MODELING QUADRILATERAL MESHES BY COMPOSITION

Alex David Hernández Maturrano

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Claudio Esperança

Rio de Janeiro
Dezembro de 2020

MODELING QUADRILATERAL MESHES BY COMPOSITION

Alex David Hernández Maturrano

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Claudio Esperança

Aprovada por: Prof. Claudio Esperança
Prof. Ricardo Guerra Marroquim
Prof. Hélio Cortes Vieira Lopes
Prof. Asla Medeiros e Sá
Prof. Anselmo Antunes Montenegro

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2020

Hernández Maturrano, Alex David

Modeling quadrilateral meshes by composition/Alex David Hernández Maturrano. – Rio de Janeiro: UFRJ/COPPE, 2020.

XI, 54 p.: il.; 29, 7cm.

Orientador: Claudio Esperança

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2020.

Referências Bibliográficas: p. 49 – 54.

1. Modeling-by-composition. 2. Boolean operation. 3. Quadrilateral meshes. 4. Quadrangulation. I. Esperança, Claudio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedicated to my family. Special feeling to my parents that waited during many years far from me in Perú until I can finish my studies.

Acknowledgment

First and foremost, thanks to God for his blessing and mercy in completing this phase of my career. I would like to express my deepest appreciation to my advisor Claudio Esperança for giving me the opportunity to do this work providing invaluable guidance throughout this research and for his patience at the end of this walk. Also, I would like to thanks to my wife who has been a constant source of support and advice for me, to my son, who despite being little, encourages me to improve myself every day and to my parents, for whom I am extremely grateful for their love, prayers, caring and sacrifices for educating and preparing me for my future. They also had a lot of patience with me.

I am highly indebted to Nico Pietroni and Paolo Cignoni for their guidance and help in many topics of this work. They proposed this challenge to my advisor and me collaborating with us with great enthusiasm. In addition, I express my sincere gratitude to the members of the thesis committee for agreeing to review this work despite the current coronavirus health crisis.

I thank the Federal University of Rio de Janeiro that, through the Systems Engineering and Computer Science Program, allowed me to fulfill my dream of complete my PhD.

Finally, the conclusion of this project could not have been accomplished without the financial support of the Coordination for the Improvement of Higher Education Personnel (CAPES), which granted me a PhD scholarship.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

MODELAGEM DE MALHAS QUADRANGULARES POR COMPOSIÇÃO

Alex David Hernández Maturrano

Dezembro/2020

Orientador: Claudio Esperança

Programa: Engenharia de Sistemas e Computação

Abordamos nesta tese o problema de modelagem de malhas quadrangulares por composição, isto é, costurando retalhos de malhas. Conseguimos construir malhas quadrangulares no espírito de Geometria Sólida Construtiva (CSG em inglês), mas trabalhando somente com malhas quadrangulares. Nossa proposta preserva a maioria dos layouts originais das partes, isto é, mantém intactos todos os quadriláteros pertencentes aos "patches" quadrangulares não envolvidos na composição, e requer um tempo de processamento de poucos segundos. Usamos várias técnicas para conseguir uma implementação estável. De forma condensada: primeiro executamos operações booleanas robustas nas malhas triangulares correspondentes; em seguida, usamos esse resultado para identificar e construir novos patches quadrangulares para pequenas regiões vizinhas às curvas de interseção; finalmente, esses patches quadrangulares recém criados são cuidadosamente quadrangulados respeitando as restrições nas bordas e levados de volta para os modelos originais. A malha resultante preserva o fluxo de arestas (*edge flow* em inglês) que, por construção, é capturado e incorporado aos novos patches quadrangulares tanto quanto possível. Finalmente, apresentamos alguns resultados que mostram o potencial de nosso protótipo para aplicações reais, em particular, para modelos desenhados para animação.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

MODELING QUADRILATERAL MESHES BY COMPOSITION

Alex David Hernández Maturrano

December/2020

Advisor: Claudio Esperança

Department: Systems Engineering and Computer Science

In this work we address the problem of modeling quadrilateral meshes by composition, i.e., by stitching together parts. We get to build pure quadrilateral meshes in the spirit of Constructive Solid Geometry (CSG), but working only with quadrilateral meshes. Our proposal to compose quadrilateral meshes preserves the majority of the original layout of the parts, i.e, it keeps untouched all the quads in the patches which are not involved in the blending, and runs at interactive rates. We use a number of well-established techniques to achieve a stable implementation, but in short, 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. At the end, we present some results showing the potential of our prototype for real applications, in particular, with models designed for animation.

Contents

List of Figures **x**

1 Introduction **1**
1.1 Classical pipeline for 3D artists 4
1.2 Character modeling for animation 5

2 Fundamentals of quad meshes **7**
2.1 Definitions 7
2.1.1 Base complex 10
2.1.2 Edge flow 12
2.1.3 Quadrangulation challenges 13

3 Literature review **18**
3.1 Boolean and Composing Operations 18
3.2 Automatic quadrangulations 21
3.3 Interactive Quadrangulation tools 23
3.4 Quad-Meshing Patches 24

4 Blending of quad meshes **26**
4.1 Optimal Patch Retraction 29
4.2 Patch Subdivision 30
4.2.1 Field-aligned Patch Tracing 31
4.2.2 Patch configuration 32
4.2.3 Cross-field generation 33
4.2.4 Concave vertex tracing 34
4.2.5 Patch splitting 34
4.3 Subdivision Optimization 34
4.3.1 On the existence of a valid solution 36
4.4 Final quadrangulation 39

5 Implementation and results **40**
5.1 Smoothing the surface nearby the intersection curve 40

5.2 Character head models 41
5.3 Running-time 43

6 Conclusions **46**
6.1 Limitations and future work 46

References **49**

List of Figures

1.1 Face modeled for animation 3
1.2 Sculpted head model 4
1.3 Animation sequence 6

2.1 Non-manifold examples 8
2.2 Dual mesh topology 9
2.3 Dual curves 10
2.4 Parallel dual curves 11
2.5 Edge ring and edge loop 11
2.6 Example of different quad layouts 12
2.7 Edge flow in polygonal meshes 13
2.8 Edge flow comparison 14
2.9 Some automatic quadrangulations methods 16

3.1 Comparison with MeshFusion 19
3.2 Comparison with QuadriFlow 20
3.3 Construction of a cross-field 22
3.4 Takayama's method 24

4.1 Overview of our processing pipeline 28
4.2 Patch layout with separatrices or motorcycle graph 29
4.3 Patch layout retraction 30
4.4 Patch tracing procedure 31
4.5 Tracing splits 32
4.6 Crossing paths 33
4.7 Concave vertex tracing 33
4.8 Optimization procedure 35
4.9 Regularization 36
4.10 Solvability 37
4.11 Triangulated part topology 38

5.1 Smoothing of the intersection curve 40

5.2 Interchanging ears 41
5.3 Man with tentacles 42
5.4 Man with python mouth and nose 42
5.5 Robustness test 43
5.6 Pairwise joins of six meshes 44
5.7 Overview of the final results obtained with our modelling tool 45
5.8 Distortions of the quads for the blended quad meshes 45

6.1 Our algorithm can mimic any boolean operation on quad meshes 47
6.2 Blending meshes of different resolution. 47
6.3 Sensitivity with respect to the initial patch layout 47
6.4 Continuity test 48

Chapter 1

Introduction

Animation, VFX, and the game industry have become some of the fastest-growing segments in the global media and entertainment market, with US$ 259 billion in 2018 and is expected to reach US$ 270 billion by 2020 [1]. Global consumers are displaying a growing appetite for engaging, high definition visual experiences. It is for that reason that the fast generation of high-quality 3D assets is a pressing need for this industry. Moreover, a significant part of the time in the production stage of films and games is destined to this task.

Several techniques to model organic shapes faster and more easily were proposed in the last decades. Depending on the objectives and resources, many ideas on how to use sketch-based techniques, 3D reconstruction from sophisticated inputs such as curve networks or line drawings, boolean operations of triangle meshes, deformation techniques, or a combination of these have been reported. However, for the films and games market, where character animation is another challenging process beyond just modeling, little progress has been made. One reason for this is the use of quadrilateral meshes as surface representation for part of the production studios throughout the modeling and animation process. The majority of the exciting advances in modeling were designed to work on triangle meshes, and the global structure of a quadrilateral meshes makes adapting such methods impossible in many cases.

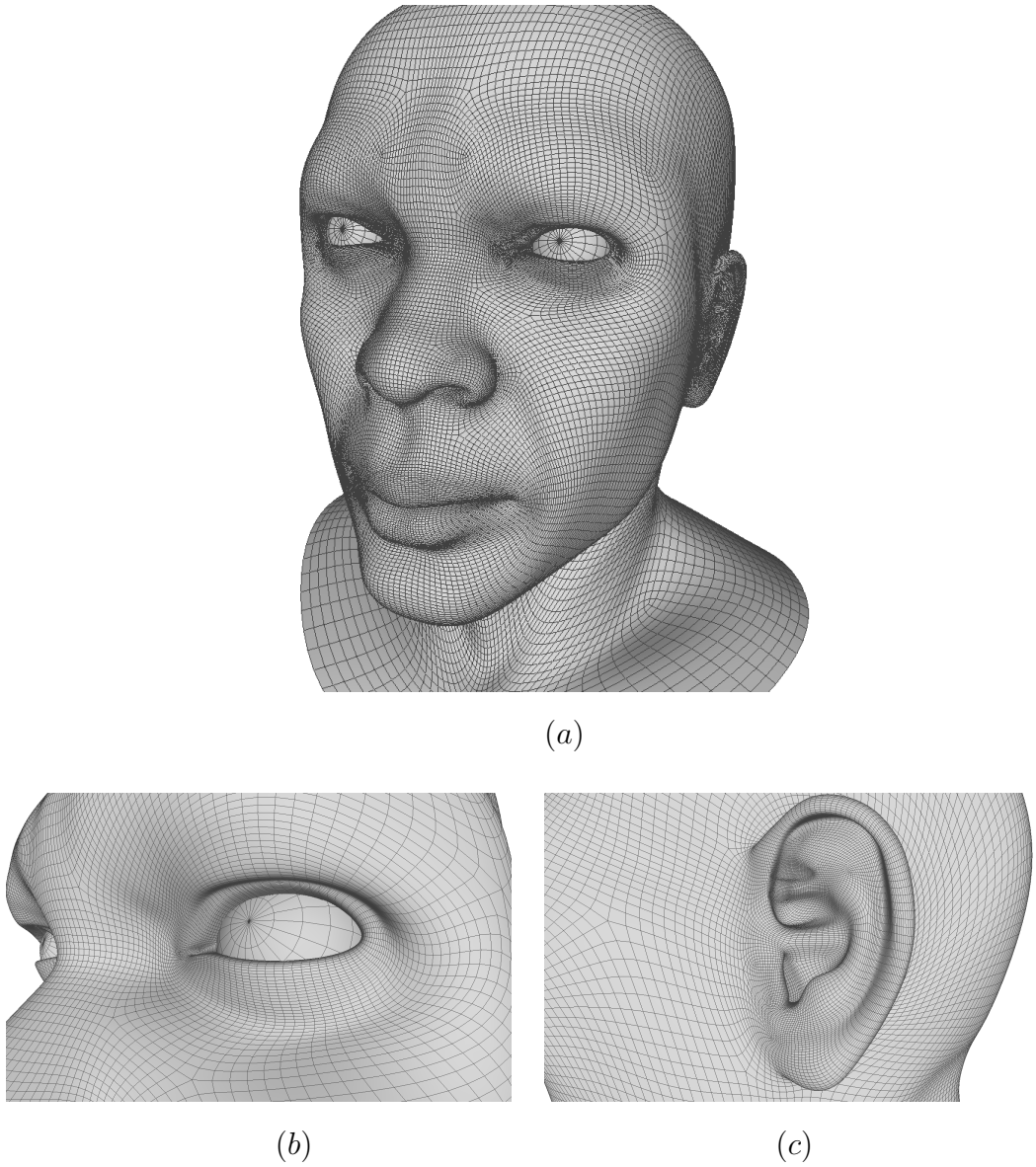
We are interested in modeling 3D assets by composition in the spirit of Constructive Solid Geometry (CSG) but working only with quadrilateral meshes designed for animation. This process of shape composition has recently gained much interest. Various techniques for combining meshes since [2] have been proposed. 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. 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 the field of modeling-by-composition is quite active and gener-

ating promising results, all the proposed solutions cannot produce a fully quadrangulated mesh. Their output is always limited to triangulated surfaces.

One reason why quadrilateral models are desirable for modeling is their simple topology, i.e, a topology with a high degree regularity and with the presence of loops of faces (we will expand on these concepts in the next chapter). That way, the model can be understood in a more straightforward manner and modified by several modelers in an extensive production line. These loops of faces are, in reality, an advantage of quadrilateral models since they allow to define more selection mechanisms (edge loops, edge rings) than with triangle-based ones. These characteristics are explored in animation by aligning the edge loops of the quadrilateral mesh with the feature lines (joints, principal curvature directions, etc) of the model and, in this way, work with a well-segmented mesh. This is crucial for a rapid and effective skinning and rigging.

On the other hand, modeling complex shapes from scratch with only quadrilaterals 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. Usually, the pipeline starts by making coarse quad layouts manually, followed later by creating a subdivision surface. Professional designers employ their semantic knowledge and experience to adjust the layout to the needs of a particular application. Typical modeling systems used in the industry (AUTODESK, PILGWAY, PIXOLOGIC, etc) allow the user to draw vertices and edges on a surface. Since this manual procedure is time-consuming and error-prone, a series of sketch-based retopology approaches [6–8] 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.

We propose to compose quadrilateral meshes designed for animation, taking inspiration from the classical boolean operations defined on triangle meshes, but with operators redesigned to work on quadrilateral ones. A quadrilateral mesh designed for animation presents a particular edge flow, and content creators still consider this feature as part of the artistic process. For this reason, the preservation of the original quadrilateral meshes during the composition process is crucial to allow the effective use of modeling-by-composition in the field of quadrilateral meshes for films and games production. In the remainder of this thesis, we shorten the phrase *quadrilateral mesh* to only *quad mesh*, meaning a mesh in which all faces are quadrilaterals.



(a)

(b) (c)

Figure 1.1: Face modeled for animation. Note the edge flow on the model. Edge loops conform to feature regions like eyelashes, nose, and ear. Also, note how the modeler placed irregular points in zones where no deformation occurs due to the animation. (a) Model designed for animation from scratch. (b) Image zoom on the left eye of model in (a). Note the contouring edge loops and the irregular point on the eye center. This is an example of how real modelers displace irregular points in such a way they don't affect the animation.(c) Image zoom of the left ear of the model in $a)$. Note how the irregular points are placed on regions of curvature change following the ear shape perfectly.

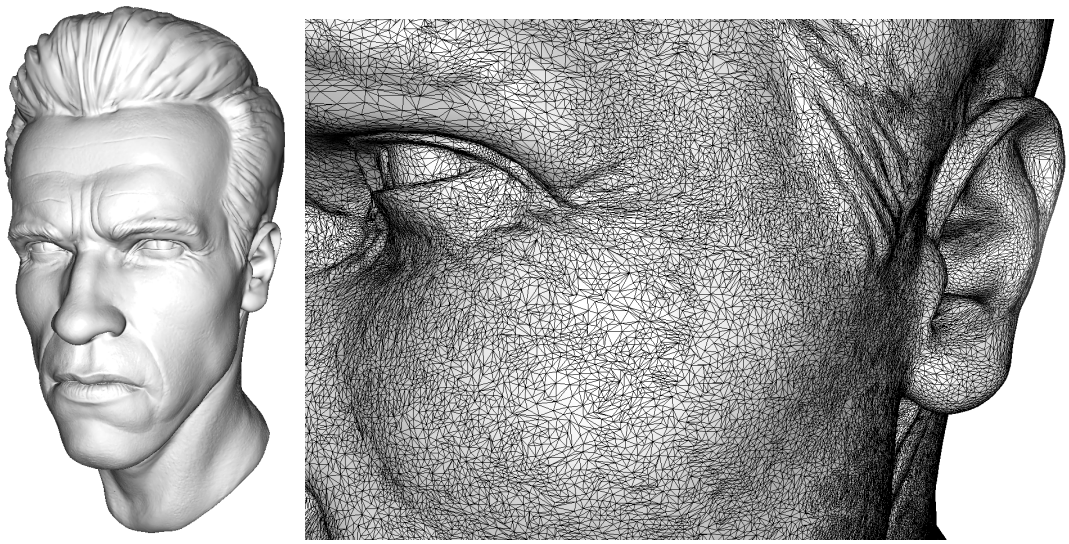


Figure 1.2: Sculpted head model. Note the richness of details, but the large number of triangles. *"The head of the sculpture use Zbrush" by imp is licensed under Creative Commons Attribution. https://skfb.ly/LDtW To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.*

1.1 Classical pipeline for 3D artists

Generating high-quality quad organic models from scratch is an art that typically begins with a 2D rough sketch of the model/character via a concept artist. Various software are used to bring that concept to the 3D world (Maya, 3DSMax, Cinema 4D, zBrush, Blender, etc.) via one or several techniques (edge-modeling, digital sculpting, box modeling). Often the software will import the 2D image in a background where the 3d modeler proceeds to build the 3D mesh by tracing the image with 3D geometry (mesh/sculpting)[1]. If the 3D model requires a lot of detail like in an animated film, this will be made into a high-resolution mesh object. Sculpting is usual for more organic designs and detailed characters. However, sculpting is considered a more intuitive and artistic way of creating a 3D model, and it is always necessary to retopologize the mesh to a quadrilateral one with a good topology for the next steps, like texturing and animation (See Figure 1.2).

