Università degli Studi di Cagliari

PHD DEGREE

Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Matematica e Informatica

Cycle XXXIII

TITLE OF THE PHD THESIS

Composing quadrilateral meshes for animation

Scientific Disciplinary Sector(s)

INF/01

PhD Student: Stefano Nuvoli

Supervisor(s): Prof. Riccardo Scateni
Prof. Nico Pietroni

Final exam. Academic Year 2019 – 2020
Thesis defence: April 2021 Session

Stefano Nuvoli gratefully acknowledges Sardinian Regional Government for the financial support of his PhD scholarship (P.O.R. Sardegna F.S.E. - Operational Programme of the Autonomous Region of Sardinia, European Social Fund 2014-2020 - Axis III Education and training, Thematic goal 10, Investment Priority 10ii), Specific goal 10.5.

Contents

Abstract

The modeling-by-composition paradigm can be a powerful tool in modern animation pipelines. We propose two novel interactive techniques to compose 3D assets that enable the artists to freely remove, detach and combine components of organic models. The idea behind our methods is to preserve most of the original information in the input characters and blend accordingly where necessary.

The first method, QuadMixer, provides a robust tool to compose the quad layouts of watertight pure quadrilateral meshes, exploiting the boolean operations defined on triangles. Quad Layout is a crucial property for many applications since it conveys important information that would otherwise be destroyed by techniques that aim only at preserving the shape. Our technique keeps untouched all the quads in the patches which are not involved in the blending. The resulting meshes preserve the originally designed edge flows that, by construction, are captured and incorporated into the new quads.

SkinMixer extends this approach to compose skinned models, taking into account not only the surface but also the data structures for animating the character. We propose a new operation-based technique that preserves and smoothly merges meshes, skeletons, and skinning weights. The retopology approach of QuadMixer is extended to work on quad-dominant and arbitrary complex surfaces. Instead of relying on boolean operations on triangle meshes, we manipulate signed distance fields to generate an implicit surface. The results preserve most of the information in the input assets, blending accordingly in the intersection regions. The resulting characters are ready to be used in animation pipelines.

Given the high quality of the results generated, we believe that our methods could have a huge impact on the entertainment industry. Integrated into current software for 3D modeling, they would certainly provide a powerful tool for the artists. Allowing them to automatically reuse parts of their well-designed characters could lead to a new approach for creating models, which would significantly reduce the cost of the process.

Chapter 1

Introduction

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

In the industry, the creation of a 3D character for animation purposes is composed of several tasks, often reserved for high-skilled professionals. Firstly, a modeler designs the surface quad layout, usually from scratch. Then, a professional rigger builds a bone structure (skeleton) that lies inside the model volume and he *paints* the skinning weights for each of its bone. Finally, an animator assigns a transformation to each bone to generate the poses for animating the model. All these processes (especially the *weight panting*) are time-consuming, error-prone, and, therefore, highly expensive [JDKL14]. Actually, the last three processes are not serially performed, but they are repeatedly iterated until a satisfying result is obtained. We refer to this delicate and challenging process as *skinning*.

Our work takes inspiration from the modeling-by-composition paradigm. The primary purpose of this approach is to directly combine parts from existing models to synthesize new ones. The result is quite intuitive and suitable for novice users, by allowing them to rapidly assemble complex 3D models. Currently, the field of modeling-by-composition is quite active and it is generating promising results.

However, all the proposed solutions cannot produce a fully quadrangulated or a quad-dominant mesh. Their output is always limited to triangulated surfaces, even if quadrilateral meshes have become a standard in several fields, including CAD, automotive design, and animation. Mostly, this is due to the fact that, for human beings, it is more natural to manipulate quadrilateral flows than unorganized triangles. Moreover, to our knowledge, there is no method for composing, along

with the surface, the skinning information needed for model animation.

Coarse quad layouts are often manually created by professional artists. Typical modeling systems used in the industry [Aut18, Pil17, Pix99] allows the user to manually draw vertices and edges on a surface. Since this manual procedure is time-consuming and error-prone, a series of sketch-based retopology approaches [CK14, TPS14, MTP+15] have been proposed. These semi-automatic approaches automate a large part of the process while allowing the user to efficiently modify the topology of the layouts without having to start from scratch.

While automatic quadrangulation and automatic skinning techniques can generate excellent results, the manual design of the characters is still considered by content creators as part of the artistic process. The artists employ their semantic knowledge and experience to adjust the layout in the context of the particular application needs. Undoubtedly, this cannot be comparable to the results of the state-of-the-art automatic methods. For this reason, the preservation of the original quad layout and skinning information during the composition process is crucial to allow the effective use of modeling-by-composition in the field of the 3D assets.

In this thesis, we propose two different methods for composing and blending quadrilateral meshes by an automatic retopology process. They are illustrated in detail in Chapter 3 and Chapter 4. Furthermore, in Chapter 4, we propose an approach to *blend* the skinning information, such as skeletons and skinning weights, along with the surface.

Part of the results obtained in this thesis have been presented at a conference and have been published in the following article [NHE+19]:
S. Nuvoli, A. Hernandez, C. Esperança, R Scateni, P Cignoni, N Pietroni.
QuadMixer: layout preserving blending of quadrilateral meshes.
2019, ACM Transactions on Graphics (TOG), vol. 38 number 6, pages 1-13.

In appendix A, we illustrate a contribution in the Digital Fabrication field. Even if it is not related to the main subject of this thesis, the candidate spent considerable time on this project during his Ph.D. studies. The proposed technique enables the fabrication of 3D complex shapes from a single block of material using 4-axis CNC milling machines. An article has been submitted to a peer-reviewed journal and it has successfully passed the revision phase.

Chapter 2

Related works

Various combining techniques that follow the modeling-by-composition paradigm have been proposed since [FKS+04]. Most of these methods allow for cutting arbitrary surface parts from one model and automatically stitching them to existing holes into another [SBSC06,SS10,ZWC+10]. In [BMBZ02] detail transfer techniques are used to copy high-frequency details.

Recently, many efforts have concentrated on the other important task of suggesting or choosing which parts to combine. Modern techniques provide fully automated frameworks to indicate the widest choice of possible results.

2.1 Quad-meshing and retopology

Robustly generating functional quad layouts and quad meshes is a well-studied research field. Extensive surveys on this topic are available [BLP+13, Cam17a] as well as shorter and more introductory papers [Cam17b].

In the following, we will concentrate the discussion on the most closely related topics: the user-assisted generation of quad meshes and the meshing of polygonal patches.

Interactive quadrangulation tools Many methods have been proposed to automatically design coarse quad layouts [BLK11, TPP+11, CBK12, BLP+13]. Most of these try to cope with specific needs in the production. For instance, [MPP+13, ZCZ+18] considers the deformation affecting a mesh during an animated sequence to generate a quad layout that remains good for all animation frames. While these automatic approaches are successful on several quantitative metrics (like singularity placement and coarseness), in production pipelines there is the need for a more art-controlled quad generation process. Due to the global nature of the problem, small changes in the user-provided initial constraints may completely

alter the generated quad mesh. The combination of this global behavior with the non-interactive nature of these automatic approaches makes the tuning of parameters an unintuitive and time-consuming task.

Therefore, many approaches have considered the issue of helping this manual quadrangulation process leaving most of the control to the final user. Bessmeltsev et al. [BWSS12] developed a technique for generating quad-dominant meshes starting from an input 3D curve network sketched by the user; with this approach, geometry and topology are defined based on regions identified by closed 3D paths.

Inspired quadrangulation [TIN+11] can also be considered related to our approach. In this work, the authors transfer quadrangulations between surfaces on a per-partition basis (e.g., head, arm, torso) via cross-parameterization. Unfortunately, this approach does not provide precise local control over the mesh layout. Instead, our method enables the direct combination of portions of quad meshes.

Finally, connectivity editing operations have been developed to enable users to modify existing quad meshes by moving pairs of irregular vertices [PZKW11]. These methods provide lower-level local operators and they can be integrated with practices of the previous class to fine-tune the mesh topology.

Quad-meshing patches In our pipelines, we face the issue of completing a partial quad mesh. This task is an essential part of all sketch-based retopology techniques [TPSS13, PBJW14, TPS14].

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 [SWZ04, NSY09, YNB+13] and finally automatically quadrangulated.

The present work also requires quadrangulating patches defined by their sides, which can be done using filling patterns generated procedurally as in [PBJW14]. In [TPSS13, TPS14], a set of manually designed patterns are expanded to tessellate arbitrary polygons with up to 6 sides.

More recently Marcias et al. [MTP+15] 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. We expect to produce triangular patches that are relatively small compared to the input meshes and we propose using Takayama's method [TPS14]. 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.

2.2 Composing surfaces

Our approaches smoothly compose portions of quadrilateral meshes in 3D assets. In the following, we will concentrate the discussion on the two different approaches for composing and extracting surfaces: mesh arrangements and implicit surface blending and reconstruction with signed distance fields.

Mesh arrangements In Chapter 3, a fundamental step of the approach is the detection and computation of the intersection between the triangulated surfaces. A vast literature exists on how to solve this problem robustly and efficiently, and we exploited the boolean operations of [ZGZJ16] for triangulated meshes. Moreover, the idea of trying to limit the modification on a mesh when performing boolean operations [PCK10], or repair the result [BK05], has been already explored. The approach of [CK10] can also perform boolean operations on generic input polygonal meshes. Finally, in [CLSA20] the authors propose a method that enables a faster computation of mesh intersection for challenging triangle meshes. However, no solutions can smoothly composite quad meshes as the proposed approaches.

Signed distance fields In Chapter 4, we propose a method for composing surfaces based on the manipulation of signed distance fields. Signed distance fields (SDF) provide a powerful implicit representation of surfaces and volumes. They are broadly used tools in Computer Graphics, with applications in collision detection [MASS15, XB16], surface reconstruction [CT11, TF18], and rendering. In recent years, researchers have been focusing on improving the time and memory efficiency in the computation of such fields [Mus11,SFP12,KDB16,MLJ+13]. Boolean operations on SDFs, such as union, difference, and intersection are common operations in geometrical design and modeling [FP05].

The process of determining the sign of the field for given 3D coordinates can be extremely challenging for surfaces with open boundaries, self-intersections, or non-manifold faces. [JKS13] proposed a robust method for an input-output segmentation that deals with these kinds of artifacts. Moreover, [BDS+18] extended this method to obtain faster evaluations, at the expenses of the approximation of the output values.

Blending implicit surfaces The blending of implicit surfaces has been widely studied by researchers in Computer Graphics. The applications vary from simulating the animation of fluids or particular substances [DG95, BGOS06] to modelling surfaces [WGG99, SPS04, dWv09, BBCW13, GBC+13, ATW+17]. The goal of these methods is to efficiently obtain a continuous and smooth resulting shape, taking into account the fidelity with respect to the input models. In the past years, the

editing of level-set implicit surfaces have been broadly studied, and many efficient solutions have been proposed [MBWB02, BMPB08, EB10].

In Chapter 4, we propose a new approach for automatically blend signed distance fields into a new one, following the skinning weight information in characters.

2.3 Automatic skinning

In Chapter 4, we illustrate a method to compose skinned models, preserving most of the original information. In the following, we will briefly discuss some methods for performing automatic skinning of characters and retargeting of skinning weights.

Automatic skeletons and weights Due to the high cost of the process, many researchers focused in the past decades to automatically generate the skeleton and its associated weights for a given surface. A more detailed review on automatic generation of skeletons and skinning weights can be found in [JDKL14].

The modern automatic skeleton extraction methods are mainly based on two approaches [JDKL14]: one takes advantage of surface flows to converge on a curved skeleton (surface flow-based methods [TAOZ12, WP02, WL08, HWCO+13]), while the other aims to find a skeleton through the segmentation of shapes (segmentation-based methods [BTST12]). Some of these approaches are example-based, having as input a given a set of example poses or the topology of the skeleton [BP07, SY07].

The automatic skinning weight process leverages on finding an optimized energy that satisfies properties of continuity and shape-awareness [DdL13, KS12, JBPS11, JWS12, LL14]. These properties are necessary to avoid artifacts during the deformation of the character. Even if state-of-the-art methods generate excellent results, the problem is yet to be solved.

The quality of the results of the automatic methods is not comparable to the artist-designed characters. We could apply these methods to the output surface of our method. However, the original skinning information would be completely lost, while our method entirely preserves most of it.

Skinning weight retargeting Weight retargeting is the process that transfers the skinning weights of a model to a new character with a similar topology. Recently, many efforts have been made to solve this problem.

In [DLG+13], the authors present a method to retarget anatomical structures from one model to another, including the skeleton bones and skinning weights. Avril et al. [ARG+16] proposes a semi-automatic method for transferring skeletons and skinning weights, also between characters with different topologies. In [PCYQ18], the authors propose a retargeting of the weights using surface matching and interpolation based on bi-harmonic distance. They obtain excellent results, by extending

the traditional point-based dynamics method (PBD) to match geometrically and physically-based constraints.

Our approach is also based on reusing the painted weights of characters to determine the weights of a new model. However, the primary difference with the methods above is that our data-driven approach deals with more than one skinned character. Moreover, we want to entirely preserve portions of the surface, skeleton, and the associated weights, and *blend* accordingly in the intersection areas. Finally, our approach natively deals with different skeleton topologies.

Chapter 3

QuadMixer: composing watertight pure quadrilateral meshes

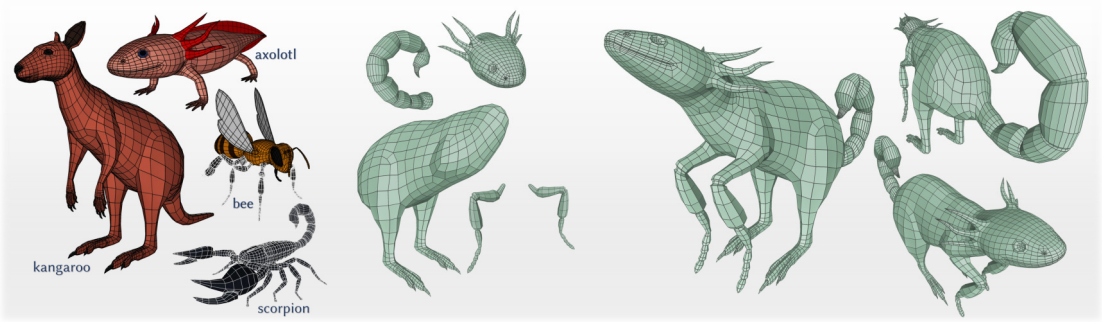


Figure 3.1: **QuadMixer, the proposed blending technique.** We can assemble pieces of different models respecting the original quad meshing. In less than ten minutes we detached and combined these body pieces to automatically obtain the pure quad mesh shown on the right.

QuadMixer is a novel technique to compose quad meshes. Our modeling principles take inspiration from the classical boolean operations defined on triangle meshes, but with operators redesigned to work on quadrilateral meshes. *QuadMixer* mimics all the conventional boolean operations that are available for triangle meshes such as union, intersection, and difference. While it is generally a challenging task to define stable boolean operations, their result can be defined precisely in the context of triangle meshes [ZGZJ16]. In contrast, this is not true for generic quad meshes. Concerning boolean operations, a first evident difference between a quad

mesh and a triangle mesh is that the former does not admit a unique piecewise discretization. Hence, the single intersection between two quadrilateral elements cannot be unequivocally defined. In this light, we might refer to our operations as *blending* to distinguish them from classical boolean operations on triangle meshes.

We summarize the main contributions of this chapter as follows:

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

3.1 Overview

Given a pair of two-manifold watertight meshes composed only of quadrilateral elements, our method *blends* them into a new, closed, two-manifold, pure quadrilateral mesh. Our approach preserves of the majority of the shapes' layout, remeshing accordingly only where needed (Figure 3.2).

As opposed to triangle meshes, performing blending on quadrilateral meshes poses extra challenges: while splitting triangles on a local basis will always result in a valid triangle mesh, quad meshes must be manipulated taking into account their entire connectivity. Indeed, the majority of applications, such as subdivision surfaces or finite element analysis, require the flow of the edges to be aligned with geometric features, imposing several conditions on the global structure and the placement of irregular vertices. Relying solely on local modifications [TPC+10] cannot, in general, enforce such global characteristics.

A simple way to tackle this problem might be to transform the quad mesh into triangles first, perform the boolean operation, and finally use a quad-meshing algorithm based on global parametrization [JTPS15, BZK09] or cross-field tracing [PPM+16, MPZ14] to obtain a sound quadrilateral mesh. However, this approach will inevitably modify the entire mesh. This result is far from ideal from a modeling point of view, as an artist would more likely prefer to preserve as much as possible the connectivity he has designed. As illustrated in Figure 3.3, a complete quadrilateral

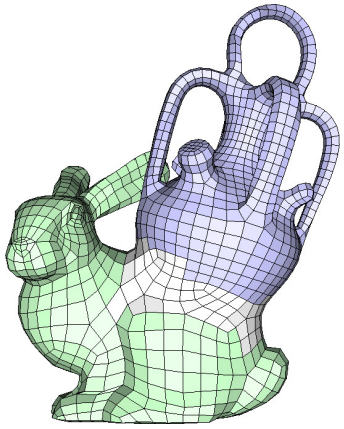


Figure 3.2: **The idea behind our method.** The blue and green surface represent the preserved quad layout of the two meshes. The remaining portion is the surface remeshed with respect to the boundary constraints.

re-meshing using the approach of [JTPS15] on the result of the boolean operation will inevitably cause the loss of most of the features designed by the artist, like the eyes of the pig or the armadillo.

Among commercial software packages, to the best of our knowledge, only the Modo suite [Vis18] provides a tool, called *MeshFusion* [Vis18], 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 (considered as triangular meshes in the intersections). Therefore, differently from our case, a quad-pure representation of the result is never explicitly computed, and the results include triangulated regions around the intersection lines. Figure 3.4 shows a comparison between the quad-dominant representation obtained by MeshFusion and the result of our method.

A fundamental task is to preserve as much as possible the connectivity of the original quadrangulated models and change the elements only in regions modified by the boolean operation. Figure 3.5 shows an overview of our entire pipeline.

- We first compute a patch decomposition of the quadrilateral meshes by using a simple *motorcycle graph* [EGKT08] tracing algorithm (see Figure 3.5.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 [ZGZJ16] (see Figure 3.5.b).

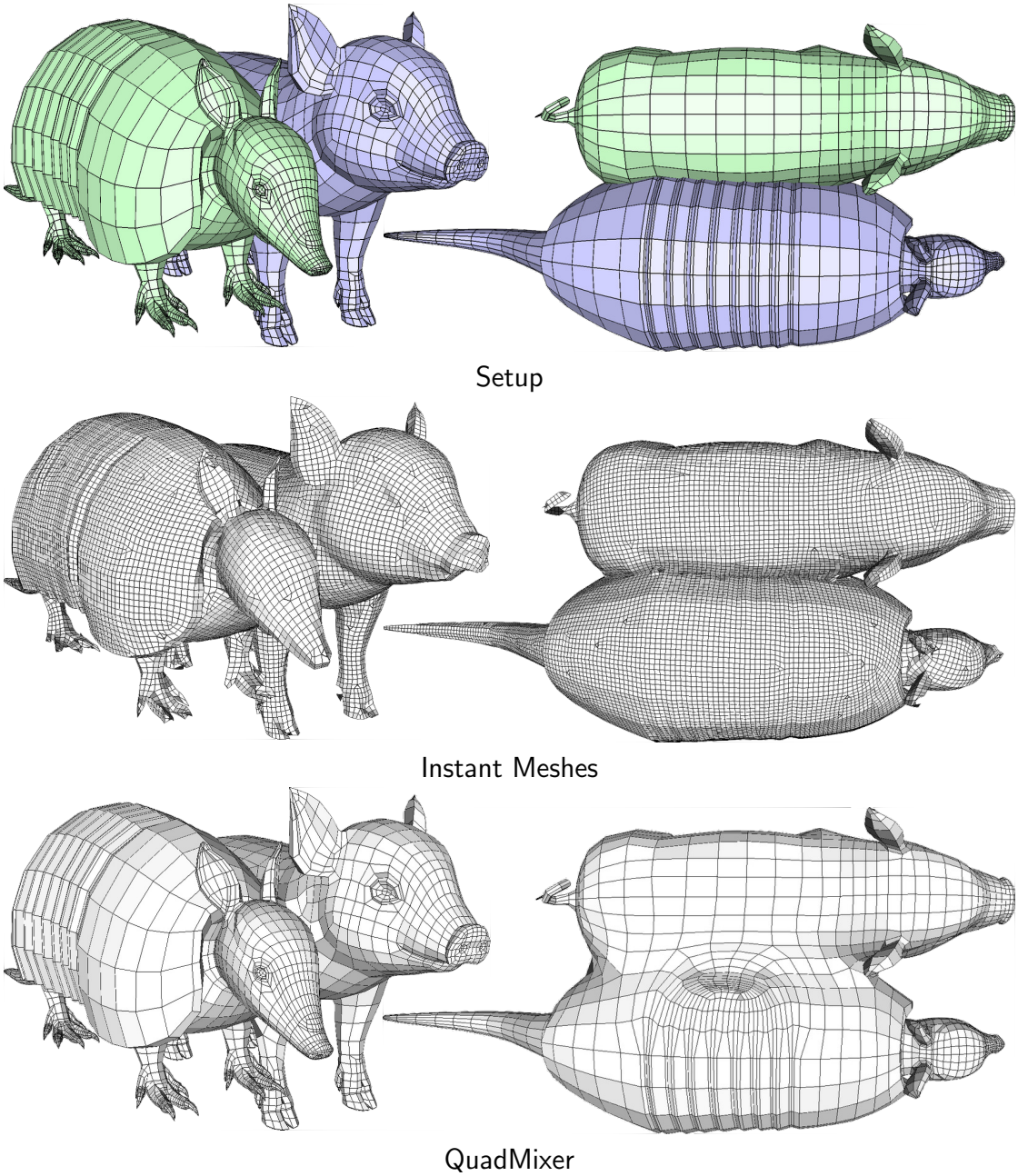


Figure 3.3: **Comparison of QuadMixer and remeshing approaches.** Current solutions ([JTPS15] for example) require to re-mesh entirely the results of the boolean operation and original connectivity is lost. Our system efficiently preserves the connectivity and is capable of blending two different connectivities

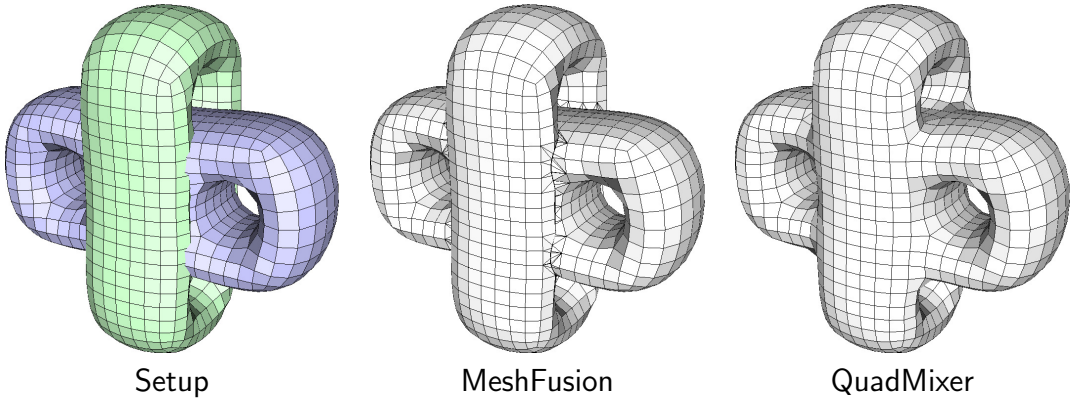


Figure 3.4: **Comparison of QuadMixer and MeshFusion [Vis18].** The difference in the intersection portion of the mesh is noticeable.

- We select the patches that have not been modified by the boolean operation. We retract the sides of the patches that are partially affected by the boolean operations. Those patches are the ones that contain only a subset of the original set of quads (see Figure 3.5.c). 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 3.5.c) which share a common boundary.

- We smooth these internal patches to generate a fair geometry surface, more straightforward to be nicely quadrangulated. The user can control the amount of introduced smoothing.

- We trace a set of internal patches $\mathcal{P}$ on the triangulated mesh $\mathcal{T}$ (see Figure 3.5.d). This step uses the definition of a cross-field [DVPS14] and applying a tracing algorithm [CBK12].

- 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 3.5.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 [TPS14] (see Figure 3.5.f) obtaining a new quadrangulated mesh $\mathcal{Q}^1$. The union of $\mathcal{Q}^1$ and $\mathcal{Q}^0$ provides the final result.

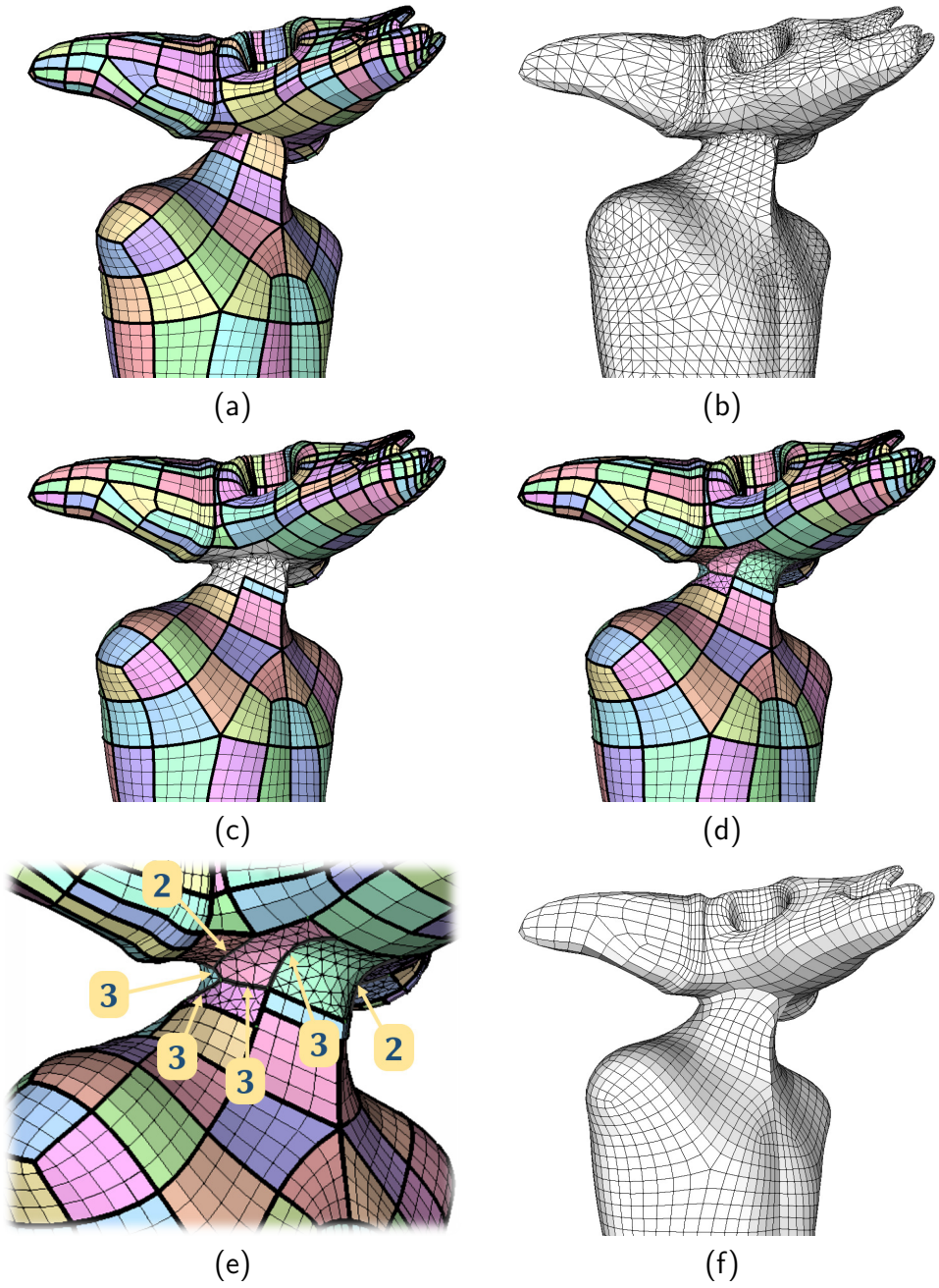


Figure 3.5: **Overview of our processing pipeline.** (a) Given two separate quadrilateral meshes we compute an initial patch layout for each; (b) then quads are split to form triangular meshes, which are then combined. (c) We update the patch layout for the patches that are affected by the boolean operation. (d) We split the triangulated portion of the surface into sub patches. (e) We derive the optimal subdivision for each side. (f) We perform the final quadrangulation.

3.2 Methods

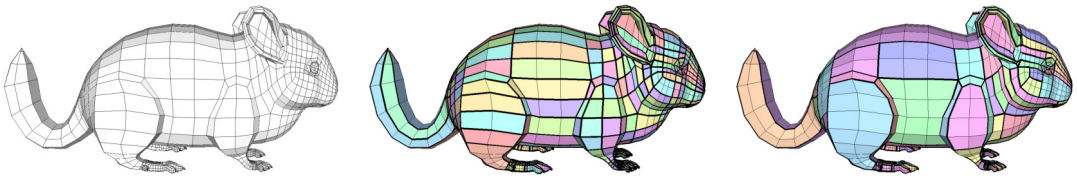


Figure 3.6: **Quad patch layout extraction.** On the left: the input quad layout; in the center: the patch layout with separatrices; on the right: the patch layout typical of motorcycle graph.

The input to our method consists of pure quadrilateral meshes. They can be the result of automatic methods or, more usually, models produced by a digital artist. The first step of our pipeline extracts a quad patch layout. This step is not difficult and consists of tracing all the separatrices stemming from irregular vertices (see Figure 3.6). Each separatrix is defined as a sequence of edges starting at a singular vertex (i.e., a vertex of valence different from four) and ending at another singular one. When applied to manually-modeled meshes, this process typically produces well-structured and compact patch decompositions, since the tools used by the artists tend to align the singularities naturally. As an alternative, we can contemporanily propagate all the separatrices and stop tracing each separatrix as soon as it crosses another one. Literature usually describes this procedure as tracing *motorcycle graphs* [EGKT08]. 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 [TPP+11, BLK11]) is not useful in this context since it might modify the original edge flow designed by the artist.

3.2.1 Optimal Patch Retraction

Given two pure quadrilateral meshes $\mathcal{Q}^A$ and $\mathcal{Q}^B$, along with their original quad patch layouts (Figure 3.7.a), we start by splitting each quad element along its smaller diagonal 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 [ZGZJ16]. 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

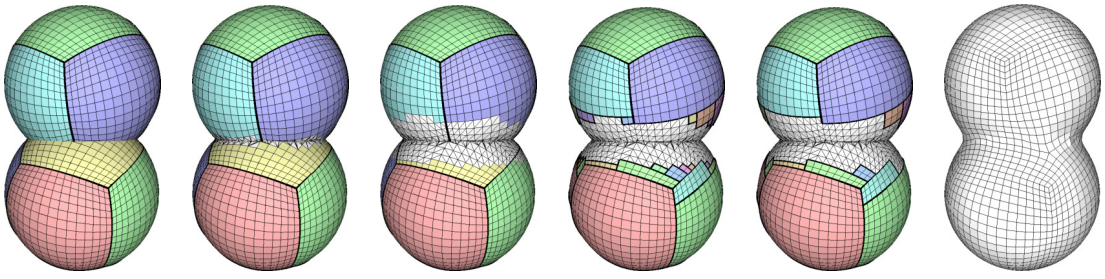


Figure 3.7: **Patch layout retraction.** (a) The initial patch layout of the two meshes. (b) The two quad meshes are split into triangle meshes to perform the boolean operation, and then the triangles are clustered to recompose the original quads when possible. (c) Original patches are retracted to maintain 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.

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 do not contain triangles not changed in the boolean operation (Figure 3.7.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 between quadrilateral layouts smoothly. For this area, we consider the quads on each side where their geodesic distance to the intersection line is below a given threshold of δ_r which is proportional to the average edge of the retracted patches (Figure 3.7.c). At this point, we have a collection of un-organized quads that we need to assemble into patches by building a new layout. The idea is to prefer the formation of large, compact rectangular patches with a regular and straight boundary with the remaining triangulated surface. For this purpose, we repeatedly search, in the set of un-organized quads, the largest inscribed rectangle composed only by quads not associated with any entirely preserved patch (Figure 3.7.d). Specifically we use the *largest rectangle in a histogram* (see [Mor94], chapter 21) algorithm to generate this new set of rectangular patches.

Finally, we perform a pruning step that eliminates all the newly created patches having a number of forming quads below a given threshold (Figure 3.7.e). We set this threshold as a fraction of the average area of the current patches.

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

3.2.2 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 [BZK09]. 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. [TPS14] requires the number of sides of the input patch to be between three and six. As a consequence, to use this method in our pipeline, we need to split the triangulated surface $\mathcal{T}^0$ into patches respecting these requirements.

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. 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 (Figure 3.8):

- 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

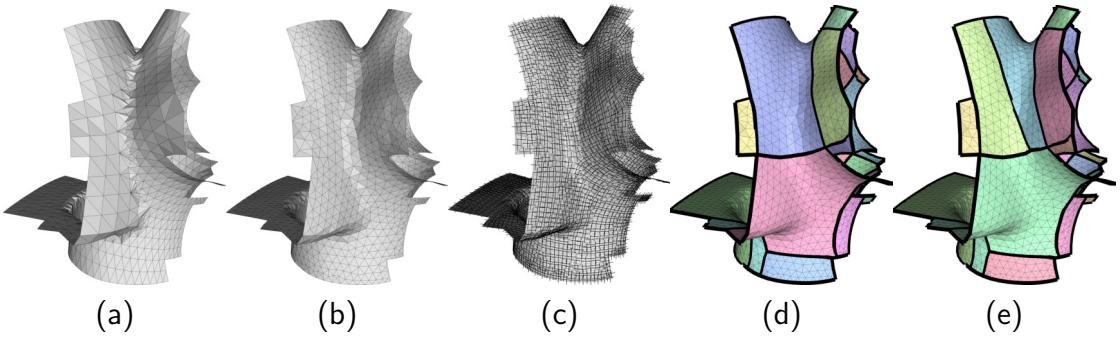


Figure 3.8: **Patch tracing procedure.** (a) The Initial patch; (b) The re-meshing step; (c) Cross field computation; (d) Concave corner tracing; (e) Final splitting of the patches to match the requirements. This patch is the one shown in the accompanying video when displaying the two intersecting fertility models.

the overall approach. We use the iterative approach included in the Meshlab framework [CCC+08].

- We compute a smooth cross field that conforms to the boundary using the poly-vector field smoothing [DVPS14]. 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 [CBK12].

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

Patch configuration After computing a cross-field on the triangle mesh $\mathcal{T}^0$ (for a more exhaustive description of cross-fields see [VCD+17]), 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 3.8 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 3.9): 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 3.9) 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.

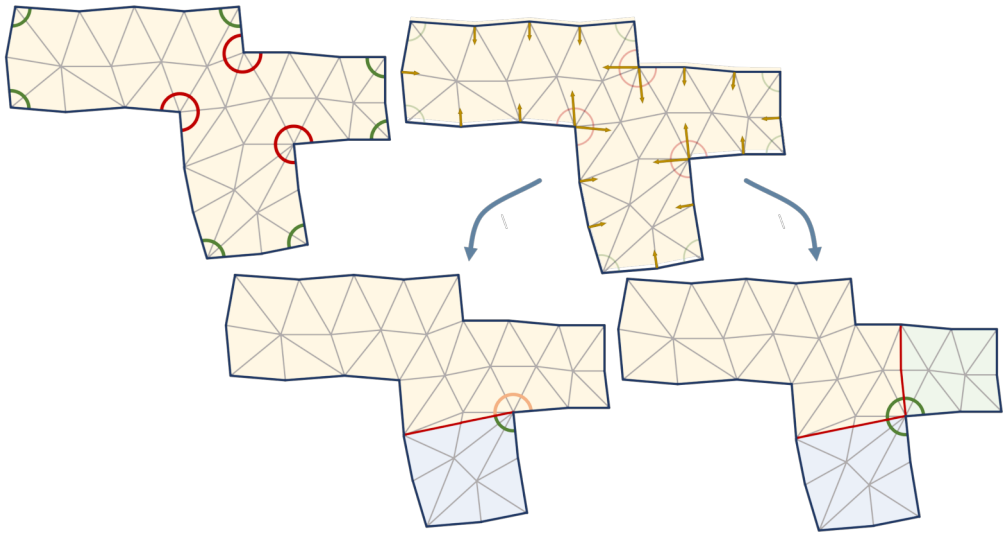


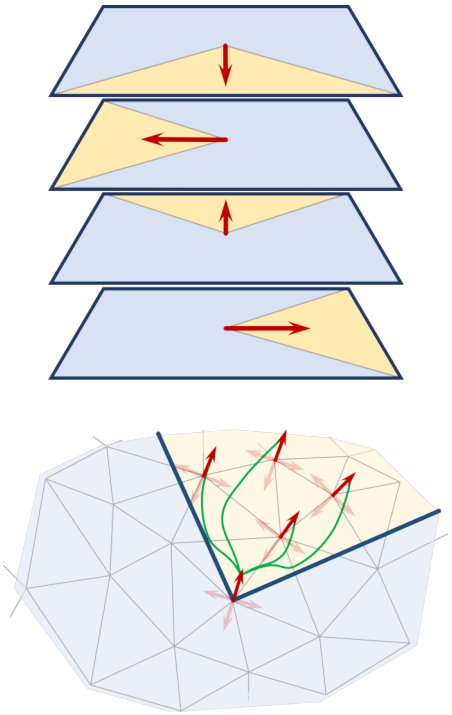
Figure 3.9: **Corner classification.** On the top left the initial corner classification: we mark each angle either as convex or as concave; on the top right we show the directions stemming from concave vertices that we use for splitting the patch in quads; on the bottom we show how we can split the 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.

Each split changes also the classification of the associated boundary vertex as follows (see Figure 3.8 for examples of this reclassification):

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

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

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 [CBK12]. Following [KNP07], given a surface M, with the exception of the singularities, we make four copies of each point $p \in M$. We associate each copy to one direction of the cross-field. Then each copy of p encodes both its position and one of the orientations of the cross field. This process creates $M4$, which is a stratification of the original manifold surface M. Space $M2$ is the quotient space of $M4$ obtained by identifying pairs of opposite directions. Hence $M2$ is composed only by two sheets. Each sheet encodes a line-field. We discretize $M4$ for tracing following the approach proposed by [CBK12]. 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 [CBK12].



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.

In our experiments, we always have accurate results and, thus, we do not use more sophisticated trace strategies (e.g., the ones described in [MPZ14] or [PPM+16]). 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 [CBK12]. 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 3.10 shows the difference between the two types of intersections.

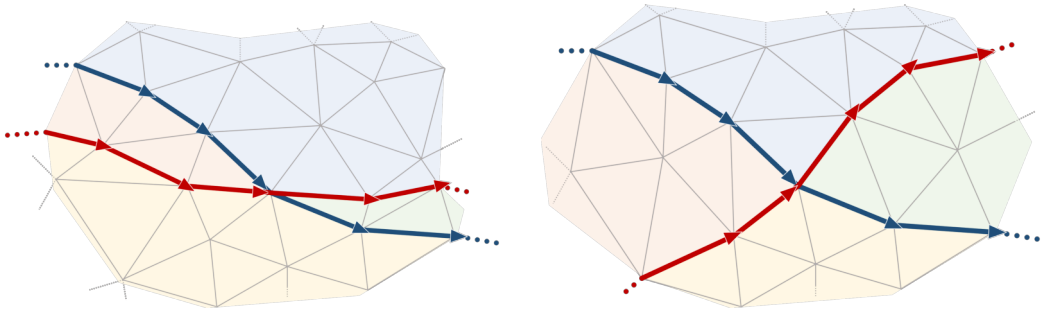
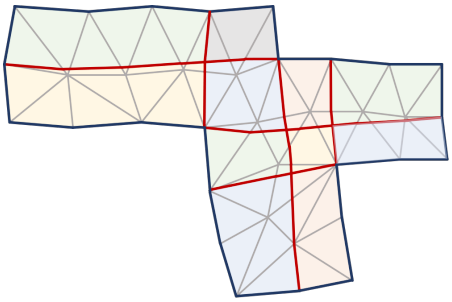


Figure 3.10: **Trace intersections.** Tangential (left) and orthogonal (right) path intersections.

