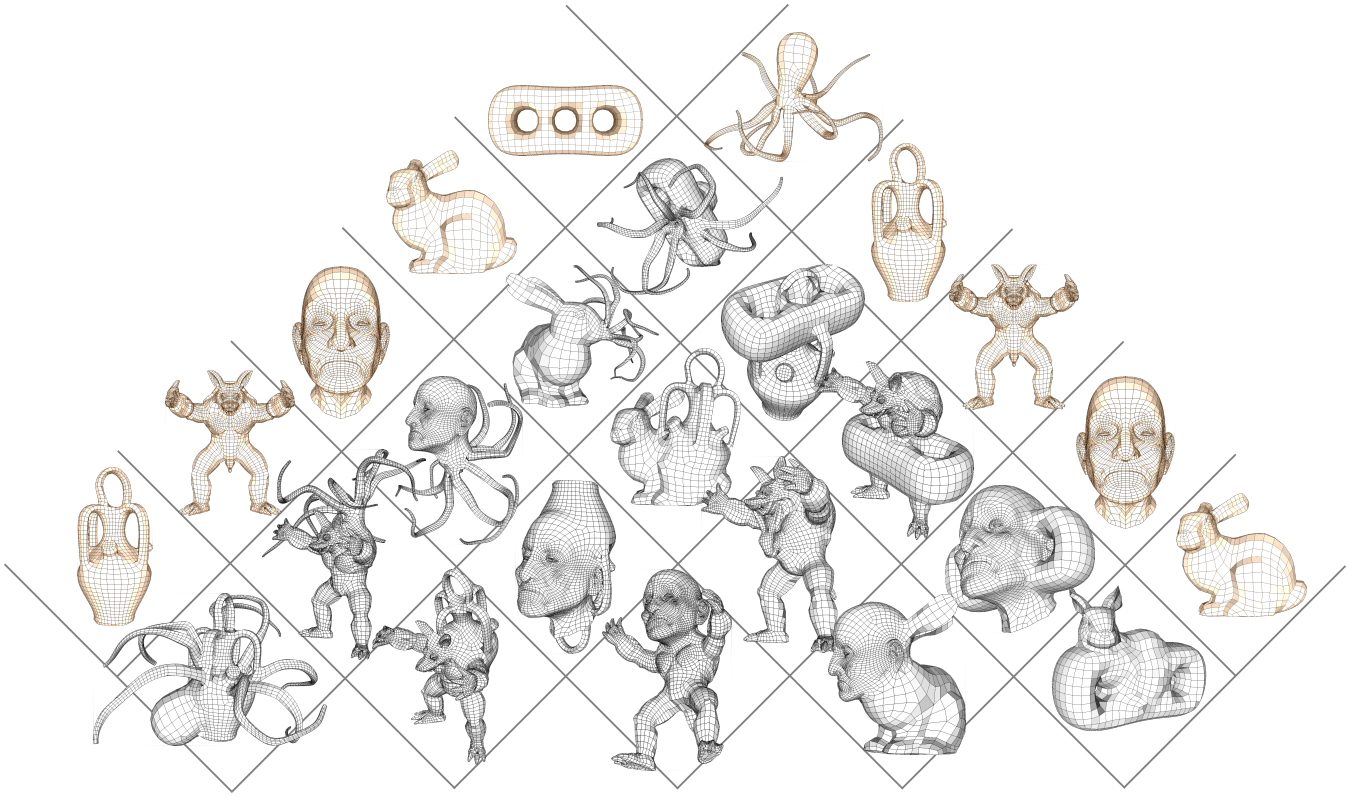


Fig. 16. All the pairwise joins of six meshes, different in genus, complexity, and details.