In this way, there are lots of manual methods for doing retopology. Also, there are semi-automatic and automatic quadrangulation methods, which are preferably used for models like rocks or trees, which won't be under as much scrutiny as characters or they will be placed at large distances from the observer. Finally, the artist can continue with UV mapping, texturing/painting and animation.

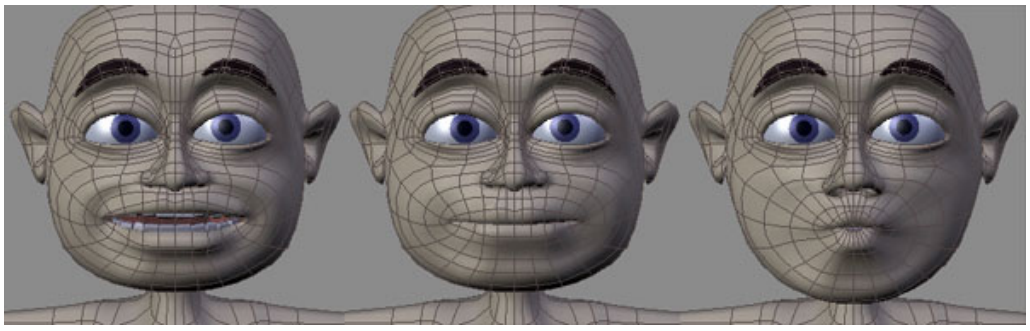
[1] An example can be seen in this tutorial.

1.2 Character modeling for animation

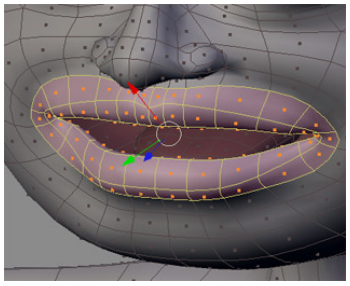
Character modeling takes its own space inside 3D modeling. Due to its particular requirements, designers apply special techniques in comparison with other types of models. Characters for animation are almost always modeled using quads. This is a difference from other characters sculpted in a popular Sculpting software. The reason for this is because a character designed to animation needs not only to look good from every angle but also in many different poses and expressions. Consider, for instance, the animation of a human model running on a track. Push-ups of the arms and legs (poses) will occur and certainly, the face will also undergo different expressions during the run (See Figure $1.3-(a)$). Building a still sculpt allows the modeler little more freedom, but it works only with characters that won't be animated where it is just necessary the model to look good in a specific pose or from a certain camera angle like in Figure 1.2.

Making a character for animation demands symmetry. Special attention must be given to the facial topology in order to grant realistic movements when the characters face deforms (see Figure1.1). Often modelers align the quad mesh edge flow in such a way that the region where the model deforms is an *edge loop*[2] from the quad mesh (Figure $1.1-(b), (c)$, Figure $1.3-(b)$), but in fact, this is still an artistic process and no automatic method exists for performing this task. Therefore, we could consider this type of model as a target to be achieved by automatic quadrangulators. Until now semi-automatic retopologizers([6–8]) are the best option to accelerate the modeling process of a character that will be animated thereafter. On the other hand, we propose a method to compute "boolean operations" between two quad meshes preserving as much as possible the original quadrangulation, thus helping in the reuse of valuable 3D assets designed previously by modelers.

[2]Check Section2.1



(a)



(b)

Figure 1.3: Example of an animation sequence (a) of a human model. As expected, the modelers aligned the edge loops with features like lips (b). *Font: WikiBooks, availabe in* `https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Advanced_Tutorials/Advanced_Animation/Guided_tour/Mesh/Shape/Sync`.

Chapter 2

Fundamentals of quad meshes

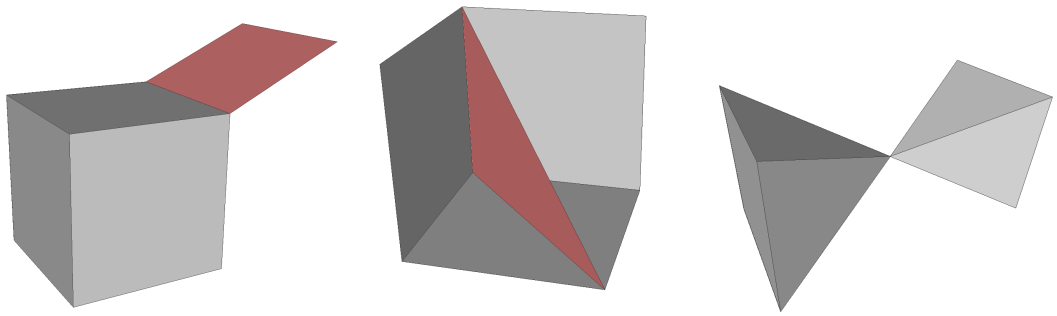
In this chapter, we briefly present the basic concepts related to quad meshes in such a way that, in the rest of the thesis, we can describe problems and challenges from a more technical point of view. We take some definitions from the survey about quad meshes by BOMMES *et al.* and the Section 2.1 of the thesis of BOMMES.

2.1 Definitions

The valence of a vertex is the number of its incident edges. Interior vertices with valence four will be called *regular vertices*, and otherwise, they will be called *extraordinary* or *irregular* vertices. Analogously, on the boundary, a *regular vertex* is characterized by a valence of three. The *star* of a vertex is the set of its incident faces and edges; the *star* of an edge is the set of its incident faces. We assume all our meshes to have a 2-*manifold* configuration, i.e., the star of any edge and of any vertex is always homeomorphic to either a disk, or a half plane. This means that any edge may be shared by either two incident faces, or just one incident face; and the set of faces connected to each vertex forms a single fan, i.e., there is no "bow tie" configuration (See Figure 2.1).

In a quad mesh, at any interior regular vertex, two pairs of opposite edges meet. Starting from an irregular vertex, any chosen edge can be followed, reaching another vertex. If that vertex is regular, then the opposite edge can be followed too (if it is not boundary), thus reaching a third vertex, and so on until an irregular vertex is eventually reached. This collection of edges is what we call a *separatrix*, i.e., a path of edges crossing regular vertices and connecting an irregular vertex to either another irregular vertex, or to the boundary.

Quad meshes can be classified into several classes based on the degree of regularity: A *regular mesh* is a mesh that can be globally mapped to a rectangular subset of a square tiling, although these meshes have a limited scope of applicability. A *semi-regular* quad mesh is obtained by gluing in a conforming way several regular 2D arrays of quads side to



(a) (b) (c)

Figure 2.1: Examples de non-manifold meshes. We observe an internal edge connected to more than two faces ((a),(b)). In (c) we see a "bow tie" configuration.

side. Each such regular submesh is called a *patch*, and the number of patches is assumed to be much smaller than the total number of faces. In a semi-regular quad mesh, all vertices that are internal to patches or lie along their boundary edges are regular. In contrast, only vertices that lie at corners of patches may be extraordinary. Semi-regular meshes represent the most important class of quad meshes in terms of applications. A quad mesh is *valence semi-regular* if most of its vertices have valence 4, and finally, a quad mesh is *unstructured* if a large fraction of its vertices are extraordinary. An unstructured mesh is obtained, for instance, from splitting each face of an arbitrary triangle mesh into three quads.

We consider *conforming meshes*, i.e., meshes in which any two faces may share either a single vertex or an entire common edge. This is a common assumption in geometry processing; but, unlike in general cases where real meshes are not conforming, quad meshes modeled by artists always are conforming because of the animation requirements.

The *Dual mesh* is a concept inspired in the dual for graphs; accordingly, the *dual quad mesh* is another mesh that we can construct as follows: Given a quad mesh $Q = (V, E, F)$, its dual is $Q^* = (V^*, E^*, F^*)$ where each vertex $v_i \in V$ is identified with a dual face $f_i^* \in F^*$, each edge $e_j \in E$ is identified with a dual edge $e_j^* \in E^*$ and each face $f_k \in F$ is identified with a dual vertex $v_k^* \in V^*$ (see Figure2.2). The connectivity of the dual is uniquely inherited by the primal. If, for example, two vertices in the primal are neighbors, so the corresponding faces will be in the dual mesh. The $4-$regularity of the faces in the primal translates into a valence regularity of the dual. Consequently, we can interpret each vertex of the dual mesh as the crossing of two dual curves (see Figure2.2$-b$). These dual curves, often called *poly-chords*, uniquely traverse bands of neighboring primal quads and induce the global connectivity of the quad mesh. While simple dual curves

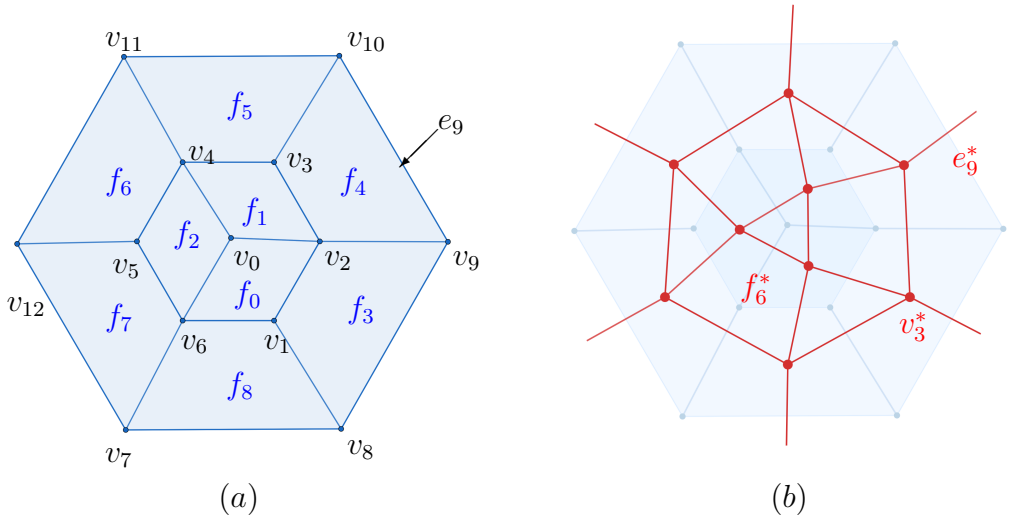


Figure 2.2: $a)$ A quad mesh Q is described by vertices v_i , edges e_i and faces f_i . In a pure quad mesh each face is four-sided while irregular vertices like v_0 are allowed to have a valence different from 4. $b)$ The dual Q^* of a quad mesh is an arrangement of curves where the face regularity translates into vertex regularity.

are usually preferred, even quad meshes which seem to be well structured often exhibit self-intersecting, long and complicated dual curves (see Figure2.3$-c$). However, since all vertices in the dual have a valence of 4, such a dual curve cannot end in the interior, i.e. each curve is either closed or crosses the boundary twice. This helps us understand why Lemma 2 is true.

The set of dual curves can be partitioned by an equivalence relation that gives us a better understanding of the global structure of quad meshes. This relation clusters *topologically parallel* curves into equivalence classes. More precisely,

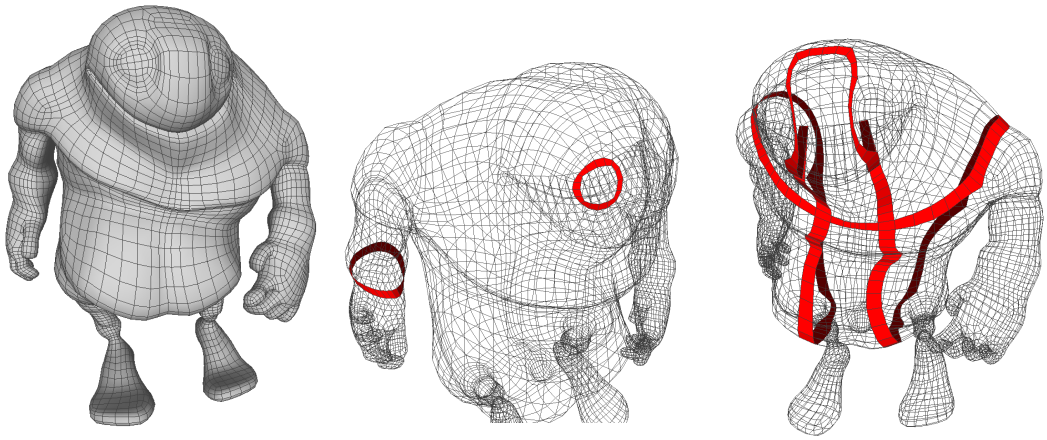
Theorem 1. *Let d_1 and d_2 be dual curves. The relation $\sim_p$ defined as follows:*

$$d_1 \sim_p d_2 \iff \textit{one is a transversal offset of the other} \tag{2.1}$$

is an equivalence relation.

Proof. Naturally $\sim_p$ is reflexive because we can consider an offset of 0 transversal edges and symmetric because, if $d_1 \sim_p d_2$, then there exists an integer n such that d_2 is a transversal offset n dual edges from d_1. Therefore, d_1 is also a transversal offset n dual edges from d_2, but in the opposite direction, i.e, $d_2 \sim_p d_1$. Finally, the transitive property holds because, if $d_1 \sim_p d_2$ and $d_2 \sim_p d_3$, then there exist integers n and m such that d_2 is an offset n dual edges from d_1 and d_2 is an offset m dual edges from d_3, hence d_3 is an ofsset $n+m$ or $|n-m|$ dual edges from d_1, i.e., $d_1 \sim_p d_3$. □

This corresponds intuitively to a "ladder" configuration (see Figure2.4), where each curve segment of d_1 forms a topological quad with two transversely intersecting dual



(a) (b) (c)

Figure 2.3: Poly-chords corresponding to the same quad mesh. We can think a poly-chord as a dual curve. (a) Quad mesh designed for animation. (b) Short dual curves. (c) Long dual curve crossing itself two times.

curves and a curve segment of d_2. It is clear this relation classifies topologically parallel curves, even more, the number of equivalence classes, i.e., the number of topologically different dual curves, is invariant under regular refinement[1] and, thus, encodes structural properties. We can, for example, have an alternative definition of the quad mesh base complex.

Finally, in the 3D modeling software context, we frequently use the term *edge ring* for quad meshes, which is quite close to our dual curve definition. An *edge ring* is a series of edges which are not directly connected, but share faces. In a quad, the only possibility is a sequence of opposite edges (Figure 2.5$-(a), (b)$), but this is just the sequence of edges crossed by a dual curve. Related to this concept, we also define the term *edge loop*, that is, a sequence edges directly connected, where each edge belongs to a different quad (Figure 2.5$-(c), (d)$). This sequence ends when an irregular vertex is found or we reach the boundary. Note that, in this terminology, a separatrix is an edge loop, and, analogously to dual curves, we can define parallel edge loops.

2.1.1 Base complex

A *quad layout* is a partitioning of an object's surface into simple networks of conforming quadrilateral patches. In the case of semi-regular quad meshes we can consider the quad layout as a kind of base structure, since the quad mesh is a regular refinement of it. This quad layout is not unique by definition and structurally does not differ from a quad mesh.

[1] A regular refinement splits each quad into four by joining opposite edges on their middle points. Given a quad, this already has a two dual curves crossing it. If we apply a regular refinement we will only add two dual curves parallel to the existing ones, that is, we will not modify the number of equivalence classes.

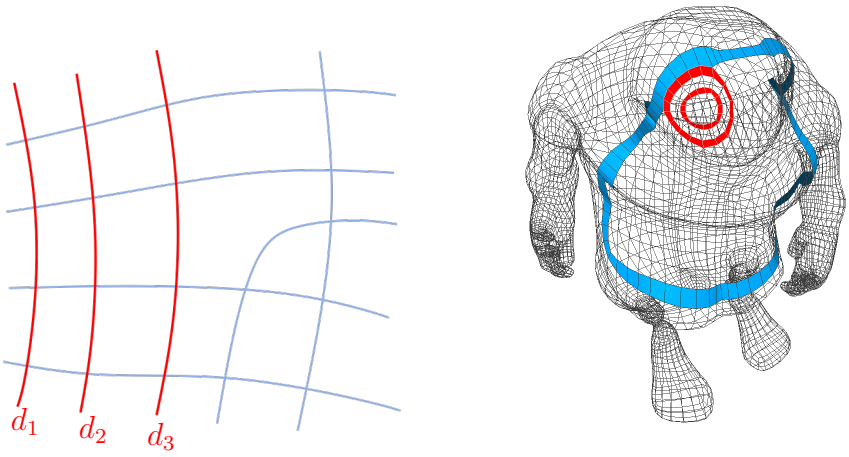


Figure 2.4: Visualization of dual curves. On the left, we see a topological representation of the dual curves space with $d_1 \sim_p d_2 \sim_p d_3$, i.e., parallel curves. On the right we also see two parallel curves (poly-chords) in red and a non-parallel curve in light-blue.