Concave vertex tracing We trace the polylines that split the patches, choosing first the ones that have an end-point in a concave vertex. When multiple alternative traces are available, we follow these selection heuristics: (i) we never add a new trace if it introduces tangential intersections; (ii) we favor traces starting from concave vertices not yet split; (ii) we prefer shorter traces to longer ones.



As we repeat the process in the sub-patches still having concave vertices, we dramatically reduce the probability of tangential collisions. At the end of the process, all concave vertices should vanish, and we have a correct convex patch layout. This condition is necessary to make the procedural quad mesh generation of [TPS14] to generate high-quality quadrilateral elements. Although we have no theoretical guarantee to be able to split all concave corners, we never encountered any failure case in our editing session, even for complex configurations.

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

Discussion Another possible strategy to procedurally quadrangulate a disk-like 3D surface has been proposed by Peng et al. [PBJW14]. This approach requires first to derive a bijective parametrization of the triangular patch; then the final patch

layout is produced by tracing straight lines in the bidimensional parametric space. While we share some ideas with that approach (like the classification between concave and convex corners and the emanating directions), we do not require any bijective parametrization, and this is a significant advantage. The decision to trace paths directly on the surface is crucial to make the method general and reliable. Constructing a low distortion bijective parametrization can be difficult and time-consuming for the general case and even particularly tricky for the thin regions which can result from the boolean operations. Moreover, designing a cross-field that aligns to boundary constraints leads the traced subdivision to blend the flow among the existing quadrangulations smoothly.

3.2.3 Subdivision Optimization

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

(i) A patch admits a quadrangulation only if the sum of boundary subdivisions is even [TPS14].

(ii) The subdivisions on the boundaries are constrained to match the preserved quadrangulated mesh.

(iii) To increase the *isometry* of the tessellation, we penalize the discrepancy of each side with respect to its ideal subdivision. We compute the ideal subdivision for each patch that has a border by averaging the edge size of adjacent quads. Then we propagate that value on internal patches, and we smooth between adjacent patches in order to discourage abrupt changes of resolution.

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

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. This variable represents the number of edges into which the sub-side will be subdivided in the final quadrangulation. 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

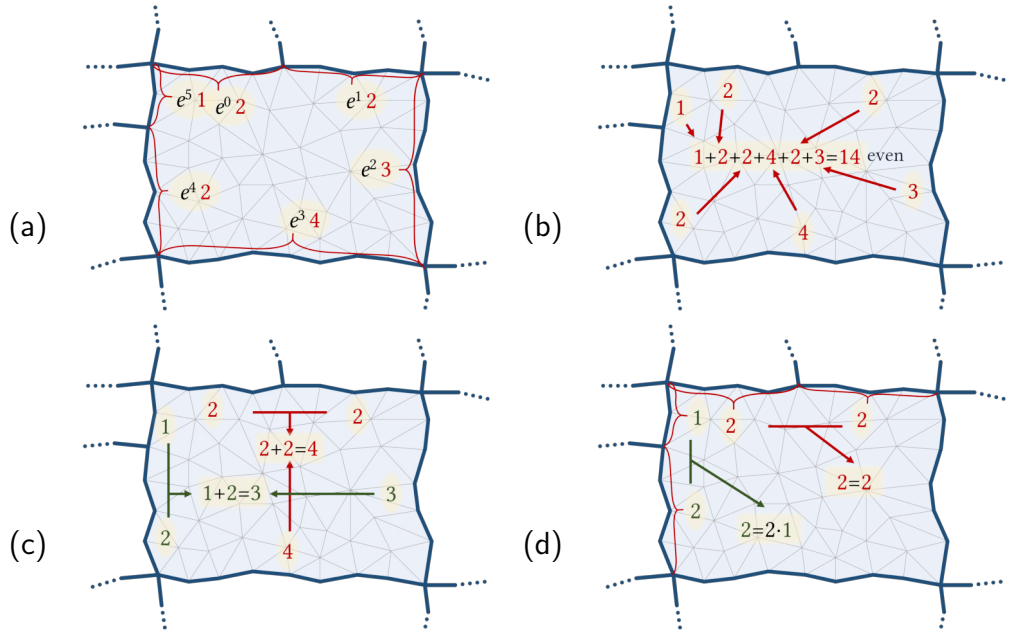


Figure 3.11: **Optimization procedure inside a patch.** The figure illustrates an example with actual values. In (a) each side is divided into sub-sides, considering all incident t-junctions of adjacent patches (numerical values in red); in (b) we show that the sum of all the edge sizes must be even; 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.

the subdivision with respect to the adjacent patches. An example of edge variable definition is shown in Figure 3.11.a.

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

$$\begin{array}{ll} \min & \sum (e_i - \hat{e}_i)^2 \\ \text{s.t.} & e_i \geq 1 \end{array} \tag{3.1}$$

The ideal size $\hat{e}_i$ is a decimal number calculated as a function of the edge size in the fixed boundaries (i.e. the subdivision values of the original quadrangulated meshes). This information is defined for each chart in the boundaries, and then it is smoothly propagated in the inner charts.

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.

$$e_i = q_i \quad \forall e_i \in B \tag{3.2}$$

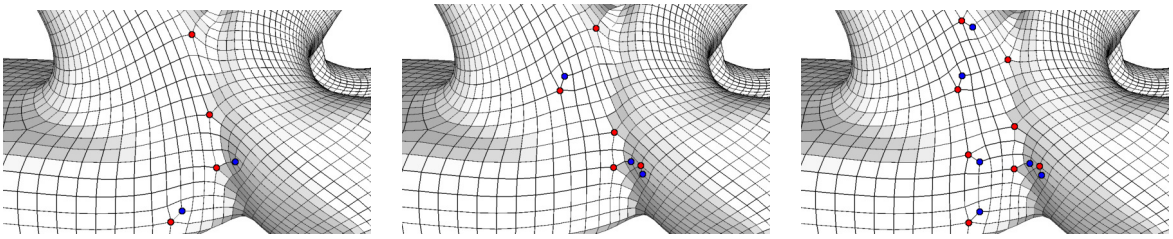
As previously stated, a quadrangulation is only possible if the sum of its subdivision is an even number. Hence, for each patch P_k, we include an additional linear constraint:

$$\begin{array}{c} \sum e_j = 2n \quad \forall e_j \in P_k \\ n \geq 1 \end{array} \tag{3.3}$$

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:

$$\begin{array}{rl} \min & (\sum\limits_{e_u \in S_0} e_u - \sum\limits_{e_w \in S_2} e_w)^2 \quad + \\ & (\sum\limits_{e_u \in S_1} e_u - \sum\limits_{e_w \in S_3} e_w)^2 \end{array} \tag{3.4}$$

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 properly normalized energy terms expressed by equation 3.1 and 3.4. The effect of this parameter on the final quadrangulation is quite intuitive (see Figure 3.12). We verified in our experiments that a value of $\alpha = 0.5$ is a good compromise between regularity and isometry. For complex quad layouts, we can use norm one minimization in equation 3.1 to speed up the entire process, in such case, both equation 3.1 and 3.4 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 3.12: **Regularization term effect.** The regularization term governs the distribution of singularities and the isometry of the tessellation.

3.2.4 On the existence of a valid solution

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

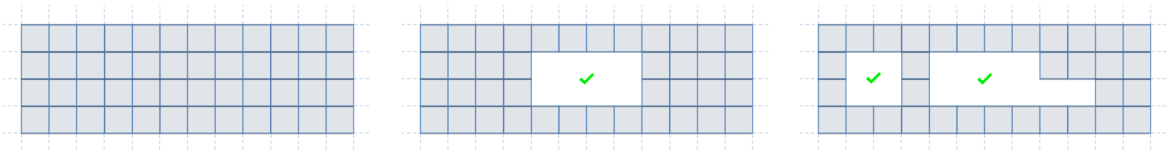


Figure 3.13: **Solvability for closed topologies with genus zero.** Removing quads from a watertight pure quadrilateral mesh always generates boundaries of an even number of subdivisions.

Luckily, this happens only in particular cases. In the general case, when blending two genus-zero quadrilateral meshes, the intersection curves will always define over each mesh disk-like regions that, by construction, will be bounded by an even number of quads (Figure 3.13). However, when higher genus meshes are involved, it can happen that the intersection curves cut out, over one of the meshes, regions with a more complex topology that are bounded by multiple boundaries. In this case, it still holds that the overall sum of the subdivisions of all the boundaries is even, but single boundaries can have an *odd* number of subdivisions. Figure 3.14 shows such an example: the intersection between a torus with a pentagonal section and a box. In this case, every single boundary over the torus will have five sides, and the connected component identified by the intersection curves over the torus (with a non disk-like topology) has an overall even number of sides (10). However, the two blending regions that we need to quadrangulate will have an odd number of sides, making their direct quadrangulation not possible. To solve this problem, we could refine the whole connected component with non disk-like topology in order to make these boundaries even. However, to make this modification minimal, we search for the shortest polychord, connecting two odd boundaries and we refine only this strip of quads [DSSC08].

Note that it is always possible to find such a connecting polychord. First consider that the overall number of the edges on the open boundary is even (by construction: they have been generated by removing quads from a closed mesh). Without loss of generality assume we have just two open boundaries A, B both with an odd edges number. Consider all the polychords exiting from the edges of A; by contradiction suppose that there is no polychord going from A to B, then all

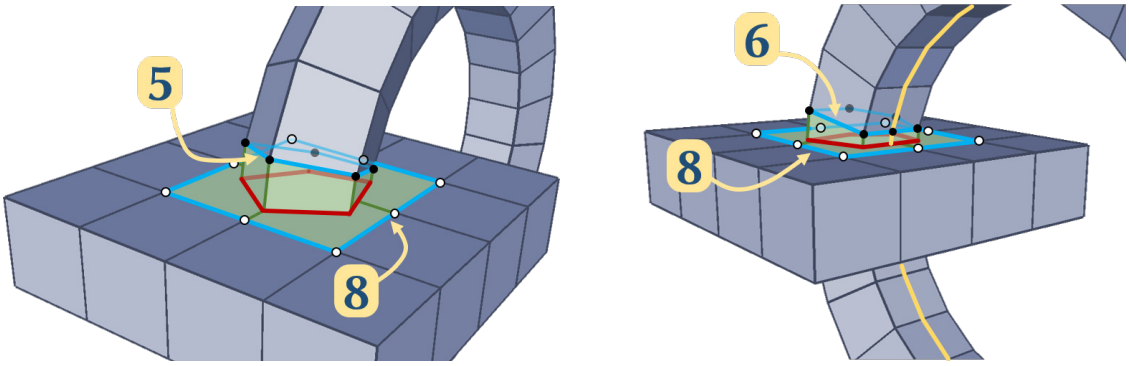


Figure 3.14: **Non-feasible example.** A torus with a pentagonal section intersects a block (left). The patch surrounding the intersection curve has a boundary with an odd number of sides (5 on the torus and 8 on the block) and therefore cannot be quadrangulated. Refining a polychord makes the boundary even (6 and 8 sides) and the patch become quadrangulable.

the polychords start and end on A covering an even number of edges, that is wrong because we assumed A has an odd number of edges. To handle the more generic case where you have more than just two open odd boundaries, first, consider that all the even boundaries could be *virtually* closed by quads in any way, leaving only an even number of odd boundaries; it is then evident that these remaining boundaries can be handled, pair by pair, as explained.

3.2.5 Final quadrangulation

At this point, we have a set of disk-like triangle patches whose sides are between the limits imposed by the quadrangulation algorithm. Also, every patch has a single boundary. We then map the boundary of the patch onto the borders of a regular polygon and use this mapping as a constraint to parameterize the interior using least-squares conformal maps [LPRM02]. We compute the quadrangulation in parametric 2D space using [TPS14], and then we interpolate the 3D positions of the vertices in parametric space.

3.3 Implementation details

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

3.3.1 The detaching tool

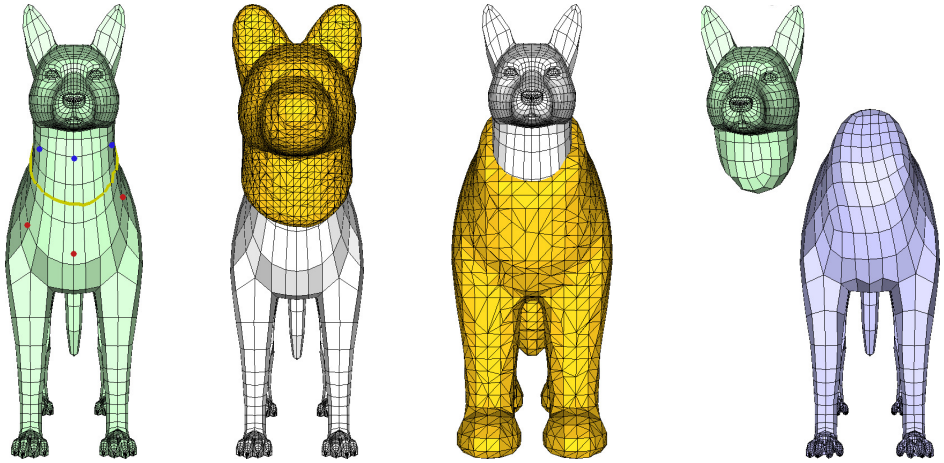


Figure 3.15: **Detaching tool.** The user select a sequence of opposite points on the model (a); Two offset surfaces are created (b) and (c); Two pieces are obtained using the intersection with the offset surfaces.

To detach components we followed an approach similar to the one proposed in [JLCW06]: the user has to select pairs of opposite points on the surface that defines a smooth polyline loop that splits the mesh into two separate components. Once we have divided the mesh into two distinct parts, we close the holes, and we create an offset surface that is used to detach the portion of the mesh using an intersection boolean operation. Figure 3.15 shows this pipeline.

3.3.2 Smoothing the surface nearby the intersection curve

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

We apply this initial smoothing step on the triangulated mesh resulting from the first boolean operation. We first select the intersection curve, then we propagate a geodesic from the intersection curve toward the interior, and we choose the subset of vertices whose geodesic distances are below a certain threshold (we use a 5% of the diagonal of the bounding box). Then we perform a Laplacian smoothing on this subset of vertices. Intuitively, the vertices that are close to the intersection lines should move more than the ones that are far away. To obtain this effect, we linearly weight the effect of the smoothing w.r.t. to the geodesic

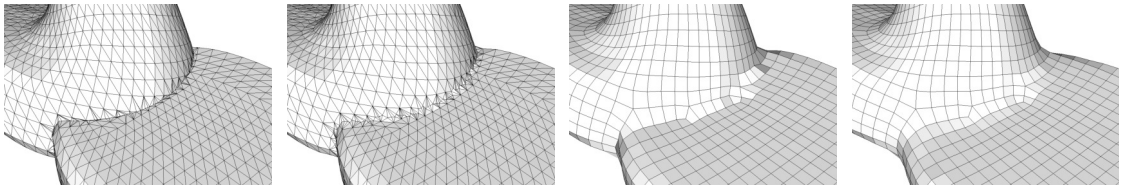


Figure 3.16: **Intersection curve smoothing effect.** (a) The result of the boolean operation; (b) the first smooth steps on the triangulated mesh; (c) The quadrangulation step with tangent space smoothing; (d) the final result after the last step of Laplacian smooth.

distance from the intersection line (see Figures 3.16.a and 3.16.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 3.16.c and 3.16.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.

3.4 Results

We performed our tests on a laptop computer with an Intel i7-8750H processor with 16GB of RAM. We used Gurobi [GO18] to solve the minimization of Section 3.2.3. All the code is single-threaded and not highly optimized; it has been implemented using the VCG Library [CNR13], CG3Lib [MN21], libigl [JP+16], Eigen [GJ+14] and CGAL [FP09]. 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 3.19 shows some of the first steps of the test. In Figure 3.17, we show a full set of combinations among six meshes having different topologies, complex connectivity, or intricate geometric details.

We tested our method on a collection of professionally designed quadrangulated animals. Some of the results are shown in Figure 3.18. The presented models have been produced through interactive editing sessions by the authors. We composed models by simply detaching portions of the body from one animal and combining them with the body of another animal. Our method always succeeded in producing a two-manifold quadrilateral surface. Most of the original quad layout has always been successfully preserved, and our method was able to create a smooth flow in

the blended portions of the meshes. The operations have always been performed within 1 second for meshes in the animal dataset.

The merging operations of Figure 3.19 required up to 15 seconds for some of the blending operations. As expected, the running time is proportional to the number of triangles in the blending area that in this specific test case can involve most of the original surface. Timings are summarized in Table 3.1.

Figure 3.20 reports the distribution of distortion in individual elements relative to the experiment shown in Figure 3.3. Distortion has been computed by using the distance with respect to the ideal quad as defined in [PTP+15]. As shown in the histograms, our method does not affect the overall quality of the quads.

Table 3.1: **Time efficiency.** Execution time of each step of the pipeline for the examples generated with QuadMixer. Times are all in millisecond with the exception of the total time which is reported in seconds.

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

3.5 Discussion

We proposed a novel powerful pipeline for freely composing quadrilateral meshes. Our method takes advantage from boolean operations to smoothly blend between

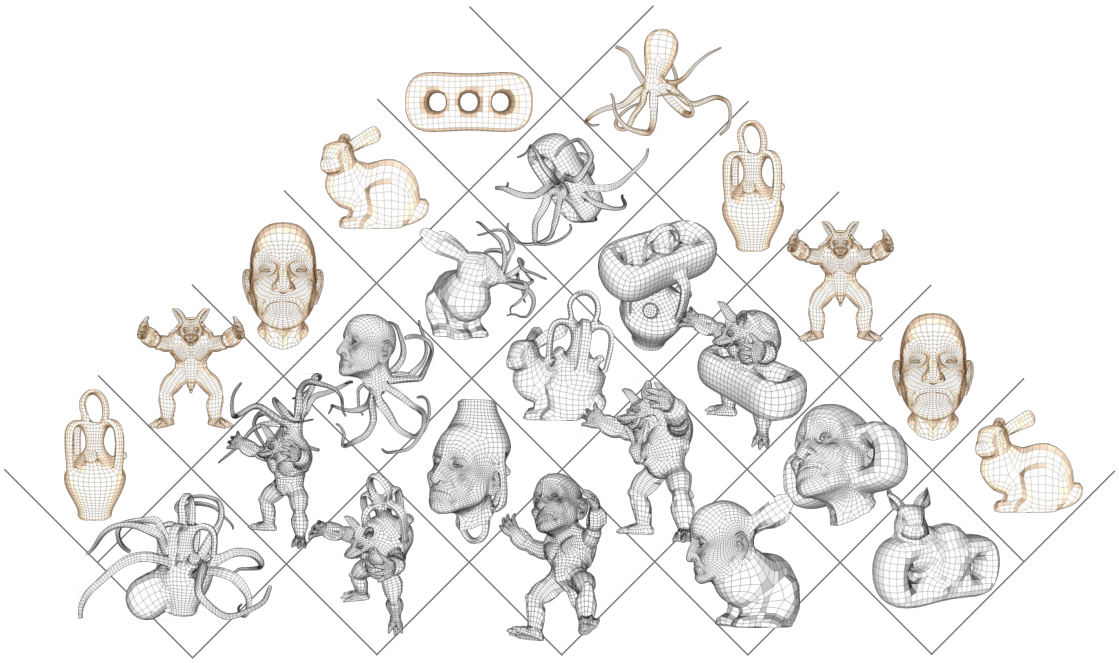


Figure 3.17: **Results.** All the pairwise joins of six meshes, different in genus, complexity, and details.

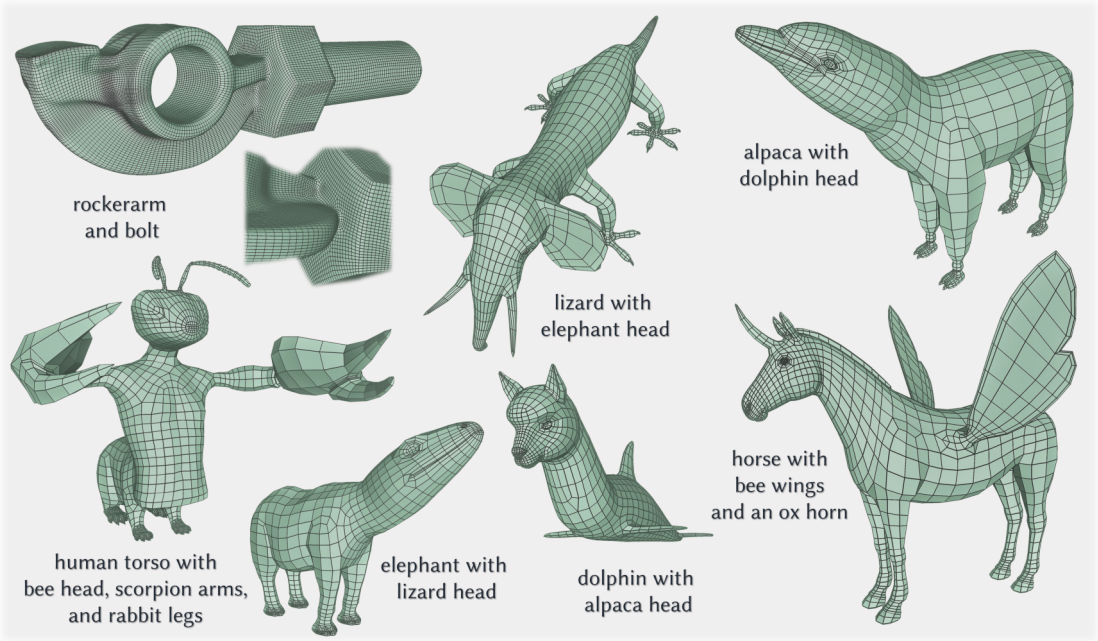


Figure 3.18: **Editing session results.** An overview of some results obtained with our modelling tool.

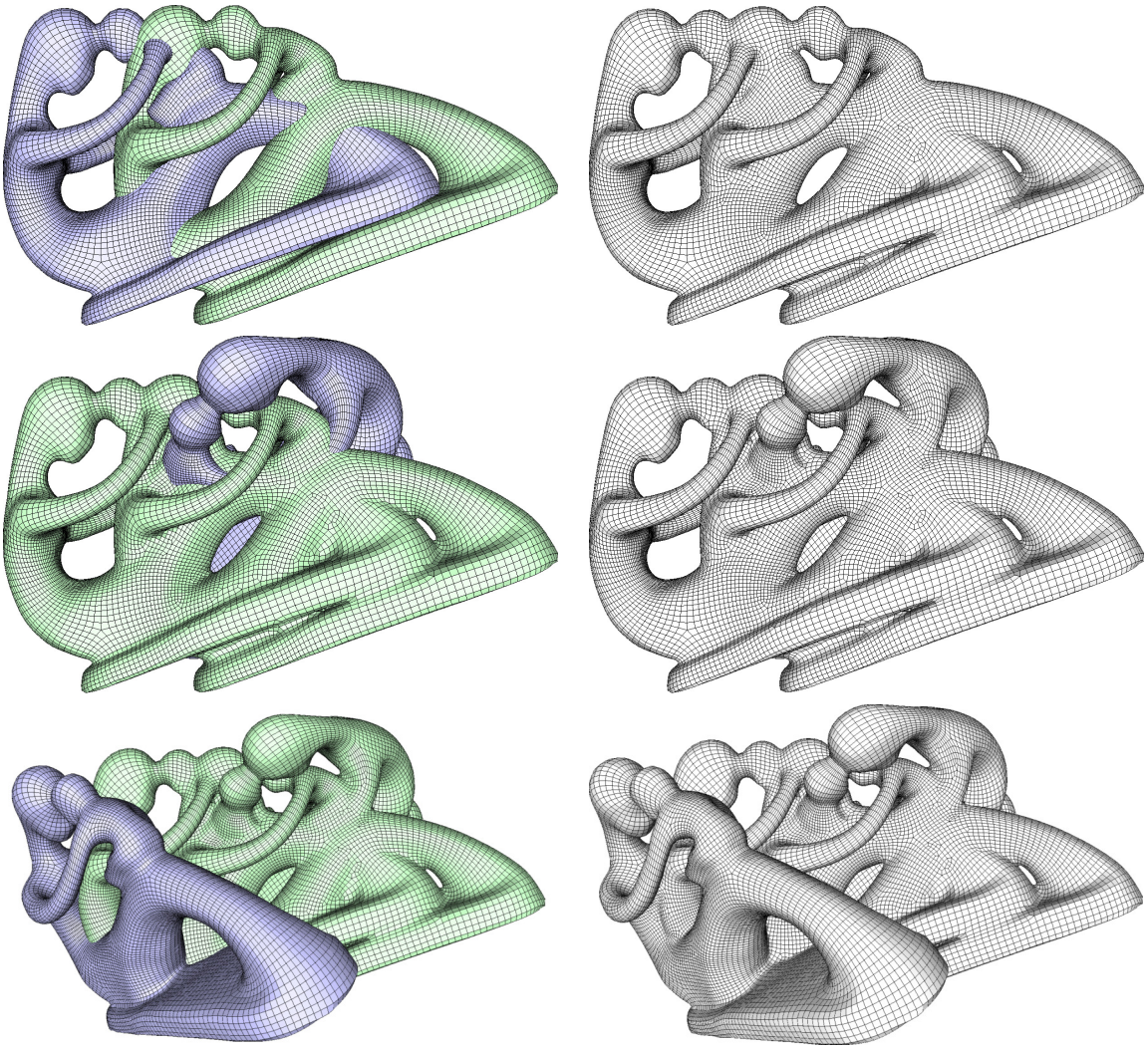


Figure 3.19: **Robustness.** Iteratively merging the fertility model to check the robustness of the proposed approach.

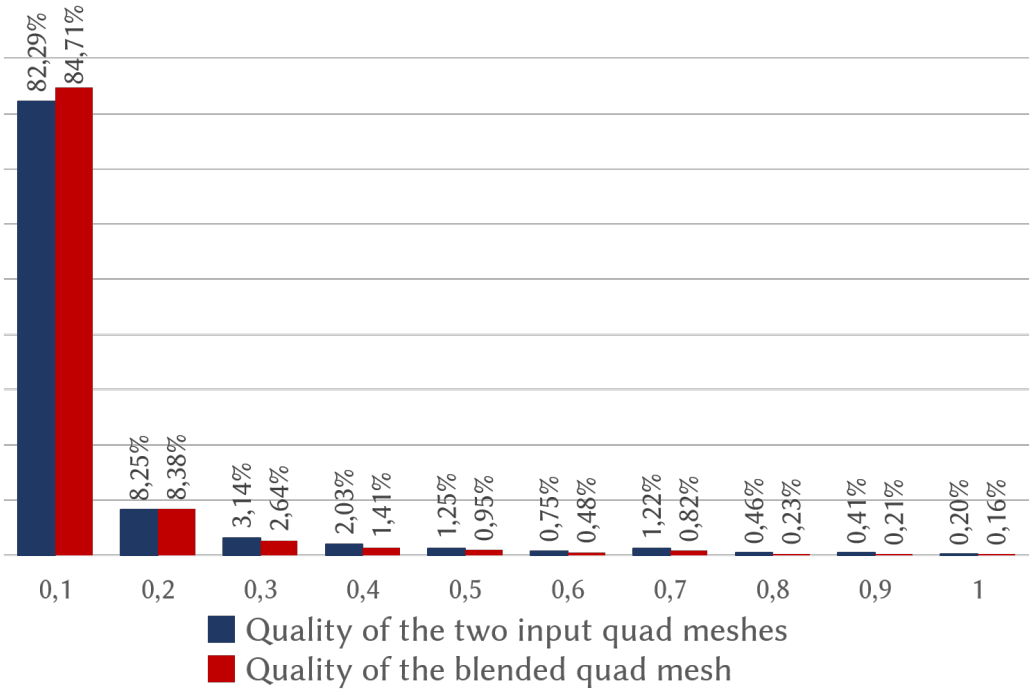


Figure 3.20: **Quad distortion.** Distortions of the quads of Figure 3.3 measured using the metric defined in [PTP+15] (0 means no distortion).

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 (Figure 3.21).

3.5.1 Limitations and Future Works

This method works only for watertight manifolds. In fact, it relies on the Boolean operations described by Zhou et al. [ZGZJ16] that are unable to process non-watertight or non-manifold shapes. Moreover, our input meshes must be composed of only quadrilaterals. Indeed for triangle or quad-dominant meshes we cannot guarantee the existence of a valid quadrangulation. In Chapter 4 we present an extension of this method that allows to blend also quad-dominant, non-watertight or non-manifold meshes.

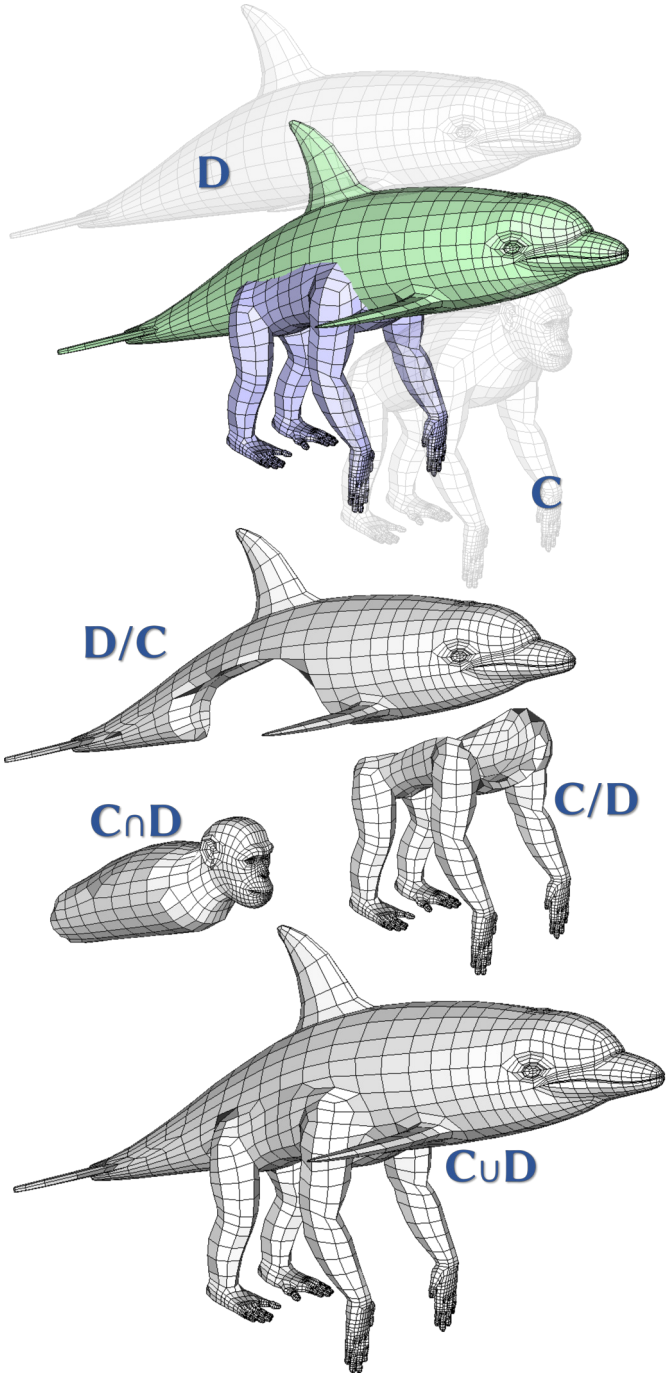


Figure 3.21: **Boolean operations on quadrilateral meshes.** Our algorithm can mimic any boolean operation on quad meshes generating a well-shaped quad mesh which reuses the most of the initial layout. We show here, from left to right: the input consisting of two quad meshes (a dolphin **D** and a chimpanzee **C**) interactively placed in the scene; the two differences; their intersection; the union, which is, typically, the most interesting operation from a semantic standpoint.

Our technique, currently, cannot efficiently preserve sharp features, as shown in Figure 3.21. An exact boolean operation will introduce sharp features, especially for the difference operation. However, it is usually not a problem for the blending of models designed by artists, where we prefer to have smooth surfaces near the intersection curve. In any case, 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 3.22), which can be attributed to the intrinsic limitations of quad mesh modeling. One practical solution to this problem consists in matching the resolutions by using some subdivision steps before performing the boolean operation. Figure 3.23 instead shows the sensitivity of the method to two different initial patch layouts.

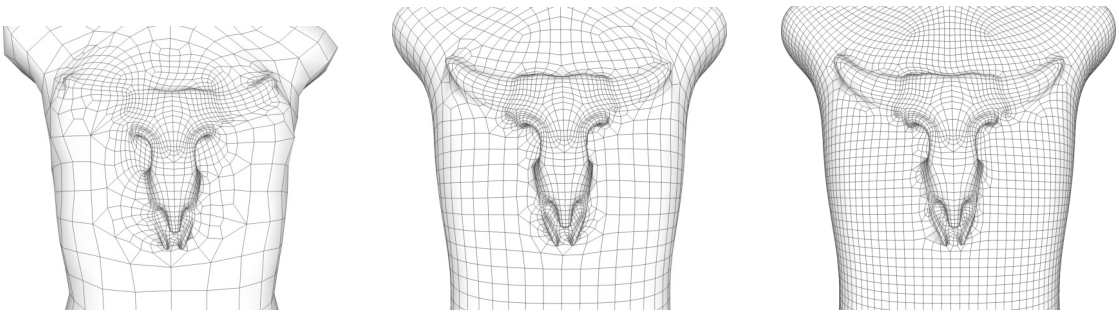


Figure 3.22: **Different resolution limitation.** Blending of meshes with different resolution can result in poor surface preservation.

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 3.24 (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,

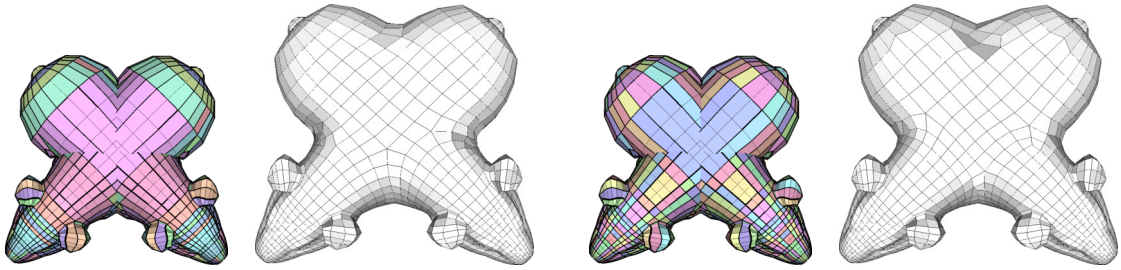


Figure 3.23: **Sensitivity with respect to two different initial patch layout.** Motorcycle graph (left) and emanating separatrices (right).

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 [PTP+15] 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 3.24 (bottom).

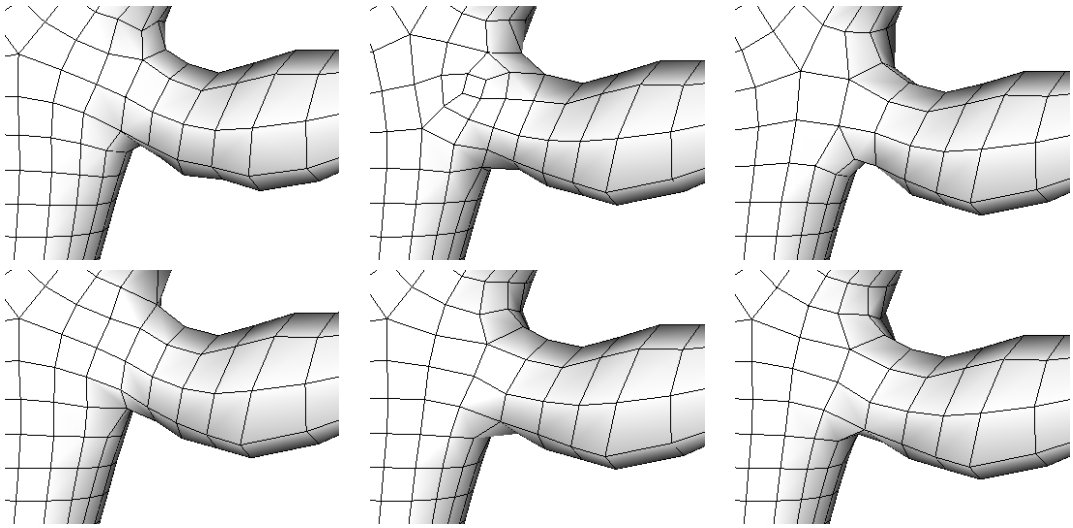


Figure 3.24: **The variation of the tessellation configurations.** Close geometric configuration might cause abrupt changes in the tessellation (top); such an artefact can be mitigated with simple improvements (bottom).

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

result since with such a configuration the original quad structure could probably not preserved.

Acknowledgments The base meshes of the animals shown in figures 3.1, 3.3, 3.6, 3.15, 3.18, and 3.23 have been professionally modeled by Paul Gonet (gonzou3d.carbonmade.com) and are available on BlenderMarket. The author granted us the right of redistributing the remixed models shown in the figures under the CC-BY-NC-SA license.

Chapter 4

SkinMixer: composing skinned models for animation

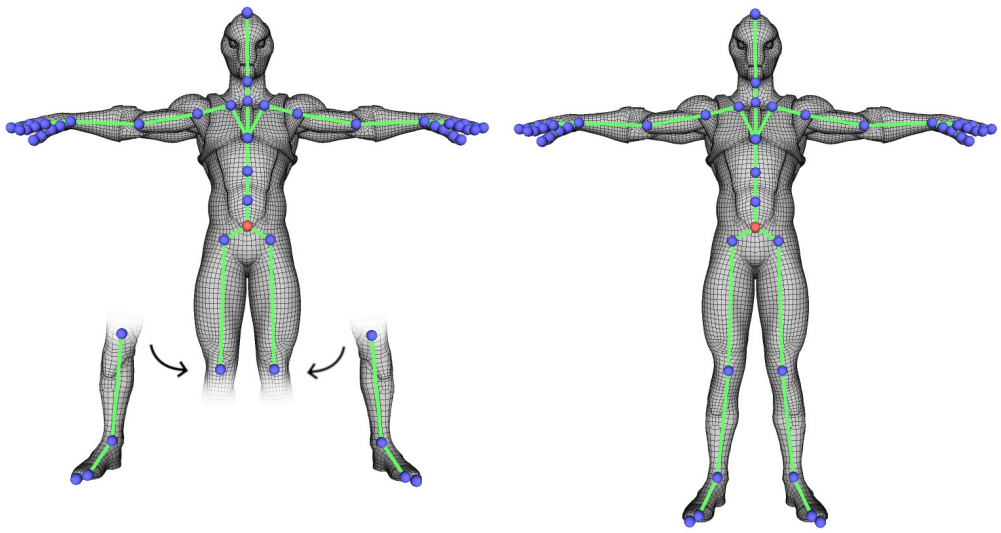


Figure 4.1: **SkinMixer, the proposed blending technique.** We can assemble components of different models with arbitrary complex geometry. In the process, also the data structures for skeletal animations are blended accordingly.