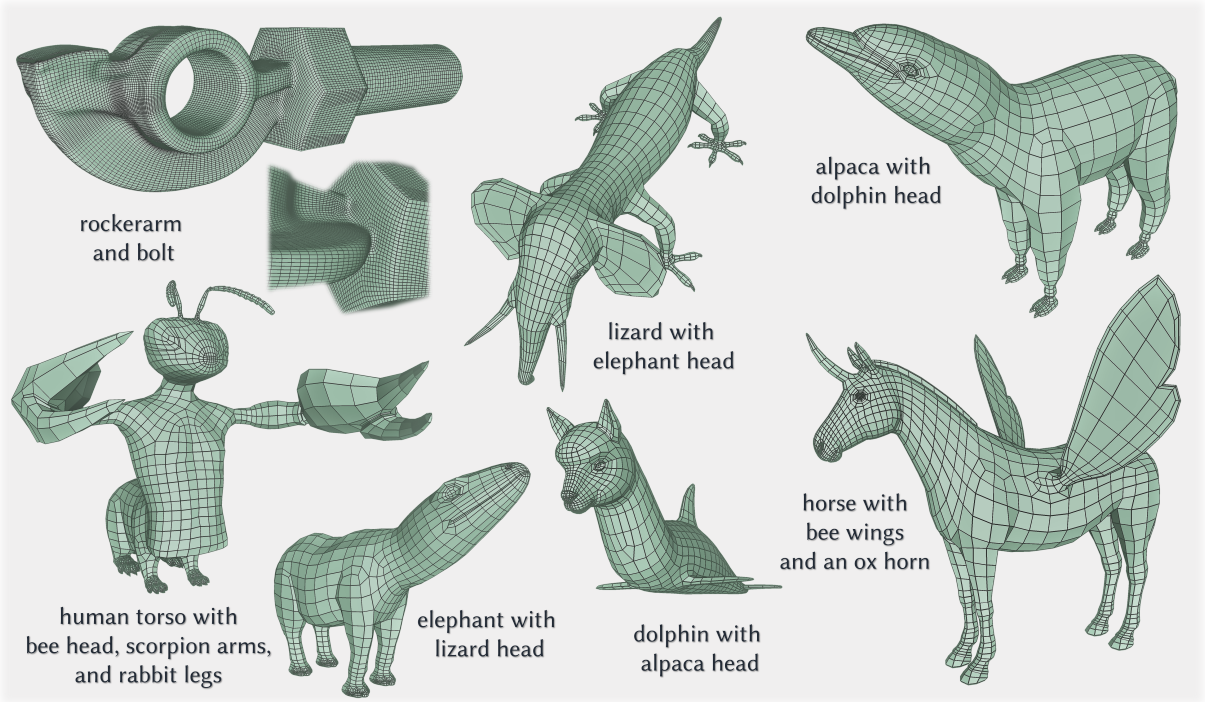


Fig. 17. An overview of the final results obtained with our modelling tool. All the displayed models are available in the additional material.

Table 1. Time of execution of each step of the pipeline for the examples generated with QuadMixer. Times are all in millisecond with the exception of the total time which is reported in seconds.

Models	K Tris	Bool	Trace	Solve	Quad	Other	Total
Dolphin ∪ Alpaca	3 / 2	92	79	25	44	11	0.25
Alpaca ∪ Dolphin	5.4 / 1.6	93	68	24	62	17	0.26
Mannequinn ∪ Alpaca	5.3 / 2	156	94	26	80	32	0.39
Lizard ∪ Elephant	6.2 / 2.4	144	174	29	93	23	0.46
Elephant ∪ Lizard	7.8 / 1.2	162	126	159	149	39	0.64
Armadillo ∪ Pig	9.1 / 4.6	338	259	160	198	86	1.04
Monkey ∪ Dolphin	10.7 / 3	328	198	702	360	54	1.64
Monkey ∩ Dolphin	10.7 / 3	313	166	202	27	32	0.74
Monkey / Dolphin	10.7 / 3	315	347	904	173	56	1.80
Monkey ∪ Mannequinn	10.7 / 5.3	391	451	1324	356	118	2.64
Monkey ∩ Mannequinn	10.7 / 5.3	379	129	51	12	28	0.60
Monkey / Mannequinn	10.7 / 5.3	392	206	162	299	53	1.11
Rockerarm ∪ Rod	47.6 / 17.7	790	822	720	701	238	3.27
Fertility ∪ Fertility	26.2 / 26.2	2649	3102	2402	438	223	8.81
Fertility^2 ∪ Fertility	39.5 / 26.2	4179	5052	4073	728	883	14.92
Fertility^3 ∪ Fertility	59 / 26.2	1628	2055	1700	920	1102	7.41