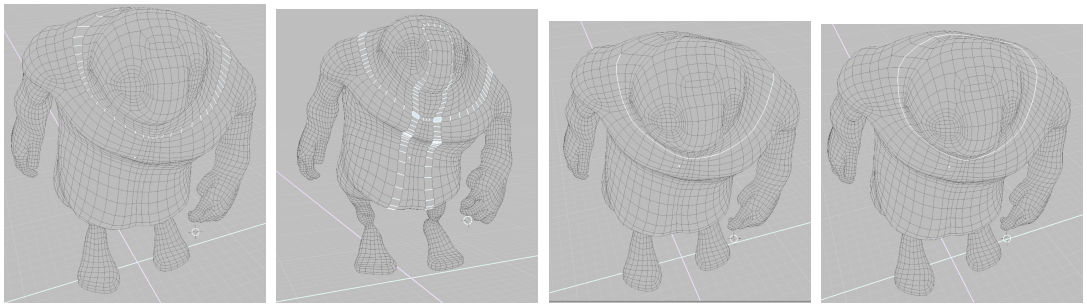
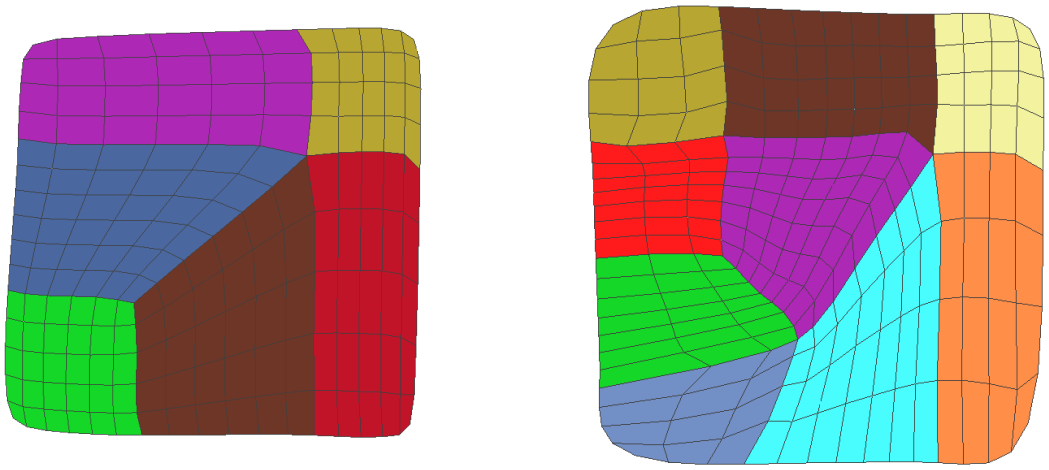


Figure 2.5: Edge Selection tools in Blender for quad meshes. $(a), (b)$ shows two edge rings. Compare with the dual curves in Figure 2.3. $(c), (d)$ shows two examples of an edge loop. Note that (c) is also a separatrix.



(a) (b)

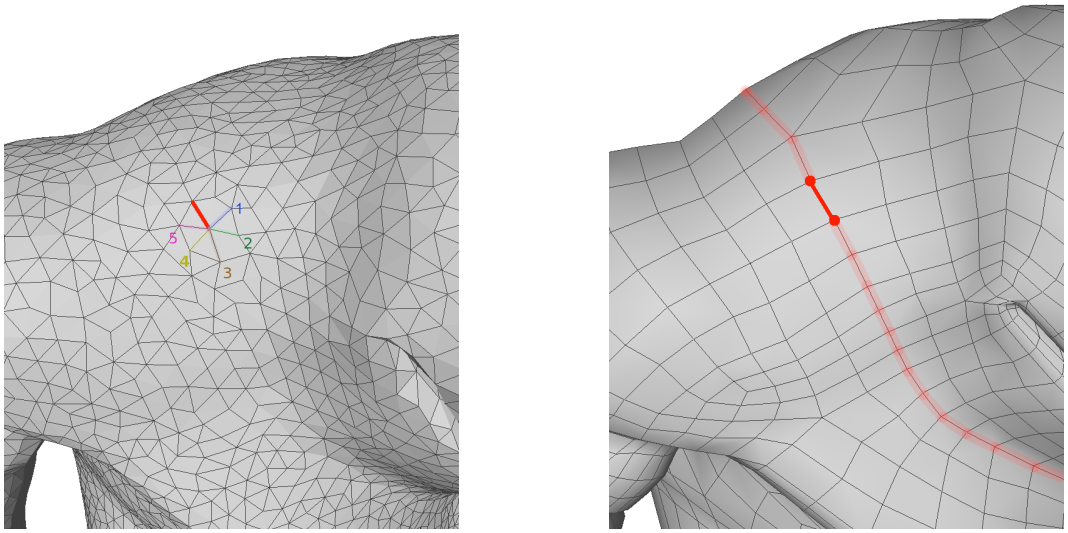
Figure 2.6: Visualization of quad layouts. Different patches are depicted with different colors. Note we have in both of cases two irregular vertices of valence 3 and 5 but different dual curves topology $a)$ Quad layout coarser. $b)$ Quad layout finer.

Rather, the difference is that patch dimensions are chosen such that the quad layout is coarse.

The *base complex* of a quad mesh Q, denoted as $\mathcal{BC}(Q)$, is the coarsest quad layout obtained from a subset of the mesh that has the same boundary and set of irregular vertices. In the primal setting, $\mathcal{BC}(Q)$ is constructed by tracing all separatrices and then removing all untraced arcs. Accordingly, the connection between irregular vertices strongly influences how many patches are required, an observation which is very helpful when optimizing the topology of quad meshes (see Figure2.6). Looking at the dual construction, we obtain the dual of the base complex $\mathcal{BC}(Q)^*$ of Q by choosing exactly one representative of all topologically parallel curves classified through $\sim_p$ and each intersection of two such representatives generates one patch P_i in the primal, i.e., the base complex $\mathcal{BC}(Q)$.

2.1.2 Edge flow

This is a term very common in the 3d modeler community, but with a vague definition. Sometimes this term means the modeling practice of ensuring that edges follow the curvature and features of the model, in particular, the human anatomy in character modeling (muscle structure). Other times, *edge flow* represents the structure of the model itself. This structure is seen as edge paths (flow) and it is a sort of path guide on how the model is formed. Certainly, this concept does not make sense in a general polygonal situation, since the edge flow's direction is undefined (Figure 2.7$-(a)$), but, in the context of quad meshes, due to the existence of the dual curves and the base complex we can define precisely what would be the *edge flow*. First, we note that, due to the existence of two dual



(a) (b)

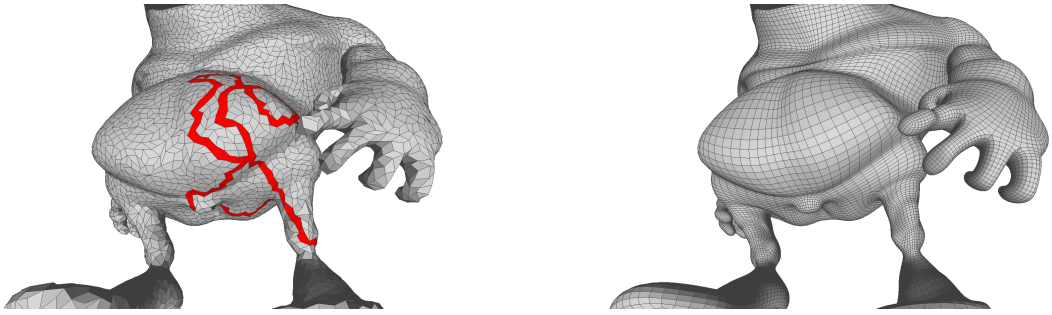
Figure 2.7: (a) In a polygonal mesh, given the red edge we cannot define unequivocally the next edge to be followed. We have 5 options, and we don't know what direction choose. (b) In a semi-regular quad mesh each edge has a flow direction well-defined. This is the direction of the dual curves associated to the adjacent faces.

curves crossing on each quad, we can define unequivocally the flow of an edge as the *edge loop* associated to that edge. Bearing that in mind, we define the *edge flow* of the quad mesh Q as the set of dual curves of the base complex $\mathcal{BC}(Q)$. Note that, this is a global concept, depending, by definition, of base complex of the mesh Q, and therefore, it is a topological concept. Note also that the edge flow becomes important when this is well-organized and has few similar flows. In this case we have a *good edge flow*. A *bad edge flow* could be obtained, for instance, in a unstructured quad mesh (Figure 2.8). .

2.1.3 Quadrangulation challenges

Quad-remeshing or retopology is a challenging open problem. It consists in generating a quad mesh given another mesh, typically a triangle mesh, in such a way the new asset preserves the geometry as much as possible. The difficulty increases because the connection between the geometry of a surface and its ideal quad mesh is weak or nonexistent, primarily application-dependent. For our purposes, if we have a character modeled for animation, its connectivity should have edge loops on its articulation and the entire topology should be optimized to reduce skinning deformation artifacts; such properties are impossible to automatically extract from static meshes([11]). Figure2.9 shows two automatic quadrangulations with state of the art methods for the same model of Figure1.1.

Since quad-remeshing is hard to do, we propose blending quad meshes to partially solve the need for the films and games industry to build quad assets from scratch. Such blending must be done preserving as much as possible the initial connectivity. In this



(a) (b)

Figure 2.8: Good and bad edge flow examples. (a) Dual curve in red for an unstructured quad mesh. We observe that this curve does not follow the muscle structure of the model and is crooked. The edge flow for this mesh is a good example of a bad edge flow. (b) The same model like in (a), but with a well-designed mesh with a good edge flow. Note the regularity, organization and better anatomy flow of the mesh.

way, modelers could perform a process that does not destroy previous work. We will try to explain in this section why blending on quad meshes poses extra challenges over its triangle counterpart.

Lemma 1 demonstrates the importance of irregular vertices in the generation of quad meshes.

Lemma 1. *Given a conforming and closed quad mesh with all vertices regular, then the genus of the surface is 1.*

Proof. In a conforming regular quad mesh without boundary the relation between the number of edges and faces is $|E| = 2\,|F|$, since each face is adjacent to exactly four edges and each edge is shared by exactly two faces due to the conforming property. Furthermore, with an analog argument we know that $|E| = 2\,|V|$ because each vertex has valence four and each edge is adjacent to exactly two vertices and consequently $|F| = |V| = 1/2\,|E|$. The Euler characteristic χ relates these quantities for a closed polyhedron in the following way to its genus g:

$$|V| - |E| + |F| = 2(1 - g)$$

then, because of our relations, $g = 1$. □

Even for genus 1 surfaces, modelers often introduce irregular vertices if the surface is more complicated than a torus, like e.g., a coffee cup. Another consequence is that we never could construct a regular mesh in the common industry case of a conforming quad mesh of genus 0, i.e., we always have a semi-regular quad mesh.

The next fact we should note is that locations and valences of irregular vertices have a global effect over the quadrangulation. Lemma 2 reveals the global nature of quadrangulations, condition not present in triangle mesh generation.

Lemma 2. *A planar and non-intersecting polygon can be quadrangulated if and only if the number of edges is even.*

Proof. The necessity of an even number of edges is consequence of the Euler formula. Let P be a planar and non-intersecting polygon and Q an arbitrary quadrangulation of P. Let F be the number of faces, E_b the number of boundary edges, and E_i the number of internal edges of Q. Since every quad has four edges and every internal edge is shared by two quads, these variables are related as $4F = E_b + 2E_i$, meaning that E_b must always be even. The sufficiency of the condition is a corollary of a more general algorithm that constructs a quadrangulation (perfect matching) from a triangulation presented in [12]. Here we use the known theorem about the existence of a triangulation for a planar polygon [13]. □

Lemma 2 can even be generalized to non-planar polygons and we have, in particular, that patches from semi-regular meshes must satisfy the parity of the number of their boundary edges. This Lemma has important consequences for the design of quad-meshing algorithms. In Section 4.3 we will face a topological consistency problem due to this requirement. In order to intuitively understand the intrinsic consistency constraint of quad meshes, it is helpful to examine Figure 2.6. We notice the same number and type of irregular vertices but in different locations and we have different quadrangulations of the same patch. When we construct the dual curves we note two different configurations and that is why the quadrangulation changes: the topology of dual curves determines a unique quad mesh, except for parallel curves (regular refinements) and this also can be expressed in the primal viewpoint. In this case, the base complex defines a unique quad mesh unless regular refinements are introduced.

In our problem, performing blending on quadrilateral meshes preserving as much as possible the initial connectivity demands working "locally" but we have just seen that we must manipulate quad meshes taking into account their global structure. In fact, local changes in the structure usually propagate globally across the whole mesh. Furthermore, preserving the edge flow from the inputs impose several conditions on the global structure and the placement of irregular vertices.

Finally, while it is generally a challenging task to define stable boolean operations, their result can be defined precisely in the context of triangle meshes [16]. 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[2]. Hence, the single intersection between two quadrilateral

[2]We can split each quad in two different ways.

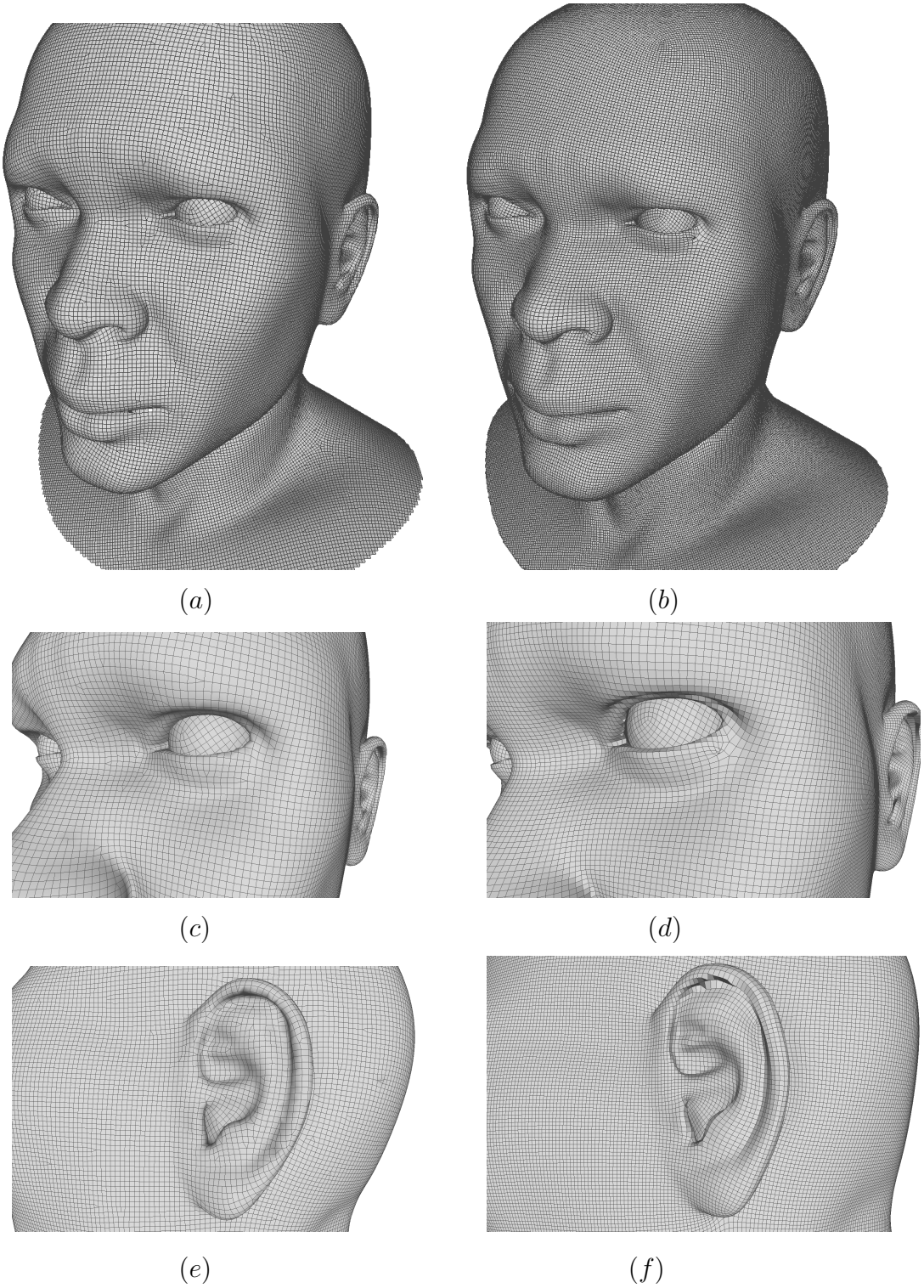


Figure 2.9: Automatic quadrangulations for the model in Figure1.1 with the methods from [14] and [15]. $a)$ Quad-dominant remeshing with the Instant meshes method[14]. $b)$ Quad-remeshing with the QuadriFlow method[15].$c)$ Left eye view of the quadrangulation with [14]. $d)$ Left eye view of the quadrangulation with [15]. $e)$ Left ear view of the quadrangulation with [14].$f)$ Left ear view of the quadrangulation with [15].