In the last chapter, we introduced a robust approach for *blending* components of watertight pure quadrilateral meshes. Unfortunately, the assets for animation in the industry (for example for video-games or films) are often composed of quad-dominant meshes with open boundaries, for which not even manifoldness is guaranteed. *QuadMixer* relies on the Boolean operations described by Zhou et al. [ZGZJ16], hence it cannot deal with this kind of meshes. Furthermore, it

considers only the surface and ignores the data structures for skeletal animations of the input models.

Successfully composing not only the geometry but also the skinning information could have a huge impact on the industry. Indeed, despite the research efforts in the past decades, the skinning process is still usually manually performed [JDKL14]. Skinning (or rigging) is the task of creating a skeleton that is embedded into the surface and determining how its bone transformations influence the deformation of each vertex in the surface mesh. As we already introduced in Chapter 1, these tasks are highly time-consuming and are usually performed by high-skilled artists. Reusing parts of high-quality designed character could certainly reduce the costs of these such expensive tasks.

Hence, we propose *SkinMixer*, a novel technique to compose parts of skinned models, taking into account not only the surface but also the data structures for skeletal animations. We perform operations on the skeleton joints of character models that allow us to remove, detach, and *merge* their components. The surface meshes, skeletons, and skinning weights are *blended* into a new model that is immediately usable in animation pipelines.

The idea behind this method is to exploit the skinning information to guide the entire *blending* process. Indeed, the skinning weights contain semantic information that link surfaces to skeletons and vice-versa. This approach allows us to automatically determine the resulting surface for each joint-based operation.

The method for the surface composition uses the retopology method described in Chapter 3, but it is extended to work on arbitrary complex surface meshes. Briefly, it *blends* volumetric signed distance fields in order to obtain an isosurface that describes the output. Differently from the boolean operation, the result does not present sharp features in the intersection curves, but natively smoothly merges the input surfaces.

We summarize the main contributions of this chapter as follows:

- We define a robust method that smoothly *blends* arbitrary complex surfaces into a quad-dominant mesh, preserving most of the layouts.

- We define a robust method that *transfers* the skinning weights of the two input models into the new model, preserving most of the original information and *blending* where necessary.

- We exploit the semantic information of the skinning weights to guide the blending processes.

4.1 Overview

Given two model characters C_1 and C_2, our method *blends* components of them into a new model C. Each model is composed of a surface mesh, a skeleton, and their related skinning weights. Similarly to *QuadMixer*, the original information of the input models are preserved as much as possible and merged accordingly only where needed.

Artists design skeletons as a hierarchical structure in which the joints (or handles) are the articulation points through which we can control the model deformations. This forms a tree structure, where nodes are the joints that are connected by several directed segments, called bones. Clearly, these joints are placed in key points that have a strong semantic meaning. For example, in a human character, we will usually find a skeleton joint for each corresponding articulation of the human body. Figure 4.2.a shows the skeleton for a human arm.

Furthermore, the skinning weights describe for each vertex the *amount of influence* of the bones, and therefore, the influence of their transformation during the animation process. Obviously, they also contain meaningful semantic information that links the skeleton joints to the mesh vertices. Figure 4.2.b shows the skinning weight values for a joint.

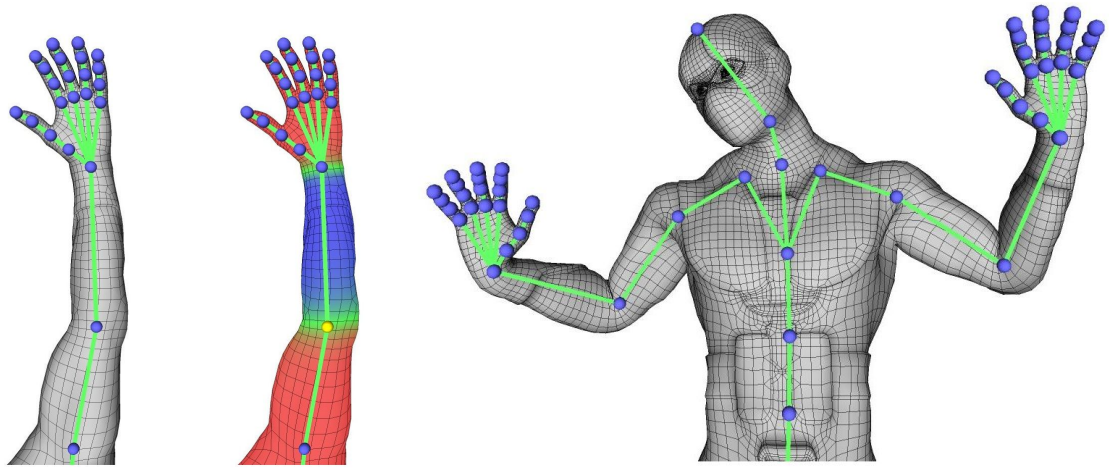


Figure 4.2: **Rigging skeleton and skinning weights.** The rigging skeleton of the arm of a human character (a), the skinning weights related to the joint in yellow (b), and a pose obtained applying a Dual Quaternion Skinning method [KCvO07] (c). In (b) the colors represent the amount of influence: blue maximum influence (1), red zero influence (0), and green middle value (0.5).

Figure 4.3 shows some of the input models used in this chapter. The humanoid characters are all taken from the Mixamo framework [Bla14], while the Elephant in Figure 4.3.d belongs to a dataset available in the Unity Asset Store (see Acknowledgments at the end of the chapter).

We propose an operation-based interactive process, in which each operation is performed on skeleton joints belonging to one or two models. Interactively performed by the user, operations allow us to detach, remove, and merge components of our characters at will. Contrarily to *QuadMixer*, in which the user had to manually select the position of the meshes or the detaching curve, we exploit the skeleton structure to automatically guide the user in his operations.

We define three operations:

- **Remove**: it removes the skeletal components that descend from the selected joint. Most of the original layout of the model is preserved, and the surface is closed near the selected joint position.

- **Detach**: it preserves all the skeletal structure that descends from the selected joint, removing each ancestor. The surface is closed accordingly, preserving most of the original layout of the model.

- **Replace**: it performs a remove operation for a model and a detach operation for the other one. Most of the original layouts are preserved, while the surface in the intersection area is the result of a blending process. In the resulting model, the two joints involved in the operation will be merged into a single one. Note that the interface automatically moves the entire character involved in the remove operation, in a way that the rest poses of the two concerned joints match with each other. Then, the user can optionally scale, translate, and rotate the two models at will.

Note that these operations cannot be performed on root or leaf nodes of the skeleton tree. Figure 4.4 shows the effect of the operations on Crypto and Zlorp models of Figure 4.3.

While the hierarchy of the bones implicitly defines our operation outcome for the skeleton, defining the resulting surface and transferring the skinning weights are more challenging tasks. The idea behind our method is to exploit the skinning weight semantic information to find the desired output surface. In particular, we assign for each mesh vertex a fuzzy value (between 0 and 1) that represents its survival in the resulting mesh. A vertex with an associated value of 1 will be entirely kept in the result, whereas a vertex with a value of 0 will be lost. Intermediate values represent their weight in the merging process. From now onward, we refer to these values as *vertex select values*.

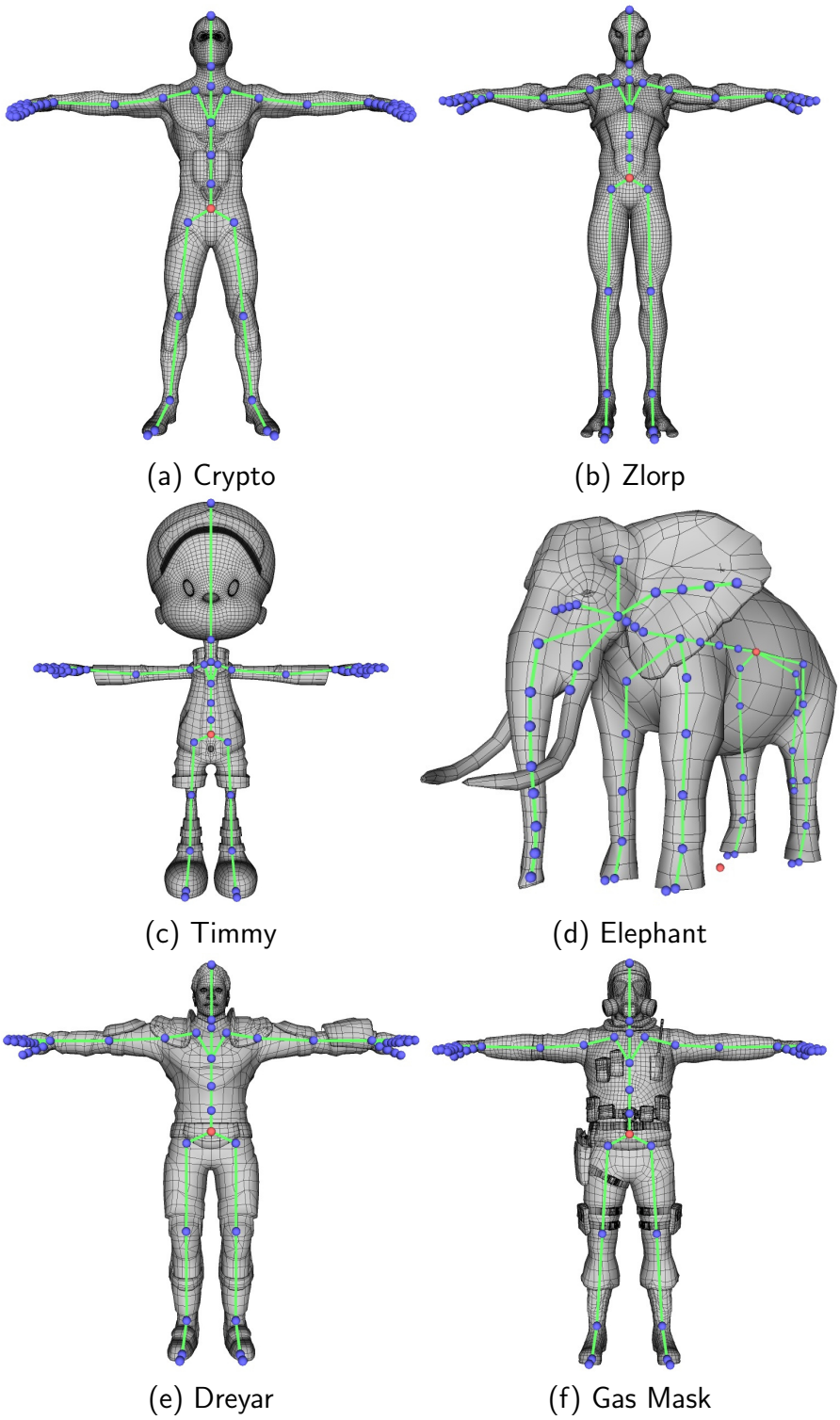
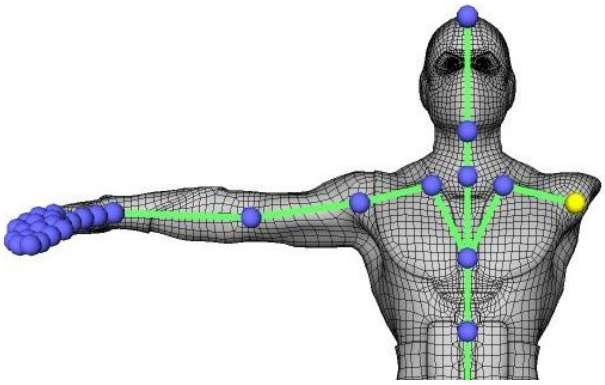
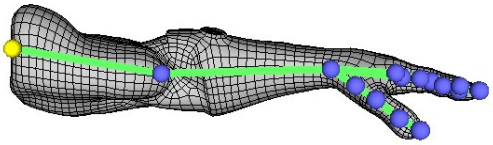


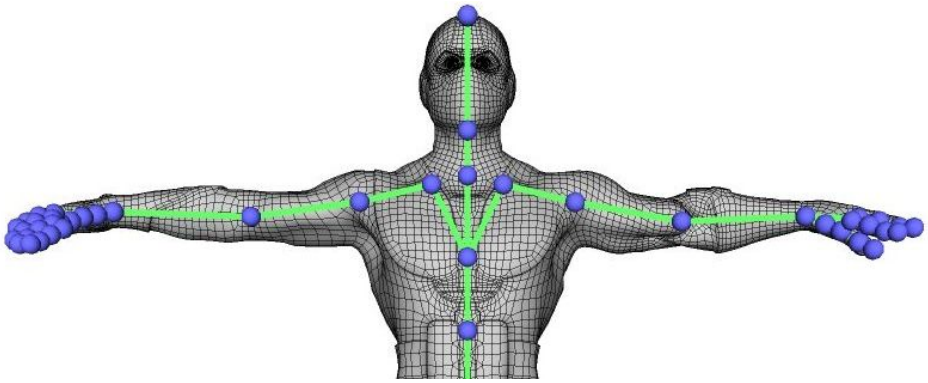
Figure 4.3: **Skinned models.** Some skinned models.



Remove Crypto arm



Detach Zlorp arm



Replace Crypto arm with Zlorp arm

Figure 4.4: **Operations.** The first row shows the result of removing the right arm from the Crypto model. The second shows the detaching of the right arm of the Zlorp model. The last Figure shows a replace operation, in which the right arm of the Crypto model is replaced with the right arm of the Zlorp model.

We summarize as follows the proposed pipeline (Figure 4.5):

- The user interactively performs *operations* to our models. The *select values* for each vertex are computed according to the operations performed (Figure 4.5.a).
- The resulting skeleton is trivially created according to the operations performed (Figure 4.5.b).
- According to the *select values*, for each operation we find the surface portion to be preserved and to be *blended* for each model (Figure 4.5.c).
- We embed the surfaces to be merged in volumetric signed distance fields, that are properly *blended* together. We attach the *blended* surface to the preserved portions of the input meshes (Figure 4.5.d).
- The surface is then quadrangulated using a similar approach to the one proposed in Chapter 3 (Figure 4.5.e).
- The skinning weights are determined for the new surface and the new skeleton, preserving most of the original information and using the *select values* to blend where necessary (Figure 4.5.f).

4.2 Methods

The input to our method consists of skinned models, each composed of a mesh, a skeleton, and its skinning weights. Conventionally, from now onward, we can refer to the character model C as the tuple $C = (M, S, W)$, where M is the mesh, S the skeleton, and W the skinning weight function. We define a mesh M as the couple $M = (V, F)$ where V and F are respectively the sets of the vertices and the faces. Each skeleton S is composed of a set of joints J and their connectivity. W represents a function that is defined for each pair of vertices and skeleton joints.

In the following Subsections, we show:

- How to create the resulting skeleton;
- How to generate the *vertex select values*;
- How to obtain the resulting surface using a distance field *blending* process;
- How we quadrangulate the surface and we stitch it to the preserved one;
- How to determine the resulting skinning weights of the model;

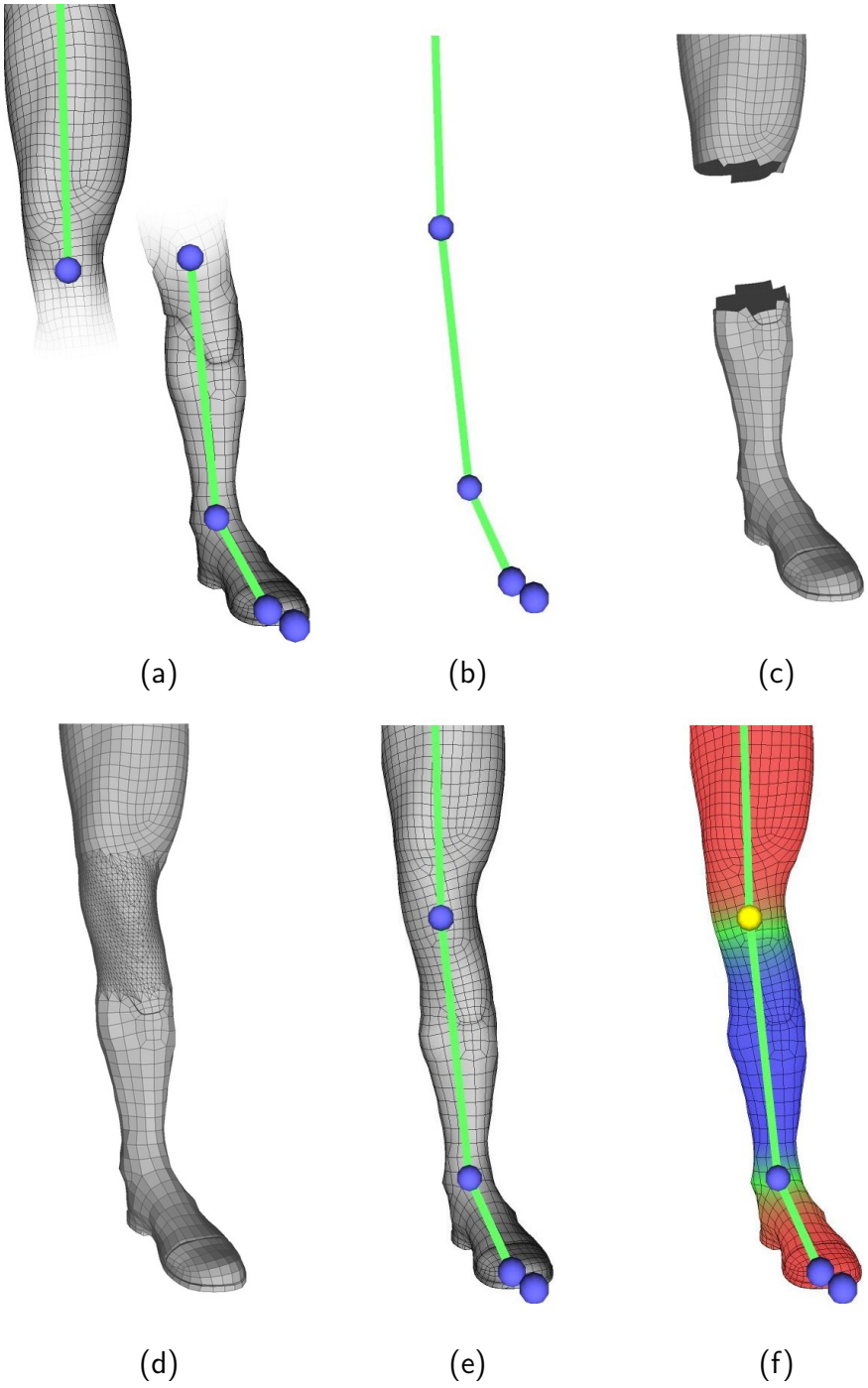


Figure 4.5: **An overview of the proposed pipeline.** The user performs operations on the models (a), the resulting skeleton is determined (b), the preserved portions of the surfaces are identified (c), a new surface is extracted by manipulating signed distance fields (d), the surface is quadrangulated (e), and the skinning weights are determined for each skeleton joint (f).

4.2.1 Determining the resulting skeleton

The resulting skeleton for the operations described in the last section is straightforward:

- In case of a remove operation, we keep all the joints that do not descend from the selected joint.
- In case of a detach operation, we keep all the joints that are descendants of the selected joint.
- In case of a replace operation, we perform a remove operation to the first model and a detach operation to the second. The two selected joints are then merged into a single one. The rest pose is determined by averaging the rest poses of the concerned joints.

We define a function $k : J \to \{0, 1\}$ for each model C for which an action has been performed. This function assigns to each joint $j \in J$ a boolean value: 1 means that the joint has survived after performing the operation, 0 otherwise. Having these values, the new skeleton can be trivially created, connecting all the components accordingly.

4.2.2 Vertex select values

For each model C, the skinning weights $W : V \times J \to [0, 1]$ is a function that maps for each pair of vertex and skeleton joint a value between 0 and 1. Note that, for each vertex $v \in V$, a common requirement of any animation pipeline is that $\sum_{j \in J} W(v, j) = 1$ [JDKL14]. For example, if this condition does not hold, skinning algorithms such as Dual Quaternion Skinning [KCvO07] would not work properly. Generally, we assume that this condition is always satisfied.

Skinning weights represent the vertex deformation with respect to each joint transformation during the animation process. We exploit these valuable insights to generate weights that are used to determine the resulting surface. In particular, we want to assign to each vertex a value from 0 to 1. This value represents the influence of that part of surface in the resulting model. Figure 4.6 shows, with a transparency effect, the preview result of each operation.

For each vertex $v \in V$, we define a function $w(v)$, called *weight function*, as:

$$w(v) = \sum_{j \in J} W(v, d) * k(j) \tag{4.1}$$

where $k(j)$ is the boolean function defined for each skeleton joint in the previous Subsection. In other words, the function w for a vertex v is simply the summation of its skinning weights related to the joints which have survived in the operation.

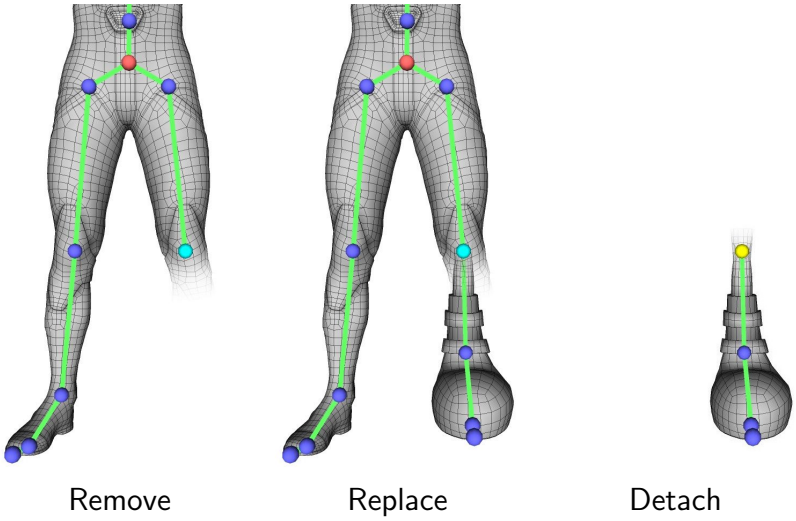


Figure 4.6: **Select values for each operation.** The transparency shows the select value assigned to each vertex. Where the value is near 0, the surface is transparent. On the other hand, the surface is completely opaque where the value is near 1.

We apply a Laplacian smoothing to this function, keeping untouched the function in the vertices with values close to 0 or 1. In particular, given a threshold t (in our examples $t = 0.99$) we smooth the function for the vertices $v \in V$ for which $1 - t < w(v) < t$ holds. Figure 4.7 paints on the surface the resulting function for the remove and detach operations on a joint.

The select value $s(v)$ is then computed for each vertex v as:

$$s(v) = w(v)^{\frac{1-o}{o}} \quad \text{with} \quad o = \frac{h+1}{2} \tag{4.2}$$

where $-1 < h < 1$ is the *hardness* parameter. Note that the value o only normalizes the parameter h from $[-1, 1]$ to $[0, 1]$, and that for $h = 0$ we have $s(v) = w(v)$.

The *hardness* parameter enables the user to choose how to smoothly distribute the intermediate values of w ($0.0 < w(v) < 1.0$) on the surface. In case of a value greater than 0, the surface will have higher *select values* and, then, more influence in the *blending* process described below. On the contrary, for a value of less than 0, the surface will have a lower influence. Figure 4.8 shows the effect of the *hardness* parameter in a detach operation. This parameter is very useful in the detach and remove operations since it determines the position where the surface will be closed. Later, we will show the effect of this parameter for the replace operation. The graphs in Figure 4.9 show the effect of the *hardness* parameter on the select value function.

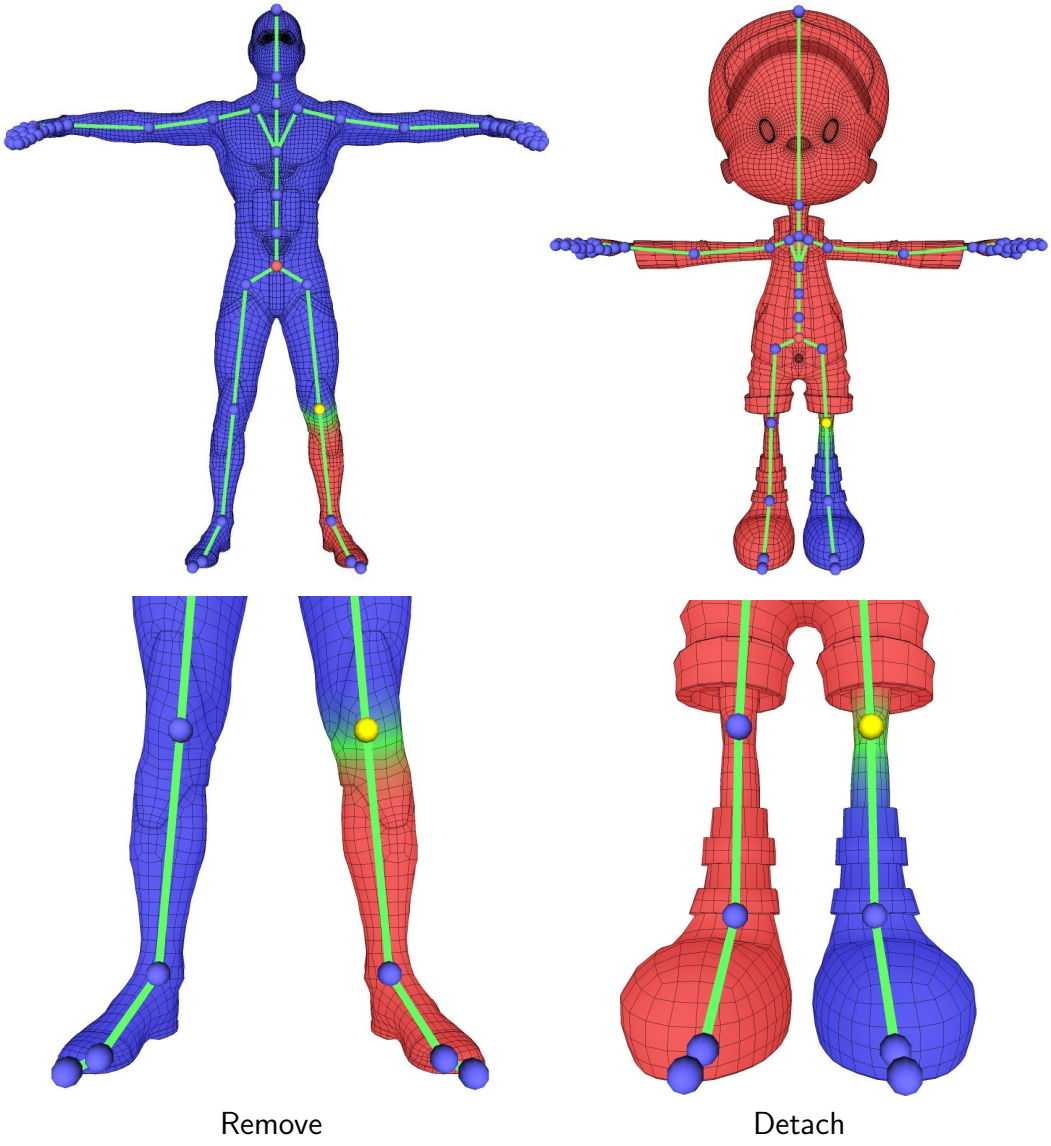


Figure 4.7: **Weight function for detach and remove operations.** The colors represent the value of the function on the surface: blue color means the values are near 1, green where the values are near 0.5, and red color represent surface where the vertex select values are near 0.

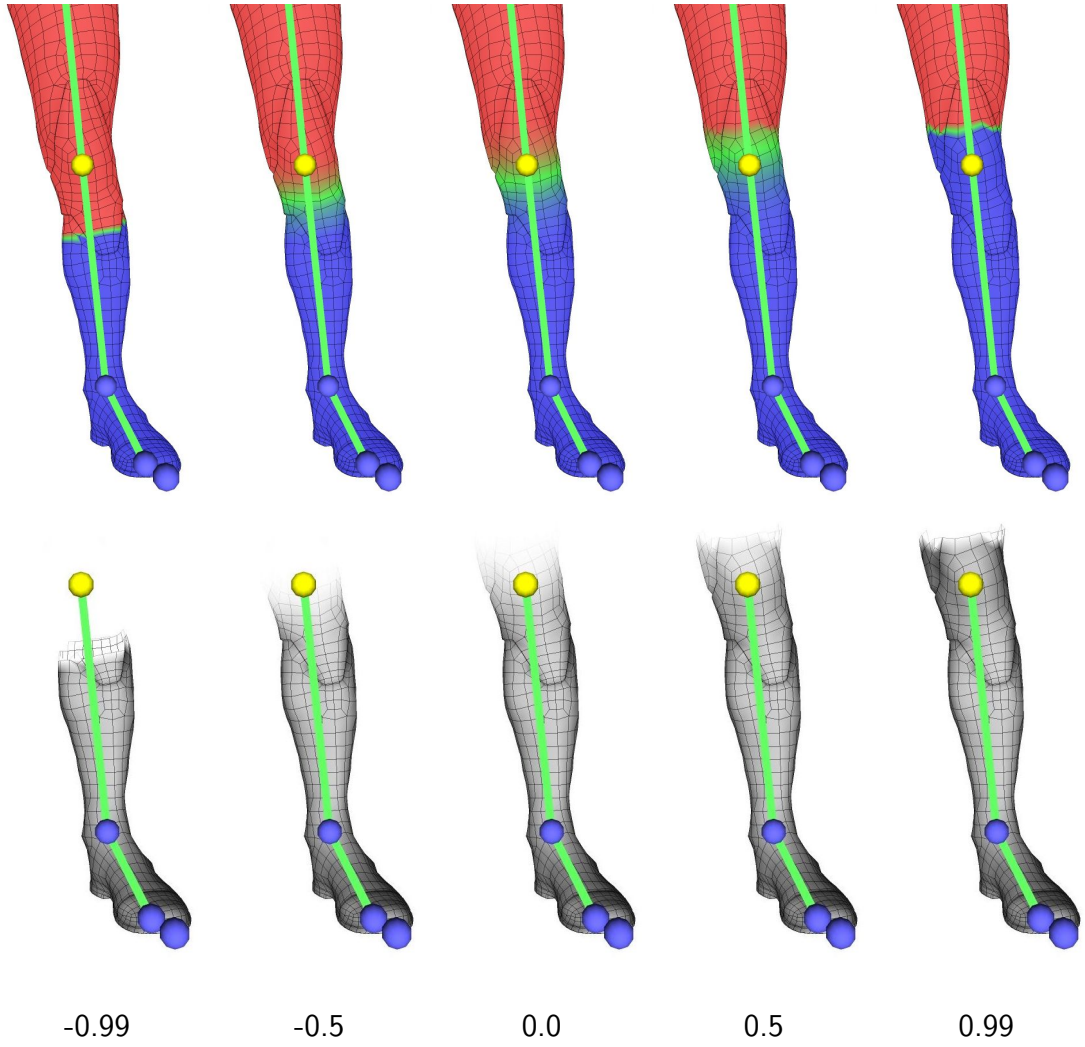


Figure 4.8: **Hardness effect on select values.** The figure shows how the hardness parameter (varying from -0.99 to 0.99) influences the select values in a detach operation. The first row shows the select values, using the same color notation previously used (blue = 1.0, green = 0.5, red = 0.0). The second row shows the select values using transparency.

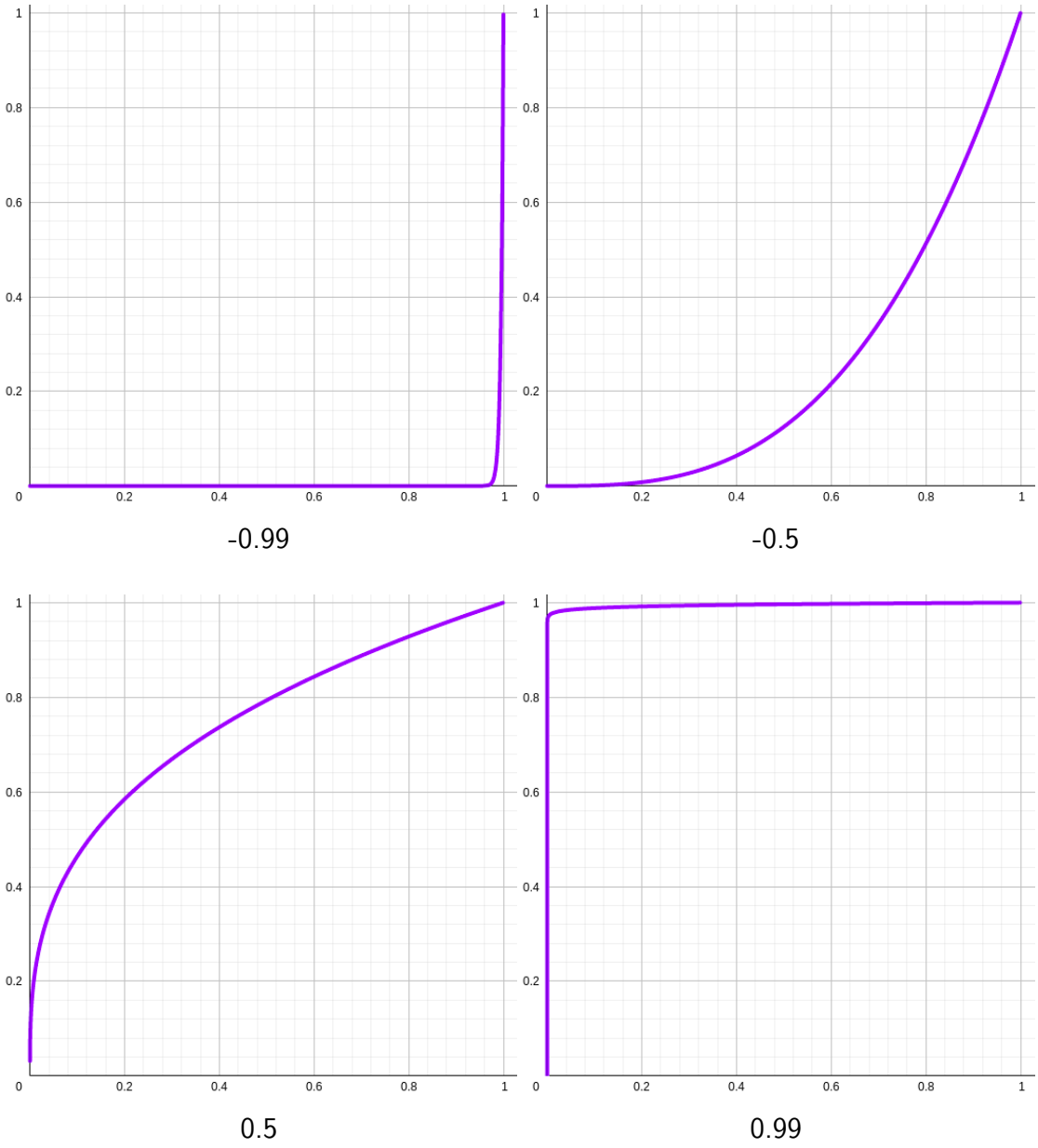


Figure 4.9: **Hardness effect in select values.** The graphs show the effect of the hardness parameters on the final select values computed as a function of the weight function. In the x-axis, there is the weight function w, in the y-axis the resulting select value s.

Rigid components As we already stated, meshes of real-world assets can be non-watertight, quad-dominant and non-manifold. Furthermore, they could be composed of several disjoint connected components. Figure 4.3.e and 4.3.f show two examples. The armor features of the Dreyar model and all devices and pockets in the Gas Mask models are portions of surfaces that are not topologically connected. From a semantic point of view, these components are less important than, for instance, the bust or the head: they are just a decorative feature. Usually, during the skeletal animations, all the vertices of these components mostly deform in the same way. For this reason, we name them *rigid* components.

Disjoint components could be a problem for our field extraction. For example, they could intersect with each other, leading to major problems while signing the field. The intersections of decorative components with more significant portions of the surface is rather common in the assets available in the market. To solve this problem, it is reasonable to have a special treatment for the *rigid* components: in our operations, they will either be entirely preserved or be discarded.

Surface topology information is not enough to identify these portions of the surface. Figure 4.10 shows a model that has several semantic meaningful portions of the surface that are not connected. Note that the semantic important components (such as bust, the head, and the legs) will deform similarly to the corresponding ones in watertight models such as Zlorp in Figure 4.3.b.

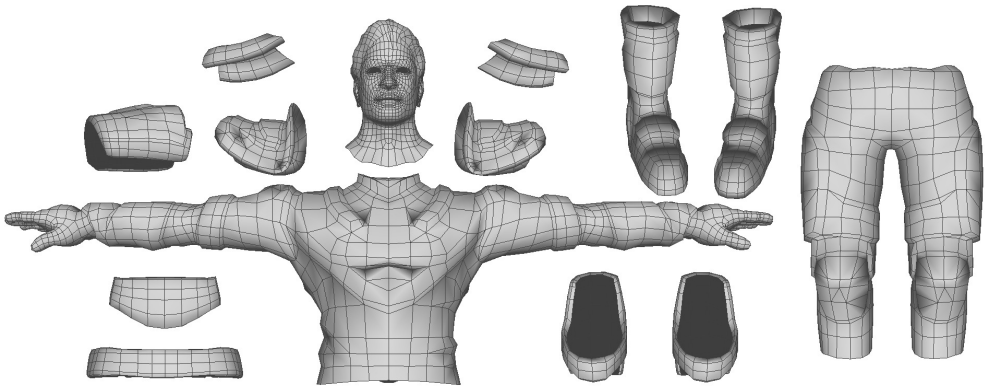


Figure 4.10: **Dreyar connected components.** The Dreyar model is composed of a high number of disjoint connected components.

For each connected component, we can determine as follows whether it is rigid or not. We find the minimum $w_{\min}$ and maximum $w_{\max}$ of the function $w(v)$ for its vertices. Given a rigidity factor r (in our examples $r = 0.8$), a component is rigid if and only if:

$$1 - r \leq w_{\min} \quad \text{and} \quad w_{\max} \geq r \tag{4.3}$$

If a component is rigid, we entirely preserve the component if the average select value s for each vertex of the component is greater than 0.5, otherwise, we discard it. Note that preserving a component means to set all its vertex select values to 1.0, whereas discarding means to set them to 0.0. Figure 4.11 shows some rigid components for the Dreyar model.

The *hardness* parameter can also influence the outcome for the rigid components, as shown in Figure 4.12. This allows us to automatically decide the outcome for each *rigid* component of the model. Nevertheless, to allow the user to interactively choose which rigid components to preserve or discard would be a reasonable feature for our modeling tool interface.

4.2.3 Determining the resulting surface

The vertex select values describe, using a value between 0 and 1, the influence of each vertex on the resulting surface. First, we find for each model the faces to be preserved and the faces to be discarded. The surface to be preserved is simply the set of the faces which have an average vertex value (calculated on the incident vertices) equal to 1. On the other hand, the faces with an average select value equal to 0 are discarded.

We apply a regularization process to avoid holes or isolated faces in the borders of the preserved surface. In particular, we consecutively perform an opening and a closing operation on the selected faces, using the mathematical morphological operators described in [CKS00]. This will increase the quality of the final quadrangulation and the robustness of the boundary stitching step.

At this point, we determined a portion of the surface that will be entirely preserved in the final model and a portion which will be entirely lost. The remaining faces, neither discarded nor preserved, will be used for determining a new surface that will be quadrangulated. Note that the discarded faces will be completely ignored in the further steps of the pipeline.