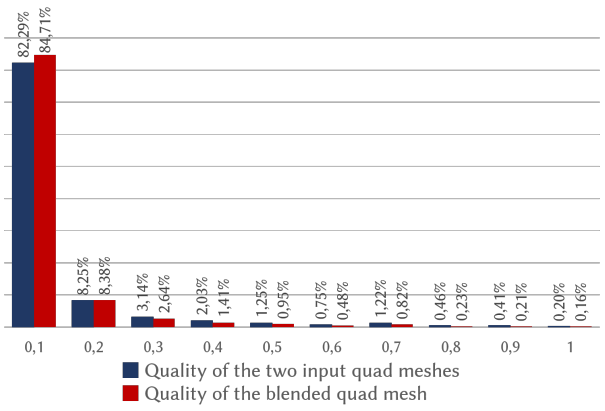


Fig. 18. Distortions of the quads of Figure 2 measured using the metric defined in [Pietroni et al. 2015] (0 means no distortion).

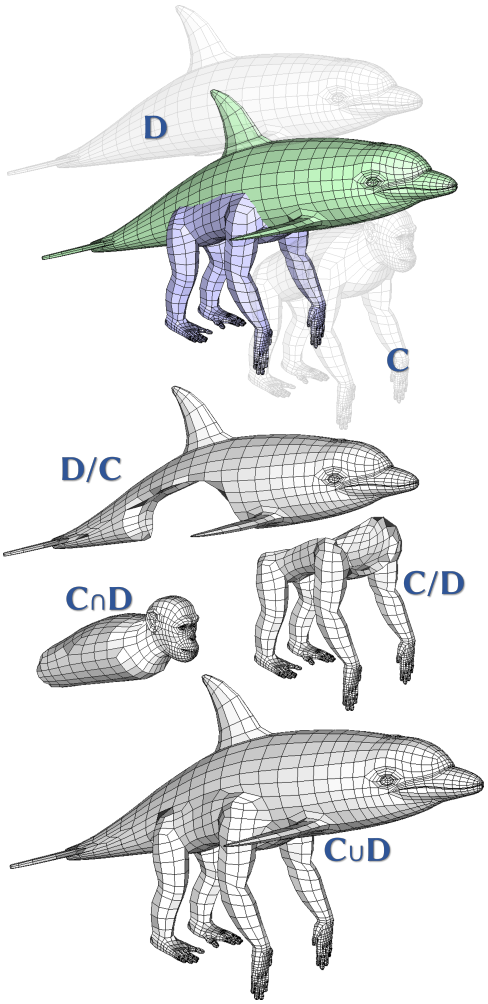


Fig. 19. Our algorithm can mimic any boolean operation on quad meshes generating a well-shaped quad mesh which reuses the most of the initial layout. We show here, from left to right: the input consisting of two quad meshes (a dolphin **D** and a chimpanzee **C**) interactively placed in the scene; the two differences; their intersection; the union, which is, typically, the most interesting operation from a semantic standpoint.

as defined in [Pietroni et al. 2015]. As shown in the histograms, our method does not affect the overall quality of the quads.

6 CONCLUSIONS

We proposed a novel powerful pipeline for freely composing quadrilateral meshes. Our method takes advantage from boolean operations to smoothly blend between quadrilateral meshes keeping as much as possible the tessellation of the original surfaces. We integrated our technique into an interactive system and tested its effectiveness in a modeling scenario. Given the robustness and the visual quality of the generated meshes, we believe that our composing technique might become a powerful tool in current production pipelines allowing artists to rapidly exploit portions of existing models instead of resorting to complete re-topology sessions.

Moreover, our method can successfully mimic all the boolean operations, such as union, difference, and intersection.

6.1 Limitations and future work

Our method, currently, cannot efficiently preserve sharp features, as shown in Figure 19. An exact boolean operation will introduce sharp features, especially for the difference operation. However, our framework can be extended to include sharp feature preservation: feature alignment can be enforced in the step of field calculation, and the features can be included as traces in the patch subdivision step. The same can be done along the intersection curve. Additionally, the field can be constrained to align with these feature lines together with boundaries. Finally, the vertices along sharp features must have a special treatment during the smoothing. These extensions would guarantee the preservation of sharp features.