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 thesis 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 define 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.

Chapter 3

Literature review

In this chapter, we review previous works relevant to our goal and describe important techniques that we use to achieve our stable blending operations between quad meshes. There are several areas of research related, but we concentrate our discourse on those which inspired the present work. We begin with a summary of boolean operations on triangle and mixed meshes. We also overview methods for automatic quadrangulations, interactive tools for quadrangulations and quad-meshing techniques for single patches.

Before we starting with the academic works, among commercial software packages, only the Modo suite [17] 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 a quad-dominant mesh, obtained by subdividing the coarse quad representations, and performing the boolean operation over the subdivided results, but considered as triangular meshes in the intersections. In this regard, 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 3.1 shows a comparison between the quad-dominant representation obtained by MeshFusion and the result of our method. Several video-tutorials[1] about Modo MeshFusion are avalaible on the internet for better comparison.

3.1 Boolean and Composing Operations

Boolean or set-theoretic operations are a natural way of constructing complex objects from simpler ones, and is the basis of Constructive Solid Geometry, where primitive solids are usually implicit objects, which naturally support such operations. However, we use the Boundary Representation *B-Rep* paradigm where surfaces are usually represented by

[1] An example: `https://www.youtube.com/watch?v=sWBpdElTeV0`

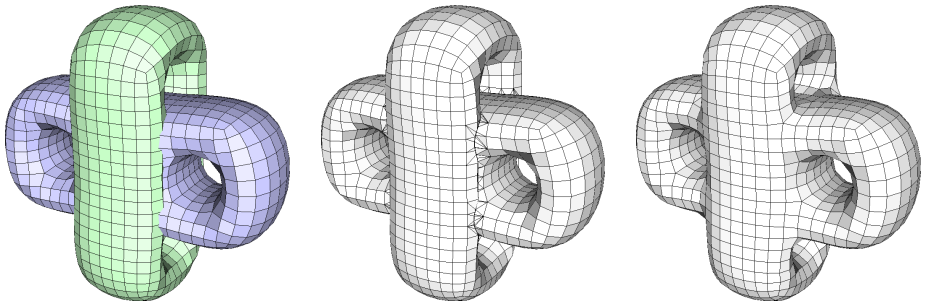


Figure 3.1: A comparison of our proposal and MeshFusion([17]): the difference in the intersection portion of the mesh is noticeable.

discrete structures such as meshes, which require considerably more effort when used as operands of set-theoretic operations. Since B-Rep is very popular in CAD and computer graphics applications in general, several techniques for computing such operations on meshes have been devised over the years.

This is a non-trivial problem, especially for complex B-Rep models as it requires many intersection tests, separating the final surface into pieces and constructing new surfaces out of these pieces. Computing the intersection curves between the input models is a central task and quite difficult if we demand accuracy. Exact arithmetic and complex techniques were used in several works addressing the issue, as in the Computational Geometry Algorithms Library (CGAL[18]) that supports robust Boolean operations on Nef polyhedra[19], which is considered a seminal reference, despite its high memory cost.

BSP-based[2] methods have been shown to be highly effective for computing boolean operations on meshes. With a careful design of predicates, a robust method was reported in [20]. The authors use an exact geometric computation approach with fixed-precision instead of arbitrary-precision arithmetic. PAVIĆ *et al.* [21] extended this technique improving its performance with an adaptive octree and also fixed-precision arithmetic. Besides, due to the BSP representation, these algorithms are able to work with polygonal meshes, not just triangle-based. However, the output mesh is completely re-tessellated. This is problematic in many contexts like ours.

More recently, DOUZE *et al.* [22] developed *QuickCSG*, a multiple polyhedral input system for boolean operations which is very fast and robust. They propose a new vertex-centric view of the problem with good results. Basically, they express a boolean solid operation using a boolean-valued function over n boolean inputs, $f : \{0,1\}^n \longmapsto \{0,1\}$. They denote $\mathbb{I}_i(x) \in \{0,1\}$ the indicator function of polyhedron P_i whose value reflects whether a point $x \in \mathbb{R}^3$ is in polyhedron P_i's inner volume. The indicator function $\mathbb{I}_f(x)$ of the final solid P_f can then be computed as $\mathbb{I}_f(x) = f(\mathbb{I}_1(x), \cdots, \mathbb{I}_n(x))$. With this

[2]Sigla de *Binary Space Partition*, ou Partição Binária do Espaço.

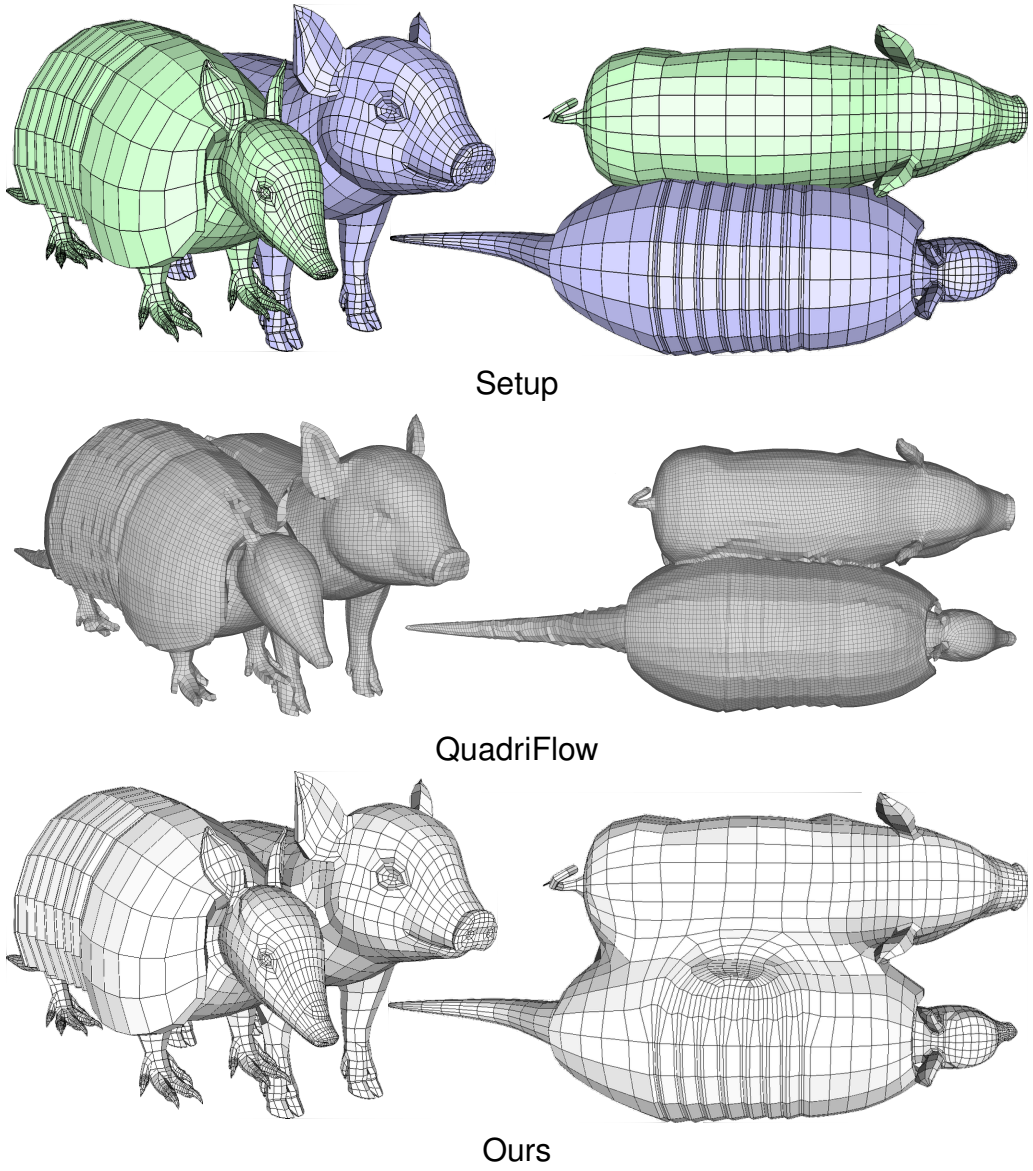


Figure 3.2: A simple way to tackle the problem of composition might be to transform the quad mesh into triangles first, perform the boolean operation, and finally use a quad-meshing algorithm to obtain a sound quad mesh. However, this approach will inevitably modify the entire mesh. This result is far 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. The solution with the QuadriFlow method[15] require to re-mesh entirely the results of the boolean operation and original connectivity is lost, in particular, features like the eyes of the pig or the armadillo. Our system efficiently preserves the connectivity and is capable of blending two different connectivities

approach and a KD-tree[3], they focus all classification and subdivision efforts on producing the final output vertices, excluding higher order primitives or intermediate vertices. Then, the size and depth of the KD-tree no longer depend on the size of the inputs but on the size of the output model. This improves performance. The reader can see more details in the project page of the work.

ZHOU *et al.* [16] developed a robust boolean system for triangle meshes using the generalized Winding Number concept restricting the problem to the class of meshes with a piecewise-constant winding number or PWN meshes. The computation is exact since they employ the CGAL exact arithmetic kernel. We base our boolean operation between the triangular parts in our method (see Section 4.1) on this work because of the exhaustive validation of this method with the online 3D printing repository of 10000 triangle meshes Thingi10K. There is an implementation of this algorithm in the LIBIGL library.

As explained in the introduction, our approach leverages on the works on the modeling-by-composition paradigm. These approaches are more interested in achieving an effective composition with an easy-to-use tool than calculating a boolean operation precisely. In this sense, *blending* is a more appropriate term to refer to these composition operations. The majority of these methods do not compute the intersection curve like in [23–25] and instead, they use ideas related to the iterative closest point (ICP) algorithm. Other works address the problem of composition like a part-to-target operation [26, 27] using more sophisticated techniques. Moreover, the idea of trying to limit the modification on a mesh when performing boolean operations [28], or repair the result [29], has been already explored. Finally, no solutions had been suggested until now to compose quad meshes smoothly and with the result being a quad mesh as in the proposed approach.

3.2 Automatic quadrangulations

A quadrangulation of a 3D polygon surface can be constructed by computing the tessellation directly or by computing a quad patch layout. For an extensive survey on quad-meshing, readers are referred to BOMMES *et al.* [9] and CAMPEN [30]. In this section, we briefly review the approaches most related to our contribution.

Several quadrangulations methods were proposed in the last years. These are comprised by **global parameterization**, **field tracing** and **quad layout generation** methods. Among the methods based on global parameterization we have the work of BOMMES *et al.* [31] that transformed the quadrangulation problem into a mixed-integer problem; KÄLBERER *et al.* [32] that developed a method to compute a global continuous parameterization for an arbitrary given simplicial $2-$manifold. CAMPEN *et al.* [33] introduced a method to generate an integral global parameterization by finding good quality quantizations on closed, orientable $2-$manifolds. More recently, FANG *et al.* [34] proposed

[3] Árvore *k*-dimensional.

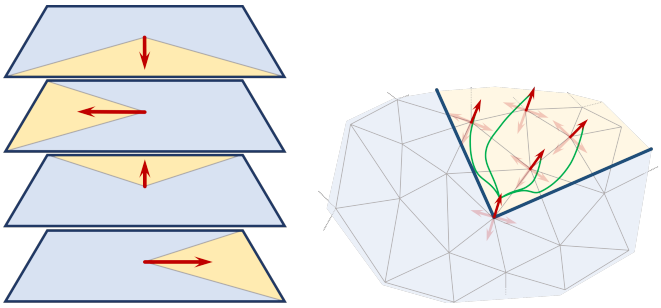


Figure 3.3: Construction of a cross-field. (a) Generation of the four sides of the cross-field. (b) Discretization like in CAMPEN *et al.* [39]

a hybrid technique that combines the theoretical guarantees of Morse-based approaches with the parameterization-based methods. The use of a four-dimensional periodic vector field allows an increase in the robustness of the parameterization part. Field tracing methods have given good results over the last decade. Starting with the work of PALACIOS e ZHANG [35] about cross-fields or 4-RoSy vector fields, we can mention PANOZZO *et al.* [36] that introduced the frame field, a generalization of cross-fields that allows generating anisotropic and non-uniform quad meshes; JIANG *et al.* [37], that improves the generation of frame fields by designing a non-Euclidean metric based on the input constraints; and, the state of the art: the Quadriflow method [15] (based on the Instant meshes method [14]). On the other hand, diverse approaches to quad layout generation exist. TARINI *et al.* [38] extract a quad layout by simplifying a given cross-field. CAMPEN *et al.* [39] exploited the relation between the dual graph and its quadrangulation, with the quad layout given by the intersection of dual loops. Also DONG *et al.* [40] used a spectral decomposition of the Laplacian to construct a Morse-Smale complex that is guaranteed to be formed by 4-sided patches. Some works were also aimed at character animation, for instance,[11, 41] consider the deformation affecting a mesh during an animated sequence to generate a quad layout that remains good for all animation frames. On the other hand, while these approaches were successful on several quantitative metrics (like singularity placement and coarseness), in production pipelines it is still necessary a more art-controlled quad generation process.

Since in our pipeline we use a cross-field like in CAMPEN *et al.* [39], it will be useful to describe briefly how we constructed that field.

Cross-field generation method

Following KÄLBERER *et al.* [32], 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 orienta-

tions of the cross field. This process creates M_4, which is a stratification of the original manifold surface M (Figure 3.3$-(a)$). Space M_2 is the quotient space of M_4 obtained by identifying pairs of opposite directions. Hence M_2 is composed only by two sheets. Each sheet encodes a line-field.

We discretize M_4 for tracing following the approach proposed by CAMPEN *et al.* [39]. 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.

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.

Finally, we have seen that automatic quadrangulation doesn't get the desired good topology that expert modelers give to their creations and, for that reason, our approach provides a solution to this problem by creating the model blending previous modeled parts created by artists preserving as much as possible the initial topology. However, even in the case that automatic quadrangulation methods advance to cover current impossible cases like in Figure1.1 our modeling by composition approach to compose quad meshes still will be of great help to modelers since our restriction of preserving as much as possible the initial topology allows reusing previous expensive work. Using an automatic method will inevitably modify the entire mesh due to the global nature of quadrangulations as we had seen. In Figure 3.2 we illustrate this with an example and compare it with our proposal.

3.3 Interactive Quadrangulation tools

The combination of the global behavior of quadrangulations with the non-interactive nature of an automatic method makes the tuning of parameters an unintuitive and time-consuming task. Therefore, many approaches have considered the issue of helping a manual quadrangulation process leaving most of the control to the final user. BESS-MELTSEV *et al.* [42] developed a technique for generating quad meshes starting from an input 3D curve network, possibly created by the user; with this approach, geometry and topology are defined based on flow-lines set by pairs of segments on a closed 3D path.

Inspired quadrangulation [43] 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 pro-

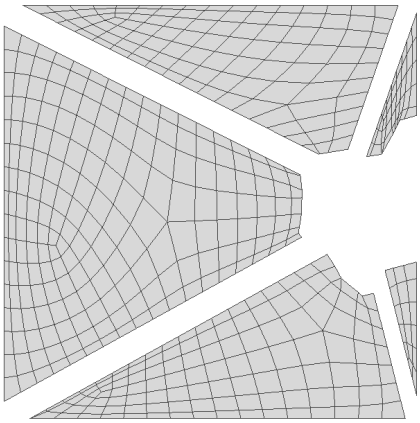


Figure 3.4: An example of a quadrangulation for adjacent patches using the Takayama's method[45].

vide 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 [44]. These methods provide lower-level local operators, and they can be integrated with practices of the previous class to fine-tune the mesh topology.

3.4 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 [7, 45, 46]. 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 [47–49] and finally automatically quadrangulated. The present work also requires quadrangulating patches defined by their sides (see Figure 3.4). In [7, 45], a set of manually designed patterns are expanded to tessellate arbitrary polygons with up to 6 sides. More recently, MARCIAS *et al.* [8] 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. Another possibility is to use filling patterns generated procedurally as in PENG *et al.* [46]. 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. As explained in Chapter 1, we expect to produce quadrangular patches that are relatively small with respect to the input meshes, and, for that reason, we propose using Takayama's method [45]. 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.