Signed field extraction For each mesh M of the operation, our objective is to extract a signed distance field, that will be used for determining a new surface to be quadrangulated. To avoid interferences in the fields caused by portions of meshes not involved in the operations, we consider only the portion of the mesh which are neither preserved nor discarded. This approach also natively solves the matter of the *rigid* components, since they are entirely either preserved or discarded. For robustness and implementation reasons, in the field calculation, we take into account also the faces of the preserved meshes that lie near the boundaries. This is performed by a growing approach: all connected faces at a given radius from a boundary vertex are involved in the field extraction. In our tests, to be as robust

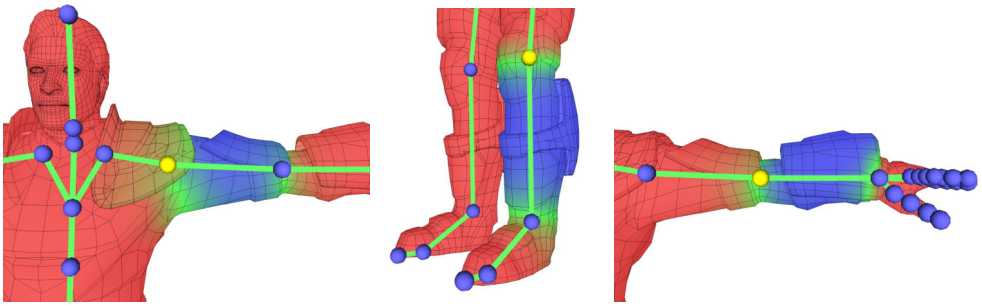


Figure 4.11: **Rigid components.** The colors (blue = 1.0, green = 0.5, red = 0.0) represent the skinning weight values on the surface related to the skeleton joint in yellow. The shown connected components have a similar value for all their vertices, therefore they are rigid: deformations have the same effect for each vertex.

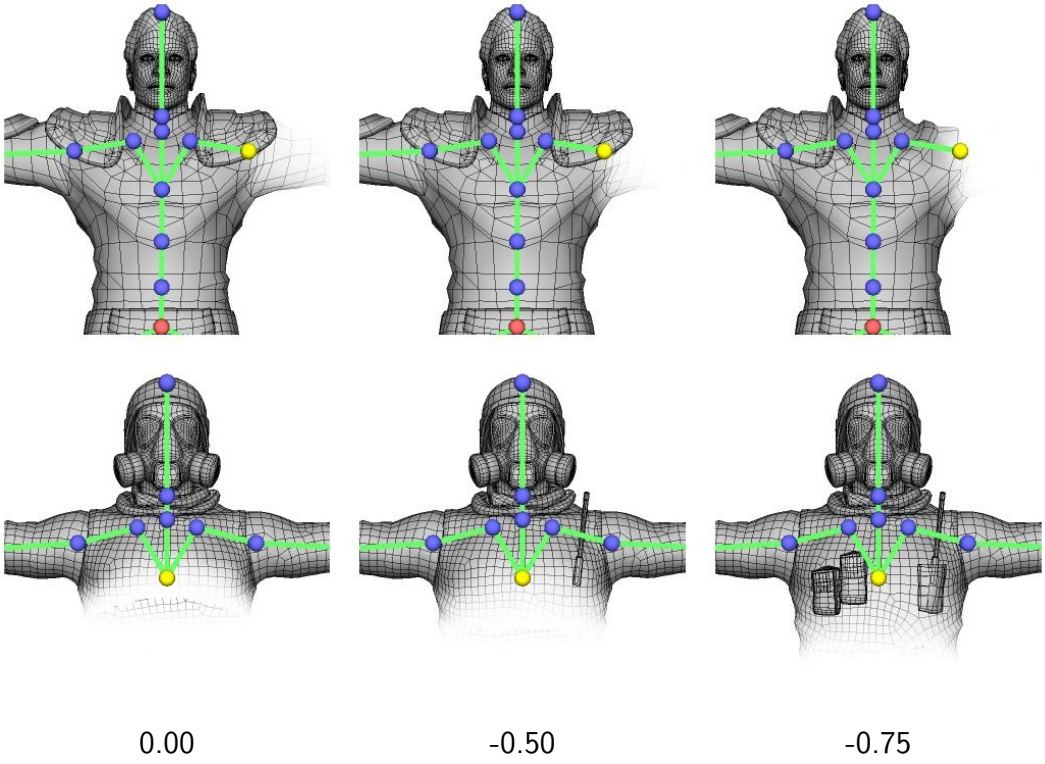


Figure 4.12: **Hardness effect in rigid components.** The hardness parameter influences the average select value in the rigid components. Hence, we can exploit this functionality to interactively preserve or discard components. In a Remove operation, a lower hardness value results in discarding surface (top). On the contrary, Detach operations with negative hardness values result in preserving more surface (bottom).

as possible in case of a tiny surface, we used a radius corresponding to the size of 15 unit voxels, even if a lower value could be used.

For each portion of mesh, we compute its unsigned distance fields using a library (OpenVDB [MLJ+13]) that allows performing robust and efficient operations on fields using a hierarchical and sparse data structure. Then, we sign the field using the fast generalized winding number described in [BDS+18]. Note that this step enables the process to deal with non-watertight meshes.

After the signing operation, the field has not continuous values in the areas where the surface was not closed, as shown in Figure 4.13.c. This is not ideal for the *blending* approach described below. Hence, we extract a closed isosurface using an efficient dual marching cube algorithm [Nie04] and we re-calculate the signed distance fields $\mathcal{F}_1$ and $\mathcal{F}_2$. This task could be performed more efficiently by a zero-preserving smoothing, but we chose to use the robust signed field extraction of the library to fully exploit the hierarchical and sparse organization of data.

Figure 4.13 illustrates the entire field extraction process for a non-watertight surface. Figure 4.14 shows some meshes obtained by the signed field extraction process in an actual replace operation.

Remove and detach operations: closing by signing the field In the case of the detach and remove operations, we have a single field F that already defines an isosurface. However, we desire to smoothly close the surface where the vertex select values are near 0.5. Indeed, the select values close to 0.5, highlighted in green in the first row of Figure 4.8, define an isoline that appropriately segments the surface in two regions.

For this reason, we compute the signed distance field as explained above selecting all the faces that are incident in vertex that have a select value in the interval $(0.5, 1)$ (the extremities of the interval are not included). The resulting surface will be stitched to the preserved mesh and then quadrangulated.

Replace operation: blending signed fields For the replace operation, the task is more challenging. Indeed, we have to find a way to manipulate the given fields to obtain an isosurface that represents the *blending* of the input meshes.

We generate the distance field for the mesh including only the faces that have an average select value (calculated on the incident vertices) in the interval $(0, 1)$. Given the signed distance fields $\mathcal{F}_1$ and $\mathcal{F}_2$, we want to *blend* them in a single field $\mathcal{F}$, following the select values s_1 and s_2.

In each field, for each voxel, we have useful information, that is the origin polygon. It identifies the face in the input meshes that lies at the minimum distance. Using these insights, we can embed the select values in two fields $\mathcal{S}_1$ and $\mathcal{S}_2$. The select values for each voxel are obtained by interpolating on the closest

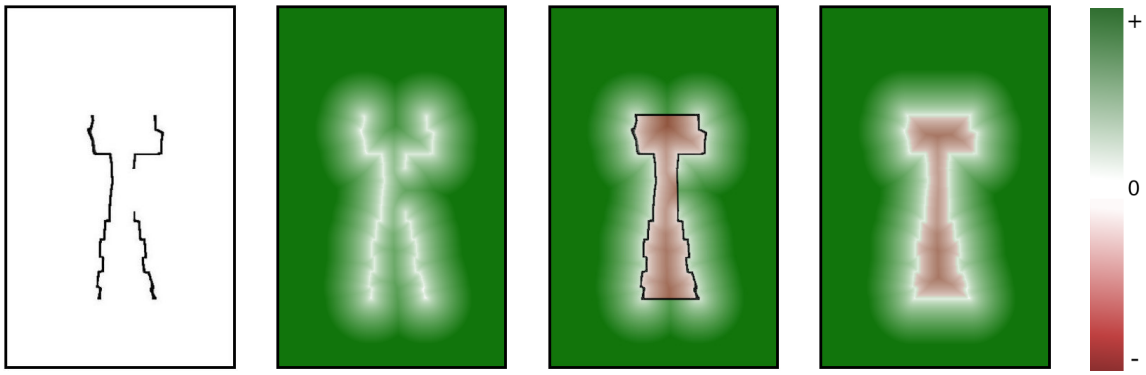


Figure 4.13: **Field extraction.** 2D representation of field extraction. Colors are interpolated in this manner: green color represents positive distance values, red negative values, and white where the field is near zero. From left to right: the original shape (a), the unsigned distance field (b), the signed field (with the isosurface in black) that presents discontinuities (c), and the final signed distance field (d).

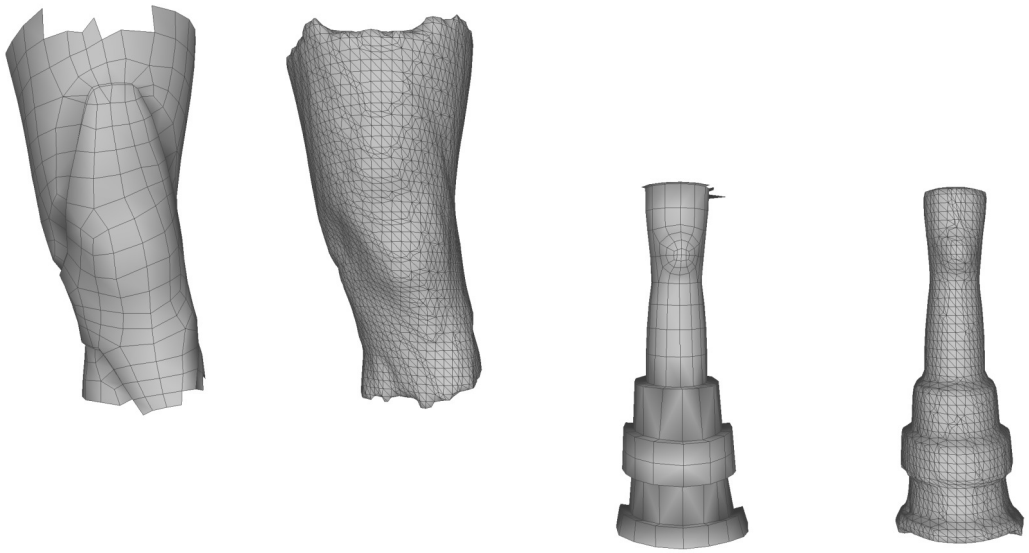


Figure 4.14: **Extracted field isosurface.** Given the faces neither preserved nor discarded, we extract a signed distance field. In this figure, we show the resulting isosurface. Note in (b) how the process properly handles the spurious border faces.

point of the origin polygon. Note that our input models are quad-dominant, and they can contain some triangular or pentagonal faces. In case of a triangle, we interpolate by the well-known triangle barycentric coordinates. The interpolation of the select values on a polygon with more than 3 sides is simply performed by triangulating the polygon using a new vertex in the barycenter (assigning to it the average select value of the polygon), and then interpolating in the triangle on which the point lies. For the sake of simplicity, we used this interpolation approach also for quadrilaterals faces, even if bilinear interpolation could give more robust and accurate results. Figure 4.15.c illustrates the color interpolation on a quadrilateral using our approach.

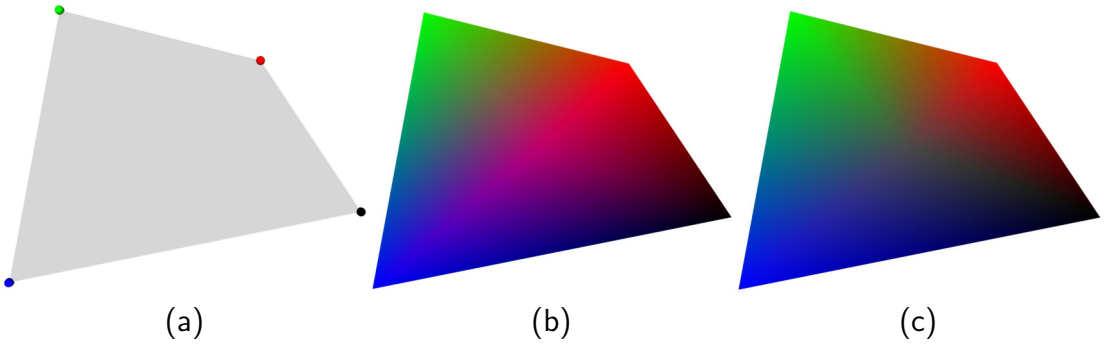


Figure 4.15: **Barycentric interpolation on a quadrilateral.** A quadrilateral with colors defined per vertex (a), interpolation by splitting the quadrilateral with one of the diagonals (b), and interpolation by splitting in the barycenter (c). The image on the right (c) illustrates the proposed interpolation method.

Given the distance fields $\mathcal{F}_1$, $\mathcal{F}_2$ and the select value fields $\mathcal{S}_1$, $\mathcal{S}_2$, for each voxel $\mathbf{X} = (x, y, z)$ the key equation that performs the field *blending* $\mathcal{F}$ is defined as:

$$\mathcal{F}(\mathbf{X}) = \frac{\mathcal{S}_1(\mathbf{X}) * \mathcal{F}_1(\mathbf{X}) + \mathcal{S}_2(\mathbf{X}) * \mathcal{F}_2(\mathbf{X})}{\mathcal{S}_1(\mathbf{X}) + \mathcal{S}_2(\mathbf{X})} \tag{4.4}$$

However, we want to guarantee that the surface is as closest as possible to the original ones where the corresponding select values are close to 1. On the other hand, we want to smoothly *blend* where both the surfaces have an intermediate value. Given a threshold t (in our examples $t = 0.99$) we compute the field F as:

- if $\mathcal{S}_1(\mathbf{X}) \geq t$ and $\mathcal{S}_2(\mathbf{X}) < t$

$$\mathcal{F}(\mathbf{X}) = \mathcal{F}_1(\mathbf{X}) \tag{4.5}$$

- if $\mathcal{S}_1(\mathbf{X}) < t$ and $\mathcal{S}_2(\mathbf{X}) \geq t$

$$\mathcal{F}(\mathbf{X}) = \mathcal{F}_2(\mathbf{X}) \tag{4.6}$$

- if $\mathcal{S}_1(\mathbf{X}) \leq 1 - t$ and $\mathcal{S}_2(\mathbf{X}) \leq 1 - t$

$$\mathcal{F}(\mathbf{X}) = 0.5 * \mathcal{F}_1(\mathbf{X}) + 0.5 * \mathcal{F}_2(\mathbf{X}) \tag{4.7}$$

- otherwise: Equation 4.4

Equation 4.4 describes the core of the *blending* process, where intermediate values are merged together to obtain the desired surface. Equation 4.5 and 4.6 guarantee that the surface is as closest as possible to the original meshes where the select values are close to one. Equation 4.7 guarantees the continuity of the field in case of low or zero select values. Figure 4.16 illustrates the entire field *blending* process with a 2D sketch.

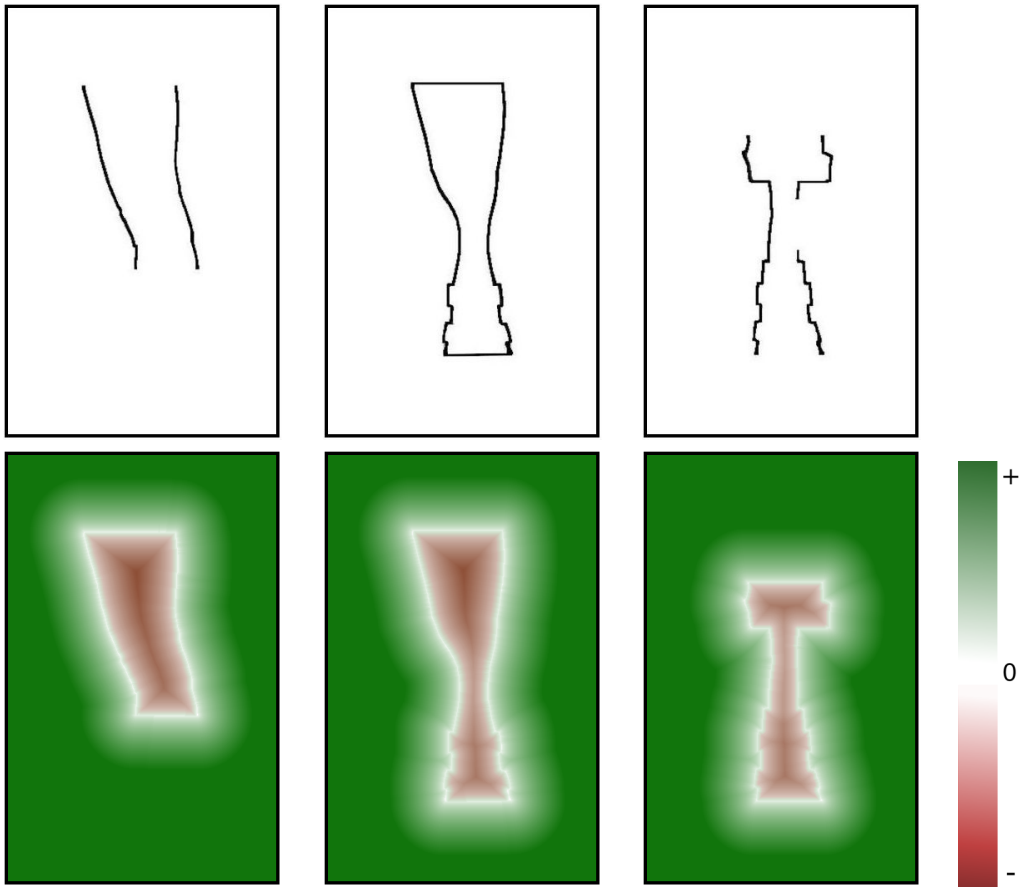


Figure 4.16: **2D representation of the field blending process.** On the left and right the two signed distance fields of the input models, in the center the resulting field. We show the shapes (top) and the distance fields using the same color notation of Fig. 4.13 (bottom).

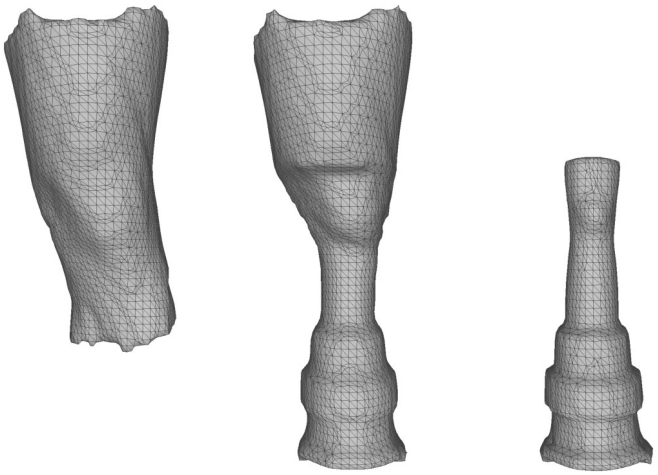


Figure 4.17: **Field blending.** Blending of two fields. On the left and right the isosurfaces of the two input fields, in the center the isosurface of the blended field.

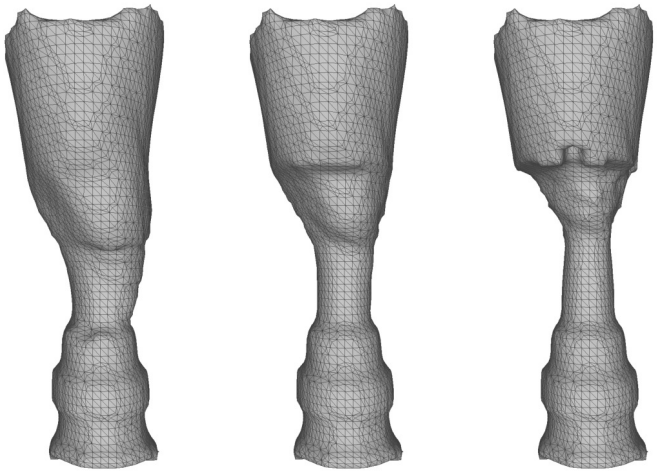


Figure 4.18: **Hardness effect in field blending.** We show the effect of the hardness parameter in the field blending on the same setup of Figure 4.17. R and D stand for the hardness for, respectively, the removed and the detached component. On the left, we preserve more of the detached component. On the right, we preserve more of the removed component. In the center, we show the zero hardness result.

At this point, we can extract the isosurface from the field $\mathcal{F}$. Figure 4.17 shows an actual field *blending* example. The outcome blended surface will be stitched to the two preserved meshes and will be quadrangulated, as described in Subsections 4.2.4 and 4.2.5.

Figure 4.18 shows the hardness parameter effect in the replace operation. Note that hardness parameters, especially for high and low values (close to -1 or 1), can reduce the quality of the resulting shape. Indeed, the select values would be less widely and uniformly distributed on the surface.

Refining the isosurface The resulting isosurface is a watertight surface. However, where the select values are near 1, we want open boundaries that will be stitched to the preserved meshes. Moreover, we want these boundaries to be as closest as possible to the borders of the preserved meshes. Hence, we delete each vertex of the isosurface mesh for which the origin polygon is a preserved face. In this way, by construction, we create boundaries that are close to the preserved mesh borders.

4.2.4 Boundary stitching

At this point, we have the quad layouts that have been preserved from the original models, and a triangle mesh resulting from the operations on the distance fields (blending for the replace operation, closing with field signing for the detach and remove operations). A requirement for applying the retopology approach proposed in Chapter 3 (Section 3.2) is that the triangle mesh and the preserved mesh must share the same boundaries. The two meshes are not connected, however, as we already stated, their boundaries are quite close by construction.

Hence, we edit the triangle mesh along its boundaries, in order to have the same boundary vertices of the preserved quad layouts. The key idea for solving the problem is to *split* the boundary edges of the triangle mesh in correspondence of each border vertex of the preserved mesh, and then perform multiple edge collapse operations in order to keep only the vertices that lie in the preserved meshes. By *splitting* an edge, we mean to add a new vertex between the two vertices that form the edge itself, appropriately adjusting the tessellation. Figure 4.19 illustrates the entire boundary stitching process.

Regularization First, we perform a pre-processing step on the triangle mesh to obtain regular and smooth borders. This is a very useful property for a stitching task. In particular, we use the opening morphological operator described in [CKS00] on the boundaries of the triangle mesh. Fig. 4.19.a shows the triangle mesh with the regularized boundaries.

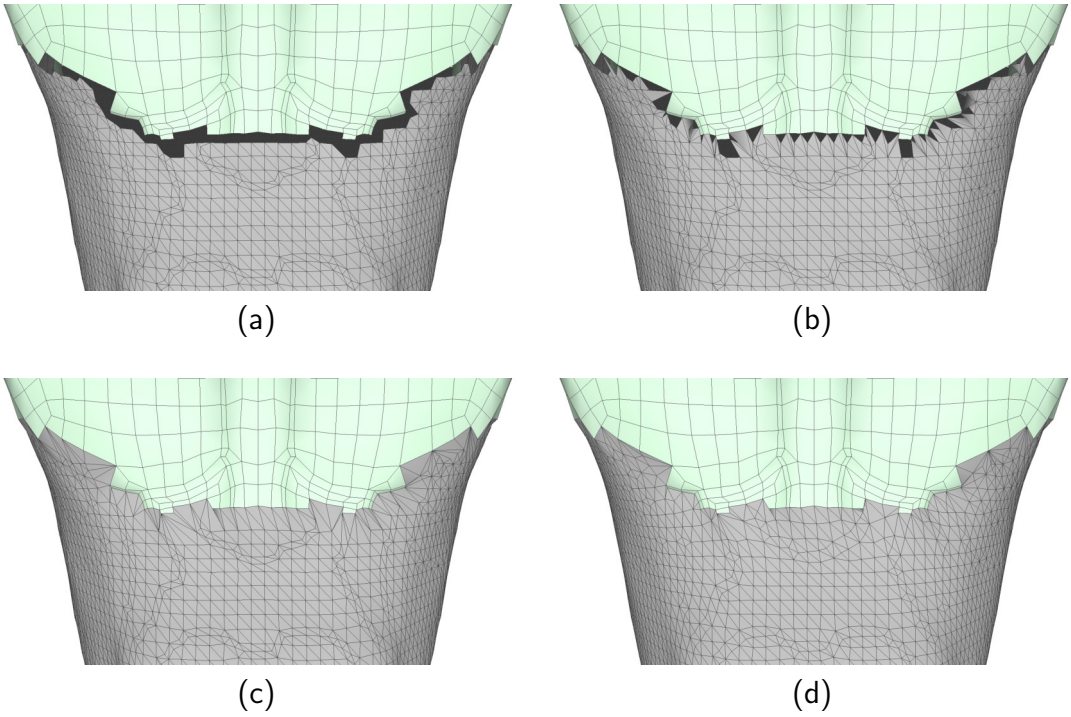


Figure 4.19: **Boundary stitching.** The preserved quad layout and the triangle mesh are stitched together by their border edge loops. The regularized input boundary loops (a), the edges are split following the parameterization (b), the edge collapse operations are performed (c), and the surface is smoothed (d).

Pairing Let E_p be the set of the border edge loops (or chains) of the preserved mesh and E_q the ones of the triangle mesh extracted from the blended implicit surface. Each of their elements is an ordered list of vertices $(v_1, ..., v_k)$ which describe the edge chains. Then, we find all the best pairs (e_p, e_q) with $e_p \in E_p$ and $e_q \in E_q$ that will be stitched together. This is performed by iteratively selecting the pairs with the least average distance between their closest points until all the elements of E_p or E_q have been assigned. Obviously, in the pairing process, we do not consider the border chains which were already boundaries of the original models. Furthermore, we discard all the connected components whose border loops have not been paired. This allows us to ignore the spurious components that may arise from the selection using the select values of the faces to be preserved or discarded.

Finding the parameterization Then, for each pair (e_p, e_q), we solve an optimization problem to find a parameterization t for the edge loop e_q of the triangle mesh. This parameterization will guide the boundary stitching process. More precisely, we want to add vertices in the boundaries of the triangle mesh e_q in

correspondence, according to the parameterization t, of each boundary vertex of e_p.

For a point v in the edge chain e_q, we define $\delta(v)$ as the projection of v in the preserved edge chain e_p. The length parametrizations start from the closest coordinates s_q, $s_p = \delta(s_q)$ in the two edge loops. Hence, we have $l_p(s_p) = 0$ and $l_q(s_q) = 0$. Then, we apply a 3D Laplacian smoothing to the edge chains and we parametrize them by their arc length in smoothed space. We refer to these length parameterizations in smoothed space as l_p and l_q.

Given the ordered list of consecutive vertices in e_q $(v_1, ..., v_k)$ for which $v_1 = s_q$, we find the parameterization t by minimizing the following energy:

$$\begin{aligned} \arg\min_{t} \quad & \alpha * D + (1 - \alpha) * R \\ \text{where} \quad & D = \sum_{i=1}^{k} (t(v_i) - l_p(\delta(v_i)))^2 \\ & R = \sum_{i=1}^{k} (t(v_i) - l_q(v_i)))^2 \\ \text{subject to} \quad & t(v_i) < t(v_{i+1}) \quad 1 \leq i < k \end{aligned} \tag{4.8}$$

The D term is the *deformation* objective term, and it encourages the parameterization t to match the parametric value of the closest point projected on the edge loop e_p. The *regularity* term R, instead, encourages t to match the length parameterization l_q of e_q. The parameter α blends between the two objective terms. In our implementation, we used $\alpha = 0.9$. For this optimization problem, we took inspiration from the approach proposed in [DYY16], even if in this case the problem is slightly different. Indeed, they deform both the edge loops to match with each other.

Edge split operations For each vertex in e_p, we perform a *split* operation on an edge of e_q, creating a new vertex with the same coordinates of the vertex in e_p. The edge to split is determined by matching the values in t with the ones of the length parameterization of e_p. Note that the constraint in the last row of Eq. 4.8 forces the parameterization to be a monotone function w.r.t. the vertex order in the edge chain. The average length of the edges in the two paired loops can be quite different since the discretization in the Marching Cubes algorithm is usually denser than the one in the original models. Our method is robust to this kind of mismatches since we create new vertices that refine the edges of the triangle mesh exactly where needed. Fig. 4.19.b shows the effect of the edge splitting operations.

Edge collapse operations Then, we iteratively perform several edge collapse operations in the triangle mesh, preserving only the new vertices resulting from

the splitting operations, i.e. the vertices lying in the preserved mesh. If it is not possible to perform the edge collapse, we add one or more triangle to close the surface with a simple hole filling algorithm. Nevertheless, due to the regular tessellation of the dual marching cubes, it never happened in our experiments. Fig. 4.19.c shows the triangle mesh after the edge collapse operations. After this step, the meshes have coinciding boundaries, as required to perform the quadrangulation approach described in Chapter 3.

Surface smoothing Finally, two Laplacian smoothing steps are performed on the surface, keeping fixed the vertices on the boundaries. The first one adaptively smooths near the boundaries, in order to avoid flipped faces after the stitching process and to encourage a smooth connection to the preserved surface. The second is a simple Laplacian smoothing applied to avoid the artifacts due to the volumetric sampling of Marching Cubes. For the first smoothing step, we want a high smoothing effect near the boundaries, and a low effect far from them. Hence, we linearly weight the effect of the smoothing w.r.t. the select value in the fields S_1 and S_2. Where the select values are lower or equal to a given threshold (0.9 in our tests), the smoothing has no effect. On the contrary, where select values are near 1, the smoothing effect is maximum. The effect is linearly blended for intermediate values between the threshold and 1. In our examples, we performed 5 smoothing iterations. The second smoothing step is a simple Laplacian smoothing with a fixed weight: in our implementation, we performed 5 iterations with weight 0.7 for each vertex. Fig. 4.19.d shows the result of the smoothing process.

4.2.5 Quadrangulation

We compute the quadrangulation as described in the Subsections 3.2.2, 3.2.3, 3.2.5. However, the feasibility of the quadrangulation cannot be guaranteed using the method proposed in Subsection 3.2.4. Indeed, the mesh could be quad-dominant, and the polychord splitting method works only for pure quad-meshes.

However, in this case, we do not have the constraint for the mesh to be composed of quadrilaterals only. Therefore, we can add triangles to guarantee the feasibility of the quadrangulation process. Particularly, if needed, we add a triangle in a chosen border side in order to increase or reduce the subdivision on the border by 1. The triangle is added in the longest edge if the subdivision value has been increased, or in correspondence of the shortest edge otherwise.

Which border side and whether to increase or reduce its subdivision is chosen by adding a term to the ILP optimization (Subsection 3.2.3). In particular, in the optimization problem, we allow each fixed border side to vary by 1 (-1 or +1), assigning a high cost in the objective function when this happens. Let b_i

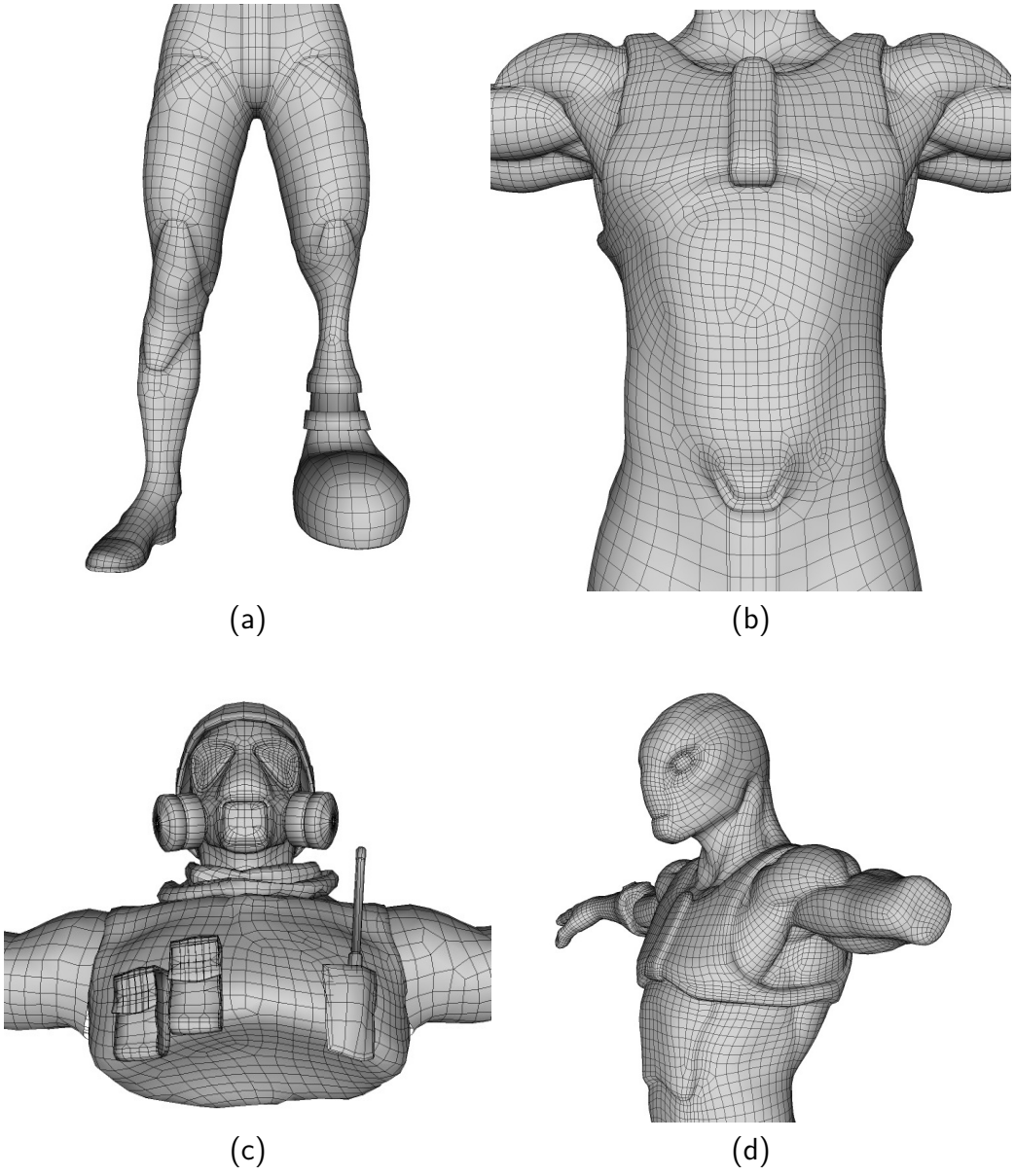


Figure 4.20: **Quadrangulation.** Some final quadrangulations. (a) and (b) show 2 replace operations respectively on the models Timmy and Crypto and the models Crypto and Zlorp. (c) and (d) are respectively a detach operation on the model Gas Mask and a remove operation on the model Zlorp.

be the fixed border vertex number, and $e_i = [b_i - 1, b_i + 1]$ be the target subside subdivision. We add to the objective function the term $|b_i - e_i| * M$ where M is a high number. In this way, we constraint the method to add a single triangle in an edge loop only when the parity hard constraint is not satisfied.

Figure 4.20 shows some final quadrangulated surfaces obtained with our method.

4.2.6 Skinning weight transfer

At this point in the pipeline, we have successfully generated the skeleton and the mesh for the resulting model. However, this model is not ready to be animated. Indeed, we have not transferred the information about the skinning weight yet. Skinning weights W_1 and W_2 are defined for each possible pair of vertex and joint of the related input model. We want to reuse this data for determining the skinning weights of the new model.

Most of the original vertices and some parts of the skeletons are preserved after performing the operations. For these preserved pairs, we simply transfer the corresponding skinning weight values. However, we have to find a way to transfer the data in the correspondence of the new surface generated by each operation.

Below, we show how we successfully perform this task. First, we find two injective mappings between the joints of the resulting skeleton and the joints of each input model. Using this information, the skinning weights are then computed using an approach similar to the *blending* process in Subsection 4.2.3.

Skeleton joint matching When we determine the resulting skeleton, we simply transfer the survived skeleton joints of each input model. Hence, we already have an implicit matching between some of two input skeleton joints and the output joints. In case of a remove or detach operation, all the joints have a corresponding birth joint in an input model. However, for the replace operation, we can only infer a partial matching between the input and the resulting skeleton.

The partial matching is enough to determine the skinning weights for the preserved surface, however, we would lose some information in the new surface that has been quadrangulated. For instance, the skinning weight information in Figure 4.21.c would be lost in correspondence of the lower part of the knee of the character in Figure 4.21.b. Indeed, differently from 4.21.a, we do not have any link between the joints in yellow of Figures 4.21.b and 4.21.c.

Hence, to properly transfer the skinning weight, we find two injective mappings $\gamma_1 : S \to S_1$ and $\gamma_2 : S \to S_1$, that link every joint of S to a joint of each input skeletons.

This mapping must be precise in the joints that are topologically close to the joint that has been merged (i.e. parents or children). On the other hand, we do

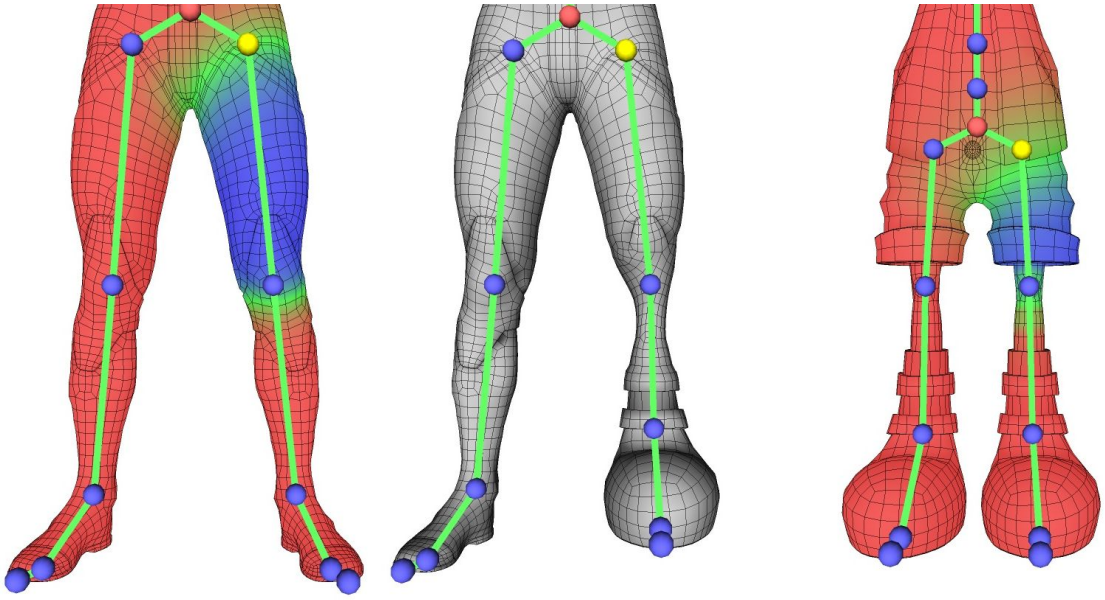


Figure 4.21: **Information lost due to the partial matching.** On the left and right (a) and (c) the input skinning weights for the joints highlighted in yellow. On the center (b) the resulting blended model. The usual color notation has been used. Some of the skinning weight information in (c) would be lost where the surface has been blended.

not need a perfect matching in topologically far joints. Indeed, it is unlikely that such a joint could contain skinning information about the new surface.

Therefore, we use a simple heuristic algorithm to find such mappings. Let γ be the desired matching function. First, we initialize γ with the partial matching inferred by construction during the skeleton creation. Therefore, for the preserved joint j_p, derived from j_o of the input model, we have $\gamma(j_p) = j_o$.