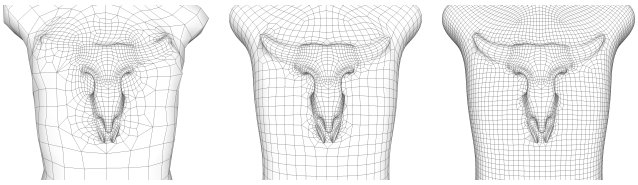


Fig. 20. Blending meshes of different resolution.

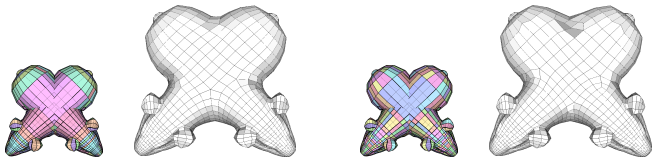


Fig. 21. The sensitivity with respect to two different initial patch layout: motorcycle graph (left) and emanating separatrices (right).

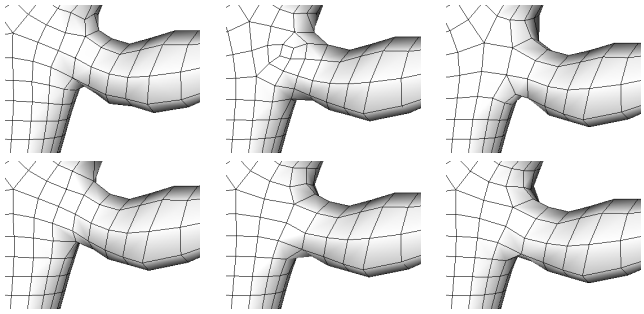


Fig. 22. The variation of the tessellation configurations: close geometric configuration might cause abrupt changes in the tessellation (top); such an artefact can be mitigated with simple improvements (bottom).

Another limitation of our method is its dependence on the initial resolution and the initial patch layout. We experimented that, as can be expected, the better results are obtained if the two meshes have a similar resolution (see Figure 20), which can be attributed to the intrinsic limitations of quad mesh modeling. One practical solution to this problem consists in matching the resolutions by using some subdivision steps before performing the boolean operation. Figure 21 instead shows the sensitivity of the method to two different initial patch layouts.

Our method cannot guarantee that the produced quadrangulation variates smoothly while the user variates the intersection configuration. The accompanying video and the examples shown in Figure 22 (top) shows this limitation: while the arm moves slowly to the bottom, the produced quadrangulation might have some unexpected change in the tessellation. This limitation might affect the overall usability. The patch layout procedure can be redesigned to variate continuously under small modifications of the intersecting region. We believe this can be a compelling topic for future work. Nevertheless, we experimented a simple procedure which already provides encouraging improvements: we randomly perturb the intersection configuration, and we select the best tessellation for a given metric. For this experiment we used as metric a linear combination between the quality of the quadrilateral elements and the number of singularities: $0.3 * Q_t + 0.7 * Av_d$, where Q_t is the average quad quality using the metric in [Pietroni et al. 2015] and Av_d is the average absolute valence deficit (the absolute difference of valence for each vertex from 4). We show the result obtained with this improvement Figure 22 (bottom).

Finally, our approach can fail in the extreme case when the boolean operations do not preserve any of the original quads, e.g., the space around the intersection lines covers all the remaining meshes: in this case our algorithm will not produce a valid patch decomposition and therefore will not be able to generate a quad meshing. However, this kind of situations is well managed by a complete re-meshing of the result since with such a configuration the original quad structure could probably not preserved.

ACKNOWLEDGMENTS

Stefano Nuvoli gratefully acknowledges Sardinia Regional Government for the financial support of his Ph.D. scholarship (P.O.R. Sardegna F.S.E.1 Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2007-2013 - Axis IV Human Resources, Objective l.3, Line of Activity l.3.1.). This work was supported in part by the Italian DSURF PRIN 2015 (2015B8TRFM) project, and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The base meshes of the animals shown in figures 1,2,5,13,17, and 21 have been professionally modeled by Paul Gonet (gonzou3d.carbonmade.com) and are available on BlenderMarket. The author granted us the right of redistributing the remixed models shown in the figures under the CC-BY-NC-SA license.

REFERENCES

Autodesk. 2018. Mudbox. https://www.autodesk.com/education/free-software/mudbox

Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. 2012. Design-driven quadrangulation of closed 3D curves. *ACM Trans. Graph.* 31, 6 (2012), 178:1–178:11.