It is important highlight that we share some ideas for our cross-field generation step (subsection 4.2.3) with the PENG *et al.* [46] approach (like the classification between concave and convex corners and the emanating directions), but unlike that work, 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 to a traced subdivision that blends the flow among the existing quadrangulations smoothly.

Chapter 4

Blending of quad meshes

In this chapter, we describe in detail our proposal to compose quadrilateral assets from other quad meshes. In particular, we address this composition in the case of quad meshes designed for animation, which present good edge flow and, until now, were impossible to generate automatically. In this way, we can supply a need in the film and game industry.

Our method assumes two input quad meshes, more usually, models produced by artists. The two main objectives are: (1) perform the composition of these models guaranteeing a pure quad mesh as a result, and (2) preserve as much as possible the connectivity of the original quad models. In order to achieve these goals, we proceed to tackle the problem "locally" around the intersection curve of the inputs. All the work will be done in a neighborhood of this curve. With that purpose, we compute a quad patch layout of the meshes to be composed and work on the intersecting patches (see Figure 4.1$-a$). Our idea is to define a new set of patches on these regions close to intersection curves, quadrangulating each patch independently. As we have seen, we have to deal with the global effects of our "local" modifications. We transform these global-local relations into an Integer Programming formulation that solves our requirements efficiently.

We based this section on our paper *"QuadMixer: Layout Preserving Blending of Quadrilateral Meshes"*[50].

In essence, our entire pipeline can be expressed as:

- We first compute a patch decomposition of the quadrilateral meshes by using a simple *motorcycle graph* [51] tracing algorithm (see Figure 4.1$-a$) or solely emanating separatrices from irregular vertices.
- We split each quad into two triangles, and we perform the boolean operation using the implementation of [16] (see Figure 4.1$-b$).
- 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.

At this stage the mesh can be divided into two sets: a quadrilateral mesh $\mathcal{Q}^0$ and a triangle mesh $\mathcal{T}$ (see Figure 4.1$-c$) which share a common boundary.

- We smooth these internal patches to generate a fair geometry surface, which can be quadrangulated nicely. 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 4.1$-d$). This step uses the definition of a cross-field [52] and applies a tracing algorithm [39].
- 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 4.1$-e$). The number of subdivisions along the border sides of $\mathcal{P}$ are constrained to match the corresponding subdivisions on $\mathcal{Q}^0$.
- We quadrangulate each patch using the data-driven approach proposed in [45] (see Figure 4.1$-f$) obtaining a new quadrangulated mesh $\mathcal{Q}^1$. The union of $\mathcal{Q}^1$ and $\mathcal{Q}^0$ provides the final result.

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 4.2). 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, such that no two consecutive edges limit the same quad. 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 simultaneously 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* [51]. 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 [38, 53]) is not useful in this context since it might modify the original edge flow designed by the artist.

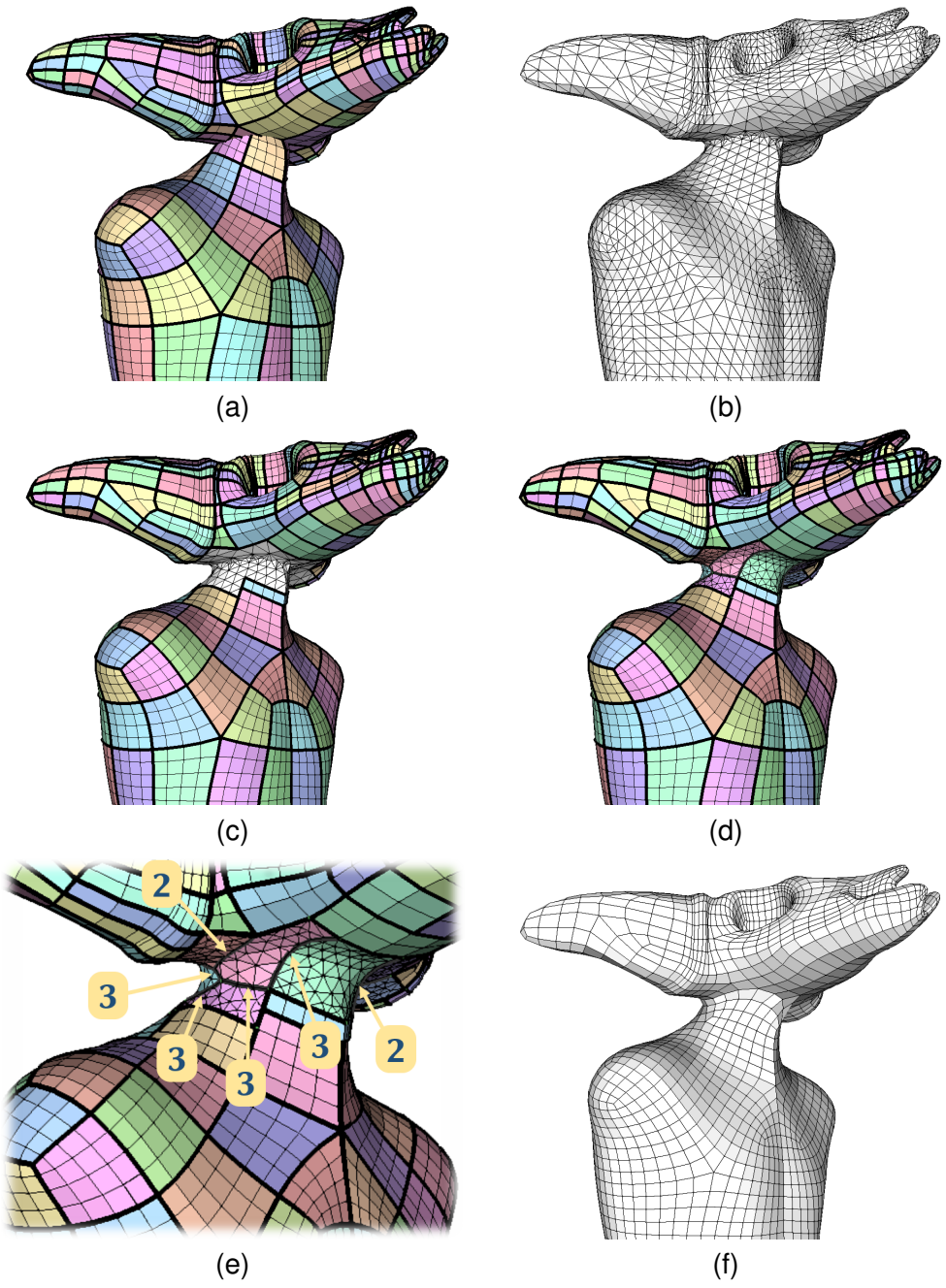


Figure 4.1: An overview of our processing pipeline: (a) Given two separate quad meshes we compute an initial patch layout for each; (b) then quads are split to form triangular meshes, which are then combined. (c) We update the patch layout for the patches that are affected by the boolean operation. (d) 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.

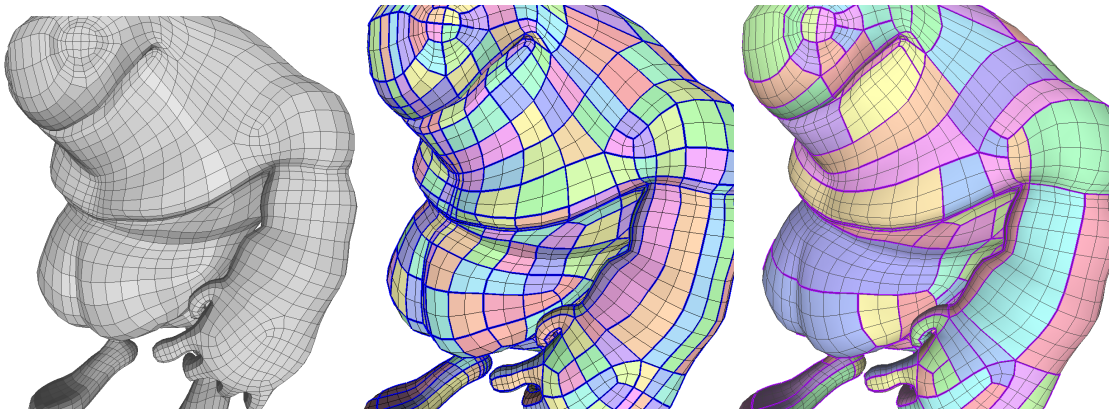


Figure 4.2: On the left: the input quad mesh; in the center: the patch layout with separatrices; on the right: the patch layout typical of a motorcycle graph.

4.1 Optimal Patch Retraction

Given two pure quadrilateral meshes $\mathcal{Q}^A$ and $\mathcal{Q}^B$, along with their original quad patch layouts (Figure 4.3$-a$), we start by splitting each quad element along its smaller diagonal[1] to transform $\mathcal{Q}^A$ and $\mathcal{Q}^B$ 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 [16]. 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 $\mathcal{Q}^A$ and $\mathcal{Q}^B$ that contain triangles not changed in the boolean operation (Figure 4.3$-b$). Then we consider, as 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 neighboring 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 length of the retracted patches (Figure 4.3$-c$). At this point, we have a collection of unorganized quads that we need to assemble into patches by building a new layout.

The idea is to favor 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 unorganized quads, the largest inscribed rectangle composed only by quads not associated with any entirely preserved patch (Figure 4.3$-d$). Specifically, we use the *largest rectangle in a histogram* (see [54], 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

[1] In this way, we obtain a more isotropic triangulation.

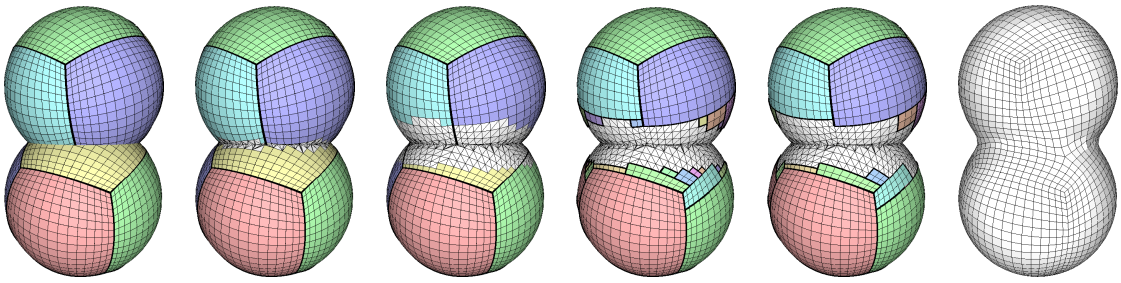


Figure 4.3: Patch layout retraction: (a) The initial patch layout of the two meshes. (b) The two quad meshes are split into triangle meshes to perform the boolean operation, and then the triangles are clustered to recompose the original quads when possible. (c) Original patches are retracted to maintain 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.

a number of forming quads below a given threshold (Figure 4.3$-e$). We set this threshold to a fraction of the average area of the current patches.

At the end of this step, we obtain a new quad-only mesh $\mathcal{Q}^0$ 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 $\mathcal{Q}^0$ coincides precisely with the boundary of $\mathcal{T}^0$.

4.2 Patch Subdivision

We now need to transform the triangulated portion of the surface $\mathcal{T}^0$ into a quad mesh $\mathcal{Q}^1$ that, once attached to the preserved quadrangulation $\mathcal{Q}^0$, 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 $\mathcal{Q}^1$, the number of subdivisions of $\mathcal{Q}^0$ 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 [31]. 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. [45] 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

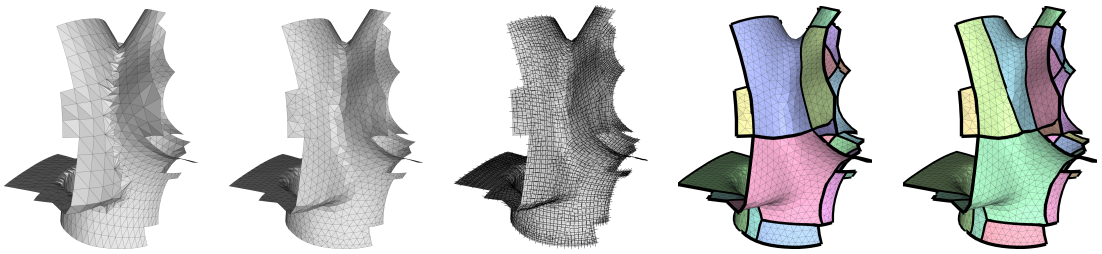


Figure 4.4: Patch tracing procedure: (a) The Initial patch; (b) The re-meshing step; (c) Cross field computation; (d) Concave corner tracing; (e) Final splitting of the patches to match the requirements.

requirements.

4.2.1 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 $\mathcal{Q}^0$ 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, for instance, a desired edge flow. 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 (see Figure 4.4) :

- 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 [55].

- We compute a smooth cross field that conforms to the boundary using the poly-vector field smoothing [52]. 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 [39].

- We iteratively re-apply this step for any sub-patch not yet respecting the conditions imposed by the constrained quadrangulation algorithm.

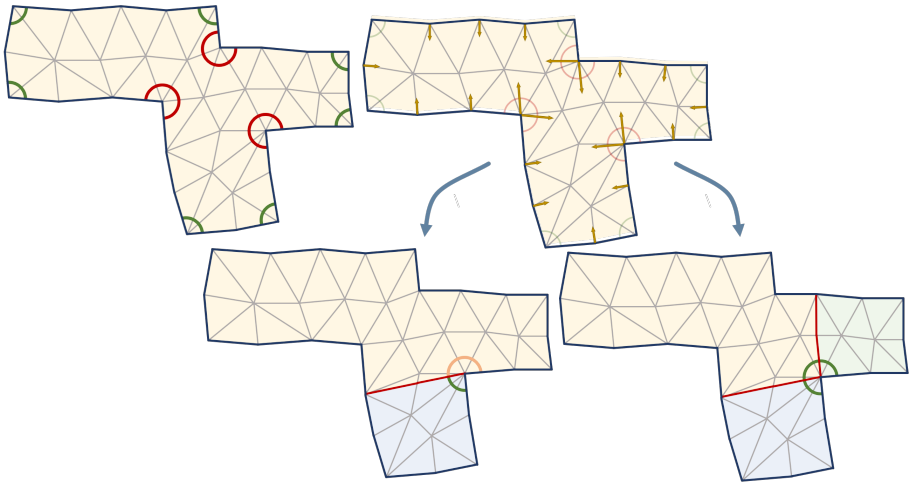


Figure 4.5: On the top left the initial corner classification: we mark each angle either as convex or as concave; on the top right we show the directions stemming from concave vertices that we use for splitting the patch in quads; on the bottom we show how we can split the concave corner into a flat and a convex corner using a single trace (left), or into three convex corners using two traces at the same time.

4.2.2 Patch configuration

After computing a cross-field on the triangle mesh $\mathcal{T}^0$ (for a more exhaustive description of cross-fields see [56]), 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 4.5 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 4.5): 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 4.5) 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:

- 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.

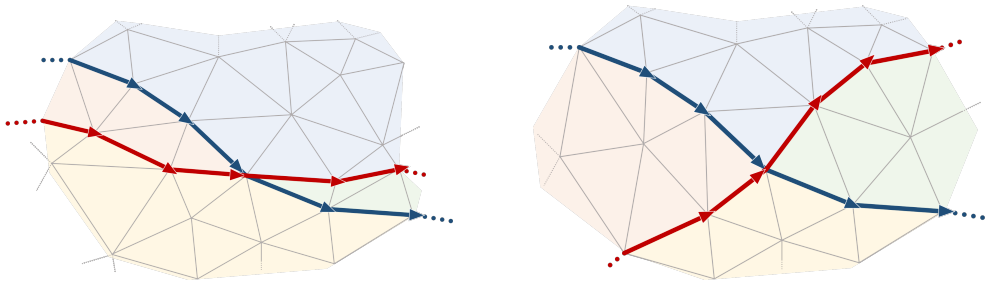


Figure 4.6: Tangential (left) and orthogonal (right) path intersections.

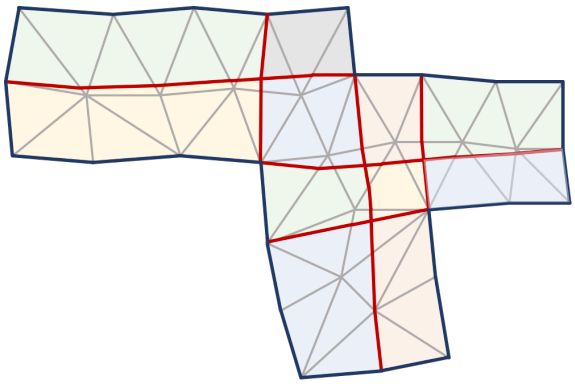


Figure 4.7: Example of our technique to trace concave vertex

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

4.2.3 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 [39]. We recap the tracing method in Section 4.2.1. In our experiments, we always have accurate results and, thus, we do not use more sophisticated trace strategies (e.g., the ones described in [57] or [58]). 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 [39]. 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 4.6 shows the difference between the two types of intersections.