Let $j \in J$ be a joint of the resulting skeleton and q a joint in the input skeleton (S_1 or S_2). The following process is repeated until γ has been computed for each joint $j \in J$. At each iteration, we assign $\gamma(j) = q$ for the values of j and q that maximize:

$$\arg\max_{j,q} \quad \alpha_T * T(j,q) + \alpha_G * G(j,q) + \alpha_M * M(q) \tag{4.9}$$

where T represents the topological matching score, G the geometrical matching score and M is a term that gives priority to the non-matched joints.

The term T encourages the mapping to follow the topology of the skeletons. It

is defined as:

$$T(j,q) = 0.5 * f(p_j, p_q) + 0.5 * (\sum_{c_j}^{C_j} \sum_{c_q}^{C_q} \frac{f(c_j, c_q)}{|C_j| * |C_q|}) \tag{4.10}$$

where p_j and p_q are respectively the parents of the joints j and q, C_j and C_q are respectively the set of children of the joints j and q; the function f is defined as:

$$f(a,b) = \begin{cases} 1 & \text{if } \gamma(a) = b \\ 0 & \text{otherwise} \end{cases}$$

The term G encourages the mapping to follow the geometry inferred by the rest poses of the skeleton. In particular, we match the directions of the bones related to the joint. We define $d_p(a)$ as the unit vector from the rest pose of the parent of a to the rest pose of the joint a. We define $d_c(a)$ as the average unit vector from the rest pose joint a to the rest pose of the children of the joint a. Therefore, the term G is defined as:

$$G(j,q) = 0.5 * (d_p(j) \bullet d_p(q)) + 0.5 * (d_c(j) \bullet d_c(q)) \tag{4.11}$$

where the symbol $\bullet$ denotes the dot products. We obtain the maximum value of 1 if the directions match.

The term M gives priority to the joints q that have not been matched yet. It is defined as:

$$M(q) = \begin{cases} 0 & \text{if the joint q has been already matched} \\ 1 & \text{otherwise} \end{cases} \tag{4.12}$$

We want the heuristic to principally depend on the tree topology information, and to follow geometrical features when there is uncertainty. Hence, in our tests we used the values $\alpha_T = 0.45$, $\alpha_G = 0.05$, $\alpha_M = 0.50$.

This matching heuristic works perfectly if the two skeleton topologies coincide. However, if the topologies are even slightly different, we have inconsistent results in joints that are topologically far from the merging joint. Nonetheless, the method works fine for the topologically close joints and, therefore, successfully solve our matching problem for determining the skinning weights. In case we want to perform animation retargeting or other more complex operations (see future works in Section 4.4), we need to find a more clever matching method.

Blending the skinning weights As we already stated, we simply transfer the skinning weight values for the preserved vertices and joints of a model. However,

each operation produces a new surface that has been quadrangulated. Replace operations *blends* portions of the two input surface into a new one, whereas remove and detach operations create a new surface that closes the shape.

Given a vertex $v \in V$ lying in the quadrangulated surface and a joint $j \in J$ in the resulting model, let $\mathcal{X} \in \mathbb{R}^3$ be the coordinates of the vertex v. The related skinning weight $W(v, j)$ can be computed by interpolating the corresponding skinning weight for the corresponding joint in the input model. The process exploits the information about the origin polygon of the distance field, using the same interpolation method with barycentric coordinates described in Subsection 4.2.3 (Figure 4.15.c).

Then, the result skinning weights for the quadrangulated new surface are computed as follow:

- For a replace operation:

$$W(v, j) = \frac{\mathcal{S}_1(\mathcal{X}) * \widetilde{W}_1(\mathcal{X}, \gamma_1(j)) + \mathcal{S}_2(\mathcal{X}) * \widetilde{W}_2(\mathcal{X}, \gamma_2(j))}{\mathcal{S}_1(\mathcal{X}) + \mathcal{S}_2(\mathcal{X})} \tag{4.13}$$

  where we refer to the interpolated skinning weights on the two input models as $\widetilde{W}_1(\mathcal{X}, \gamma_1(j))$ and $\widetilde{W}_2(\mathcal{X}, \gamma_2(j))$. γ_1 and γ_2 are the corresponding joint mappings obtained as described above. Obviously, as in Subsection 4.2.3, $\mathcal{S}_1$ and $\mathcal{S}_2$ are the volumetric fields containing the interpolated select values.

- For the remove and detach operations:

$$W(v, j) = \widetilde{W}(\mathcal{X}, \gamma(j)) \tag{4.14}$$

  where $\widetilde{W}(\mathcal{X}, \gamma(j))$ are the interpolated skinning weights and γ the joint mapping of the single input model. However, the result of a remove operation could have leaf joints, originating from a non-leaf handle in the input. Conventionally, leaf joints are usually a decorative feature for the skeleton, and should not represent any deformation. Therefore, we simply accumulate to their parent the related skinning weight values.

Note that we handle the select value borderline cases as we described in Subsection 4.2.3. Finally, we perform a per-vertex normalization of the skinning weights, to guarantee that the constraint $\sum_{j \in J} W(v, j) = 1$ is satisfied for each vertex $v \in V$.

Figure 4.22 shows a pose of an animation for the resulting model of Figure 4.23. Figure 4.23 and 4.24 show some results of the skinning weights *blending* process for two replace operations. In the second row of Figure 4.24, it is clear how the flows of the skinning weights are captured and *blended* together using the select values. Finally, Figure 4.25 and 4.26 show the resulting skinning weights for, respectively, a remove and a detach operation.

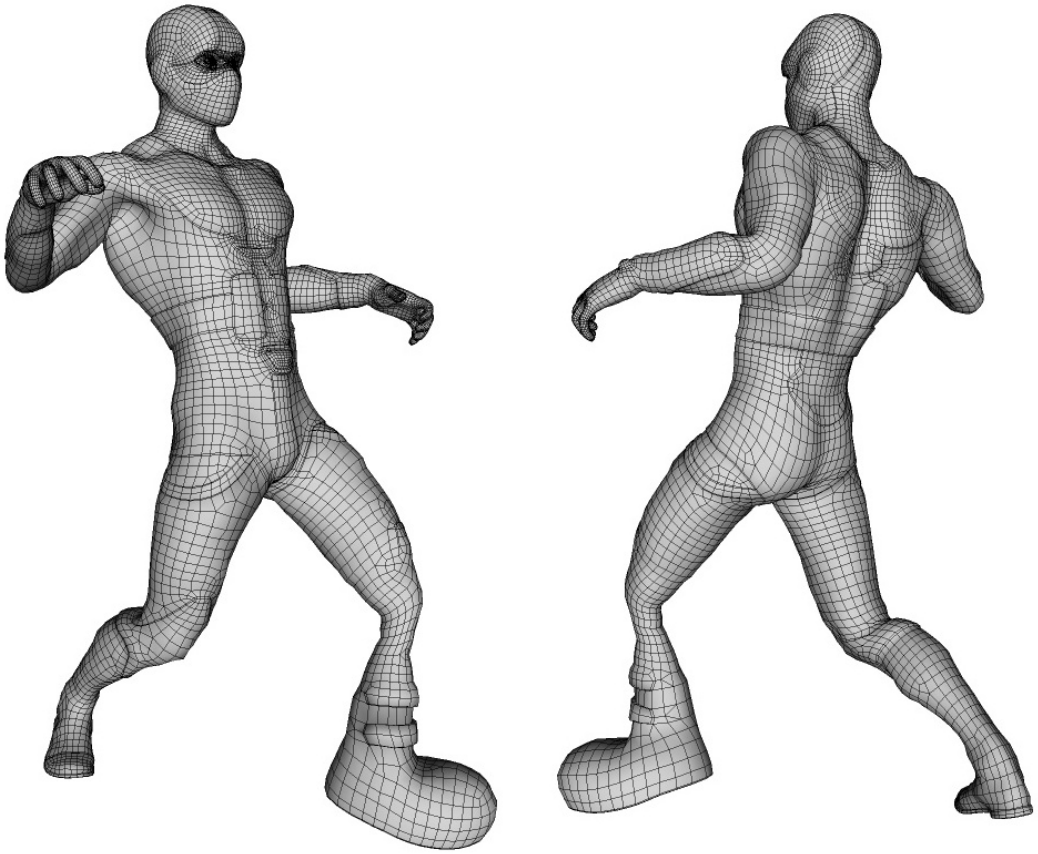


Figure 4.22: **A pose of the resulting model.** The animation is provided by the Mixamo [Bla14] framework. The pose for the resulting mesh is obtained simply by transferring the animation keyframes local transformation of the survived joints of each model. We generated the pose using the Dual Quaternion skinning [KCvO07] algorithm.

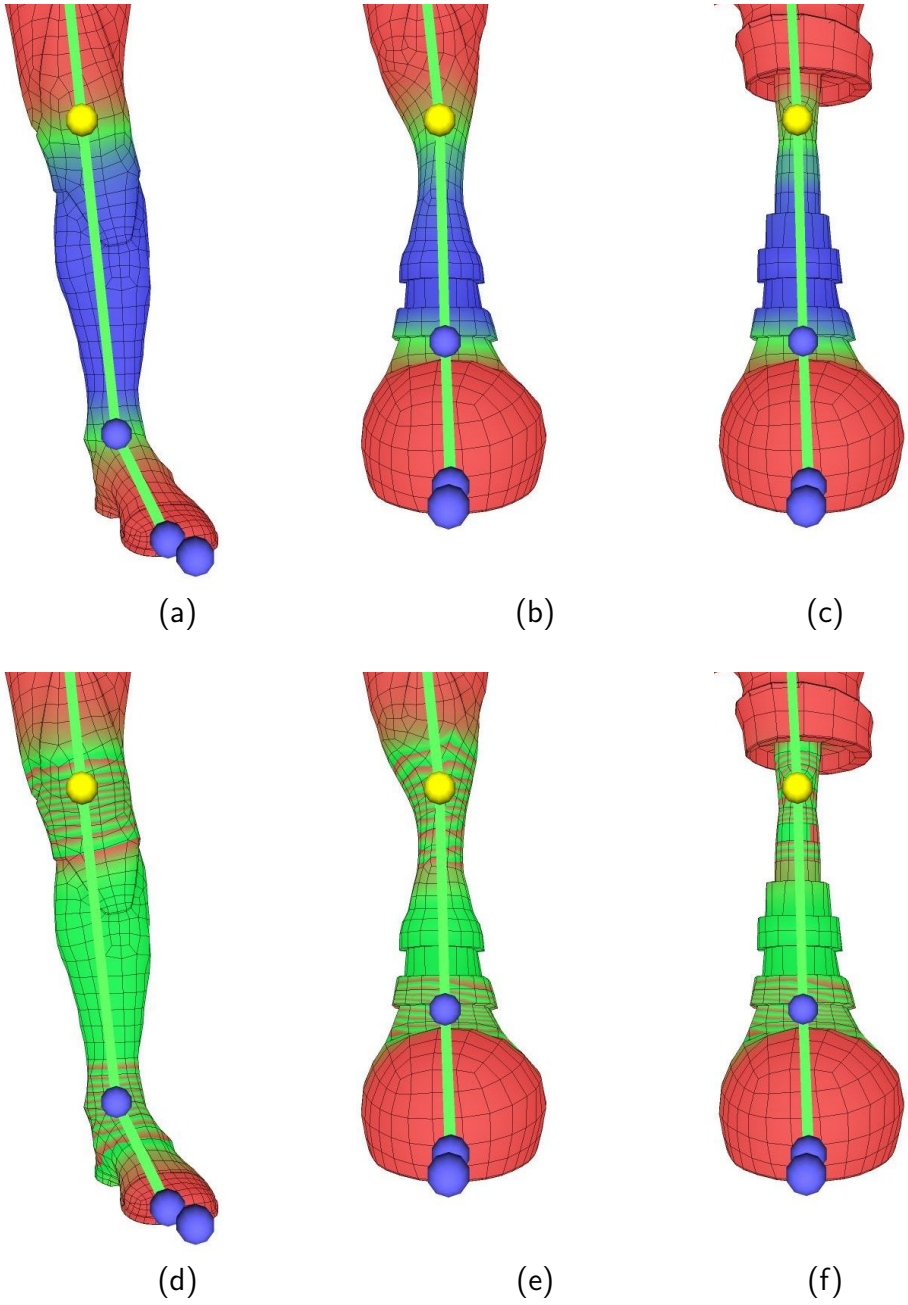


Figure 4.23: **Skinning weight blending.** Skinning weight of the resulting model of a replace operation for the models Crypto and Timmy. On the left and right are shown the input skinning weights, on the center the resulting skinning weights. The first row shows the skinning weights for the joint in yellow, using the usual color notation. The second row shows the so-called *contour* shader, that shows the isolines of the fractional part of the values, blending from red (0.00) to green (0.99).

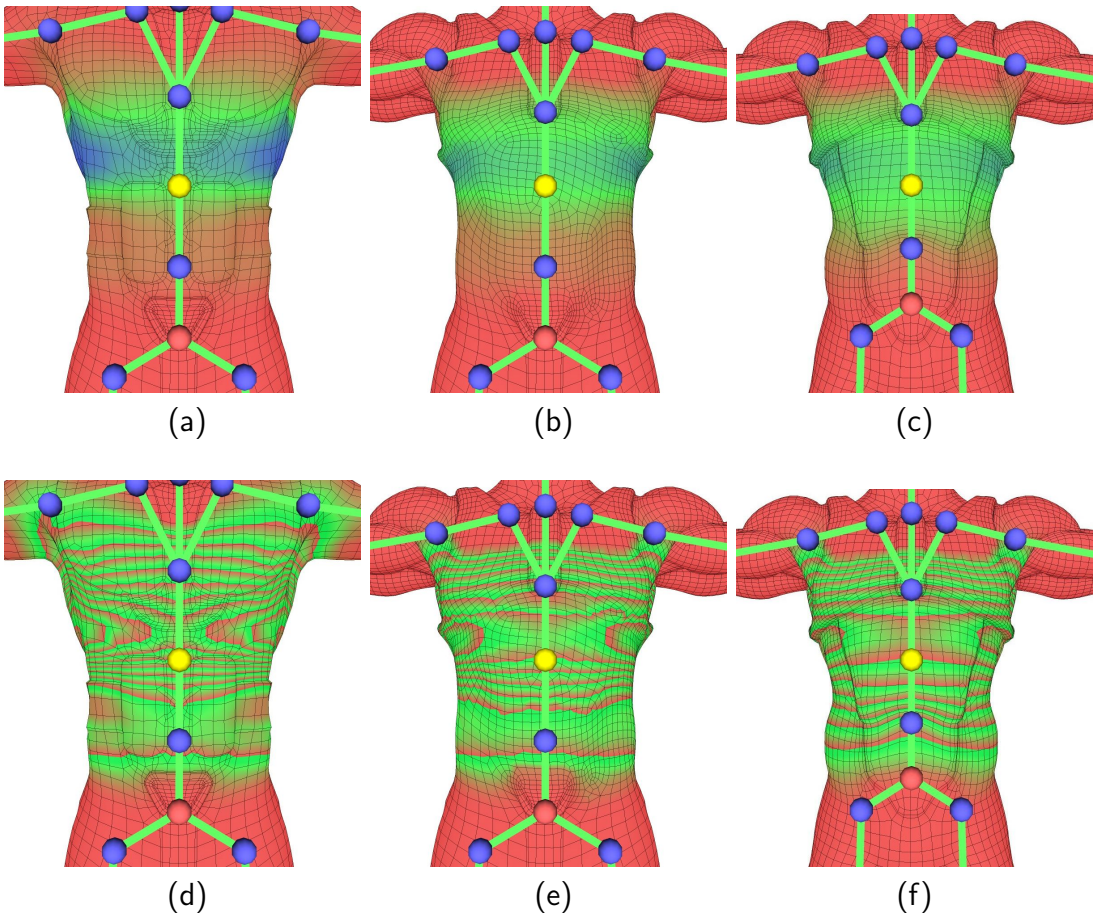


Figure 4.24: **Skinning weight blending.** Skinning weight of the resulting model of a replace operation for the models Crypto and Zlorp. On the left and right are shown the input skinning weights, on the center the resulting skinning weights. The first row shows the skinning weights for the joint in yellow, using the usual color notation. The second row shows the so-called *contour* shader, that shows the isolines of the fractional part of the values, blending from red (0.00) to green (0.99).

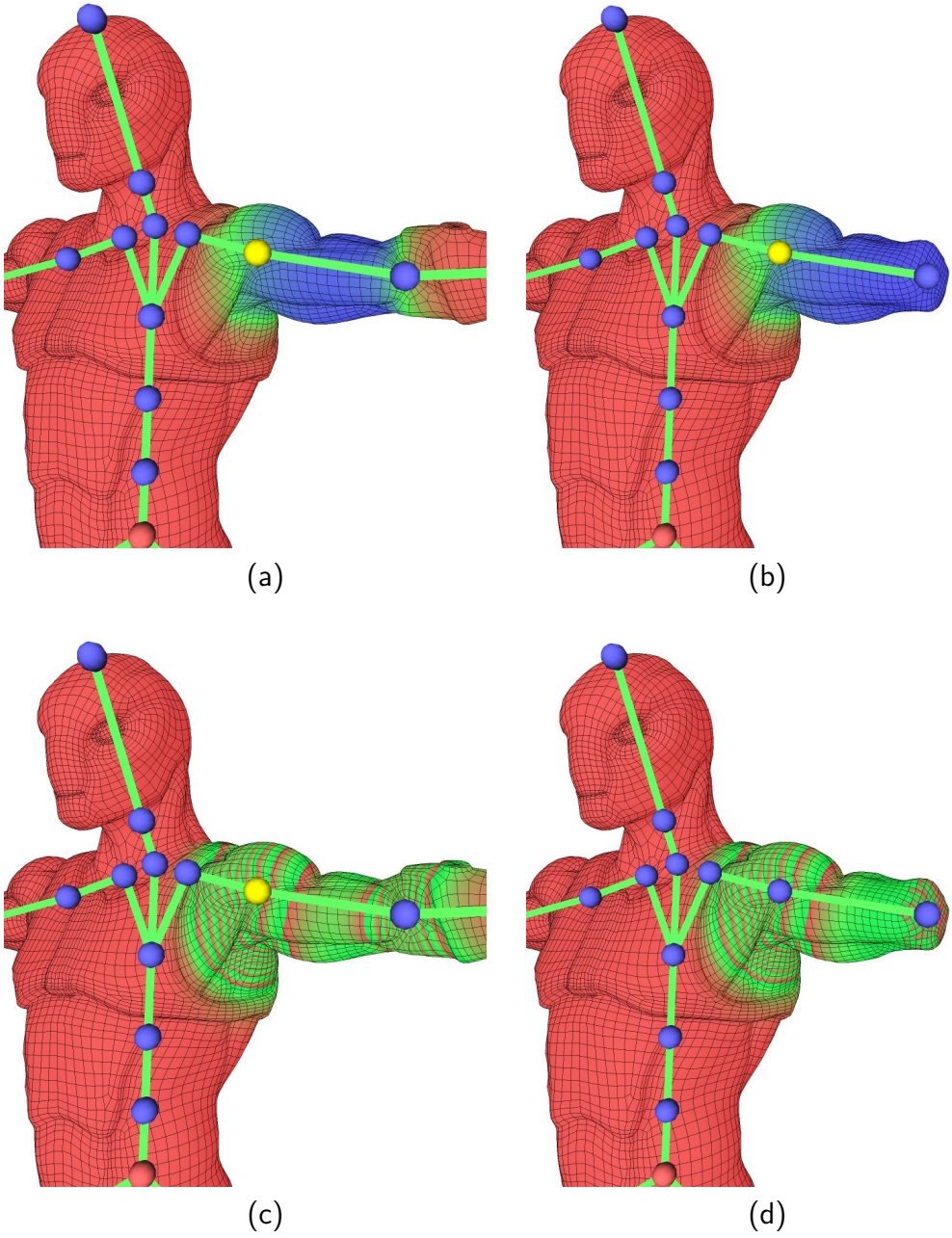


Figure 4.25: **Skinning weight transfer.** Skinning weight of the resulting models of a detach operation for the model Gas Mask. Note that the leaf joint has no influence on any vertex of the mesh. On the left the input skinning weights, on the right the resulting skinning weights. The first row shows the skinning weights for the joint in yellow, using the usual color notation. The second row shows the so-called *contour* shader, that shows the isolines of the fractional part of the values, blending from red (0.00) to green (0.99).

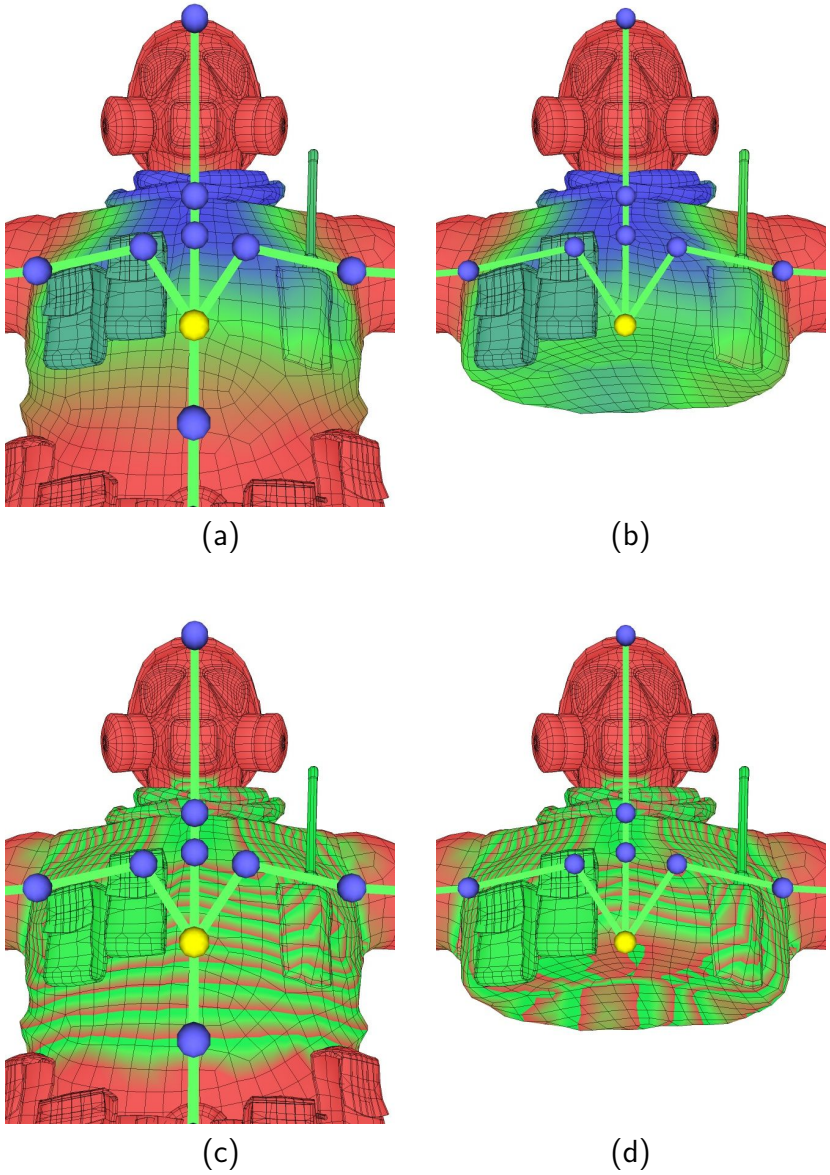
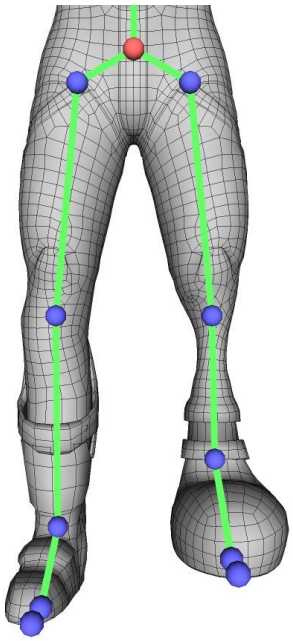


Figure 4.26: **Skinning weight transfer on a detach operation.** Skinning weight of the resulting models of a remove operation for the model Zlorp. Note that the two joints corresponding to the shoulders have some amount of influence where the model has been detached. On the left the input skinning weights, on the right the resulting skinning weights. The first row shows the skinning weights for the joint in yellow, using the usual color notation. The second row shows the so-called *contour* shader, that shows the isolines of the fractional part of the values, blending from red (0.00) to green (0.99).

4.2.7 Implementation details

We developed an interface which allows the user to load characters and perform operations on them. The user can apply several operations to multiple models, however, they are not immediately performed. On the contrary, a preview of the result using transparency is provided. When the user is satisfied, he can generate the output results at once.

Each operation is performed separately. In particular, we generate and manipulate a separate volumetric field for each remove, detach or replace operation. Even if this results in lower performances, this allows us to perform multiple operations on the same models without having any interference in the field operations. The Figure on the right shows an example of two replace operations on the Crypto model for which a single volumetric field could generate interferences due to the closeness of the involved surfaces. The left leg belongs to the Dreyar model, while the right leg belongs to the Timmy model. Such a result could not be possible by manipulating a single volumetric field.



Border stitching and skinning weight *blending* are also performed separately for each operation. On the other hand, the final preserved faces in the input mesh and the final new surface to be quadrangulated are found by gathering and joining together the information of each field. Therefore, the quadrangulation is performed in a single step.

Note that our method does not work properly if there is not any preserved region of the input models that divide and lie among each of the new surfaces generated by different operations. Indeed, this would result in components that are not connected together. Nevertheless, this can happen only for operations on consecutive joints (i.e. a joint and one of its children) and/or for *hardness* parameters near -1 or 1. For this reason, we did not handle this case for the time being. This could be handled by connecting the disconnected components or by avoiding the absence of preserved faces between the new surfaces.

The voxel size of the fields is determined as a function of the average length of the meshes. We find the minimum average edge length among the models involved in the operations. Then, the chosen voxel size corresponds to this value multiplied by a factor 0.7.

4.3 Results

We performed our tests on a laptop computer with an Inteli7-6700HQ processor with 16GB of RAM. We used OpenVDB [MLJ+13] for the high-performance field operations. Gurobi [GO18] to solve all the minimization problems. In the implementation, also the following libraries have been used: VCG Library [CNR13], CG3Lib [MN21], libigl [JP+16], Eigen [GJ+14] and CGAL [FP09].

Figure 4.27 shows some of the results obtained with our method. The generation of these results required up to about 15 seconds. The running time is influenced by the number of operations performed, the voxel size and the size of the intersected area. The most expensive operations in terms of time and memory footprint are the extraction and blending of the fields. This is because we cannot fully exploit the sparsity and hierarchical organization of data during the field signing and blending operations. Timings are summarized in Table 3.1. Note that all the code is single-threaded and not highly optimized. Moreover, a dense volumetric sampling has been used, even if wider voxels could have been used without any problem.

Table 4.1: **Time efficiency.** The execution time of some steps of the pipeline for the examples generated in Figure 4.27. Times are all in seconds. FE stands for Field Extraction, SB for surface blending, BQ for border attaching and quadrangulation, and WB for the skinning weight blending.

Models	FE	SB	BQ	WB	Total
Elephant - Timmy (stomach)	1.5	2.1	1.9	1.0	6.5
Timmy - Dreyar - Crypto (arms and head)	3.8	6.3	4.2	1.3	15.6
Timmy - Crypto (leg)	0.8	1.4	1.2	0.7	4.1
Crypto - Zlorp (stomach)	1.1	1.9	1.4	0.9	5.3
Gas Mask - Zlorp (legs)	1.8	2.7	2.0	1.2	7.7

Resulting skinning weight evaluation Figures 4.28, 4.29 and 4.30 show a comparison of the skinning weights obtained with our *blending* method and the skinning weights generated by the automatic method of Maya [Aut] ("closest distance"). In each figure, the same pose has been calculated for both results with the Dual Quaternion Skinning [KCvO07] method. A particular shading to highlight artifacts on the surface has been used.

Figures 4.28 and 4.29 show that our resulting weights are usually better than the ones obtained with the automatic tool. Both figures illustrate how the automatic generation creates weights that result in a volume shrinking during the deformation.

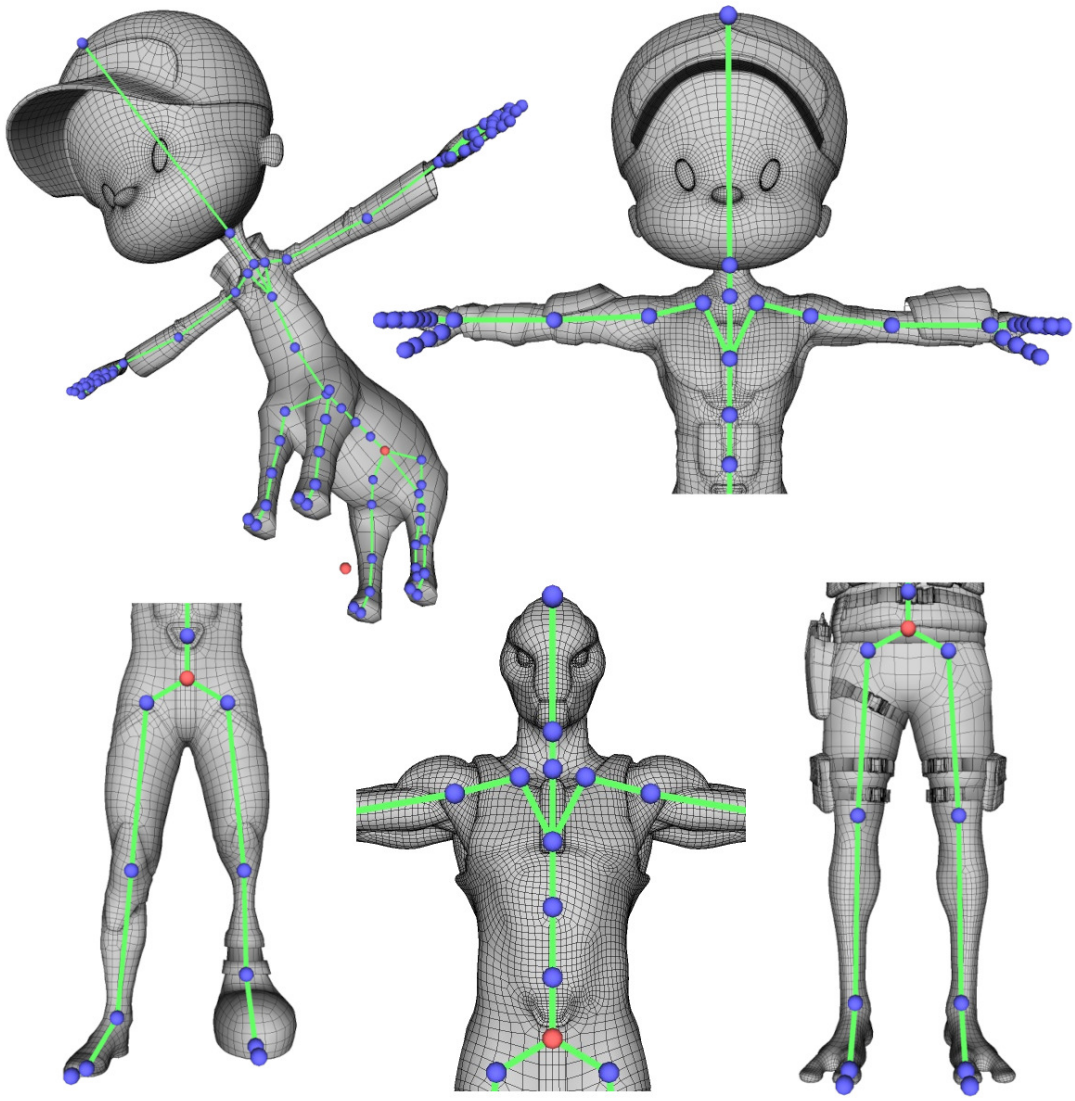


Figure 4.27: **Results.** Some results generated with our method. Every input model of Figure 4.3 has been used at least once. Some characters are the outcome of several operations applied on more than two models.

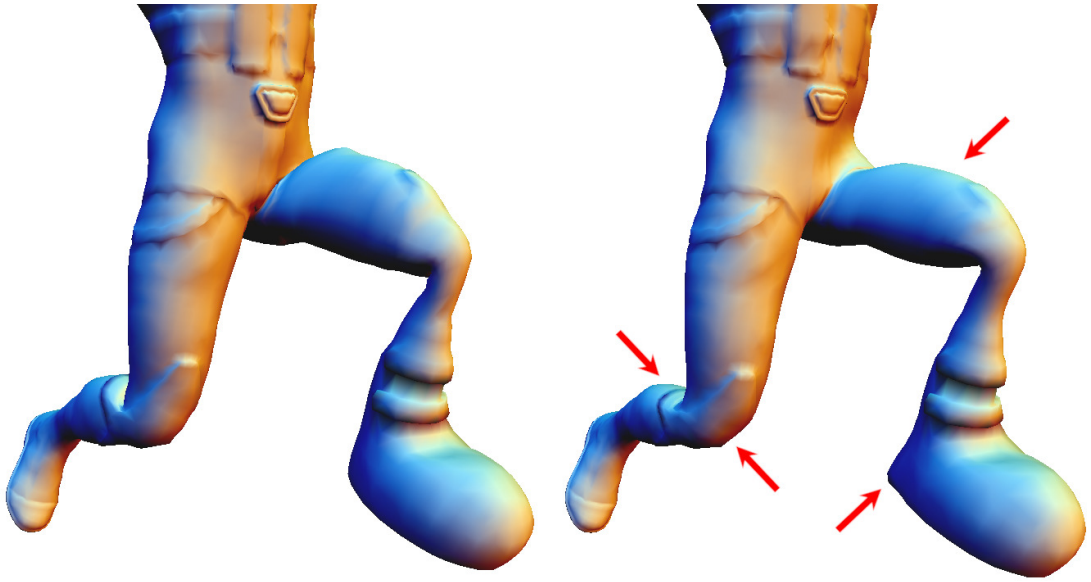


Figure 4.28: **Skinning weight comparison.** A comparison between the skinning weights generated by our method (left) and the automatically generated weights (right) for the model Timmy - Crypto (leg) of Figure 4.27. Red arrows indicate some artifacts and flaws on the deformed surfaces. The automatic tool produced volume shrinking, while our method preserved most of the original skinning weights.

Moreover, in Figure 4.29, the features in the bust, in the stomach and in the knees present visible artifacts when deformed. In the results of our *blending* approach, the surface and the skinning weights not lying in the intersection area are deformed exactly how the artists meant because we entirely preserved those components.

In Figure 4.29 the pictures in the bottom show a zoom of the intersection area. In this case, our method successfully blended and incorporated the weights of the original models, producing an undoubtedly better result than the automatic tool. On the other hand, Figure 4.30 shows a case where our method can create artifacts in the intersection area. However, also in this case, the automatic generation produced artifacts unwanted by the artist in correspondence of the components we preserved, while our method was flawless.

Our method to determine skinning weights works properly in case we replace components of characters with components of a similar shape. In the operation that merges the Timmy and the Elephant models, the shapes involved in the *blending* were too different. See Section 4.4.1 for a detailed discussion on this matter.

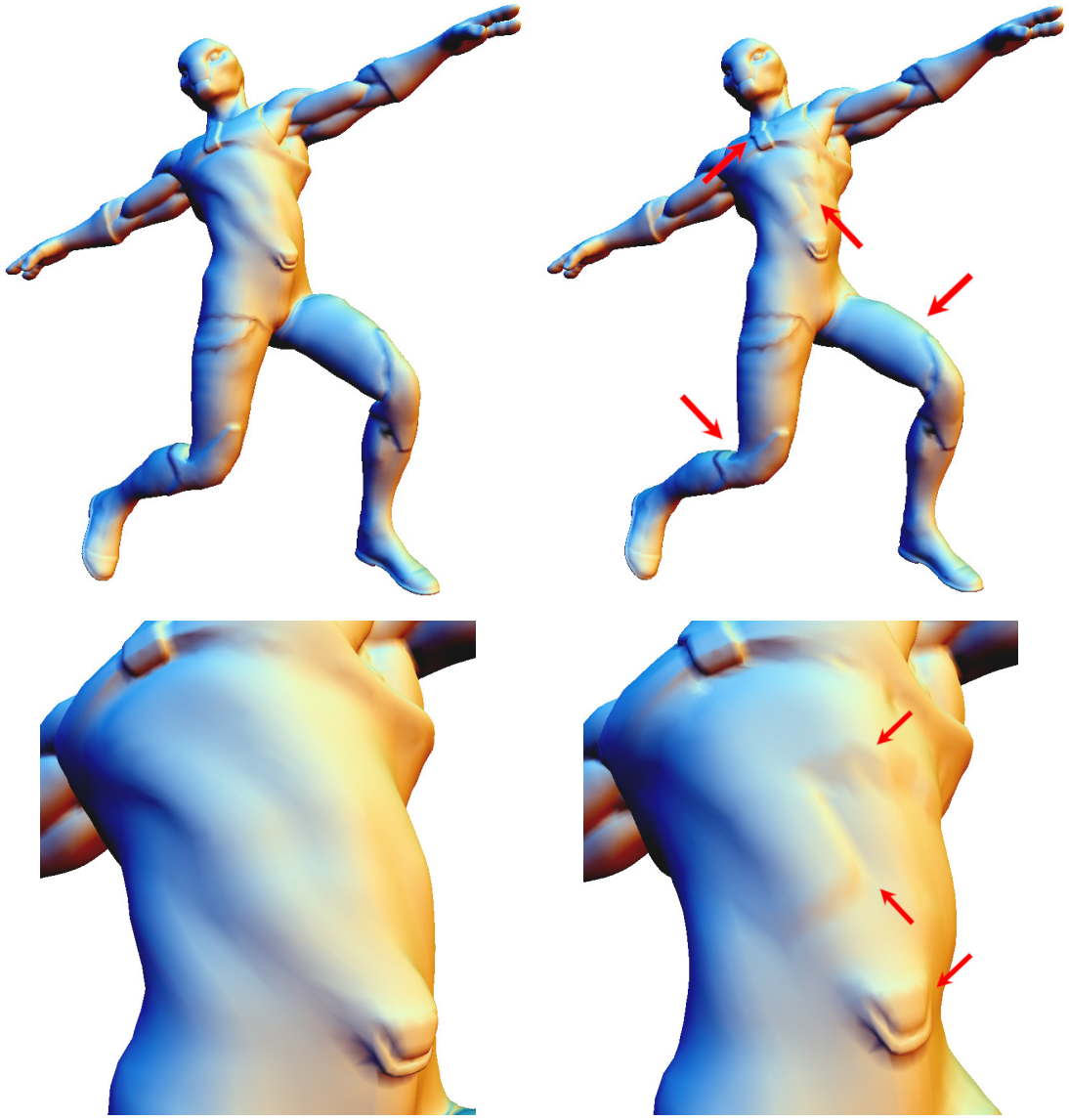


Figure 4.29: **Skinning weight comparison.** A comparison between the skinning weights generated by our method (left) and the automatically generated weights (right) for the model Crypto - Zlorp (stomach) of Figure 4.27. Red arrows indicate some artifacts and flaws on the deformed surfaces. The automatic tool result presents several artifacts.