Henning Biermann, Ioana M. Martin, Fausto Bernardini, and Denis Zorin. 2002. Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph.* 21, 3 (2002), 312–321.

Stephan Bischoff and Leif Kobbelt. 2005. Structure Preserving CAD Model Repair. *Comput. Graph. Forum* 24, 3 (2005), 527–536.

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. 2013. 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. 2017a. Partitioning Surfaces Into Quadrilateral Patches: A Survey. *Comput. Graph. Forum* 36, 8 (2017), 567–588.

Marcel Campen. 2017b. Tiling the Bunny: Quad Layouts for Efficient 3D Geometry Representation. *IEEE Computer Graphics and Applications* 37, 3 (2017), 88–95.

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 and Leif Kobbelt. 2010. Exact and Robust (Self-)Intersections for Polygonal Meshes. *Comput. Graph. Forum* 29, 2 (2010), 397–406.

Marcel Campen and Leif Kobbelt. 2014. Dual strip weaving: interactive design of quad layouts using elastica strips. *ACM Trans. Graph.* 33, 6 (2014), 183:1–183:10.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference 2008, Salerno, Italy, 2008.* 129–136.

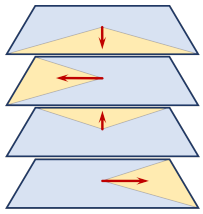
CNR. 2013. The Visualization and Computer Graphics Library. http://vcg.isti.cnr.it/vcglib/.

Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. 2008. Quadrilateral mesh simplification. *ACM Trans. Graph.* 27, 5 (2008), 148:1–148:9.