4.2.4 Concave vertex tracing

We trace the polylines that split the patches, choosing first the ones that have an endpoint 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; (iii) 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 (see Figure 4.7). This condition is necessary to make the procedural quad mesh generation of [45] 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.

4.2.5 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 $\mathcal{T}^0$.

4.3 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:

1. A patch admits a quadrangulation only if the sum of boundary subdivisions is even (Lemma 2).

2. The subdivisions on the boundaries are constrained to match the preserved quadrangulated mesh.

3. 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. The ideal subdivision would be $\lceil \text{side length}/\text{average edge size} \rceil$, where $\lceil\ \rceil$ is the ceil

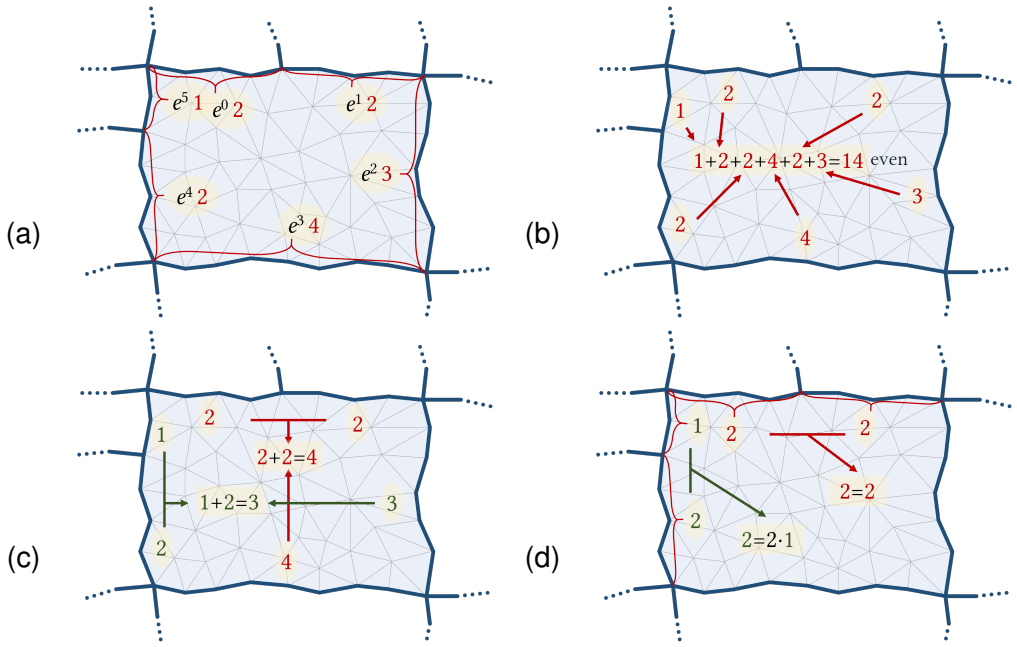


Figure 4.8: An example, with actual values, of the optimization procedure inside a patch. In (a) each side is divided into sub-sides, considering all incident T-junctions of adjacent patches (numerical values in red); in (b) we show that the sum of all the edge sizes must be even; in (c) we show how 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.

function. Then we propagate that value on internal patches, and we smooth between adjacent patches in order to discourage abrupt changes of resolution.

4. To increase the *regularity* of the tessellation, we favor the equality of opposite edges for every 4-sided patches.

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 4.8$-a$.

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

$$\begin{aligned} \min \quad & \sum (e_i - \hat{e}_i)^2 \\ \text{s.t.} \quad & e_i \geq 1 \end{aligned} \tag{4.1}$$

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.

$$e_i = q_i \quad \forall e_i \in B \tag{4.2}$$

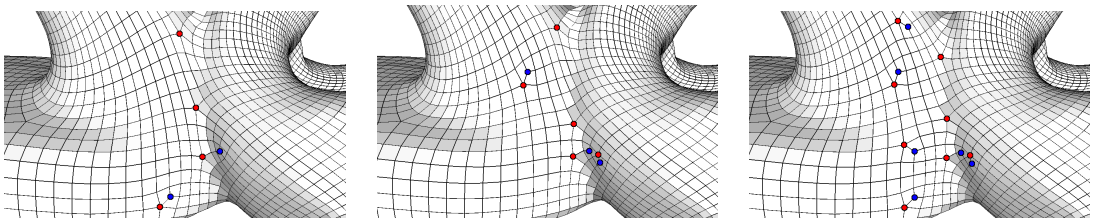
As previously stated, a quadrangulation is only possible if the sum of its subdivision is an even number. Hence, for each patch P_k, we include an additional linear constraint:

$$\begin{aligned} \textstyle\sum e_j &= 2n \quad \forall e_j \in P_k \\ n &\geq 1 \end{aligned} \tag{4.3}$$

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:

$$\min(\sum_{e_u \in S_0} e_u - \sum_{e_w \in S_2} e_w)^2 + (\sum_{e_u \in S_1} e_u - \sum_{e_w \in S_3} e_w)^2 \tag{4.4}$$

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 4.1 and 4.4. The effect of this parameter on the final quadrangulation is quite intuitive (see Figure 4.9). 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 minimize the equation 4.1 using the $L1$ norm to speed up the entire process, in such case, both equation 4.1 and 4.4 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 ~ 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.



$\alpha = 0.01$ $\alpha = 0.5$ $\alpha = 0.99$

Figure 4.9: The regularization term governs the distribution of singularities and the isometry of the tessellation.

4.3.1 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

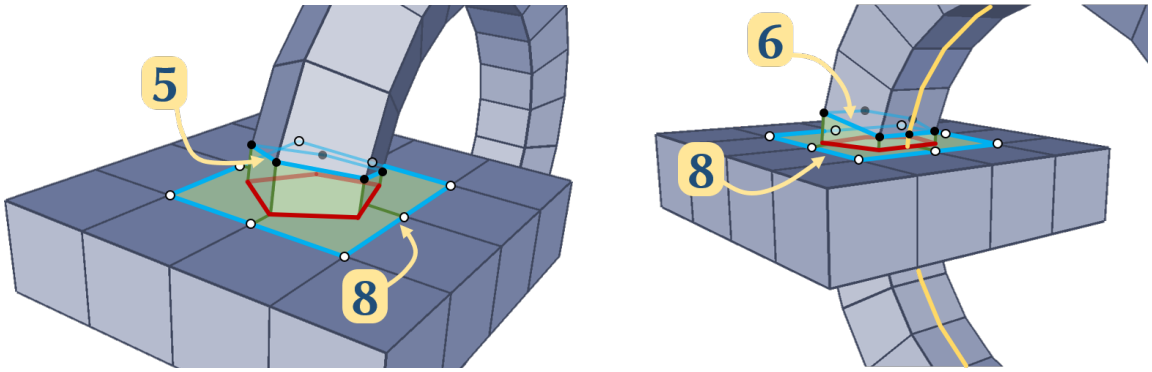


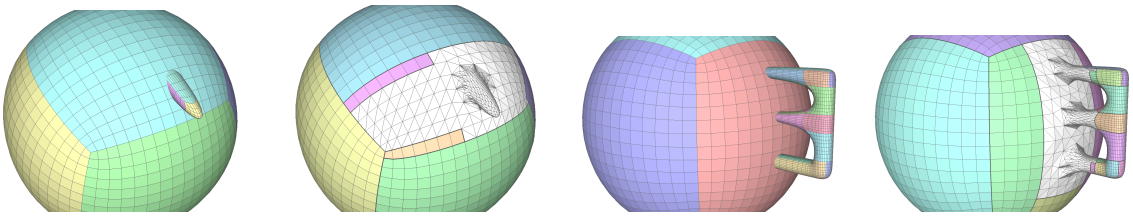
Figure 4.10: A torus with a pentagonal section intersects a block (left). The patch surrounding the intersection curve has a boundary with an odd number of sides (5 on the torus and 8 on the block) and therefore cannot be quadrangulated. Refining a polychord makes the boundary even (6 and 8 sides) and the patch become quadrangulable.

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. In order to better understand this, imagine we have one connected component on the triangulated part. This region is isomorphic to (1) a triangulation of a closed planar polygon or (2) a triangulation of a planar polygon with holes (see Figure 4.11). In the first case the boundary has an even number of edges because this was also the boundary of a quadrangulated region (the quads removed). In the second case we can face the situation of and odd number of edges at the boundary. Lemma 3 statements that we can never quadrangulate that region.

Lemma 3. *Given a planar polygon O with a holes. If O has and odd number of edges, then O can not be quadrangulated.*

Proof. We can convert the polygon O into a closed polygon without holes by making cuts from each hole to the boundary. It is always possible to make these cuts without intersection. Consider the new polygon C formed by a circulation of the edges passing through the cuts twice. As we passed twice by the cuts we did not modify the parity of the final boundary, i.e., the final boundary has also an odd number of edges. By Lemma 2 C can not be quadrangulated, then O also can not be quadrangulated, since if O could be quadrangulated we could join the holes to the boundary by a sequence of edges (from the quadrangulation) and we could find a polygon C' like in our construction that can be quadrangulated, a contradiction. □

Lemma 4. *Given a planar polygon O with a holes and with the outer boundary with an even number of edges. If O has and even number of edges, then the number of holes with an odd number of edges is even.*



(a) (b) (c) (d)

Figure 4.11: (a) and (b) show an example of a triangulated part with a simple boundary in the composition of two quad meshes. In (c) and (d) we see the other possibility: the triangulated part has a complex boundary with many loops. Note that the region is isomorphic to a planar triangulation of a polygon with holes.

Proof. Since O has an even number of edges and the outer boundary also has an even number of edges, then, the sum of the number of edges of the holes must be even. Finally, the number of holes with an odd number of edges have to be even. □

Fortunately, even in the case (2), we always can quadrangulate the triangulated part. First, note that it still holds that the overall sum of the subdivisions of all the boundaries is even (what can happen is that single boundaries can have an *odd* number of subdivisions). It's easy to see why this is true. Looking at the remaining quad meshes after the patch retraction in the case (2). The outer boundary has an even number of edges because the same argument like in case (1). The inner boundaries(loops) are the boundary of an open quad mesh. As we had seen, the dual curves are loops or cross the boundary twice, then the overall number of edges at the boundary (the inner boundaries for the triangulated part) is even. We show in Figure 4.10 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 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 [59].

Note that it is always possible to find such a connecting polychord. By Lemma 4, the number of odd boundaries in the triangulated part is even. Suppose, by contradiction, that there is a boundary A with an odd number of edges such that there is no polychord going from A to any other boundary, then all the polychords starting on A are also ending on A, thus, covering an even number of edges, that is wrong because we assumed has an odd number of edges. Finally, how the number of odd boundaries is even, we can pair all of them by a polychord, and, then, we can always quadrangulate a composition of two quad meshes.

4.4 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 [60]. We compute the quadrangulation in parametric 2D space using [45], and then we interpolate the 3D positions of the vertices in parametric space.

Chapter 5

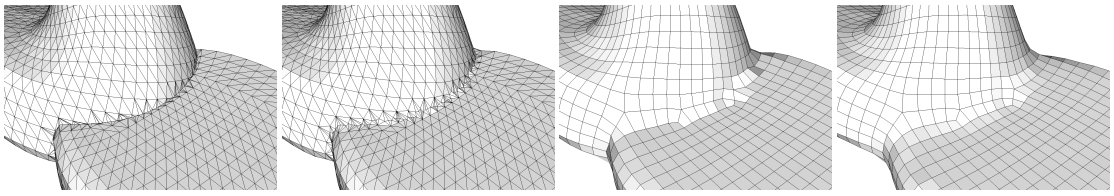
Implementation and results

In this chapter we expose our results and some implementation details of our small interactive system to test our proposed technique to blend quad meshes preserving as much as possible their initial conectivity. We performed our test on a desktop computer Intel Core i7-8750H with 16GB of RAM. We used Gurobi [61] to solve the minimization of Section 4.3. All the code is single-threaded and not highly optimized; it has been implemented using the VCG Library [62], CG3Lib [63], libigl [64], and CGAL [18].

5.1 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. Intu-



(a) (b) (c) (d)

Figure 5.1: 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.

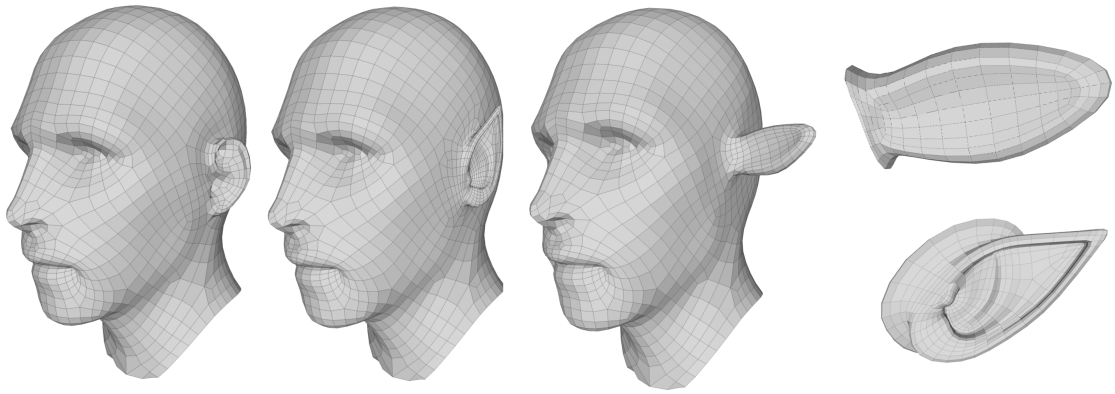


Figure 5.2: In the centre, we can see the result for the composition of the head on the left and the ears on the right.

itively, 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 with respect to the geodesic distance from the intersection line (see Figures $5.1-a$ and $5.1-b$). 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 $5.1-c$ and $5.1-d$ 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.

5.2 Character head models

We held a small session with a character animation model in order to demonstrate the application of our system in this area. We also used animal heads and try to get interesting humanoid head models. We pre-process the models with the purpose of gathering the necessary isolated parts and deleting whatever artifact like non-2-manifoldness or non-pure quad meshing.

A nice application we found for our system was the exchange of well-separated parts of the head for different 3d models. Figure 5.2 illustrates how we can change the ear of our human head by two new ones.

We also could generate a less intuitive new model by merging two models with an appropriate translation and rotation. In Figure 5.3 and Figure 5.4 we could see examples of this process.

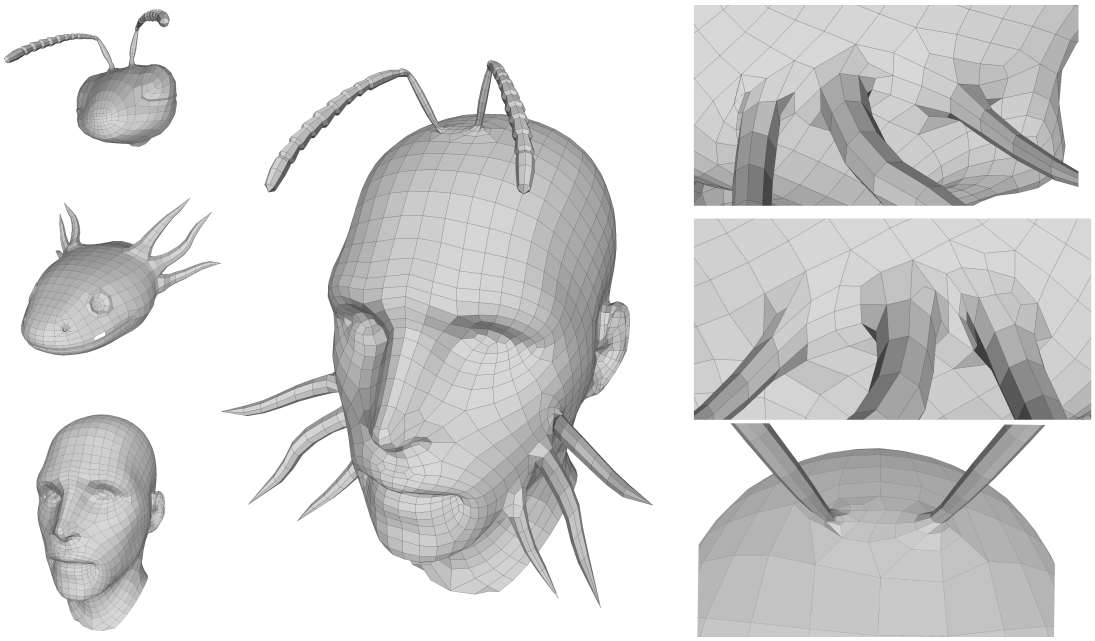


Figure 5.3: We join three models to generate a man with tentacles and antennae.