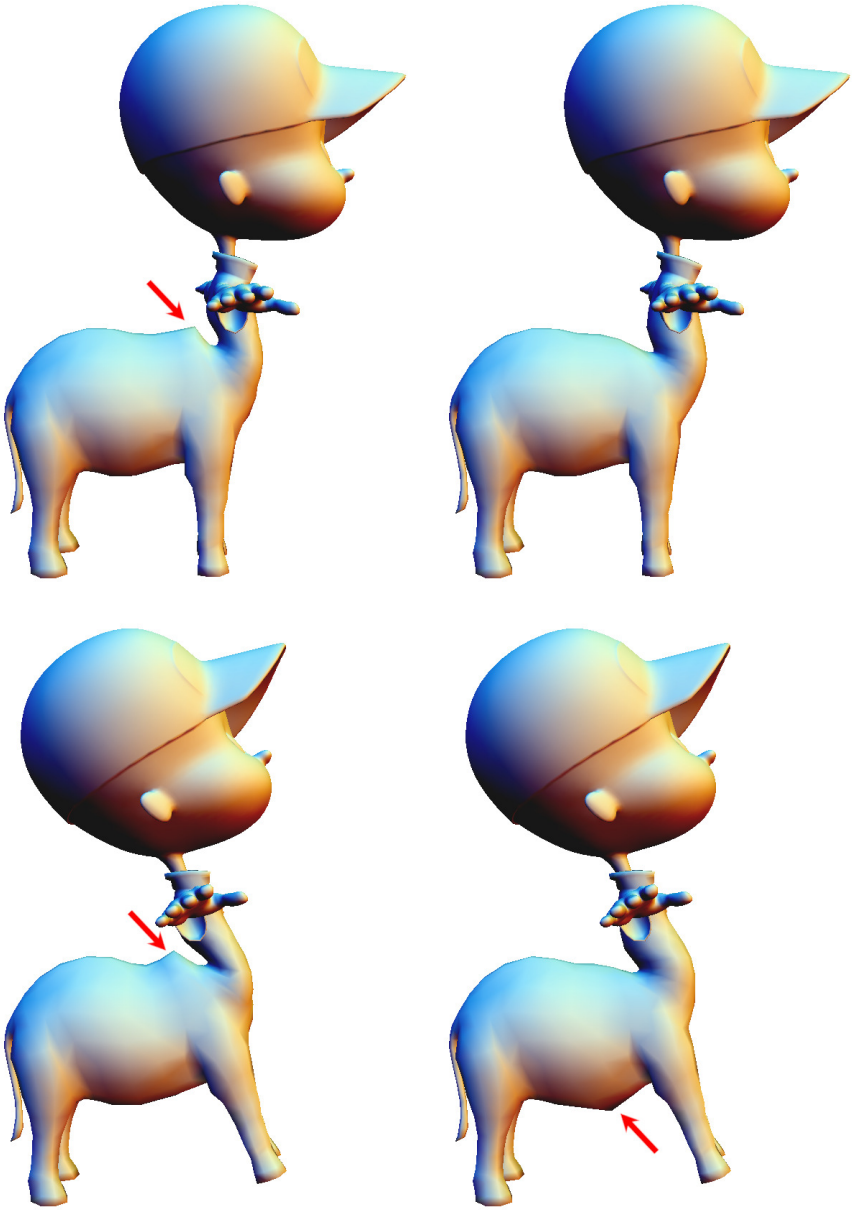


Figure 4.30: **Skinning weight comparison.** A comparison between the skinning weights generated by our method (left) and the automatically generated weights (right) for the model Elephant - Timmy (stomach) of Figure 4.27. Red arrows indicate some artifacts and flaws on the deformed surfaces. Our method presents worse results in the intersection area, but it perfectly performs for the components which have been preserved.

4.4 Discussion

We propose a novel method to compose skinned model components into new characters ready to be used in animation pipelines. We extended the surface retopology approach of Chapter 3 to natively *blend* arbitrary complex surfaces. Furthermore, we present some techniques to compose also the data structures necessary to animate the characters, such as skeletons and skinning weights. *SkinMixer* produces excellent results in most of the cases, however, more work should be done to increase performances and robustness. In the following subsections, we discuss some limitations and possible future works.

4.4.1 Limitations

As we anticipated at the end of the last chapter, our method can fail in determining proper skinning weights in case of the composition of shapes too different from each other. Actually, this is a problem that can be found in the surface *blending* process as well. Indeed, to obtain a proper result, the characters in a replace operation should be adequately positioned. For example, we want to avoid that the involved shapes intersect orthogonally or have a too different orientation. Furthermore, the vertex select values in the two fields should vary monotonically along a direction in a way that they are complementary with respect to each other. For example, it would not be ideal that, in a given voxel, we have select values near 0 or 1 in both the fields.

Figure 4.31 shows a problematic replace operation for the models Timmy and Elephant. The involved surfaces do not match in their orientation, and they orthogonally intersect with each other. The resulting surface is not satisfactory. In Figure 4.27 the outstanding result has been obtained by manually positioning the input characters in order to have a similar orientation and a proper match between the select value fields. In this case, even if our method performed very well in *blending* the geometry, the skinning weights in the intersection area are not flawless, as shown in Figure 4.30. This is due to how we interpolate select values and skinning weights using the origin polygons of the fields.

However, Figure 4.32 shows a difficult case for which our blending approach produced a satisfactory result. Even if the surfaces have not a similar orientation, the select values in the two fields are distributed well enough to allow the method to produce an acceptable result.

When the surfaces involved in the replace operation are in a proper position and have a similar orientation, the results are impressive. For the replace operations of Figure 4.27, we never performed any manual positioning, except for the Timmy - Elephant character. Figure 4.33 shows a successful result, in which we replace a leg of the Timmy model with an arm of the Crypto model.

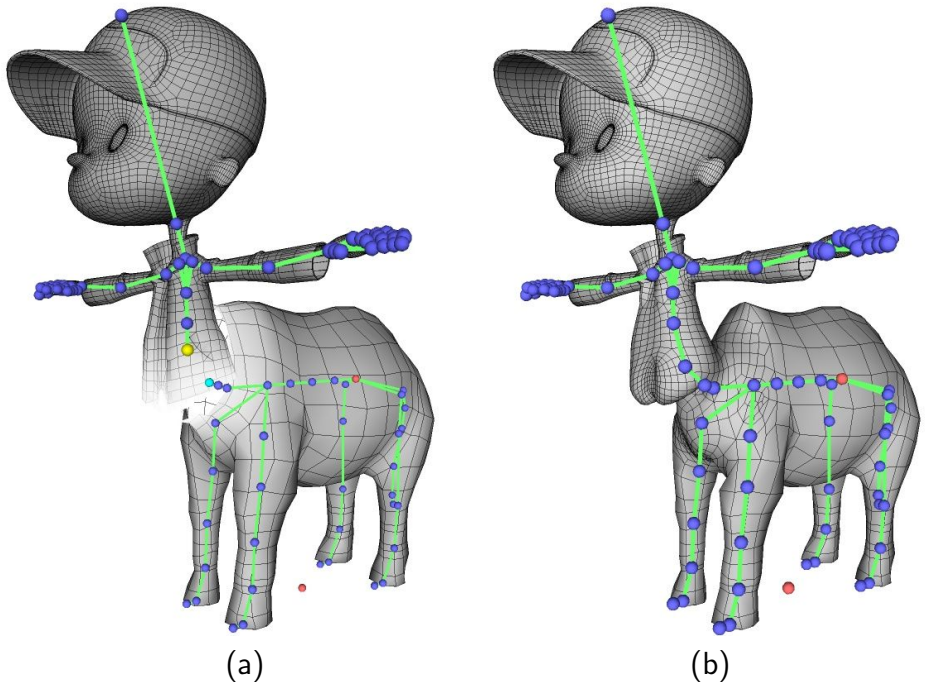


Figure 4.31: **Surface blending limitation.** The involved surfaces intersect with each other and the select values of the two model do not vary accordingly. Our method cannot produce a satisfactory output.

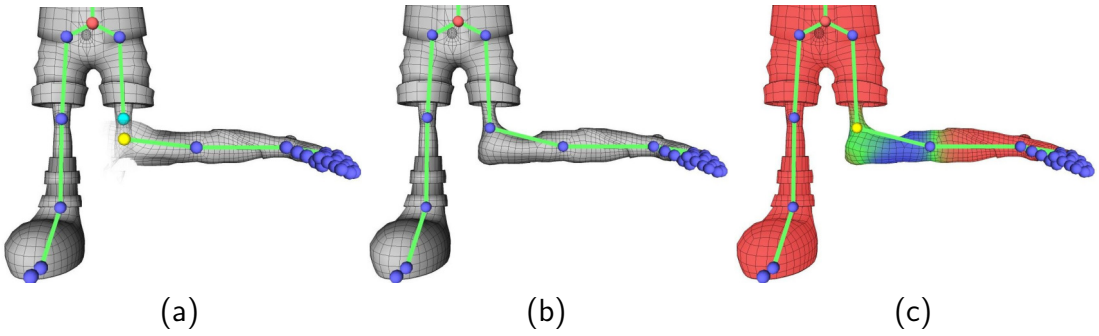


Figure 4.32: **Difficult case for our blending approach.** Even if the two surfaces were not oriented accordingly, the method handled well this case. From left to right are shown: operation setup (a), resulting model (b), skinning weights related to the joint in yellow (c).

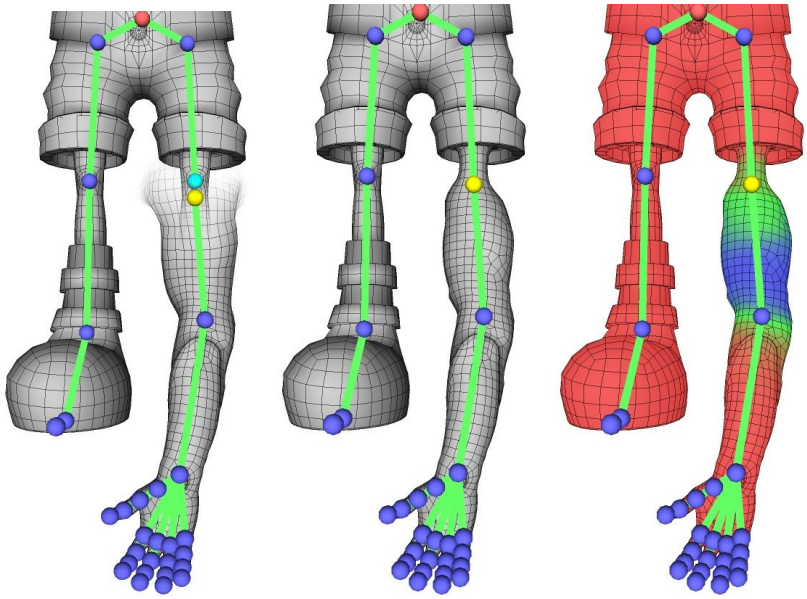


Figure 4.33: **Operations on semantically different components.** Our method performs well when the surface orientation and the vertex select values of the models accordingly match in the volume. From left to right are shown: operation setup (a), resulting model (b), skinning weights of the joint in yellow (c).

Operations on volumetric fields can be expensive in terms of time and memory complexity. As we already stated, we choose the voxel size as a function of the average edge length in the input meshes. However, we must be careful to do not choose a too low value. Indeed, a too dense sampling in the volumetric field could lead to a huge memory complexity during the process.

4.4.2 Future works

In the previous subsection, we presented the limitation of the method in the case of shapes not oriented properly or too different from each other. Furthermore, we prefer selecting values in the two shapes that monotonically grow in opposite directions. It would be interesting to automatically find and suggest to the user the best orientation and position for the detached model in a replace operation. This could be performed by solving an optimization problem to find the best affine transformation to be applied to the characters. The objective function could be a value that describes how the vertex select values of the two models "match" in the field.

Our method does not take into account the symmetry of the tessellation. For example, Figure 4.20.b shows a symmetric shape for which the quadrangulation does not present symmetry. Further research on this matter would be very interesting and it would significantly improve the visual quality of our results. A possible solution to this problem would be to detect symmetric regions of the shape analyzing the geometric features (we could also consider the tessellation of the input models) and force the patch subdivision procedure (Subsection 3.2.2) and the subdivision optimization (Subsection 3.2.3) to generate symmetric results. While, due to its heuristic nature, it could be challenging to guarantee this property in the tracing procedure, we can obtain symmetric results in the ILP optimization by simply adding a cost in the objective function that forces symmetric patches to have the same subdivision values. The state of the art report in [MPWC13] shows some candidate methods for symmetry detection.

In Chapter 3 we used boolean operations for composing the two quadrilateral meshes. In this Chapter, instead, we determine the new surface by a smooth *blending* of signed distance fields. It would be interesting to have the option to perform the union of the shapes, similarly to *QuadMixer*. This union operation can be performed directly on the distance fields, by selecting the minimum distance value for each voxel. This approach could be implemented by only changing a little step of the pipeline.

In our pipeline, we did not take into account the animations already defined for each character. While it is trivial to simply transfer the local transformation of each input animation for the corresponding skeleton joints, it would be great to find a way to automatically retarget the complete animations on the new character.

Finally, we have not taken into account the 3D texture of the assets. For the preserved surface, it is trivial to transfer the texture data since we have a direct mapping between vertices. However, determining new texture for intersection area is challenging. It would be interesting to apply some data-driven texture synthesis algorithms to generate such textures.

Acknowledgments Many thanks to Fabrizio Corda for helping me with the animation pipeline and the conversion of the datasets. Most of the rigged models showed in this chapter are taken from Mixamo [Bla14]. The African Elephant has been professionally designed by Jan Pecnik (https://connect.unity.com/u/jan-pecnik) and it is available for purchase in the Unity Asset Store.

Chapter 5

Conclusions

In this thesis, we presented novel techniques in the context of the modeling-by-composition paradigm. We proposed two operation-based methods that allow the user to freely compose parts of input 3D assets. In the current pipeline used in the industry, generating high-quality 3D assets is a daunting and expensive task, performed by high-skilled artists. The proposed methods enable to perform operations as detaching, removing and merging of input character components, automatically preserving most of the valuable work performed by the professional designers.

In Chapter 3 we introduced a robust method for *blending* geometrical components of pure quadrilateral meshes, taking inspiration from the boolean operations defined for triangle meshes. Chapter 4 extends this retopology approach to work on complex shapes with a quad-dominant tessellation. Moreover, it illustrates a novel method to preserve and merge the data structures for skeletal animations during the composition.

The methods are robust and produce output characters of excellent visual quality. For SkinMixer, we also obtained remarkable results during the model animation. However, in the intersection regions, the resulting surface and skinning weights could be imperfect. Nonetheless, the output of our methods can always be an exceptional starting point for a refinement process of the artist.

By analyzing the impressive results obtained, we believe that the proposed approaches could be a powerful tool in the entertainment industry. Integrated into current software packages for model design, the artists would be able to rapidly assemble parts of their model into a new one ready to be animated. This would greatly reduce the cost of the design process, that usually consists of a complete editing session starting from scratch.

Appendix A

Automatic surface segmentation for seamless fabrication using 4-axis milling machines

We introduce a novel geometry-processing pipeline to guide the fabrication of complex shapes from a single block of material using 4-axis CNC milling machines. This setup extends classical 3-axis CNC machining with an extra degree of freedom to rotate the object around a fixed axis. The first step of our pipeline identifies the rotation axis that maximizes the overall fabrication accuracy. Then we identify two height-field regions at the rotation axis's extremes that are used to secure the block on the rotation tool. We segment the remaining portion of the mesh into a set of height-fields whose principal directions are orthogonal to the rotation axis. The segmentation balances the approximation quality, the boundary smoothness, and the total number of patches. Additionally, the segmentation process takes into account the object's geometric features, as well as saliency information. The output is a set of meshes ready to be processed by off-the-shelf software for the 3-axis tool-path generation. We present several results to demonstrate the quality and efficiency of our approach to a range of inputs.

A.1 Introduction

Digital fabrication technologies, which aim at producing physical objects from their digital representation, have significantly progressed in recent years. Thanks to the coupling between the increase in automation, accuracy and flexibility and the reduction in reproduction costs, applications of digital fabrication are booming in many areas [NKI+18].

The most common technologies for physically producing 3D shapes can be

broadly subdivided in two classes: *Additive Manufacturing* techniques (*3D printing*), which build objects by adding material layer by layer, and *Subtractive Manufacturing* techniques (*CNC machining*), which construct objects by cutting material away from a solid block.

Additive technologies have the major advantage that they decouple the manufacturing process from the geometric complexity of the fabricated object. Computer graphics and geometry processing boosted the use of 3D printing in multiple contexts and have recently explored its limits for the production of complex functional shapes [SEPC16]. The increasing availability of inexpensive 3D printers has further widened the potential user base, leading to a booming market.

While 3D printing may be considered optimal for small-scale production or rapid prototyping [NKI+18], industrial productions often require the manufactured products to be produced in large numbers and to comply with physical constraints that are inconsistent with additive technology (e.g., robustness or usage of natural materials such as wood or stone). Contrarily to 3D printing, CNC machining is usually faster and capable of producing massive parts using a wide range of materials. Moreover, since shapes are carved out from solid blocks of material, the resulting objects are structurally more robust than those created with a layer-depositing process. Therefore, subtractive manufacturing remains the dominant process in industrial settings.

Unfortunately, subtractive manufacturing has strict geometric constraints that impose severe limitations on the class of fabricated objects. In particular, traditional 3-axis CNC machining requires the manufactured object to be a height-field. Consequently, the mass production of intricate shapes is challenging, if not impossible, since it needs an expert to plan, then manually adjust the object's orientation to accommodate the different working directions. For these reasons, considerable effort has been spent on extending the fabrication DOFs. However, this task often requires expensive machines (e.g., 5-axis milling tools or robotic arms) controlled by complex automated planning systems [Tan14], making the user base of these advanced solutions considerably narrower from that of 3D printing.

In this context, 4-axis CNC milling has a particular position in the spectrum of fabrication technology. It operates on the same axes of a 3-axis machine, but also includes the rotation around one of the axes. By enabling the rotation of the volume between each carving operation, it supports the manufacturing of a much broader range of forms, at a buying and maintenance expense that is only marginally higher than that of the classical 3-axis solutions. The primary purpose of path planning in 4-axis CNC machinery is to derive the optimal rotation axis and the minimum setup directions. Unfortunately, the few solutions that exist in the state-of-the-art are not capable of producing complex shapes, forcing the extensive use of manual interventions (see Sec. A.2).

We introduce a novel pipeline to fabricate complex shapes from a single block of material using a 4-axis CNC machine. We first devise the rotation axis that maximizes the surface's orthogonality to the milling directions while fitting the object in the fabrication workspace (i.e., the raw material block). Then, we fabricate the object's largest area by milling towards a sequence of directions orthogonal to the rotation axis. Finally, we mill the extremities secured to the rotation axis. To derive the optimal set of directions, we segment the object into multiple height-fields, which are then ordered into a milling sequence. The subdivision jointly optimizes individual height-field approximation quality, boundary smoothness, and the total number of patches, while avoiding unwanted collisions between the milling tool and the surface. Additionally, this segmentation process exploits saliency information (automatically-generated or user-provided) to avoid possibly visible seams in highly-detailed or semantically important surface regions. Figure A.1 shows, from left to right, the proposed pipeline.

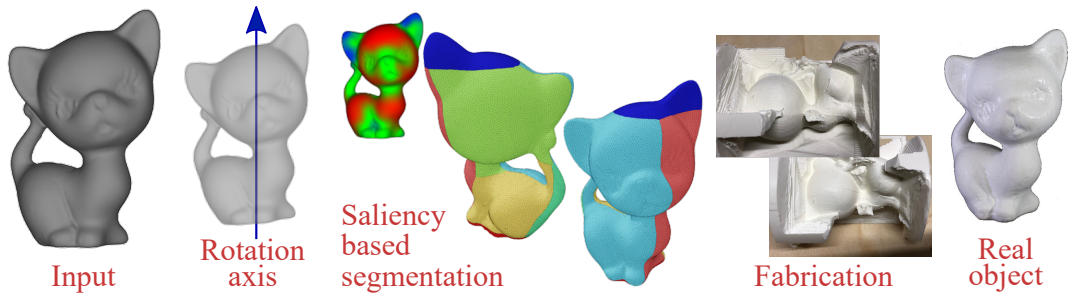


Figure A.1: **Our processing pipeline in a nutshell** (left to right): we first compute the rotation axis on the input mesh; we then partition the mesh in millable height-field portions (including the top and bottom ones) taking also into account the information of the saliency map; we compute a milling sequence and fabricate the object using the 4-axis milling machine; we clean up the result to obtain the final real object.

Our main contribution is a novel integrated optimization scheme that extends height-field decomposition approaches to the 4-axis fabrication setup. The proposed method is effective and practical. By exploiting the extra rotation axis, we extended the range of shapes that can be manufactured to a single solid piece of material, making subtractive fabrication an appealing solution for a wide range of users. In contrast to previous techniques that split objects into multiple components that are assembled afterward, our method produces seamless and robust shapes (see Sec. A.4). Furthermore, by reducing 4-axis fabrication to a sequence of 3-axis milling processes, we can directly use any of the many 3-axis fabrication drivers available in the industry, making our solution immediately applicable in practice.

A.2 Related work

There is a vast literature on CNC machining in Computer-aided design and mechanical engineering. Most of that research focuses on optimizing the tools and their path for the milling process and does not emphasize the surface decomposition. The task of planning the global carving process, and ensure the tool accessibility, is usually allocated to the experts. The increase in modeling capabilities and the resulting complexity of available 3D shapes made this manual task harder. So a lot of literature in Computer graphics studied how to automatically decompose the surface or the milled volume into parts that are suitable for CNC fabrication.

A.2.1 Height-field surface decomposition

A common strategy to overcome the limitations of CNC machines is to decompose the object into multiple height-fields. A 3-axis CNC machine can fabricate each height-field part individually, and then the pieces are manually assembled afterward. One of the first methods that followed this idea is the decomposition proposed by Alemanno et al. [ACP+14]. However, this method requires the user to specify the surface decomposition manually. Since it does not consider the assembly process, its usage is mostly limited to the simple cases of 3D-assembled low-reliefs. Most of the literature on height-field decomposition for fabrication is related to moldable pieces. Herholz et al. [HMA15] decompose the surface into different height fields, defining the components of a rigid mold. This approach allows the user to extract the casted objects. Flexmolds [MPBC16] and Metamolds [AMG+18] use a relaxed visibility constraint to produce a more general patch decomposition that involves the use of deformable molds. While the task of planning a casting and a milling process have significant overlap (they both strive for high-field patch arrangements), the latter requires special care to guarantee that the tool can access the carving area at any moment during fabrication and that the milling angles produce good quality carvings.

A.2.2 Volume decomposition

Several methods consider the volume of the fabricated object during the optimization process [SEPC16]. Most of the techniques that use additive manufacturing decompose the object into multiple components that can fit the 3D printer's workspace. Chopper [LBRM12] decomposes the model into pyramidal parts that minimize the use of supports [HLZCO14]. Dapper [CZL+15] optimizes the packing of the parts in the printing volume. Recently, Filoscia et al. [FAG+20] proposed a new method to decompose a 3D-printed object into portions that minimize the visual impact of the seams. However, these methods are not directly related to

the planning of subtractive fabrication. The decomposition strategy proposed by Araujo et al. [ACA+19] guarantees an assembly sequence. Regarding molding, composite molds [AMG+19] are the only ones that consider explicitly the volume surrounding the casted object.

Most of these methods aim for height-field approximations or consider the volumetric access to allow for assembly. However, they are designed for additive fabrication or casting, while subtractive techniques pose additional challenges. Since the milling tool carves the object from the outside toward the inside instead of merging planar layers, extra constraints should guide the milling angles and secure the milling tool's accessibility. Fanni et al. [FCM+18] proposed a decomposition method based on polycube mappings and analyzed the manufacturability of their decomposition for additive and subtractive techniques, taking into account both 3- and 4-axis milling. Since their method does not take into account special constraints relative to subtractive techniques, their final decompositions prove feasible in practice just for additive techniques. The approach proposed by Muntoni et al. [MLS+18] decomposes, instead, a 3D object into height fields, then projects the decomposition toward the interior, covering the entire volume and, at the same time, ensuring each piece to be manufacturable with 3-Axis CNC machines. The final model is obtained by reassembling the pieces after fabrication, leading to visible seams and reducing the physical robustness of the fabricated model. To avoid decomposing the object into multiple parts, DSCarver [ZZX+18] uses a (3+2) axis milling machine, that acts like a traditional 3-axis milling machine during carving, but supports two-degree-of-freedom (2-DOF) rotation of the drilling tool between each milling pass. DSCarver automatically plans and optimizes the different phases and rotation angles by covering the input surface with a minimum number of accessible regions and then extracting a set of machinable patches from each accessible region. Thanks to the extra degrees of freedom, this method can produce significantly complex objects in a single piece, including high-genus ones. Their approach, however, requires full exploitation of the 3+2 DOFs.

To the best of our knowledge, none of the proposed methods is specifically designed for 4-axis CNC machines, a popular choice for industrial production.

A.3 Automatic fabrication planning

A 4-axis CNC machine extends the classical 3-axis CNC device with an extra degree of freedom corresponding to the object's rotation around an axis. In a typical manufacturing setup, as illustrated in Figure A.2, the carved volume is fixed to a rotating shaft through a solid flat base. Thus, the fabrication of a general 3D object starting from a solid block of material (called *stock*) must involve at least three phases to cover the entire surface. In particular, while we can fabricate the

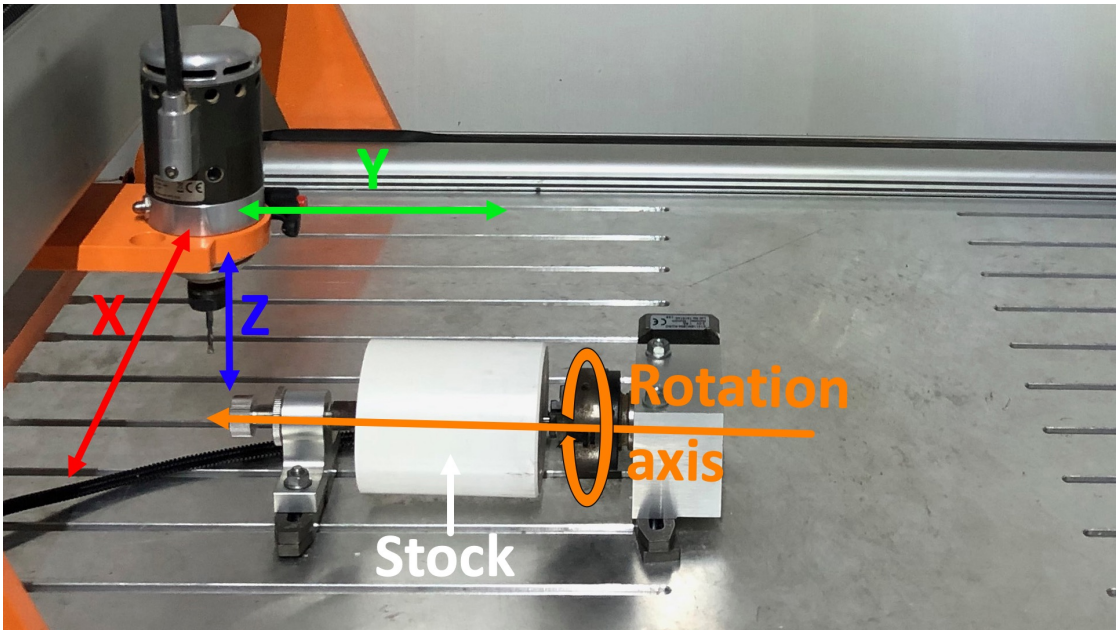


Figure A.2: **Our fabrication setup.** The milling tool on the left has three degrees of freedom in the operational field of the machine; the rotating shaft adds the fourth one.

large majority of a shape by exploiting the different milling directions provided by the rotation axis, two additional fabrication phases are necessary to carve the top and bottom segments that were not accessible during the first phase. Figure A.3 illustrates the entire process.

Our automatic fabrication planning method aims at finding the most efficient and accurate plan to organize the milling process. We decompose the fabrication process into a sequence of 3-axis milling processes. Each fabrication stage produces a suitably oriented height-field representation of a small surface region. In particular, for a given rotation axis α, we partition the object into three different regions: a *side* region, a *top* region, and a *bottom* region. We then approximate each of the bottom and top parts as a single height field whose elevation is aligned with α, and the side region as a set of height fields whose elevations are orthogonal to α. Hence, our automatic fabrication planning involves the following steps:

- We prefilter the geometric detail of the object to produce a regularized representation suitable for further processing (Sec. A.3.1).

- We find the rotation axis α that produces the best height-field approximations in the three regions (top, bottom, and side). Intuitively, in this optimization process, we favor the orthogonality of the fabricated surface to the resulting

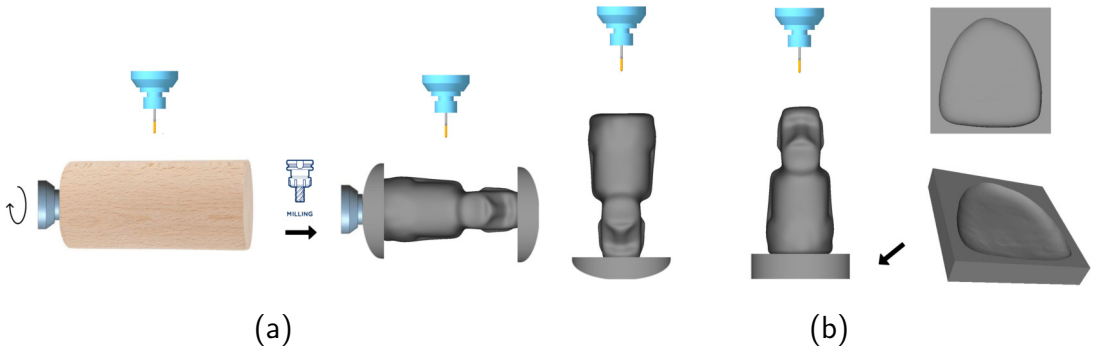


Figure A.3: **Fabrication process outline.** We first carve the set of height-fields on the side of the model from the raw block of material (a); then, we unpin the block from the rotation tool, and we mill the two height-fields (top and bottom) (b). Notice that we use a mold to keep the model correctly in place for milling the second between top and bottom.

milling directions (Sec. A.3.2).

- Given the rotation axis α, we segment the model into a set of non-overlapping height fields. We first derive the top and bottom regions that can be fabricated as a single height field. Then, using a graph-cut algorithm, we segment the remaining side regions into a set of height fields that can be fabricated by rotating around α, taking into account fabricability, efficiency, and accuracy criteria (Sec. A.3.3).
- We recover the details lost in the prefiltering phase (Sec. A.3.4).
- We produce the final fabrication plan, determining the order of fabrication of the individual charts and the process to produce each of them (Sec. A.3.5).

Details on each of these different steps are provided in the following sections.

A.3.1 Prefiltering

Since a rough meshing can cause several problems to segmentation algorithms, we initially re-mesh the input shape to a dense and regular uniform tessellation. Additionally, high-frequency details might induce excessive fragmentation in the final height-field decomposition. To tackle this issue, we use the same approach as Muntoni et al. [MLS+18], that allows users to sacrifice the fidelity to the input to enable a more natural and faster fabrication. We first remove high-frequency surface details using the low-pass Taubin filter [Tau95]. Then, after the segmentation, we reintroduce high-frequency features while enforcing the fabricability constraints

(see Sec. A.3.4 for more information). This pre-processing step is optional and controlled by the user.

A.3.2 Determining the best overall milling orientation

The choice of the rotation axis α has a massive impact on the quality of the final segmentation and fabrication (see Figure A.4 for an illustration of the effect of selecting two different rotation axes). The first step of our method is, therefore, the selection of the axis that provides the best fabrication accuracy while tightly fitting the model inside the stock. To do so, we must solve two problems. First of all, we must determine how a given rotation axis partitions the surface into the side, top, and bottom fabrication regions (Sec. A.3.2). Then, we must foresee how a given partitioning will impact the overall fabrication accuracy (Sec. A.3.2).

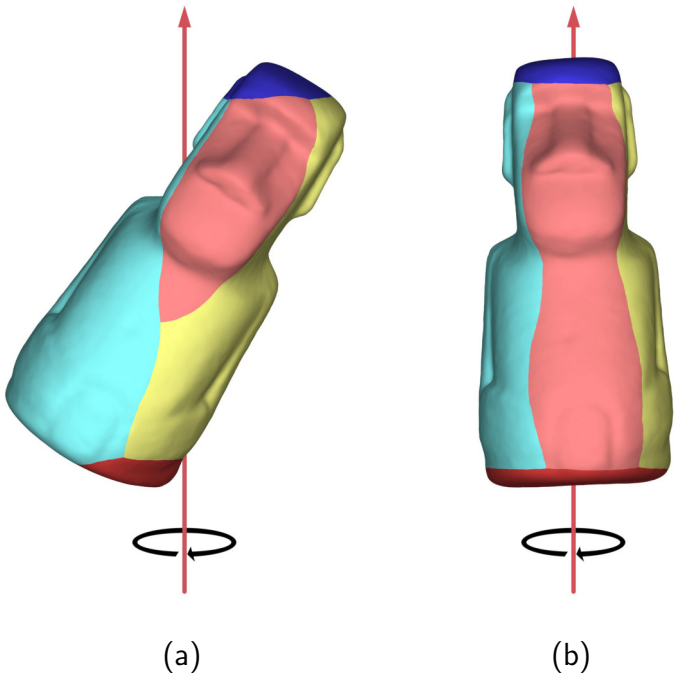


Figure A.4: **Effect of axis selection.** The choice of the axis in (a) leads to height-fields of varying height, thus worsening the tool-path creation; the axis shown in (b), while keeping the same number of height-fields to carve, leads to more homogeneous, in size and depth, ones.

Partitioning into top, bottom, and side regions

As we previously stated, given a rotational axis, we should partition the input geometry into three regions: the top and bottom areas (fixed to the rotation axis) and the side region (milled by exploiting the extra rotation degree of freedom). For a given rotation axis α, we fabricate the top and bottom regions along with the directions $+\alpha$ and $-\alpha$. Given an input 3D mesh $\mathcal{M}$, we search for two disjoint top $\mathcal{P}_{top} \subset \mathcal{M}$ and bottom $\mathcal{P}_{bottom} \subset \mathcal{M}$ regions. A face $f_i \in \mathcal{M}$ is *visible* along a direction d if the dot product between face normal n_i and d is less than zero and it is not occluded along d. This visibility definition allows taking into account simultaneously for fabricability and access to the milling tool.

Given a rotation axis α, the top $\mathcal{P}_{top}$ and bottom $\mathcal{P}_{bottom}$ regions should be composed by the largest set of connected faces which are *visible* along $-\alpha$ and $+\alpha$ respectively. Additionally, those regions should also be as close as possible to the shape extremities along the rotation axis. Thus, we first initialize the top and bottom regions using the extremity faces (i.e., the face incident to the top-most or bottom-most vertex) along the rotation axis. We then iteratively grow each of those regions until it reaches the visible face having the largest distance along the rotation axis. The remaining faces compose the side region. Notice that, given a rotation axis, it is always possible to select two extremity faces to initialize the top and bottom regions. Notice also that, by definition, the three regions are disjoint. Figure A.5 shows the resulting decomposition.

Finding the best axis

Determining a provably optimal axis, in theory, would require the evaluation of all the effects it has on fabrication. This is however very costly, and could lead to repeatedly segment the model for several orientations, as the objective function can have many local minima (see Sec. A.3.3). We propose here a more efficient approximate solutions, that decouples axis selection from segmentation, rapidly determining a good, if not provably optimal, axis before the segmentation using only geometric measures on the mesh.

Intuitively, the rotation axis that will produce the highest quality is the one that maximizes the overall alignment of the fabricated surface normal with the milling directions. The alignment is calculated separately for each component. For each of the top and bottom regions, where milling is aligned with the axis direction, given a candidate rotation axis d, we compute the alignment as the sum of the dot

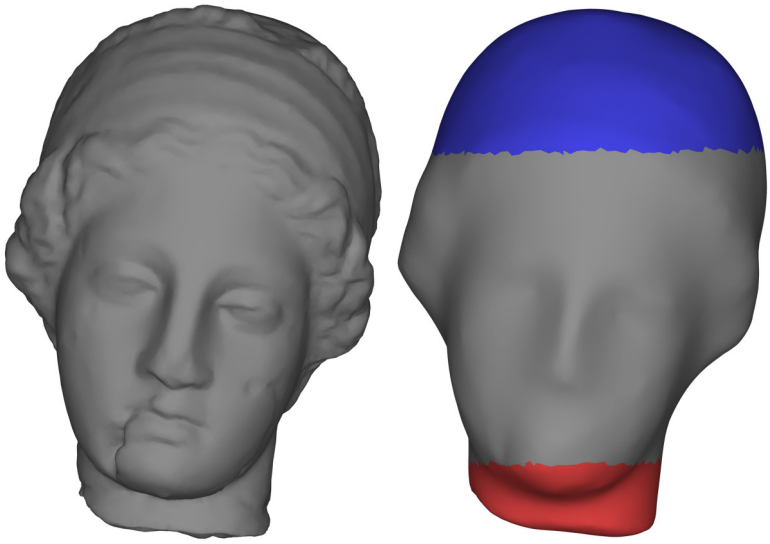


Figure A.5: **Decomposition into regions given an axis.** (a) Original model. (b) Smoothed model with top height field in blue, bottom height field in red, and side region in gray,

product between face's normal and the rotation axis, respectively:

$$A_{top} = \sum_{f_i \in \mathcal{P}_{top}} a_i * (n_i \bullet d) \tag{A.1}$$

$$A_{bottom} = \sum_{f_i \in \mathcal{P}_{bottom}} a_i * (n_i \bullet -d) \tag{A.2}$$

where a_i and n_i are the respectively the area and the normal of the face f_i.

Instead, for the remaining faces on the side, we must favor the orthogonality of the face normals to the rotation axis. This copes with the fact that the milling machine will work on a shaft, which will be orthogonal to the rotation direction.

$$A_{side} = \sum_{f_i \in \mathcal{P}_{side}} a_i * (1 - |n_i \bullet d|) \tag{A.3}$$

To compute the axis, we compute a large set of candidate direction $d_1, ..., d_k$. The k directions, with $k = 2000$ for the examples in this chapter, are uniformly distributed on a hemisphere using a Fibonacci sphere algorithm [SJP06]. We then choose the one that maximizes $A = A_{top} + A_{bottom} + A_{side}$.

A.3.3 Determining the optimal decomposition into height fields

Once we establish the rotation axis, we can partition the input geometry into the three regions using the method of Sec. A.3.2. The top and the bottom areas are height fields fabricated in a single pass by milling along the main direction of the rotation axis. However, the side area is a complex surface, whose fabrication requires to rotate the object during fabrication. The space of the possible milling directions for the side region is composed of all the directions which are orthogonal to the rotation axis. However, in order to plan an efficient and accurate milling process, we should promote the formation of large smooth patches, as they support a continuous motion of the milling tool [ZZX+18]. Hence, the main task of this segmentation process is to choose an optimal subset of directions and, at the same time, associate each direction with a surface patch composed of a set of faces. The patches constitute a partitioning of the side region, and they should not overlap. This class of problems can be efficiently solved using a multi-label graph-cut optimization [BVZ01] using discretized milling directions as labels.

Segmentation

We sample a set of uniformly generated fabrication directions $\mathcal{L}$ orthogonal to the chosen rotation axis (we used 120 possible directions in all the examples shown in this chapter) and we define a label for each direction. We have to find a labelling function $\ell\colon \mathcal{P} \to \mathcal{L}$ that assigns a direction (label) to each face in $\mathcal{P}$. The graph-cut algorithm derives the optimal labeling by minimizing a function:

$$\arg\min_{\ell} \quad \sum_{f \in \mathcal{P}} D\left(f, \ell\right) + \sum_{(p,q) \in E} S\left(p, q, \ell\right)$$

where E is the set of edges. $D(t, \ell)$ is the fidelity (or data) term and describes the cost of assigning the fabrication direction ℓ to the face f. Intuitively, this term tends to label each face in its optimal direction. $S(p, q, \ell)$ is the smooth term and describes the cost of assigning a given label to two adjacent faces p and q. This term favors the formation of a compact and smooth patch layout and holds the total number of patches.