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.
David Eppstein, Michael T. Goodrich, Ethan Kim, and Rasmus Tamstorf. 2008. Motorcycle Graphs: Canonical Quad Mesh Partitioning. *Comput. Graph. Forum* 27, 5 (2008).
Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David P. Dobkin. 2004. Modeling by example. *ACM Trans. Graph.* 23, 3 (2004), 652–663.
LLC Gurobi Optimization. 2018. Gurobi Optimizer Reference Manual. http://www.gurobi.com
Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013a. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.* 32, 4 (2013), 33:1–33:12.
Alec Jacobson, Daniele Panozzo, et al. 2013b. 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.
Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. 2006. Easy Mesh Cutting. *Comput. Graph. Forum* 25, 3 (2006), 283–291.
Felix Kälberer, Matthias Nieser, and Konrad Polthier. 2007. QuadCover - Surface Parameterization using Branched Coverings. *Comput. Graph. Forum* 26, 3 (2007), 375–384.
Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.* 21, 3 (2002), 362–371.
Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. 2013. Animation-Aware Quadrangulation. *Comput. Graph. Forum* 32, 5 (2013), 167–175.
Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine-Hornung, Enrico Puppo, and Paolo Cignoni. 2015. Data-driven interactive quadrangulation. *ACM Trans. Graph.* 34, 4 (2015), 65:1–65:10.
Carroll Morgan. 1994. *Programming from Specifications (2Nd Ed.).* Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK.
Alessandro Muntoni, Stefano Nuvoli, et al. 2019. CG3Lib: A structured C++ geometry processing library. https://github.com/cg3hci/cg3lib.
Ashish Myles, Nico Pietroni, and Denis Zorin. 2014. Robust field-aligned global parametrization. *ACM Trans. Graph.* 33, 4 (2014), 135:1–135:14.
Ahmad H. Nasri, Malcolm A. Sabin, and Zahraa Yasseen. 2009. Filling N-Sided Regions by Quad Meshes for Subdivision Surfaces. *Comput. Graph. Forum* 28, 6 (2009), 1644–1658.
Darko Pavic, Marcel Campen, and Leif Kobbelt. 2010. Hybrid Booleans. *Comput. Graph. Forum* 29, 1 (2010), 75–87.
Chi-Han Peng, Michael Barton, Caigui Jiang, and Peter Wonka. 2014. Exploring quadrangulations. *ACM Trans. Graph.* 33, 1 (2014), 12:1–12:13.
Chi-Han Peng, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. 2011. Connectivity editing for quadrilateral meshes. *ACM Trans. Graph.* 30, 6 (2011), 141:1–141:12.
Nico Pietroni, Enrico Puppo, Giorgio Marcias, Roberto Scopigno, and Paolo Cignoni. 2016. Tracing Field-Coherent Quad Layouts. *Comput. Graph. Forum* 35, 7 (2016).
Nico Pietroni, Davide Tonelli, Enrico Puppo, Maurizio Froli, Roberto Scopigno, and Paolo Cignoni. 2015. Statics Aware Grid Shells. *Comput. Graph. Forum* 34, 2 (2015).
Pilgway. 2017. 3DCoat. https://3dcoat.com/home/
Pixologic. 1999. ZBrush. http://pixologic.com
Scott Schaefer, Joe D. Warren, and Denis Zorin. 2004. Lofting Curve Networks using Subdivision Surfaces. In *Second Eurographics Symposium on Geometry Processing, Nice, France, July 8-10, 2004.* 103–114.
Ryan Schmidt and Karan Singh. 2010. Meshmixer: An Interface for Rapid Mesh Composition. In *ACM SIGGRAPH 2010 Talks (SIGGRAPH '10).* ACM, New York, NY, USA, Article 6, 1 pages.
Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. 2006. SnapPaste: an interactive technique for easy mesh composition. *The Visual Computer* 22, 9-11 (2006), 835–844.
Kenshi Takayama, Daniele Panozzo, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. 2013. Sketch-based generation and editing of quad meshes. *ACM Trans. Graph.* 32, 4 (2013), 97:1–97:8.
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, 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.
The CGAL Project. 2019. *CGAL User and Reference Manual* (4.14 ed.). CGAL Editorial Board.
Julien Tierny, Joel Daniels II, Luis Gustavo Nonato, Valerio Pascucci, and Cláudio T. Silva. 2011. Inspired quadrangulation. *Computer-Aided Design* 43, 11 (2011), 1516–1526.
Amir Vaxman, Marcel Campen, Olga Diamanti, David Bommes, Klaus Hildebrandt, Mirela Ben-Chen, and Daniele Panozzo. 2017. Directional field synthesis, design, and processing. In *SIGGRAPH '17 Courses.* 12:1–12:30.
The Foundry Visionmongers. 2018. Modo 12.1. http://www.thefoundry.co.uk/products/modo
Zahraa Yasseen, Ahmad H. Nasri, W. Boukaram, Pascal Volino, and Nadia Magnenat-Thalmann. 2013. Sketch-based garment design with quad meshes. *Computer-Aided Design* 45, 2 (2013), 562–567.
Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng, and Xue-Cheng Tai. 2010. Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow. *Comput. Graph. Forum* 29, 2 (2010), 517–526.
Jiaran Zhou, Marcel Campen, Denis Zorin, Changhe Tu, and Cláudio T. Silva. 2018. Quadrangulation of non-rigid objects using deformation metrics. *Computer Aided Geometric Design* 62 (2018), 3–15.
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* 35, 4 (2016), 39:1–39:15.

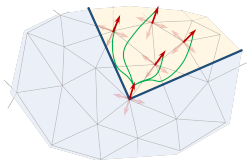
A TRACING THE FIELD

Following [Kälberer et al. 2007], given a surface M, with the exception of the singularities, we make four copies of each point $p \in M$. We associate each copy to one direction of the cross-field. Then each copy of p encodes both its position and one of the orientations of the cross field. This process creates $M4$, which is a stratification of the original manifold surface M. Space $M2$ is the quotient space of $M4$ obtained by identifying pairs of opposite directions. Hence $M2$ is composed only by two sheets. Each sheet encodes a line-field.



We discretize $M4$ for tracing following the approach proposed by [Campen et al. 2012]. Given an input manifold surface equipped with a per-vertex cross-field, we create a graph with four nodes on every vertex, one for each direction of the cross-field [Campen et al. 2012].

We connect each node with the neighbors whose position is within the visibility cone of its emanating direction, for each position, we choose the copy having the more aligned direction. We also augment the graph with vertices belonging to the 1-ring to provide more degrees of freedom to the tracing process.



Given an initial node (which corresponds to a vertex and a field direction), the tracing process is a propagation process where at every step we select the connected nodes whose position is the most aligned with the field. This way, we have a fast and robust tracing setup.