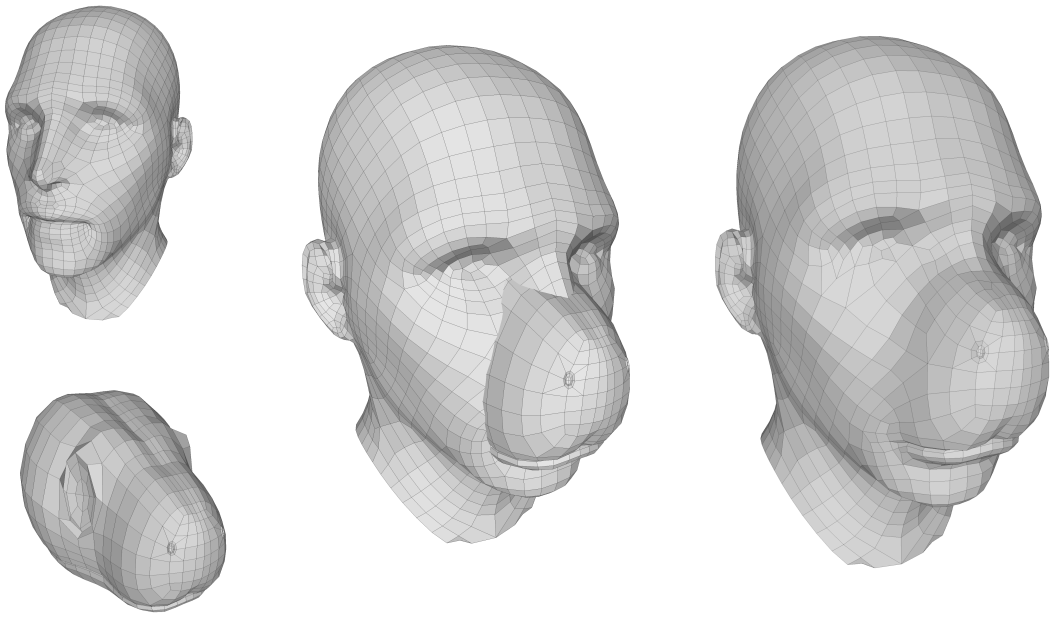


Figure 5.4: We join two models to generate a man with mouth and nose of a python.

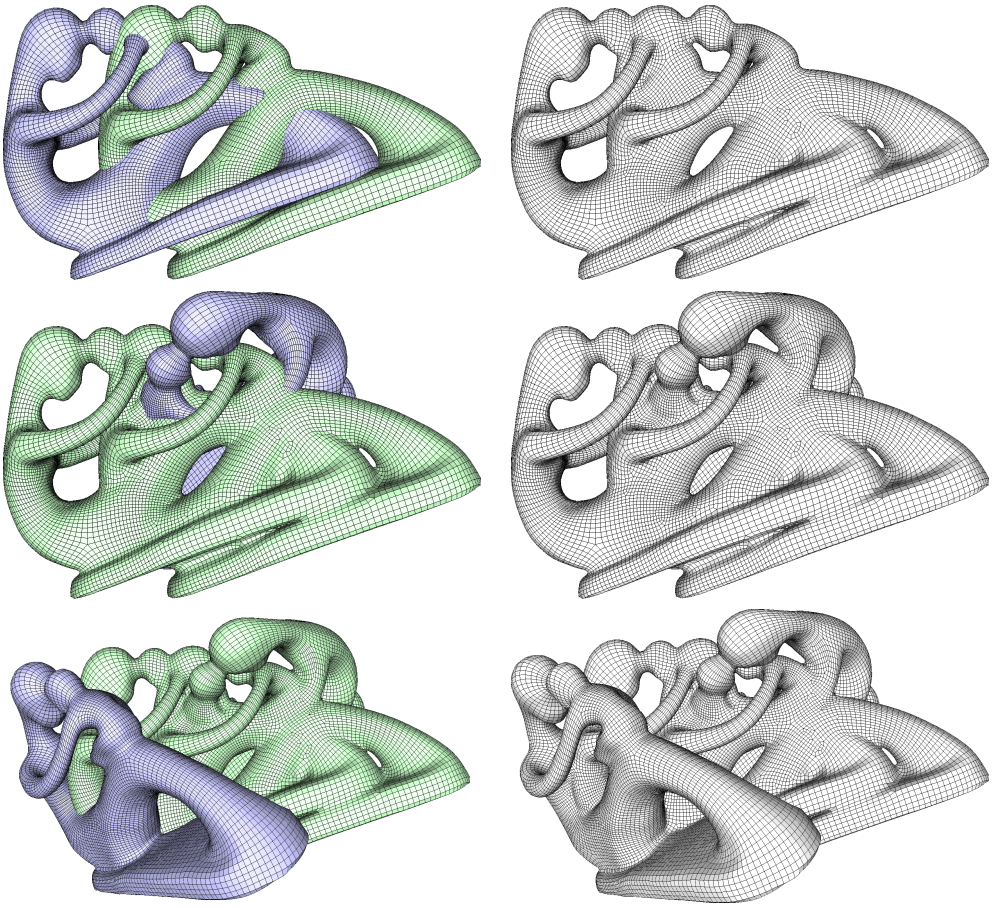


Figure 5.5: Iteratively merging the fertility model to check the robustness of the proposed approach.

5.3 Running-time

In order 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 5.5 shows some of the first steps of the test. In Figure 5.6, we show a full set of combinations among six meshes having different topologies, complex connectivity, or intricate geometric details.

Figure 5.8 reports the distribution of distortion in individual elements relative to the experiment shown in Figure 3.2. Distortion has been computed by using the distance with respect to the ideal quad as defined in [65]. As shown in the histograms, our method does not affect the overall quality of the quads.

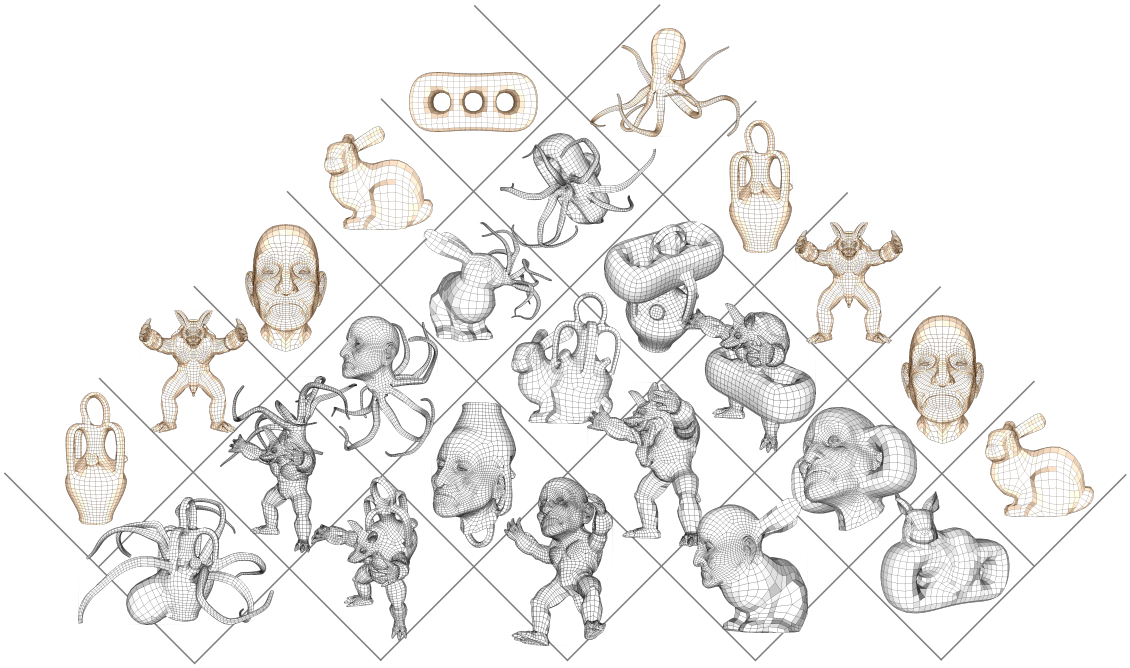


Figure 5.6: All the pairwise joins of six meshes, different in genus, complexity, and details.

Table 5.1: Time of execution of each step of the pipeline for the examples generated with our system. 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

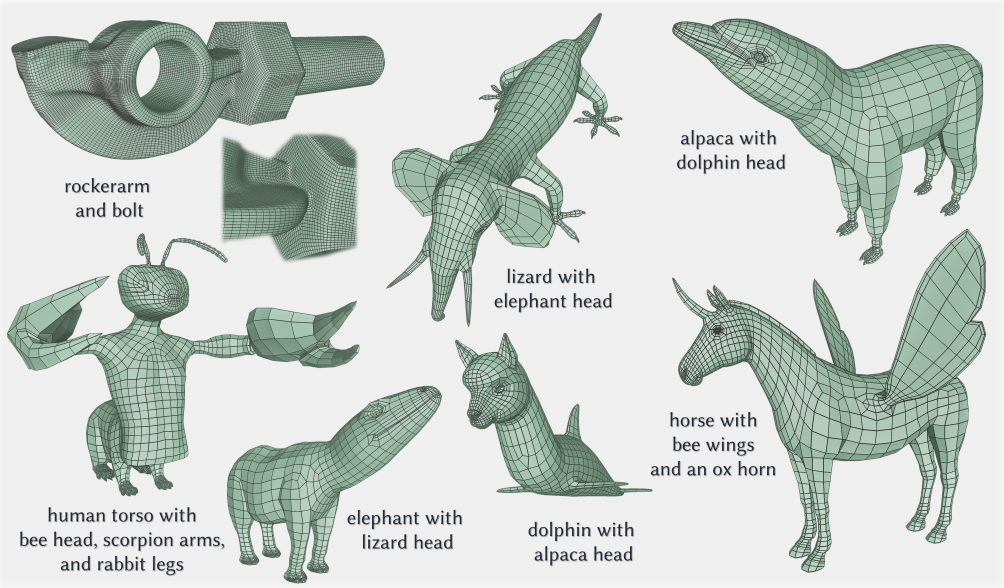


Figure 5.7: An overview of the final results obtained with our modelling tool. All the displayed models are available in the additional material.

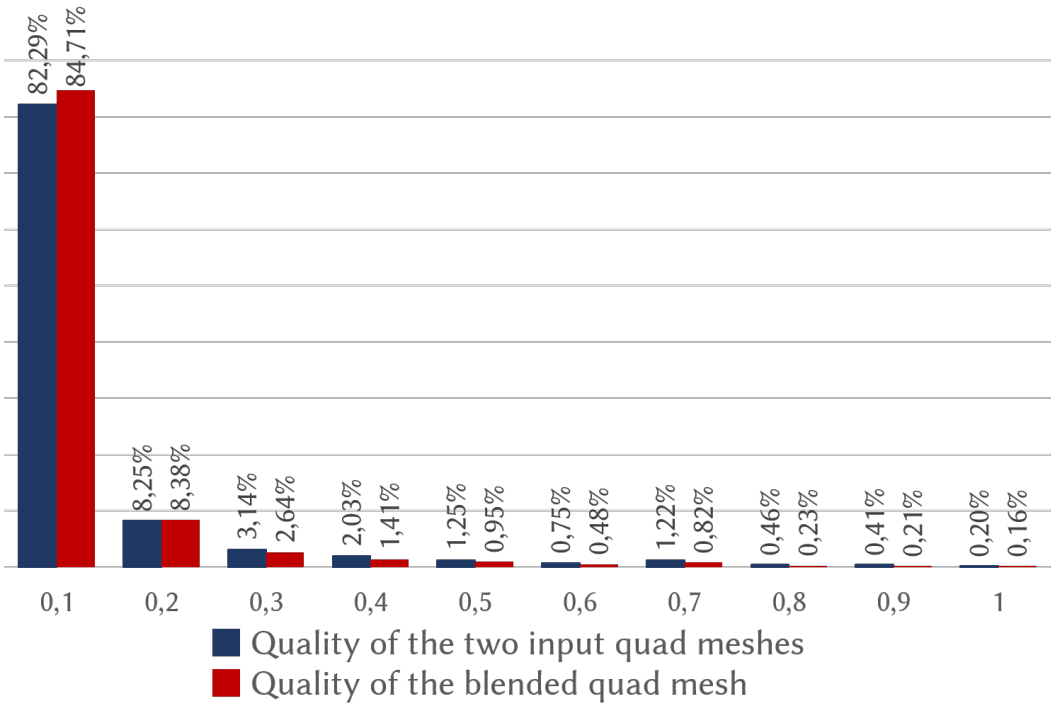


Figure 5.8: Distortions of the quads of Figure 3.2 measured using the metric defined in [65] (0 means no distortion).

Chapter 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 6.1. 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.

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 6.2), 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

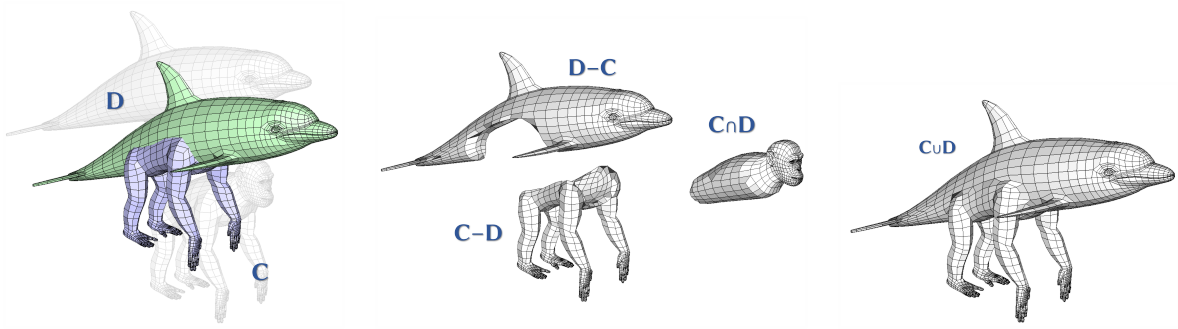


Figure 6.1: Our algorithm can mimic any boolean operation on quad meshes generating a well-shaped quad mesh which reuses the most of the initial layout. We show here, from left to right: the input consisting of two quad meshes (a dolphin **D** and a chimpanzee **C**) interactively placed in the scene; the two differences; their intersection; the union, which is, typically, the most interesting operation from a semantic standpoint.

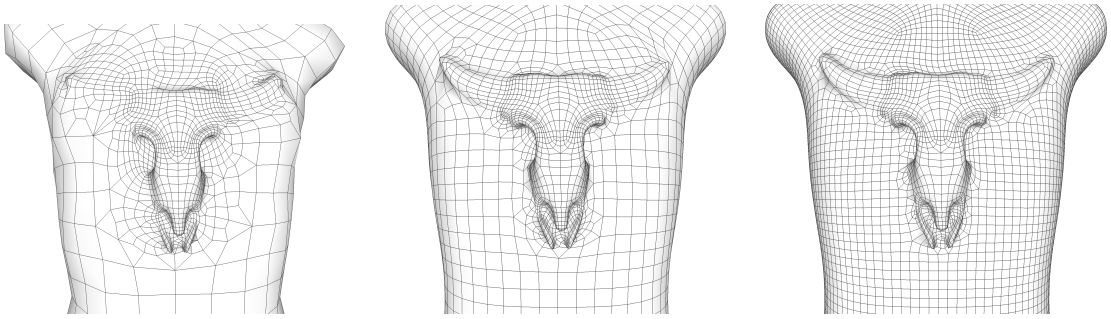


Figure 6.2: Blending meshes of different resolution.

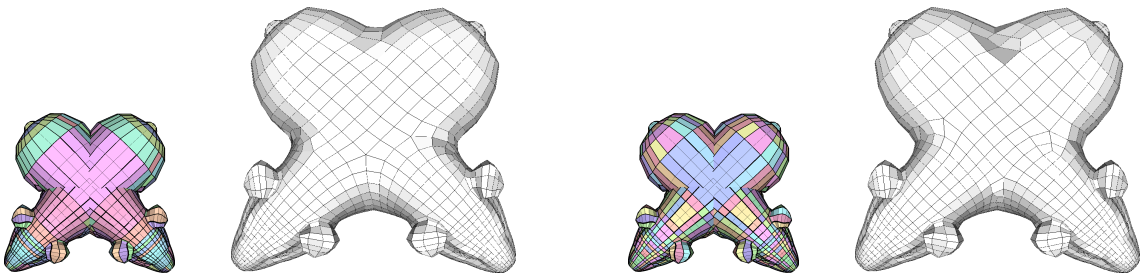


Figure 6.3: The sensitivity with respect to two different initial patch layout: motorcycle graph (left) and emanating separatrices (right).