The fidelity term should consider how much the face normals are aligned with the milling direction. Secondarily, we should avoid associating a face to a direction along which it is occluded by other object portions; indeed, that will prevent the milling tool from reaching the target point in the volume during the milling process. The visibility is precomputed per-face, per-direction using collision detection accelerated by an AABB Tree structure. In case of collision, we simply

assign an infinite cost. We define the fidelity term as:

$$D(f,\ell) = \begin{cases} 1 - n_f \bullet \ell\,(f) & \text{if } f \text{ is visible from } \ell\,(f) \\ \infty & \text{otherwise} \end{cases} \tag{A.4}$$

where n_f is the normal of the triangle.

The smoothness term $S(p,q,\ell)$ minimizes the overall boundary lengths and determines the shape and location of the boundary between one patch and the others.

Since patch boundaries are the regions where one milling sequence is interrupted and then continued from another orientation, to improve the manufacturing process's quality, we should concentrate them in areas with low detail. In order to take into account this factor, we assume that our mesh is enriched by *saliency* information, providing at each face a measure of regional importance [LVJ05] (see Sec. A.3.3). We define the smoothness term as:

$$S(p,q,\ell) = \begin{cases} c + d\,\frac{\mathscr{C}(p)+\mathscr{C}(q)}{2} & \text{if } \ell\,(p) \neq \ell\,(q) \\ 0 & \text{otherwise} \end{cases} \tag{A.5}$$

where c is the compactness term, d is the saliency factor and $\mathscr{C}$ the face saliency. The compactness term c, by influencing the number of patches, controls the smoothness of the segmentation. The saliency factor d is multiplied by the average saliency of the concerned faces, which is higher if the face is considered perceptually more important, and prevents the formation of boundaries that separate a salient area (see Sec. A.3.3).

Saliency computation

Our algorithm can use any per-face saliency factor to encode semantic and/or geometric information. It can be the result of an automatic, semi-automatic, or interactive processes. In our framework, we have implemented both a fully automatic method and an interactive tool.

For automatic saliency computation, we start by computing at each vertex the multi-scale saliency metric $\mathscr{C}(v)$ proposed by Lee et al. [LVJ05], dividing it by the number of scales to normalize the value between 0 and 1. Sinche the multi-scale approach tends to localize the high frequencies in small areas (Figure A.6a), our final saliency field is found by propagating these peak values to their neighborhood through a diffusion process. In practice, we first propagate the maximum value of each vertex v to its neighborhood N_v, repeating this step a fixed number of times (Figure A.6b), and then perform a Laplacian smoothing (Fig. A.6c). For the results of this chapter, our experiments led us to apply 5 propagation and 20 smoothing

steps. We finally compute the saliency $\mathscr{C}(f)$ for each face as the arithmetic means of the value at its vertices.

We also support user-driven saliency determination by letting the user to interactively select/unselect with a brushing tool the areas of interests (Figure A.6d). This makes it possible, in particular, to have users mark semantically important areas.

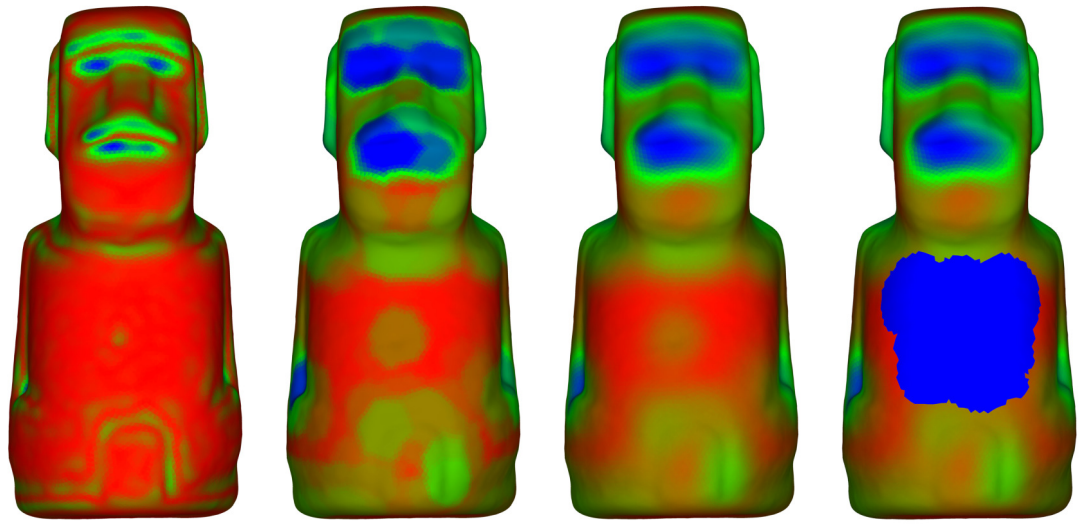


Figure A.6: **Saliency field computation.** We show how we generate mesh saliency for the Moai model. The salience values are between 0 (red), and 1 (blue).

Figure A.7 shows that the saliency term makes a large difference in the segmentation, providing quality in the segmentation in a fully automatic way. The last column presents the two final segmentations used to actually manufacture the Batman and David models.

Charts optimization

Once we have our per-face segmentation ℓ, we can derive the pair-wise disjoint charts $C_1, C_2, C_3, ...C_n$. Each chart $C_i \subset P$ is composed of a connected set of faces assigned to a fabrication direction in $\mathcal{L}$. Note that there could be multiple charts associated with the same direction. The objective function for graph-cut segmentation favors the formation of a small number of charts with regular boundaries and usually avoids forming charts with multiple boundaries.

However, even if we use a large compactness term, we might end up with small isolated charts, mostly due to occlusions in complex shapes. In order to improve

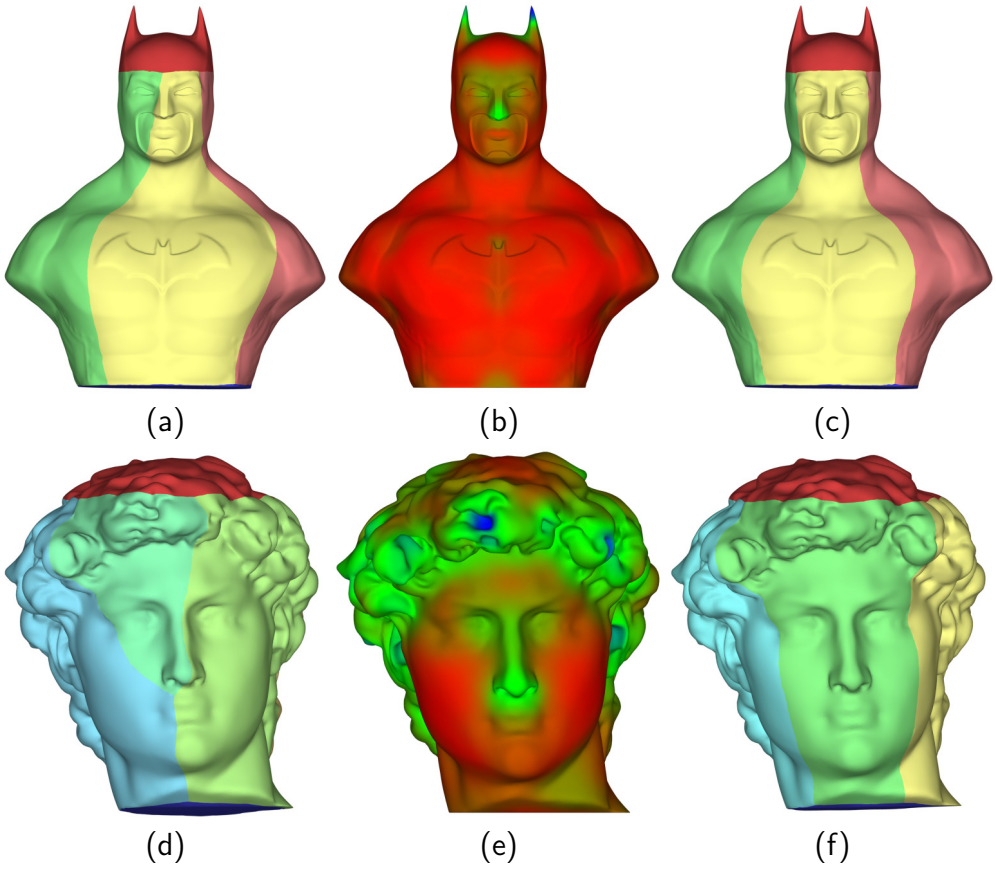


Figure A.7: **Effect of the saliency term on the segmentation.** From left to right: segmentation without considering saliency, final saliency values, segmentation with saliency.

the efficiency of the fabrication process, we delete charts with an area of less than a given threshold (in our experiments we used 0.01% of the total shape area). When we delete a chart, we iteratively re-assign the faces to the best adjacent charts, starting from the border and propagating in the interior. This strategy can assign some face to a direction from which it is non-visible. In this case, we sacrifice the exact fidelity of the shape to allow a simpler fabrication. Notice that this step, similarly to pre-filtering (Sec. A.3.1), is optional and controllable by the user through the tuning of the merging threshold.

The smoothness of the chart boundaries, moreover, plays a crucial role in a high-quality result. Even if the graph-cut usually produces acceptable results, the boundaries might depend on the initial tessellation (Figure A.8a). Therefore, we perform a Laplacian smoothing of the boundary lines, re-projecting the coordinates in the shape's tangent space and refining the original triangle mesh. Figure A.8

shows the effect of this process.

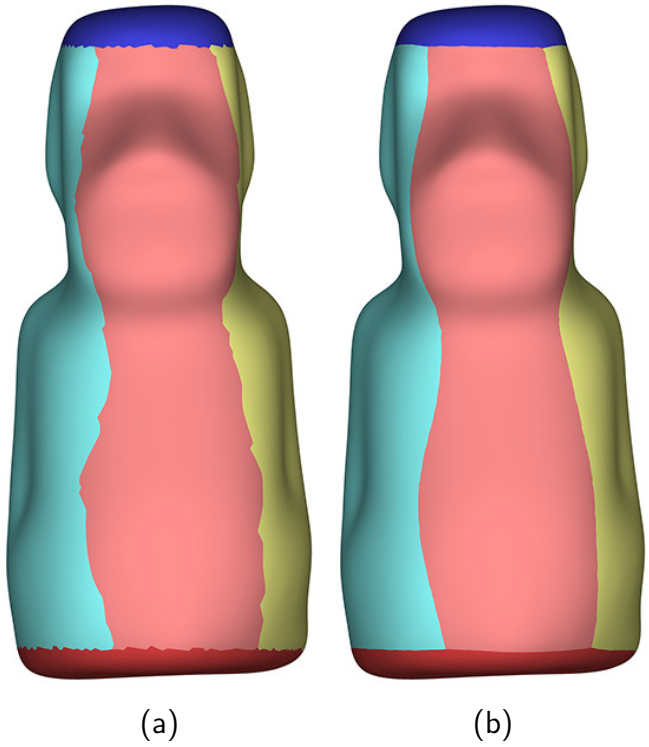


Figure A.8: **Boundary smoothing process.** (a) Original segmentation. (b) Segmentation after boundary smoothing.

A.3.4 Detail recovery

Once we have obtained the decomposition, for each chart, we reintroduce the high-frequency details lost during pre-filtering (Sec. A.3.1) by using Laplacian surface reconstruction framework [Sor06]. As in Muntoni et al. [MLS+18], we constrain the reconstruction with height-field constraints that ensure fabricability, without taking into account that high-frequency detail reintroduction could cause some triangles to be no longer visible due to slight occlusions (Figure A.9). Again, we sacrifice the exact fidelity of the shape to simplify the process. The tool-path generation software will automatically produce the closest fabricable shape.

In particular, we iteratively minimize a given energy via coordinate descent, deforming each vertex at a time and freezing the others. Let v be a vertex in the deforming mesh $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ and w its corresponding vertex in the original mesh.

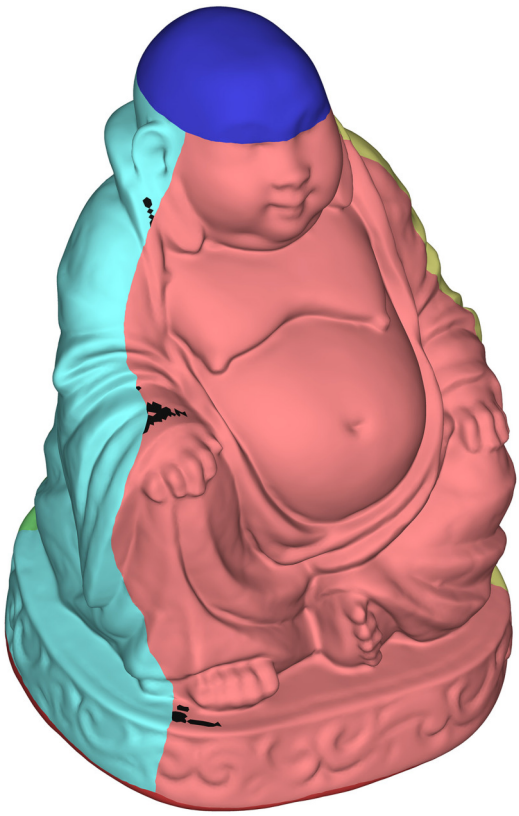


Figure A.9: **Occlusion effects.** Black faces represent triangles no longer visible from the spindle after restoring the high frequencies.

Then we solve the following optimization problem:

$$\begin{aligned} &\arg\min_{v} ||\delta v - \delta w||^2 \quad \text{s.t.} \\ &\forall f \in \mathcal{F} \text{ incident in } v \quad n_f \bullet \ell(f) \geq 0 \end{aligned} \tag{A.6}$$

where δ denoted the differential coordinates and n_f denotes the face normal of f in the mesh to be deformed. The new vertices generated by the boundary smoothing have not a corresponding vertex in the original mesh. Thus, at each iteration, we deform them in order to keep their differential coordinates unchanged in the deforming mesh.

A.3.5 Generating the fabrication sequence

Our optimization pipeline allows us to use 4-axis machines by partitioning the fabrication into a sequence of 3-axis milling processes. Thus, we can easily use any available 3-axis fabrication driver to compute the tool-path for each height-field.

The physical fabrication of objects composed of many charts poses, however, extra considerations. As shown in Figure A.10a, depending on the depth of the surface to mill, the drill bit could not reach that depth without tool collisions. For each chart, we must guide the tool-path generator to engrave portions of volume, ensuring accessibility and avoiding milling surfaces that belong to other charts. We must thus embed each chart surface in the stock, computing the volume that has to be carved to mill it (Sec. A.3.5), order the charts into a fabrication sequence (Sec. A.3.5), and finally construct the final fabrication plan (Sec. A.3.5), By solving these problems, we make it possible to use our pipeline with any of the available 3-axis software.

Embedding chart surfaces into the stock

In order to be fabricated, each shape to mill must embed its chart into the given stock, meeting the following constraints:

- it contains the entire surface of the chart to mill;
- it does not contain surface assigned to other charts, or, when not possible, it is minimum;
- it can be milled entirely without collisions of the drill bit.

Note that, in our examples, we used a cylinder as stock to maximize the final result size and minimize scrap material, but any other shape is allowed.

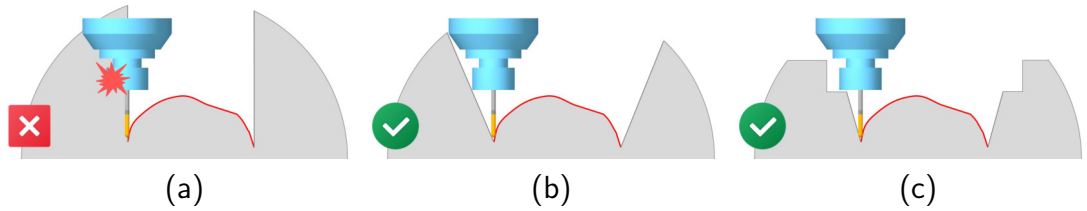


Figure A.10: **2D representation of how we generate the surface for embedding the chart surface into the stock**. (a) Limitations of a surface that is parallel with respect with the fabrication direction. (b) Surface that form a given angle guarantees fabricability and manufacturing quality. (c) Reducing manufacturing time with steps orthogonal to the fabrication direction.

For each chart, we have to propagate the surface from its boundaries to the extremes of the fabrication volume. This propagated surface must guarantee both physical access of the milling tool and high precision near the boundaries. As illustrated in Figure A.10.a, the trivial solution could be to project the extrema of

each chart in the direction orthogonal to the rotation axis (we would obtain parallel *walls*) to obtain the portion of stock to mill. This choice would lead to collisions and impossibility to mill near the boundaries. Instead, we tilt the walls until they form an angle with the fabrication direction that avoids collisions when milling the borders of the chart (Figure A.10.b). We tilt the walls only in the proximity of the chart boundaries, while once at a safe distance we simply propagate orthogonal walls until we reach the external surface of the stock (Fig. A.10c). This solution allows for the physical access of the milling tool and, at the same time, minimizes the possibility to collide with other patches. Figure A.11 shows an example of a chart embedded in a stock.

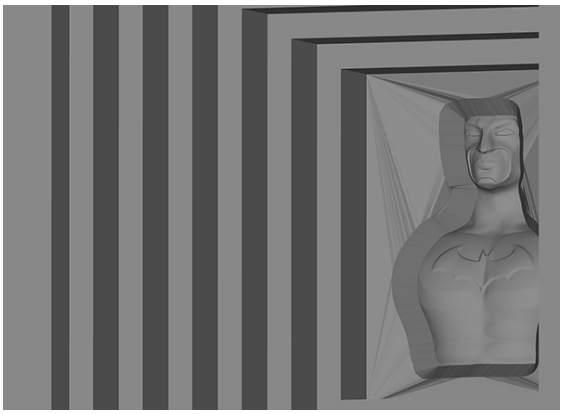


Figure A.11: **Chart embedded in the stock.** Tilted walls are erected and then step-like orthogonal walls are propagated until the external surface of the stock is reached.

Ordering the charts

Each of the resulting meshes generated by the embedding procedure contains the respective chart's surface. However, as shown in Figure A.12, oblique walls might intersect with other charts. In this case, we can improperly mill other portions of the final surface. This interference must be solved by properly ordering the charts.

To derive the best ordering, given the n charts $C_1, ..., C_n$ forming the decomposition of the side of our input shape, we define a $n \times n$ conflict matrix $\mathcal{M}$, whose entries $\mathcal{M}(c, d)$ (we consider only $c \neq d$) are defined as the total area of the surface belonging to a chart d that would be disrupted while milling the chart c:

$$\mathcal{M}(c, d) = \sum_{f \in c} a_f \quad \text{s.t. } f \text{ is reachable and } f \in d \tag{A.7}$$

where a_f denotes the face area in faces f of the embedded shape. In other words, $\mathcal{M}(c, d)$ represents the cost, in terms of surface area, of fabricating a chart c before

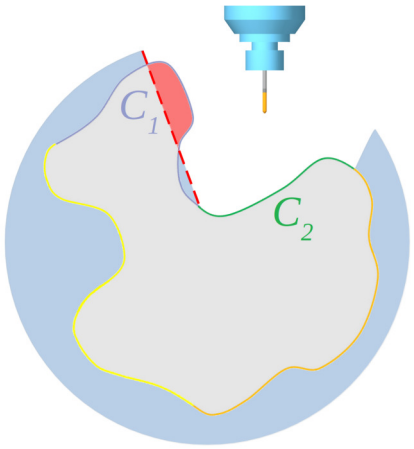


Figure A.12: **Effect of improper charts ordering.** The oblique boundary of chart C_2 intersects a portion of the surface of another C_1 (highlighted in red). In this case, the red area would be unintentionally removed when performing the milling of C_2.

a chart d. We want to find a chart ordering $(c_1, ..., c_n)$ that minimizes the overall cost:

$$\arg\min_{(c_1,...,c_n)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathcal{M}(c_i, c_j) \tag{A.8}$$

We solve this optimization problem with a greedy approach. Conventionally, let $\mathcal{M}(c, d) = 0$ for $c = d$. At each iteration i, let $\mathcal{C} = \{C_1, ..., C_n\} \setminus \{c_1, ..., c_{i-1}\}$ be the set of charts that have been not processed yet. Then, we select the chart c_i as follows:

$$c_i = \arg\min_{c \in \mathcal{C}} \sum_{d \in \mathcal{C}} \mathcal{M}(c, d) \tag{A.9}$$

The cost term in the summation represents the surface of the chart c_i having conflicts during the current iteration. In case of equal costs, we choose the chart c_i that maximizes $\sum_{d \in \mathcal{C}} \mathcal{M}(d, c)$, that is the area which will not conflict anymore with other charts in the next iterations. Although this heuristic does not guarantee to find the optimal solution, it works very well in practice. In all our experiments, we were able to find a result with zero overall cost in all the cases, except for the ones with cross-conflicts between charts. The part of the object belonging to charts not yet milled is usually entirely hidden by our embedding method.

Generating the final fabrication plan

Once we derive the best order, we can create the final fabrication plan. The 3-axis driving software typically requires two input shapes per chart: one representing the initial block of raw material to be carved (the stock) and the desired target shape. Hence, by determining these meshes, we can easily produce our shapes with any widely available 3-axis driver, completely abstracting from the usage of a 4-axis machine.

Starting from the initial block and the first chart, we iteratively process each target shape. In case it was not possible to avoid all the conflicts between charts, the concerned surface is included in the target shape. In this way, we choose to fabricate it with less accuracy from another direction, instead of losing a portion of the final object. At each iteration, we use the resulting mesh of the previous step as the initial stock. We use robust Boolean operations described by Zhou et al. [ZGZJ16] to perform these operations. Fig. A.13 shows a simple 2D representation of the process.

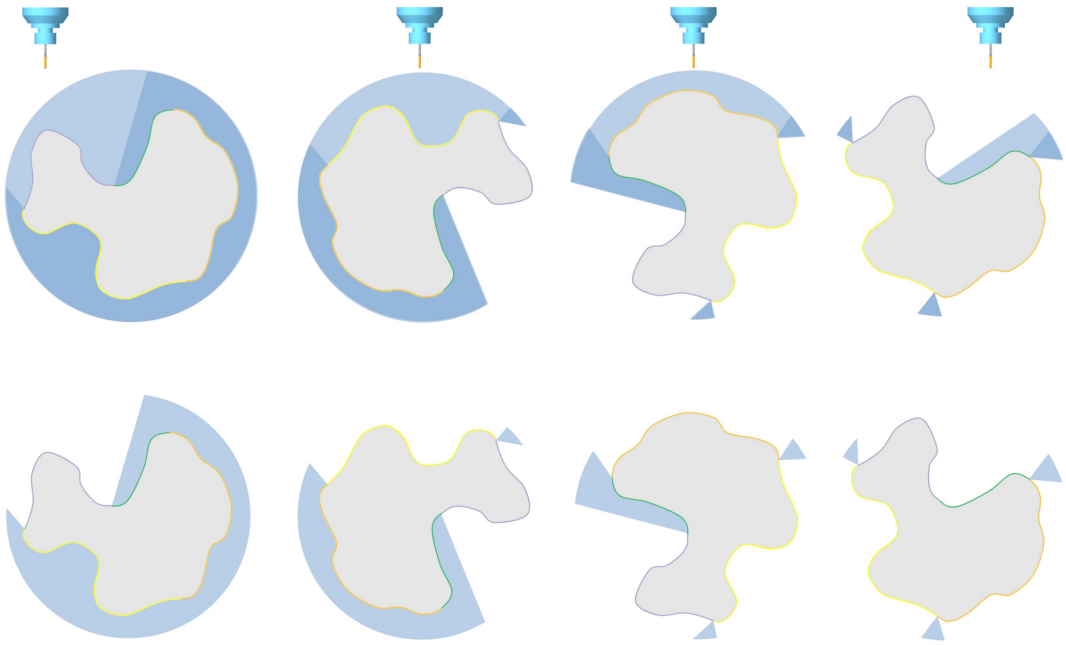


Figure A.13: **Set of meshes generated by our method.** On top, the stocks (with the part that will be milled in a lighter color), and on bottom the resulting shape. At each step, the resulting shape will be rotated and used as stock for the next step. The ordering of the charts is computed in order to minimize the area fabricated from a wrong direction.

Finally, we generate the stock and target meshes for the top and bottom regions. The process is similar, but we remove the volume around to easily remove the waste material at the end of the manufacturing process. Indeed, the remaining stock portions in the final target shape can be easily detached at the end of the fabrication process, saving manufacturing time. The user can choose to process first either the top or bottom region. We generate a mold for the first one to secure and align the model to the machine during the fabrication of the second region. Figure A.14 shows all the meshes generated for the Batman model.

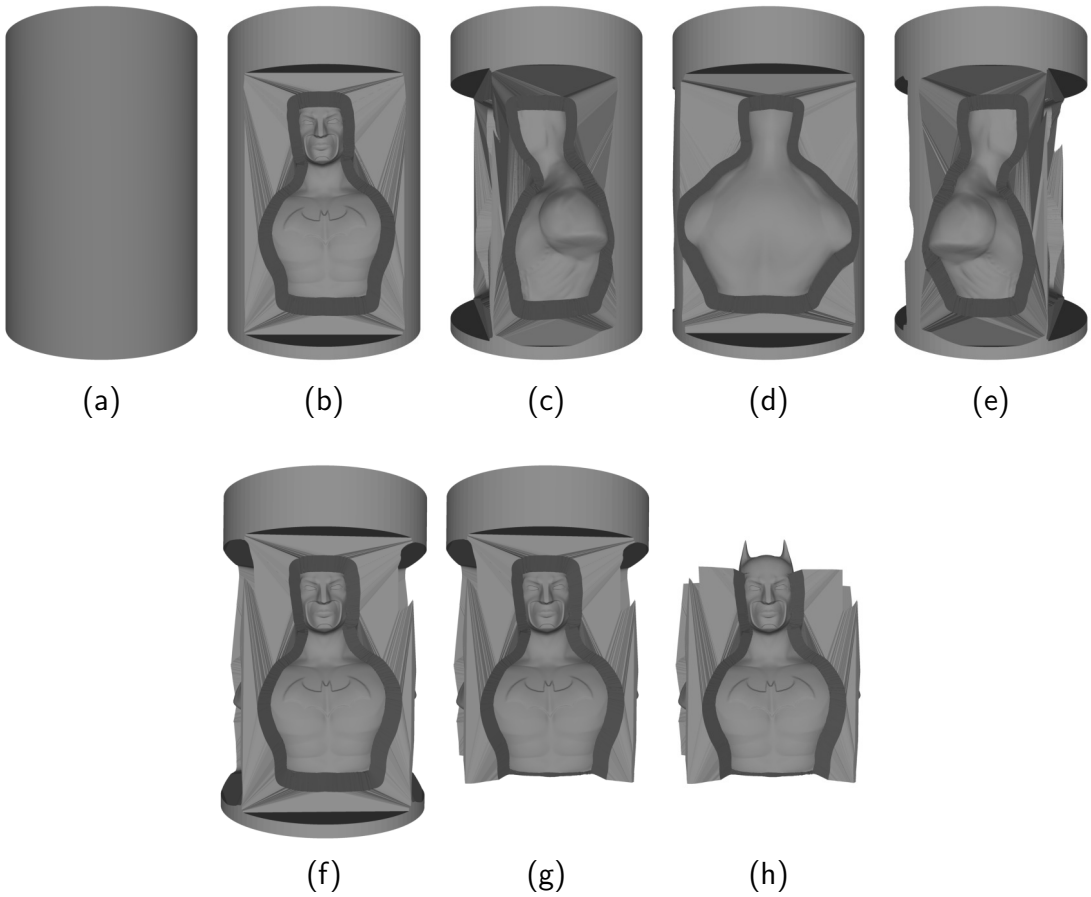


Figure A.14: **Fabrication sequence.** The first row shows the initial stock (a) and target shapes for side region (b, c, d, e). Each target shape is rotated properly and used as stock for the next step. The second row shows the resulting mesh after 4-axis fabrication and the target shapes for the bottom (g) and top (h) regions.

A.4 Implementation and results

Our approach has been implemented in C++, using cg3lib [MN21] and VCG [CNR13] for geometry processing functions, Eigen [GJ+14] for linear algebra routines, libigl [JP+16] and CGAL [FP09] for mesh booleans. In this section, we present results obtained on a broad range of organic models.

A.4.1 Computational and machining setup

All the fabrication plans have been computed on a workstation with a 6-cores Intel i5-8600K processor clocked at 3.6 GHz and 16 GB of memory.

While our methods of general use, and can be applied to any 4-axis CNC machines, all the models presented here were fabricated using a CNC machine Stepcraft 2 840, with a working area (X, Y, Z) of 600 × 840 × 140 mm with an added 4-axis module and an HF spindle 500 W by Stepcraft. For the roughing phase, we employed a 3 mm flat cutter with two flutes, while for the finishing stage, we used a 1 mm ball cutter with two flutes. The machine imposes limits on the manufacturing size of the model. Our models must fit inside a cylinder of 70mm in height and diameter. To drive the machine, we use Fusion 360 by Autodesk for the tool-path generation.

A.4.2 Performance and quality evaluation

Our method has been evaluated on a variety of shapes (see Table A.1). Figure A.15 shows the four results that have been manufactured and Figure A.16 presents several results of the segmentation process. The proposed pipeline can successfully fabricate objects with genus zero or higher.

Tab. A.1 provides statistics about our experiments. For each model, we report the processing time. We split the timing into four intervals:

(i) computation of the saliency map to avoid placing seams on semantically relevant portions of the shape;

(ii) identification of the rotation axis;

(iii) segmentation of the shape to associate each triangle to a single map linked to a height-field;

(iv) partitioning the 4-axis process in a sequence of 3-axis milling.

All times are in seconds. As it is possible to observe, the segmentation is the task requiring the highest computational effort. In the last column, we list the number

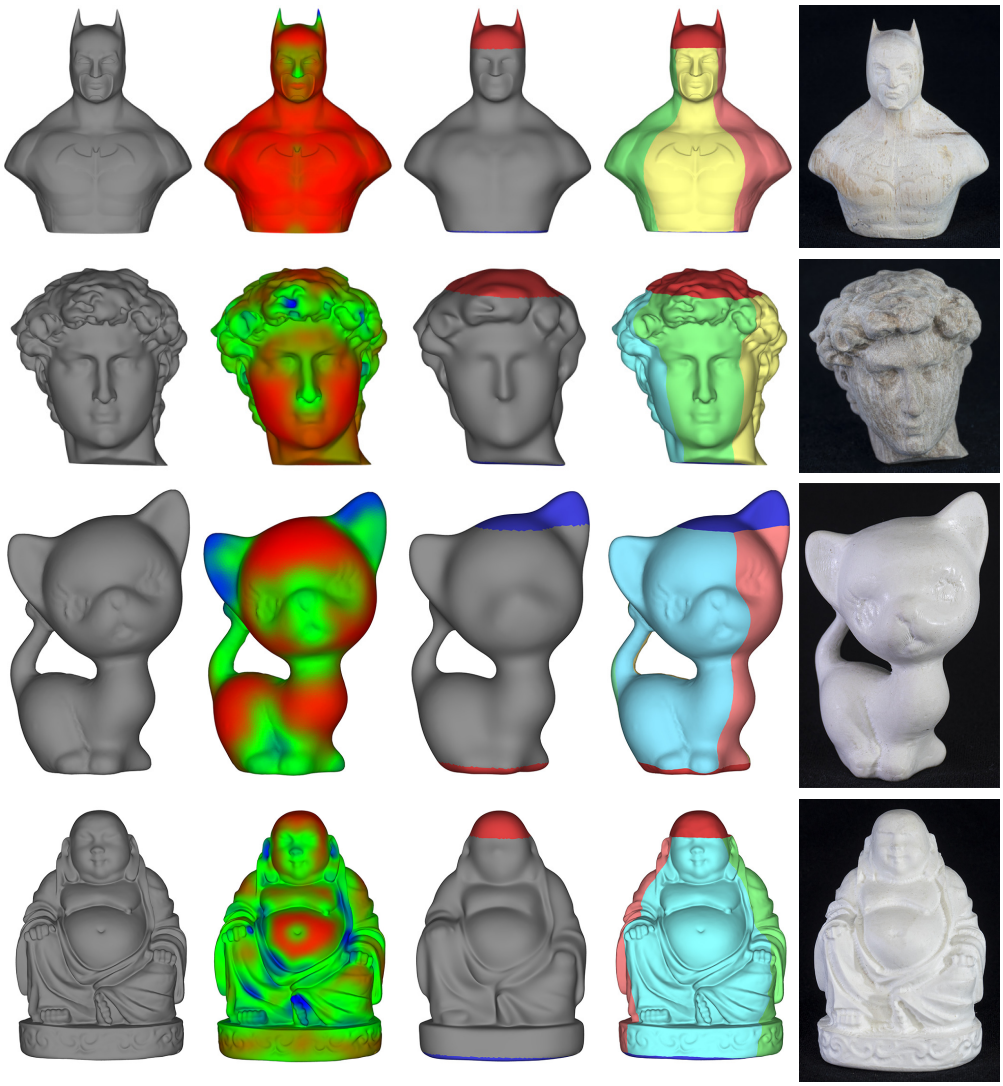


Figure A.15: **Fabricated models.** For the four fabricated models we show here (left to right): the initial shape; its saliency map; the two ends (in red and blue) displayed on the smoothed model; the decomposition of the model in height-fields; the fabricated object (in PVC for the kitten and the Buddha models, and wood for the batman and David models).

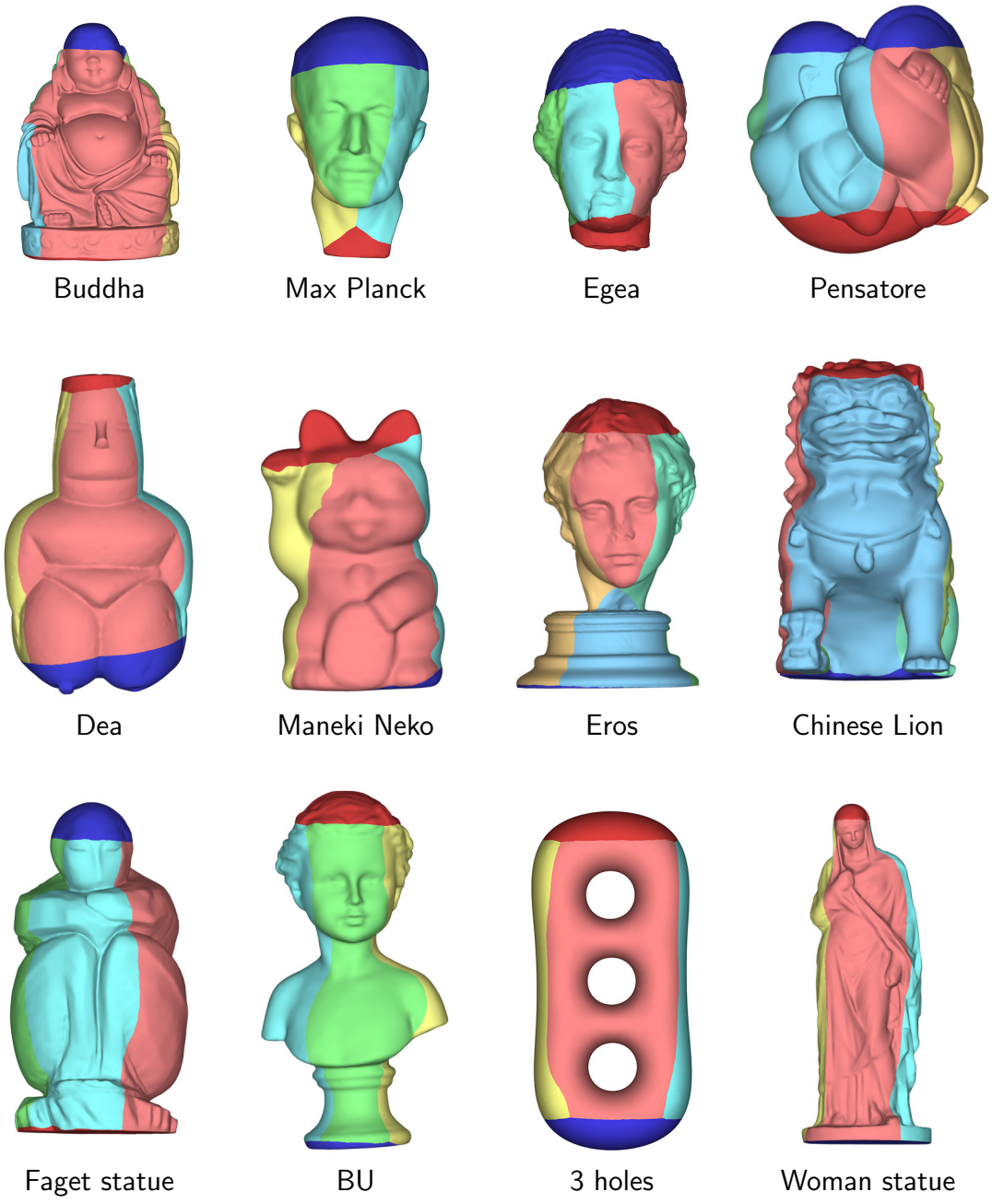


Figure A.16: **Several final segmentations.** In our experiments we used compactness term $c = 30$ and saliency factor $d = 25$.

of resulting charts. As we can see, the number of charts is maintained very low despite the shape complexity.

In our results, we applied the prefiltering described in Sec. A.3.1. To test their fidelity, we computed the distance between the input surfaces and the results after the reintroduction of high frequencies, using Metro [CRS98]. The resulting distance is less than 1×10^{-2} on average, with 2×10^{-5} variance and a maximum value of 1.7×10^{-2}. All these values are relative to the bounding box diagonal of the input models. We can thus conclude that, for the tested models, our fabrication plan produces meshes that are very close to the original mesh. It should also be noted that this accuracy is higher than what achievable in practice by our milling machine (see Sec. A.4.1).

We also computed the total surface area that would be manufactured from a direction that is different from the one planned in our segmentations, due to the chart ordering problem described in Sec. A.3.5. This surface is on average 0.7% w.r.t. the total surface of the input model, and 11 models have 0% of this surface, showing the effectiveness of the greedy sorting method. In any case, manufacturing a chart from a different milling direction from the planned one only reduces quality and does not cause other major issues on the final result. For example, we successfully manufactured the kitten model that had 2.9% of surface area manufactured from a different direction.

In Tab. A.2, we report some of the physical measures regarding the fabrication process of the objects we manufactured.

A.4.3 Limitations

Our pipeline can work on a reasonable range of shapes that can be found in industrial applications. However, because of the intrinsic limitations of 4-axis milling, it might fail when large portions of the model are not visible (due to occlusions) from any direction generated by the 4^{th} axis rotation. Figure A.17 shows a straightforward failure example.

Furthermore, our result is highly influenced by the choice of the rotation axis. Since the proposed method relies on a heuristic, and occlusions are not fully taken into account, we cannot guarantee the optimality of the result. For example, Figure A.18 and Figure A.19 show the segmentation of two complex shape that, due to occlusions, presents an excessive number of small charts. Additionally, removing them as explained in Sec. A.3.3 could result in too many faces that are not visible, leading to poor fidelity in the fabrication. The black faces in Figure A.18 represent the surface that is non-visible. In general, for this kind of shapes, single-block fabrication on 4-axis machines is very difficult if not impossible, and it would be ideal to split the model into multiple components that can be fabricated separately and assembled afterward.

Table A.1: **Model Statistics.** The first column shows the number of faces (NF), the next four columns (S, saliency; O optimal orientation; A face association; M fabrication sequence and mesh generation) list the timing of the computation, in the last column we report the number of obtained charts (NC). Times are in seconds.