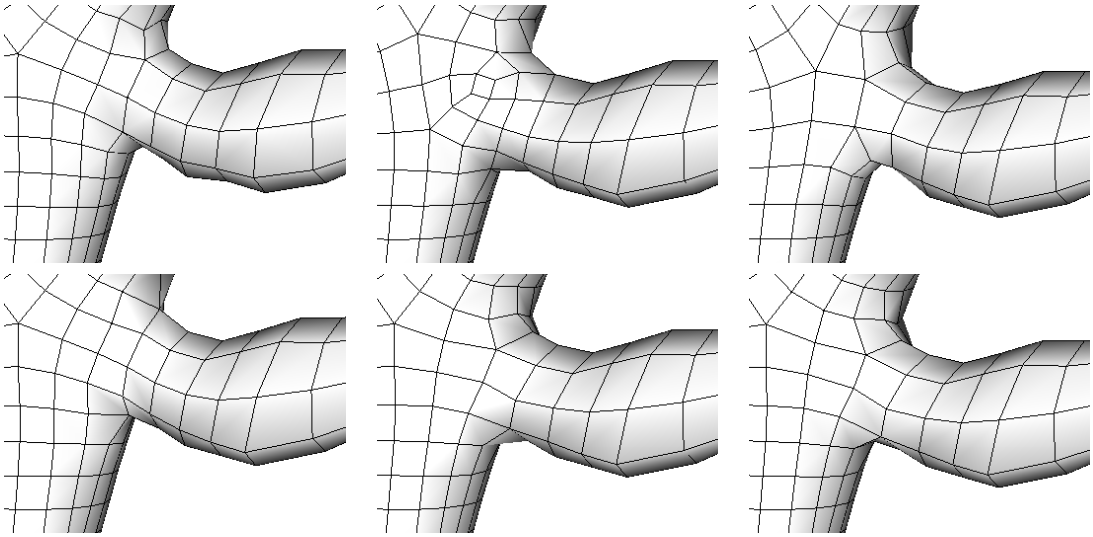


Figure 6.4: The variation of the tessellation configurations: close geometric configuration might cause abrupt changes in the tessellation (top); such an artefact can be mitigated with simple improvements (bottom).

boolean operation. Figure 6.3 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 6.4 (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 [65] 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 6.4 (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.

References

[1] RESEARCH, AND MARKETS. "Global Animation, VFX & Games Industry: Strategies, Trends & Opportunities, 2019". 2019. https://www.researchandmarkets.com/reports/4721808/global-animation-vfx-and-games-industry. Accessed: 2019-10-17.

[2] FUNKHOUSER, T. A., KAZHDAN, M. M., SHILANE, P., et al. "Modeling by example", *ACM Trans. Graph.*, v. 23, n. 3, pp. 652–663, 2004.

[3] AUTODESK. "Mudbox". 2018. Disponível em: <https://www.autodesk.com/education/free-software/mudbox>.

[4] PILGWAY. "3DCoat". 2017. Disponível em: <https://3dcoat.com/home/>.

[5] PIXOLOGIC. "ZBrush". 1999. Disponível em: <http://pixologic.com>.

[6] CAMPEN, M., KOBBELT, L. "Dual strip weaving: interactive design of quad layouts using elastica strips", *ACM Trans. Graph.*, v. 33, n. 6, pp. 183:1–183:10, 2014.

[7] TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, A., et al. "Sketch-based generation and editing of quad meshes", *ACM Trans. Graph.*, v. 32, n. 4, pp. 97:1–97:8, 2013.

[8] MARCIAS, G., TAKAYAMA, K., PIETRONI, N., et al. "Data-driven interactive quadrangulation", *ACM Trans. Graph.*, v. 34, n. 4, pp. 65:1–65:10, 2015.

[9] BOMMES, D., LÉVY, B., PIETRONI, N., et al. "Quad-Mesh Generation and Processing: A Survey", *Comput. Graph. Forum*, v. 32, n. 6, pp. 51–76, set. 2013. ISSN: 0167-7055. doi: 10.1111/cgf.12014. Disponível em: <http://dx.doi.org/10.1111/cgf.12014>.

[10] BOMMES, D. *Quadrilateral Surface Mesh Generation for Animation and Simulation*. Phd thesis, Faculty of Mathematics, Computer Science and Natural Sciences / RWTH Aachen University, 2012.

[11] MARCIAS, G., PIETRONI, N., PANOZZO, D., et al. “Animation-Aware Quadrangulation”, *Comput. Graph. Forum*, v. 32, n. 5, pp. 167–175, 2013.

[12] REMACLE, J.-F., LAMBRECHTS, J., SENY, B., et al. “Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm”, *International Journal for Numerical Methods in Engineering*, v. 89, pp. 1102 – 1119, 02 2012. doi: 10.1002/nme.3279.

[13] FOURNIER, A., MONTUNO, D. Y. “Triangulating Simple Polygons and Equivalent Problems”, *ACM Trans. Graph.*, v. 3, n. 2, pp. 153–174, abr. 1984. ISSN: 0730-0301. doi: 10.1145/357337.357341. Disponível em: <https://doi.org/10.1145/357337.357341>.

[14] JAKOB, W., TARINI, M., PANOZZO, D., et al. “Instant field-aligned meshes”, *ACM Trans. Graph.*, v. 34, n. 6, pp. 189:1–189:15, 2015.

[15] HUANG, J., ZHOU, Y., NIESSNER, M., et al. “QuadriFlow: A Scalable and Robust Method for Quadrangulation”, *Computer Graphics Forum*, 2018. ISSN: 1467-8659. doi: 10.1111/cgf.13498.

[16] ZHOU, Q., GRINSPUN, E., ZORIN, D., et al. “Mesh arrangements for solid geometry”, *ACM Trans. Graph.*, v. 35, n. 4, pp. 39:1–39:15, 2016.

[17] VISIONMONGERS, T. F. “Modo 12.1”. 2018. Disponível em: <http://www.thefoundry.co.uk/products/modo>.

[18] THE CGAL PROJECT. *CGAL User and Reference Manual*. 4.14 ed. , CGAL Editorial Board, 2019. Disponível em: <https://doc.cgal.org/4.14/Manual/packages.html>.

[19] HACHENBERGER, P., KETTNER, L. “3D Boolean Operations on Nef Polyhedra”. In: Board, C. E. (Ed.), *CGAL User and Reference Manual*, 4.9.1 ed., 2016. Disponível em: <http://doc.cgal.org/latest/Nef_3/index.html>.

[20] BERNSTEIN, G., FUSSELL, D. “Fast, Exact, Linear Booleans”. In: *Proceedings of the Symposium on Geometry Processing*, SGP ’09, pp. 1269–1278, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association. Disponível em: <http://dl.acm.org/citation.cfm?id=1735603.1735606>.

[21] PAVIĆ, D., CAMPEN, M., KOBBELT, L. “Hybrid Booleans”, *Computer Graphics Forum*, v. 29, n. 1, pp. 75–87, 2010. ISSN: 1467-8659. doi: 10.1111/

j.1467-8659.2009.01545.x. Disponível em: <http://dx.doi.org/10.1111/j.1467-8659.2009.01545.x>.

[22] DOUZE, M., FRANCO, J.-S., RAFFIN, B. *QuickCSG: Arbitrary and Faster Boolean Combinations of N Solids*. Research Report RR-8687, Inria - Research Centre Grenoble – Rhône-Alpes ; INRIA, mar. 2015. Disponível em: <https://hal.inria.fr/hal-01121419>.

[23] SHARF, A., BLUMENKRANTS, M., SHAMIR, A., et al. "SnapPaste: an interactive technique for easy mesh composition", *The Visual Computer*, v. 22, n. 9-11, pp. 835–844, 2006.

[24] TSAI, M., LU, T. "A Rapid Mesh Fusion Method to Create 3D Virtual Characters in Games". In: *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pp. 393–398, 2009.

[25] SCHMIDT, R., BROCHU, T. "Adaptive Mesh Booleans", *ArXiv e-prints*, maio 2016.

[26] SCHMIDT, R., SINGH, K. "Meshmixer: An Interface for Rapid Mesh Composition". In: *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, pp. 6:1–6:1, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0394-1.

[27] ZHANG, J., WU, C., CAI, J., et al. "Mesh Snapping: Robust Interactive Mesh Cutting Using Fast Geodesic Curvature Flow", *Comput. Graph. Forum*, v. 29, n. 2, pp. 517–526, 2010.

[28] PAVIC, D., CAMPEN, M., KOBBELT, L. "Hybrid Booleans", *Comput. Graph. Forum*, v. 29, n. 1, pp. 75–87, 2010.

[29] BISCHOFF, S., KOBBELT, L. "Structure Preserving CAD Model Repair", *Comput. Graph. Forum*, v. 24, n. 3, pp. 527–536, 2005.

[30] CAMPEN, M. "Partitioning Surfaces Into Quadrilateral Patches: A Survey", *Comput. Graph. Forum*, v. 36, n. 8, pp. 567–588, 2017.

[31] BOMMES, D., ZIMMER, H., KOBBELT, L. "Mixed-integer quadrangulation", *ACM Trans. Graph.*, v. 28, n. 3, pp. 77, 2009.

[32] KÄLBERER, F., NIESER, M., POLTHIER, K. "QuadCover - Surface Parameterization using Branched Coverings", *Comput. Graph. Forum*, v. 26, n. 3, pp. 375–384, 2007.

[33] CAMPEN, M., BOMMES, D., KOBBELT, L. "Quantized Global Parametrization", *ACM Trans. Graph.*, v. 34, n. 6, out. 2015. ISSN: 0730-0301. doi: 10.1145/2816795.2818140. Disponível em: <https://doi.org/10.1145/2816795.2818140>.

[34] FANG, X., BAO, H., TONG, Y., et al. "Quadrangulation through Morse-Parameterization Hybridization", *ACM Trans. Graph.*, v. 37, n. 4, jul. 2018. ISSN: 0730-0301. doi: 10.1145/3197517.3201354. Disponível em: <https://doi.org/10.1145/3197517.3201354>.

[35] PALACIOS, J., ZHANG, E. "Rotational Symmetry Field Design on Surfaces". In: *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, p. 55–es, New York, NY, USA, 2007. Association for Computing Machinery. ISBN: 9781450378369. doi: 10.1145/1275808.1276446. Disponível em: <https://doi.org/10.1145/1275808.1276446>.

[36] PANOZZO, D., PUPPO, E., TARINI, M., et al. "Frame Fields: Anisotropic and Non-Orthogonal Cross Fields", *ACM Trans. Graph.*, v. 33, n. 4, jul. 2014. ISSN: 0730-0301. doi: 10.1145/2601097.2601179. Disponível em: <https://doi.org/10.1145/2601097.2601179>.

[37] JIANG, T., FANG, X., HUANG, J., et al. "Frame Field Generation through Metric Customization", *ACM Trans. Graph.*, v. 34, n. 4, jul. 2015. ISSN: 0730-0301. doi: 10.1145/2766927. Disponível em: <https://doi.org/10.1145/2766927>.

[38] TARINI, M., PUPPO, E., PANOZZO, D., et al. "Simple quad domains for field aligned mesh parametrization", *ACM Trans. Graph.*, v. 30, n. 6, pp. 142:1–142:12, 2011.

[39] CAMPEN, M., BOMMES, D., KOBBELT, L. "Dual loops meshing: quality quad layouts on manifolds", *ACM Trans. Graph.*, v. 31, n. 4, pp. 110:1–110:11, 2012.

[40] DONG, S., BREMER, P.-T., GARLAND, M., et al. "Spectral Surface Quadrangulation". In: *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, p. 1057–1066, New York, NY, USA, 2006. Association for Computing Machinery. ISBN: 1595933646. doi: 10.1145/1179352.1141993. Disponível em: <https://doi.org/10.1145/1179352.1141993>.

[41] ZHOU, J., CAMPEN, M., ZORIN, D., et al. "Quadrangulation of non-rigid objects using deformation metrics", *Computer Aided Geometric Design*, v. 62, pp. 3–15, 2018.

[42] BESSMELTSEV, M., WANG, C., SHEFFER, A., et al. "Design-driven quadrangulation of closed 3D curves", *ACM Trans. Graph.*, v. 31, n. 6, pp. 178:1–178:11, 2012.

[43] TIERNY, J., II, J. D., NONATO, L. G., et al. "Inspired quadrangulation", *Computer-Aided Design*, v. 43, n. 11, pp. 1516–1526, 2011.

[44] PENG, C., ZHANG, E., KOBAYASHI, Y., et al. "Connectivity editing for quadrilateral meshes", *ACM Trans. Graph.*, v. 30, n. 6, pp. 141:1–141:12, 2011.

[45] TAKAYAMA, K., PANOZZO, D., SORKINE-HORNUNG, O. "Pattern-Based Quadrangulation for *N*-Sided Patches", *Comput. Graph. Forum*, v. 33, n. 5, pp. 177–184, 2014.

[46] PENG, C., BARTON, M., JIANG, C., et al. "Exploring quadrangulations", *ACM Trans. Graph.*, v. 33, n. 1, pp. 12:1–12:13, 2014.

[47] SCHAEFER, S., WARREN, J. D., ZORIN, D. "Lofting Curve Networks using Subdivision Surfaces". In: *Second Eurographics Symposium on Geometry Processing, Nice, France, July 8-10, 2004*, pp. 103–114, 2004.

[48] NASRI, A. H., SABIN, M. A., YASSEEN, Z. "Filling *N*-Sided Regions by Quad Meshes for Subdivision Surfaces", *Comput. Graph. Forum*, v. 28, n. 6, pp. 1644–1658, 2009.

[49] YASSEEN, Z., NASRI, A. H., BOUKARAM, W., et al. "Sketch-based garment design with quad meshes", *Computer-Aided Design*, v. 45, n. 2, pp. 562–567, 2013.

[50] NUVOLI, S., HERNANDEZ, A., ESPERANÇA, C., et al. "QuadMixer: Layout Preserving Blending of Quadrilateral Meshes", *ACM Trans. Graph.*, v. 38, n. 6, nov. 2019. ISSN: 0730-0301. doi: 10.1145/3355089.3356542. Disponível em: <https://doi.org/10.1145/3355089.3356542>.

[51] EPPSTEIN, D., GOODRICH, M. T., KIM, E., et al. "Motorcycle Graphs: Canonical Quad Mesh Partitioning", *Comput. Graph. Forum*, v. 27, n. 5, pp. 1477–1486, 2008.

[52] DIAMANTI, O., VAXMAN, A., PANOZZO, D., et al. "Designing *N*-PolyVector Fields with Complex Polynomials", *Comput. Graph. Forum*, v. 33, n. 5, pp. 1–11, 2014.

[53] BOMMES, D., LEMPFER, T., KOBBELT, L. "Global Structure Optimization of Quadrilateral Meshes", *Comput. Graph. Forum*, v. 30, n. 2, pp. 375–384, 2011.

[54] MORGAN, C. *Programming from Specifications (2Nd Ed.)*. Hertfordshire, UK, UK, Prentice Hall International (UK) Ltd., 1994. ISBN: 0-13-123274-6.

[55] CIGNONI, P., CALLIERI, M., CORSINI, M., et al. "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference 2008, Salerno, Italy, 2008*, pp. 129–136, 2008.

[56] VAXMAN, A., CAMPEN, M., DIAMANTI, O., et al. "Directional field synthesis, design, and processing". In: *SIGGRAPH '17 Courses*, pp. 12:1–12:30, 2017.

[57] MYLES, A., PIETRONI, N., ZORIN, D. "Robust field-aligned global parametrization", *ACM Trans. Graph.*, v. 33, n. 4, pp. 135:1–135:14, 2014.

[58] PIETRONI, N., PUPPO, E., MARCIAS, G., et al. "Tracing Field-Coherent Quad Layouts", *Comput. Graph. Forum*, v. 35, n. 7, pp. 485–496, 2016.

[59] DANIELS, J., SILVA, C. T., SHEPHERD, J., et al. "Quadrilateral mesh simplification", *ACM Trans. Graph.*, v. 27, n. 5, pp. 148:1–148:9, 2008.

[60] LÉVY, B., PETITJEAN, S., RAY, N., et al. "Least squares conformal maps for automatic texture atlas generation", *ACM Trans. Graph.*, v. 21, n. 3, pp. 362–371, 2002.

[61] GUROBI OPTIMIZATION, L. "Gurobi Optimizer Reference Manual". 2018. Disponível em: <http://www.gurobi.com>.

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

[63] MUNTONI, A., NUVOLI, S., OTHERS. "CG3Lib: A structured C++ geometry processing library." 2019. https://github.com/cg3hci/cg3lib.

[64] JACOBSON, A., PANOZZO, D., OTHERS. "libigl: A simple C++ geometry processing library". 2013. http://igl.ethz.ch/projects/libigl/.

[65] PIETRONI, N., TONELLI, D., PUPPO, E., et al. "Statics Aware Grid Shells", *Comput. Graph. Forum*, v. 34, n. 2, pp. 627–641, 2015.