Model	**NF**	**S**	**O**	**A**	**M**	**NC**
3 holes	43K	0.77	5.88	23.71	12.53	6
Batman	120K	11.49	34.63	279.29	31.33	6
BU	72K	4.88	19.14	118.77	21.99	6
Buddha	120K	8.54	36.82	151.73	30.04	6
Chinese Lion	60K	2.87	15.32	67.30	33.59	8
David	60K	2.66	14.70	72.79	19.93	6
Dea	40K	4.56	21.47	206.55	22.69	6
Egea	30K	1.32	7.30	22.75	12.05	6
Eros	120K	9.23	35.68	201.58	41.87	8
Faget statue	60K	2.58	15.24	85.90	16.72	6
Kitten	37K	1.40	8.48	66.68	17.06	7
Max Planck	54K	2.25	13.84	80.68	17.20	6
Moai	40K	1.28	9.00	60.00	13.77	6
Maneki neko	44K	1.66	10.07	28.47	14.84	6
Pensatore	60K	2.64	15.52	59.01	23.17	7
Woman statue	60K	3.67	13.83	73.43	16.67	6

Table A.2: **Model dimensions.** We report, for the four fabricated models, the size in millimeters of the target model and the initial stock.

Model	Model height	Stock diameter	Stock length
Batman	70	72	92
David	54	62	86
Buddah	70	72	86
Kitten	70	72	88

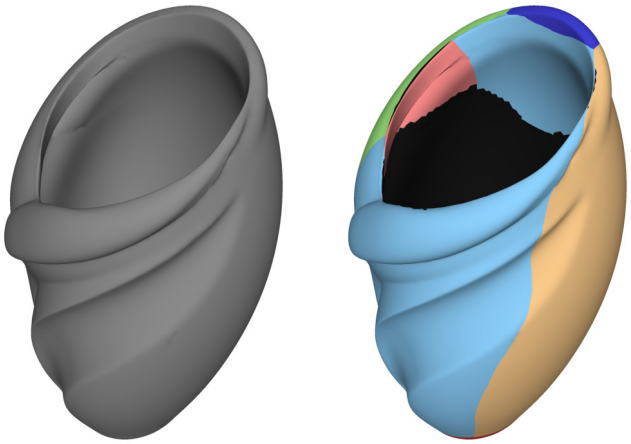


Figure A.17: **Failure case due to intrinsic limitations of 4-axis milling.** In the depicted cup-shaped surfaces, faces in black are non-visible from any possible direction.

Moreover, having a small bottom and top regions might results in practical issues during the fabrication. These regions must be wide enough to sustain the entire object weight during the entire manufacturing process. For example, the blue and red regions in Figure A.19 could be a problem for the fabrication of a tiny object. We can address this problem by simply adding a term in the optimization problem described in Sec. A.3.2, that penalizes the formation of small top and bottom areas. Alternatively, we can also discard the axis that generates top and bottom regions smaller than a certain threshold. However, we cannot guarantee to have a suitable solution for the general case.

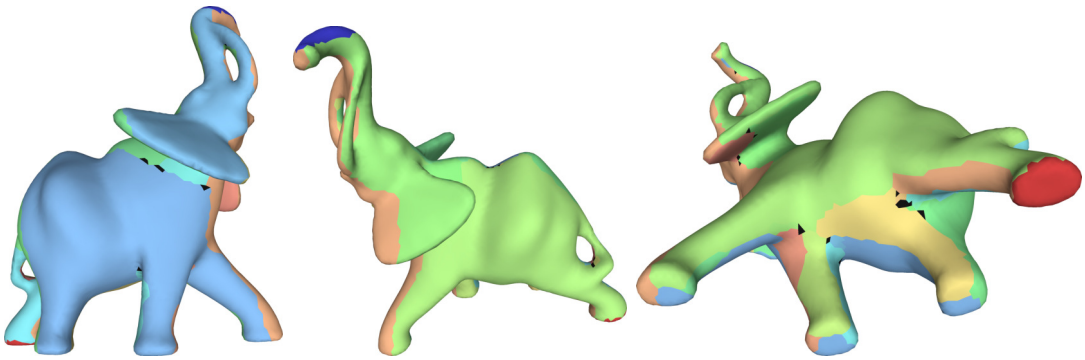


Figure A.18: **Limitations due to occlusions.** A large occluded area could lead to an excessive number of small charts and non-visible faces.

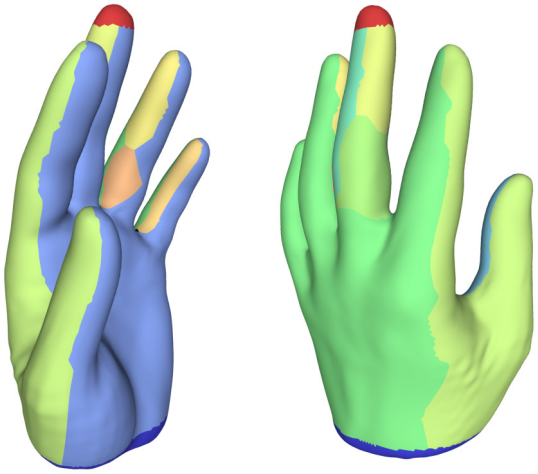


Figure A.19: **Limitations due to small top and bottom areas.** The regions must be wide enough to sustain the object during the fabrication.

A.4.4 Illustration of the fabrication process

In order to illustrate the practical usage of our pipeline in a CNC machining workshop, we illustrate the complete fabrication process of a model.

The inset on the right shows how the block is initially secured on the rotation tool (see also Figure A.2). We used PVC cylinders as stock for the kitten and the Buddha models, and wood for the batman and David models. The cylinder is mounted in the rotation axis between the step motor and the tail-stock.

We start milling the side regions of the model, first performing the roughing step using the 3mm flat cutter for each direction, and then performing a finishing step with a 1mm ball cutter. We show in Figure A.20 and Figure A.21 respectively the roughing process for the side regions of the kitten model.

We then remove the block and place it on the plane, and we perform the fabrication of the top, the mold, and the bottom of the shape. Each step is composed of a roughing and a finishing phase. The precise alignment of the model at sub-millimeter tolerance, a strict requirement of this phase, is facilitated by the creation of the mold. Figure A.22

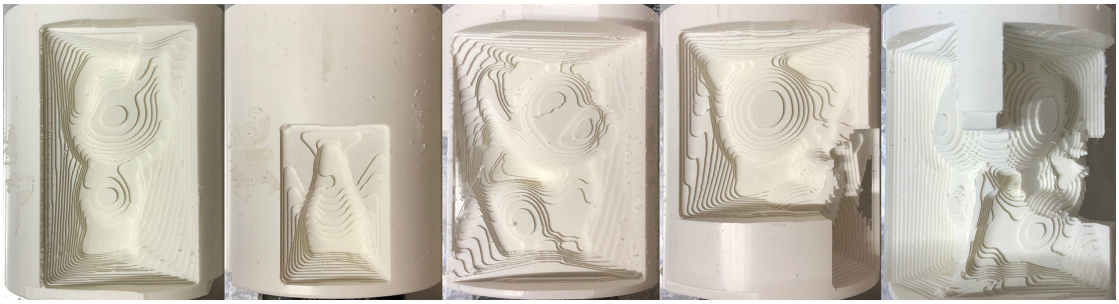


Figure A.20: **Roughing phase.** Side regions finished with a 3mm flat cutter.

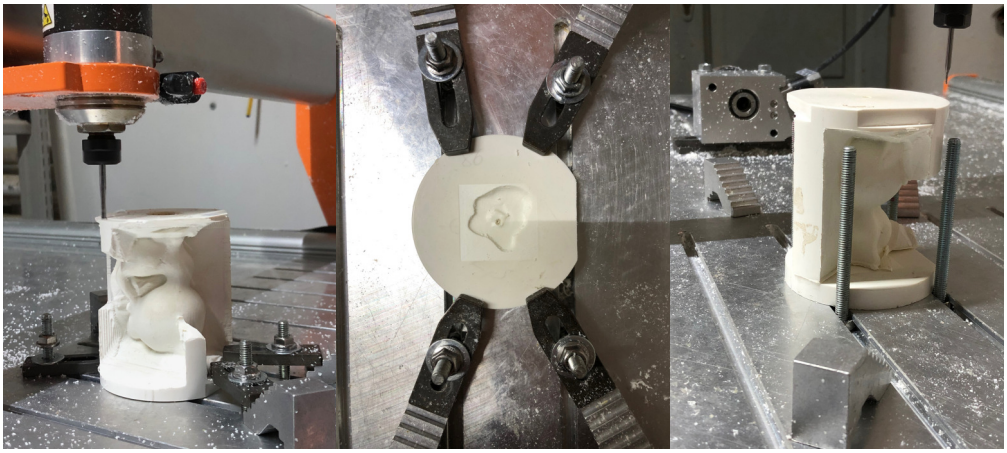
Figure A.21: **Finishing phase.** Side regions finished with a 1mm ball cutter.

shows these three fabrication steps and how each block is secured to the machine plane.

It's worth noticing that the model's small size (a few cm for the kitten) influences its quality since the cutter cannot be smaller than 1mm. The ratio between the cutter size and the shape size is an essential parameter of the fabrication step. In Figure A.23a, we show that, at the end of the milling, a portion of the stock initially surrounding the shape may still remain attached to it, due to limitations in machine accuracy. The connection is very thin, and we can easily remove it (Figure A.23b). To bring the model to a smooth shape in those cases, we can use a rasp or sandpaper to remove the material along the seams (Figure A.23c). Despite this disadvantage, we can obtain excellent results.

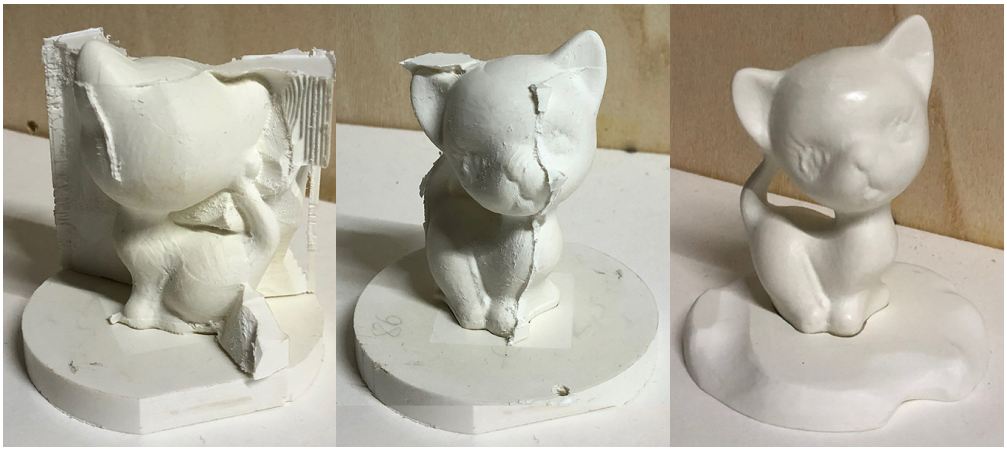
A.5 Conclusions

We introduced a novel automatic method for the fabrication of non-trivial shapes using 4-axis milling machines. Our approach enables the fabrication of highly detailed models in a single-block without visible seams, in contrast to previous methods based on decomposition and reassembly. The introduction of saliency information guarantees the fabrication of certain important areas of the model in a



(a) (b) (c)

Figure A.22: **Fabrication phases.** Manufacturing the first of the two ends of the model (a), and the second end of the model (c) using its mold (b).



(a) (b) (c)

Figure A.23: **Fabricated model before and after the final sanding pass.** In (a) the model has still part of the stock that, initially, was surrounding it; in (b) only the portion that is to be sanded is present. in (c), the final polished model, using the mold as a base

single pass.

The method has been designed to be easily integrable in current pipelines. First of all, by letting the user control the accuracy and fidelity of the manufactured model through user-controllable detail filtering thresholds, we enable the possibility to sacrifice small details when necessary in order to obtain effective fabrication plans. Moreover, the method has been designed to effectively fit within current industrial fabrication settings with minimal upgrades.

With respect to the traditional methods relying on 3-axis milling machines, we exploit an additional rotation axis, which has been specifically designed in order to subdivide the entire process into a sequence of 3-axis carving passes. This leads our approach to be immediately available to users that possess a 3-axis machine. Indeed, the only requirement is the installation of a rotation shaft, which is a common additional accessory for CNC machines. Furthermore, they can use, without any restriction, any available 3-axis driver for path planning. We thus expect that our proposed technique can immediately result in practical industrial applications, as it significantly extends the range of shapes that can be automatically fabricated on common low-cost machines.

Despite the significantly expanded class of fabricable models, not all meshes can be produced using our single-pass approach. This limitation is generally due to the fact that four DOFs may be not enough for manufacturing such shapes in a single block. Extending to another DOF and using a 5-axis machine is not as straightforward as passing from 3 to 4-axis machines. One possible solution to overcome these limitations could be to decompose the input shape into multiple cylindrical components that can be fabricated independently using our method and assembled afterward. With this approach, we would aim at producing free-form shapes, with significantly fewer components than common height-field decomposition methods.

Bibliography

[ACA+19] Chrystiano Araújo, Daniela Cabiddu, Marco Attene, Marco Livesu, Nicholas Vining, and Alla Sheffer. Surface2volume: Surface segmentation conforming assemblable volumetric partition. *ACM Transaction on Graphics*, 38(4), 2019. 91

[ACP+14] Giuseppe Alemanno, Paolo Cignoni, Nico Pietroni, Federico Ponchio, and Roberto Scopigno. Interlocking pieces for printing tangible cultural heritage replicas. In *Proc. GCH*, pages 145–154. Eurographics Association, 2014. 90

[AMG+18] Thomas Alderighi, Luigi Malomo, Daniela Giorgi, Nico Pietroni, Bernd Bickel, and Paolo Cignoni. Metamolds: computational design of silicone molds. *ACM Transactions on Graphics*, 37(4):136:1–136:??, August 2018. 90

[AMG+19] Thomas Alderighi, Luigi Malomo, Daniela Giorgi, Bernd Bickel, Paolo Cignoni, and Nico Pietroni. Volume-aware design of composite molds. *ACM Trans. Graph*, 38(4):110:1–110:12, 2019. 91

[ARG+16] Quentin Avril, S. Ribet, Donya Ghafourzadeh, Olivier Dionne, Srinivasan Ramachandran, M. Lasa, Sahel Fallahdoust, and E. Paquette. Animation setup transfer for 3d characters. *Computer Graphics Forum*, 35, 2016. 8

[ATW+17] Baptiste Angles, Marco Tarini, Brian Wyvill, Loïc Barthe, and Andrea Tagliasacchi. Sketch-based implicit blending. *ACM Trans. Graph.*, 36(6), November 2017. 7

[Aut] Autodesk, INC. Maya. 75

[Aut18] Autodesk. Mudbox, 2018. 4

[BBCW13] Adrien Bernhardt, Loïc Barthe, Marie-Paule Cani, and Brian Wyvill. Implicit blending revisited. *Computer Graphics Forum*, 29:367–376, 05 2013. 7

[BDS+18] Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Trans. Graph.*, 37(4), July 2018. 7, 55

[BGOS06] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1):19–38, January 2006. 7

[BK05] Stephan Bischoff and Leif Kobbelt. Structure preserving CAD model repair. *Comput. Graph. Forum*, 24(3):527–536, 2005. 7

[Bla14] Sue Blackman. *Rigging with Mixamo*, pages 565–573. Apress, Berkeley, CA, 2014. 42, 69, 83

[BLK11] David Bommes, Timm Lempfer, and Leif Kobbelt. Global structure optimization of quadrilateral meshes. *Comput. Graph. Forum*, 30(2):375–384, 2011. 5, 17

[BLP+13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Cláudio T. Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Comput. Graph. Forum*, 32(6):51–76, 2013. 5

[BMBZ02] Henning Biermann, Ioana M. Martin, Fausto Bernardini, and Denis Zorin. Cut-and-paste editing of multiresolution surfaces. *ACM Trans. Graph.*, 21(3):312–321, 2002. 5

[BMPB08] A. Brodersen, K. Museth, S. Porumbescu, and B. Budge. Geometric texturing using level sets. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):277–288, 2008. 8

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72–es, July 2007. 8

[BTST12] Gaurav Bharaj, Thorsten Thormählen, Hans-Peter Seidel, and Christian Theobalt. Automatically rigging multi-component characters. *Comput. Graph. Forum*, 31(2pt3):755–764, May 2012. 8

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001. 97

[BWSS12] Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. Design-driven quadrangulation of closed 3d curves. *ACM Trans. Graph.*, 31(6):178:1–178:11, 2012. 6

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77, 2009. 12, 19

[Cam17a] Marcel Campen. Partitioning surfaces into quadrilateral patches: A survey. *Comput. Graph. Forum*, 36(8):567–588, 2017. 5

[Cam17b] Marcel Campen. Tiling the bunny: Quad layouts for efficient 3d geometry representation. *IEEE Computer Graphics and Applications*, 37(3):88–95, 2017. 5

[CBK12] Marcel Campen, David Bommes, and Leif Kobbelt. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph.*, 31(4):110:1–110:11, 2012. 5, 15, 20, 22

[CCC+08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference 2008, Salerno, Italy, 2008*, pages 129–136, 2008. 20

[CK10] Marcel Campen and Leif Kobbelt. Exact and robust (self-)intersections for polygonal meshes. *Comput. Graph. Forum*, 29(2):397–406, 2010. 7

[CK14] Marcel Campen and Leif Kobbelt. Dual strip weaving: interactive design of quad layouts using elastica strips. *ACM Trans. Graph.*, 33(6):183:1–183:10, 2014. 4

[CKS00] R. Christian, L. Kobbelt, and H. Seidel. Extraction of feature lines on triangulated surfaces using morphological operators. 2000. 53, 60

[CLSA20] Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. Fast and robust mesh arrangements using floating-point arithmetic. *ACM Trans. Graph.*, 39(6), November 2020. 7

[CNR13] CNR. The visualization and computer graphics library, 2013. http://vcg.isti.cnr.it/vcglib/. 30, 75, 108

[CRS98] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. In *Computer graphics forum*, volume 17, pages 167–174. Wiley Online Library, 1998. 111

[CT11] F. Calakli and Gabriel Taubin. Ssd: Smooth signed distance surface reconstruction. *Computer Graphics Forum*, 30:1993 – 2002, 11 2011. 7

[CZL+15] Xuelin Chen, Hao Zhang, Jinjie Lin, Ruizhen Hu, Lin Lu, Qixing Huang, Bedrich Benes, Daniel Cohen-Or, and Baoquan Chen. Dapper: decompose-and-pack for 3D printing. *ACM Transactions on Graphics*, 34(6):213:1–213:??, November 2015. 90

[DdL13] Olivier Dionne and Martin de Lasa. Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, page 173–180, New York, NY, USA, 2013. Association for Computing Machinery. 8

[DG95] Mathieu Desbrun and Marie-Paule Gascuel. Animating soft substances with implicit surfaces. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, page 287–290, New York, NY, USA, 1995. Association for Computing Machinery. 7

[DLG+13] Ali-Hamadi Dicko, Tiantian Liu, Benjamin Gilles, Ladislav Kavan, Francois Faure, Olivier Palombi, and Marie-Paule Cani. Anatomy transfer. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 32(6), 2013. 8

[DSSC08] Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. Quadrilateral mesh simplification. *ACM Trans. Graph.*, 27(5):148:1–148:9, 2008. 27

[DVPS14] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Designing N-polyvector fields with complex polynomials. *Comput. Graph. Forum*, 33(5):1–11, 2014. 15, 20

[dWv09] Erwin de Groot, Brian Wyvill, and Huub van de Wetering. Locally restricted blending of blobtrees. *Computers & Graphics*, 33(6):690–697, 2009. 7

[DYY16] Noah Duncan, Lap-Fai Yu, and Sai-Kit Yeung. Interchangeable components for hands-on assembly based modelling. *ACM Trans. Graph.*, 35(6), November 2016. 62

[EB10] Manolya Eyiyurekli and David Breen. Interactive free-form level-set surface-editing operators. *Computers & Graphics*, 34(5):621–638, 2010. CAD/GRAPHICS 2009 Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference Vision, Modeling & Visualization. 8

[EGKT08] David Eppstein, Michael T. Goodrich, Ethan Kim, and Rasmus Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. *Comput. Graph. Forum*, 27(5):1477–1486, 2008. 13, 17

[FAG+20] Irene Filoscia, Thomas Alderighi, Daniela Giorgi, Luigi Malomo, Marco Callieri, and Paolo Cignoni. Optimizing object decomposition to reduce visual artifacts in 3d printing. *Computer Graphics Forum*, 39(2), May 2020. 90

[FCM+18] Filippo A Fanni, Gianmarco Cherchi, Alessandro Muntoni, Alessandro Tola, and Riccardo Scateni. Fabrication oriented shape decomposition using polycube mapping. *Computers & Graphics*, 77:183–193, 2018. 91

[FKS+04] Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David P. Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, 2004. 5

[FP05] Sarah Frisken and Ronald Perry. Designing with distance fields. volume 2005, pages 58–59, 01 2005. 7

[FP09] Andreas Fabri and Sylvain Pion. Cgal: The computational geometry algorithms library. In *Proc. ACM SIGSPATIAL conference on advances in geographic information systems*, pages 538–539, 2009. 30, 75, 108

[GBC+13] Olivier Gourmel, Loic Barthe, Marie-Paule Cani, Brian Wyvill, Adrien Bernhardt, Mathias Paulin, and Herbert Grasberger. A gradient-based implicit blend. *ACM Trans. Graph.*, 32(2), April 2013. 7

[GJ+14] Gael Guennebaud, Benoit Jacob, et al. Eigen: a c++ linear algebra library, 2014. http://eigen.tuxfamily.org. 30, 75, 108

[GO18] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018. 30, 75

[HLZCO14] Ruizhen Hu, Honghua Li, Hao Zhang, and Daniel Cohen-Or. Approximate pyramidal shape decomposition. *ACM Transactions on Graphics*, 33(6):213:1–213:??, November 2014. 90

[HMA15] Philipp Herholz, Wojciech Matusik, and Marc Alexa. Approximating free-form geometry with height fields for manufacturing. *Comput. Graph. Forum*, 34(2):239–251, 2015. 90

[HWCO+13] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4), July 2013. 8

[JBPS11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4), July 2011. 8

[JDKL14] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*, 2014. 3, 8, 40, 47

[JKS13] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.*, 32(4):33:1–33:12, 2013. 7

[JLCW06] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy mesh cutting. *Comput. Graph. Forum*, 25(3):283–291, 2006. 29

[JP+16] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2016. https://libigl.github.io. 30, 75, 108

[JTPS15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6):189:1–189:15, 2015. 12, 13, 14

[JWS12] Alec Jacobson, Tino Weinkauf, and Olga Sorkine. Smooth shape-aware functions with controlled extrema. *Comput. Graph. Forum*, 31(5):1577–1586, August 2012. 8

[KCvO07] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, page 39–46, New York, NY, USA, 2007. Association for Computing Machinery. 41, 47, 69, 75

[KDB16] Dan Koschier, Crispin Deul, and Jan Bender. Hierarchical hp-adaptive signed distance fields. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, page 189–198, Goslar, DEU, 2016. Eurographics Association. 7

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover - surface parameterization using branched coverings. *Comput. Graph. Forum*, 26(3):375–384, 2007. 22

[KS12] Ladislav Kavan and Olga Sorkine. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph.*, 31(6), November 2012. 8

[LBRM12] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: partitioning models into 3d-printable parts. *ACM Trans. Graph*, 31(6):129:1–129:9, 2012. 90

[LL14] Zohar Levi and David Levin. Shape deformation via interior rbf. *IEEE Transactions on Visualization and Computer Graphics*, 20(7):1062–1075, July 2014. 8

[LPRM02] Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002. 28

[LVJ05] Chang Ha Lee, Amitabh Varshney, and David W. Jacobs. Mesh saliency. *ACM Trans. Graph.*, 24(3):659–666, July 2005. 98

[MASS15] Nathan Mitchell, Mridul Aanjaneya, Rajsekhar Setaluri, and Eftychios Sifakis. Non-manifold level sets: A multivalued implicit surface representation with applications to self-collision processing. *ACM Trans. Graph.*, 34(6), October 2015. 7

[MBWB02] Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. Level set surface editing operators. *ACM Trans. Graph.*, 21(3):330–338, July 2002. 8

[MLJ+13] Ken Museth, Jeff Lait, John Johanson, Jeff Budsberg, Ron Henderson, Mihai Alden, Peter Cucka, David Hill, and Andrew Pearce. Openvdb: An open-source data structure and toolkit for high-resolution volumes. In *ACM SIGGRAPH 2013 Courses*, SIGGRAPH '13, New York, NY, USA, 2013. Association for Computing Machinery. 7, 55, 75

[MLS+18] Alessandro Muntoni, Marco Livesu, Riccardo Scateni, Alla Sheffer, and Daniele Panozzo. Axis-aligned height-field block decomposition of 3d shapes. *ACM Trans. Graph*, 37(5):169:1–169:15, 2018. 91, 93, 101

[MN21] Alessandro Muntoni and Stefano Nuvoli. Cg3lib: A c++ geometry processing library, January 2021. 30, 75, 108

[Mor94] Carroll Morgan. *Programming from Specifications (2Nd Ed.)*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994. 18

[MPBC16] Luigi Malomo, Nico Pietroni, Bernd Bickel, and Paolo Cignoni. Flexmolds: automatic design of flexible shells for molding. *ACM Trans. Graph*, 35(6):223:1–223:12, 2016. 90

[MPP+13] Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. Animation-aware quadrangulation. *Comput. Graph. Forum*, 32(5):167–175, 2013. 5

[MPWC13] Niloy Mitra, Mark Pauly, Michael Wand, and Duygu Ceylan. Symmetry in 3d geometry: Extraction and applications. *Computer Graphics Forum*, 32, 09 2013. 83

[MPZ14] Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):135:1–135:14, 2014. 12, 22

[MTP+15] Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine-Hornung, Enrico Puppo, and Paolo Cignoni. Data-driven interactive quadrangulation. *ACM Trans. Graph.*, 34(4):65:1–65:10, 2015. 4, 6

[Mus11] Ken Museth. Db+grid: a novel dynamic blocked grid for sparse high-resolution volumes and level sets. page 51, 08 2011. 7

[NHE+19] Stefano Nuvoli, Alex Hernandez, Claudio Esperança, Riccardo Scateni, Paolo Cignoni, and Nico Pietroni. Quadmixer: Layout preserving blending of quadrilateral meshes, nov 2019. 4

[Nie04] Gregory M. Nielson. Dual marching cubes. In *Proceedings of the Conference on Visualization '04*, VIS '04, page 489–496, USA, 2004. IEEE Computer Society. 55

[NKI+18] Tuan D. Ngo, Alireza Kashani, Gabriele Imbalzano, Kate T.Q. Nguyen, and David Hui. Additive manufacturing (3d printing): A review of materials, methods, applications and challenges. *Composites Part B: Engineering*, 143:172 – 196, 2018. 87, 88

[NSY09] Ahmad H. Nasri, Malcolm A. Sabin, and Zahraa Yasseen. Filling N-sided regions by quad meshes for subdivision surfaces. *Comput. Graph. Forum*, 28(6):1644–1658, 2009. 6

[PBJW14] Chi-Han Peng, Michael Barton, Caigui Jiang, and Peter Wonka. Exploring quadrangulations. *ACM Trans. Graph.*, 33(1):12:1–12:13, 2014. 6, 23

[PCK10] Darko Pavic, Marcel Campen, and Leif Kobbelt. Hybrid booleans. *Comput. Graph. Forum*, 29(1):75–87, 2010. 7

[PCYQ18] Junjun Pan, Lijuan Chen, Yuhan Yang, and Hong Qin. Automatic skinning and weight retargeting of articulated characters using extended position-based dynamics. *Vis. Comput.*, 34(10):1285–1297, October 2018. 8

[Pil17] Pilgway. 3dcoat, 2017. 4

[Pix99] Pixologic. Zbrush, 1999. 4

[PPM+16] Nico Pietroni, Enrico Puppo, Giorgio Marcias, Roberto Scopigno, and Paolo Cignoni. Tracing field-coherent quad layouts. *Comput. Graph. Forum*, 35(7):485–496, 2016. 12, 22

[PTP+15] Nico Pietroni, Davide Tonelli, Enrico Puppo, Maurizio Froli, Roberto Scopigno, and Paolo Cignoni. Statics aware grid shells. *Comput. Graph. Forum*, 34(2):627–641, 2015. 31, 34, 37

[PZKW11] Chi-Han Peng, Eugene Zhang, Yoshihiro Kobayashi, and Peter Wonka. Connectivity editing for quadrilateral meshes. *ACM Trans. Graph.*, 30(6):141:1–141:12, 2011. 6

[SBSC06] Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. Snappaste: an interactive technique for easy mesh composition. *The Visual Computer*, 22(9-11):835–844, 2006. 5

[SEPC16] Asla Medeiros e Sá, Karina Rodriguez Echavarria, Nico Pietroni, and Paolo Cignoni. State of The Art on Functional Fabrication. In A. Medeiros e Sa, N. Pietroni, and K. Rodriguez Echavarria, editors, *Eurographics Workshop on Graphics for Digital Fabrication*. The Eurographics Association, 2016. 88, 90

[SFP12] Mathieu Sanchez, Oleg Fryazinov, and Alexander Pasko. Efficient evaluation of continuous signed distance to a polygonal mesh. In *Proceedings of the 28th Spring Conference on Computer Graphics*, SCCG '12, page 101–108, New York, NY, USA, 2012. Association for Computing Machinery. 7

[SJP06] Richard Swinbank and R James Purser. Fibonacci grids: A novel approach to global modelling. *Quarterly Journal of the Royal Meteorological Society*, 132(619):1769–1793, 2006. 96

[Sor06] Olga Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006. 101

[SPS04] Benjamin Schmitt, Alexander Pasko, and Christophe Schlick. Constructive sculpting of heterogeneous volumetric objects using trivariate b-splines. *The Visual Computer*, 20:130–148, 05 2004. 7

[SS10] Ryan Schmidt and Karan Singh. Meshmixer: An interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, SIGGRAPH '10, pages 6:1–6:1, New York, NY, USA, 2010. ACM. 5

[SWZ04] Scott Schaefer, Joe D. Warren, and Denis Zorin. Lofting curve networks using subdivision surfaces. In *Second Eurographics Symposium on Geometry Processing, Nice, France, July 8-10, 2004*, pages 103–114, 2004. 6

[SY07] S. Schaefer and C. Yuksel. Example-based skeleton extraction. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 153–162, Goslar, DEU, 2007. Eurographics Association. 8

[Tan14] Tran Duc Tang. Algorithms for collision detection and avoidance for five-axis nc machining: A state of the art review. *Computer-Aided Design*, 51:1–17, 2014. 88

[TAOZ12] Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. Mean curvature skeletons. *Comput. Graph. Forum*, 31(5):1735–1744, August 2012. 8

[Tau95] G. Taubin. Curve and surface smoothing without shrinkage. In *Proceedings of the Fifth International Conference on Computer Vision*, ICCV '95, page 852, USA, 1995. IEEE Computer Society. 93

[TF18] Yizhi Tang and Jieqing Feng. Multi-scale surface reconstruction based on a curvature-adaptive signed distance field. *Computers & Graphics*, 70:28 – 38, 2018. CAD/Graphics 2017. 7

[TIN+11] Julien Tierny, Joel Daniels II, Luis Gustavo Nonato, Valerio Pascucci, and Cláudio T. Silva. Inspired quadrangulation. *Computer-Aided Design*, 43(11):1516–1526, 2011. 6

[TPC+10] Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. Practical quad mesh simplification. *Comput. Graph. Forum*, 29(2):407–418, 2010. 12

[TPP+11] Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni. Simple quad domains for field aligned mesh parametrization. *ACM Trans. Graph.*, 30(6):142:1–142:12, 2011. 5, 17

[TPS14] Kenshi Takayama, Daniele Panozzo, and Olga Sorkine-Hornung. Pattern-based quadrangulation for N-sided patches. *Comput. Graph. Forum*, 33(5):177–184, 2014. 4, 6, 15, 19, 23, 24, 28

[TPSS13] Kenshi Takayama, Daniele Panozzo, Alexander Sorkine-Hornung, and Olga Sorkine-Hornung. Sketch-based generation and editing of quad meshes. *ACM Trans. Graph.*, 32(4):97:1–97:8, 2013. 6

[VCD+17] Amir Vaxman, Marcel Campen, Olga Diamanti, David Bommes, Klaus Hildebrandt, Mirela Ben-Chen, and Daniele Panozzo. Directional field synthesis, design, and processing. In *SIGGRAPH '17 Courses*, pages 12:1–12:30, 2017. 20

[Vis18] The Foundry Visionmongers. Modo 12.1, 2018. 13, 15

[WGG99] Brian Wyvill, Andrew Guy, and Eric Galin. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. *Comput. Graph. Forum*, 18:149–158, 06 1999. 7

[WL08] Yu-Shuen Wang and Tong-Yee Lee. Curve-skeleton extraction using iterative least squares optimization. *IEEE Transactions on Visualization and Computer Graphics*, 14:926–936, 2008. 8

[WP02] Lawson Wade and Richard E. Parent. Automated generation of control skeletons for use in animation. *Vis. Comput.*, 18(2):97–110, April 2002. 8

[XB16] Hongyi Xu and Jernej Barbic. 6-dof haptic rendering using continuous collision detection between points and signed distance fields. *IEEE Transactions on Haptics*, PP:1–1, 09 2016. 7

[YNB+13] Zahraa Yasseen, Ahmad H. Nasri, W. Boukaram, Pascal Volino, and Nadia Magnenat-Thalmann. Sketch-based garment design with quad meshes. *Computer-Aided Design*, 45(2):562–567, 2013. 6

[ZCZ+18] Jiaran Zhou, Marcel Campen, Denis Zorin, Changhe Tu, and Cláudio T. Silva. Quadrangulation of non-rigid objects using deformation metrics. *Computer Aided Geometric Design*, 62:3–15, 2018. 5

[ZGZJ16] Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. Mesh arrangements for solid geometry. *ACM Trans. Graph.*, 35(4):39:1–39:15, 2016. 7, 11, 13, 17, 34, 39, 106

[ZWC+10] Juyong Zhang, Chunlin Wu, Jianfei Cai, Jianmin Zheng, and Xue-Cheng Tai. Mesh snapping: Robust interactive mesh cutting using fast geodesic curvature flow. *Comput. Graph. Forum*, 29(2):517–526, 2010. 5

[ZZX+18] Haisen Zhao, Hao (Richard) Zhang, Shiqing Xin, Yuanmin Deng, Changhe Tu, Wenping Wang, Daniel Cohen-Or, and Baoquan Chen. Dscarver: decompose-and-spiral-carve for subtractive manufacturing. *ACM Trans. Graph*, 37(4):137:1–137:14, 2018. 91, 97

Acknowledgments

This thesis is dedicated to all the ones I loved who cannot be with me anymore. I wish you were here and I would give everything to be able to share this moment with you. I miss you.

Thanks to my Ph.D. advisor Riccardo, for his continuous and brilliant support during my Ph.D. studies. My most sincere gratitude for your patience and immense knowledge. Thanks for everything you do for your students, I have never seen such a good teacher in my entire life.

Thanks to my Ph.D. advisor Nico, for his invaluable guidance throughout my Ph.D. studies. My results would be not possible without your knowledge, your insightful advice, and your encouragement. Thanks for your exemplary welcome in Sydney, you and Paola made me feel at home. Thank you, Paola!

Thanks to Paolo, Marco, Fabrizio, Alessandro M. and Alessandro T. for the contribution in the works presented in this thesis.

The order of the following acknowledgments is completely random. I swear I applied the shuffle algorithm in the C++ standard library std::shuffle (citation needed). All of you are extremely important to me, without your support and your belief in me I would not have had a chance to accomplish this goal. Any resemblance to real events and/or to real persons, living or dead, is not purely coincidental at all.

Thanks to Roberto, Isabella, Riccardo, Beatrice, and Leonardo, for being the cheerful, brilliant, and surprising kids you are. Grown-ups never understand anything by themselves, and it is tiresome for children to be always and forever explaining things to them (cit.). You're the hope for the new generations.

Thanks to Carlotta, for brightening my days more than the sunlight. You just unexpectedly came into my life, and you already found your way into my heart. Thanks for constantly pushing me to improve myself and for our insightful and astonishing conversations about life. Your deep sincerity, your infinite strength, your thoughtful consideration, your sweet smile, and, especially, your immense heart, are immeasurably precious to me.

Thanks to Roberta, for showing me what it means to be a pure person who cares about people. Your goodness to me and to other people has always inspired

me, you proved to me that sensitivity is a value.

Thanks to Massimo, for always reminding me of my talents and my strengths. I owe you, I am who I am today thanks to you: you showed me how deep a friendship can be and you taught me how to love and value myself and others. Even if we are geographically as farthest as we could be, a +10 time-zone is not enough at all to make me love you less. You still are and you will always be an important point of reference in my life. Thanks to Sarah, for being a joy in Massimo's life and for bringing to life his most beautiful dream Noah.

Thanks to Mom, for her perseverance and all the sacrifices to give us everything we need or wish. Even if it has been really difficult, you successfully gave me the opportunities to accomplish my goals. For this, I will always be deeply grateful. Often, we take everything for granted and I am sorry if sometimes I had.

Thanks to Valentina, for proving to me, without a shadow of a doubt, that a deep friendship between men and women can exist. In your thesis acknowledgments, you wrote you and I are like cats and dogs: that's totally nonsense! On the opposite, we are like bees and flowers: we are totally different, but I need you, you need me and we benefit from each other. In case you are wondering, you are the bee and I am the flower.

Thanks to Bruno, for being an awesome neighbour and a great friend, the first friend I have ever had. Thank you for all the days spent chilling and playing videogames. Thank you for your hospitality, always flawless. I think I owe you half of your electricity bills for the past 15 years!

Thanks to Simona and Giacomo, for being the sister and the brother-in-law that everyone would like to have. Thanks to Simona, for being such a good sister and being always present, even if we are far. Thanks to Giacomo for his friendship and all the time spent together when we were young. If I cultivated my passion for Computer Science and I accomplished these goals, it is also because of you.

Thanks to Alberto, for being a great flatmate and, especially, a great friend. I admire you for being able to bear with me during our long cohabitation, even if several times you tried to kill me in the most ingenious and various ways.

Thanks to Fabio, Alessandro T., Alessandro M., and Martina for welcoming me to BatCaverna and for all the evenings spent together. I was totally a stranger in the office, and I'm aware I am not an easy person sometimes. You welcomed me like a friend, and for this, I will always thankful to you! My time in Cagliari would not have been the same without you.

Thanks to Giuseppe B. and Anna, for being the best friends I could have in Cagliari. Thank you for all the fun we had and for all the insightful conversations. We can talk about politics, travels, relationships, life, etc... and I would always learn something useful and new. Thank you Anna for your help while writing this thesis! Thanks to Pierpaolo, for being the coolest dad I have ever met and for all

the delicious dinners!

Thanks to Giuseppe F., for being an awesome flatmate and friend. You're always been a friend I could count on. Thanks for all the games in LOL, even if you should thank me for the boosting.

Thanks to Pietro, for being a great friend. I always admired your generosity and your goodness to your friends. Thank you for being a friend I could always count on. By the way, even if you make fun of me for being a nerd, we all know that you are more nerd than me.

Thanks to my uncles, for giving me the awesome childhood full of love that is the dream of every kid. Thank you for always satisfying my infinite curiosity when I was a child. Thanks to you I found my passion and, therefore, I accomplished my goals.

Thanks to Carlo, for always being there everytime I needed a drink and a chat. Thanks for being the youngest (inside) among my friends.

Thanks to Elle, for being, sometimes, the only one who is able to understand and value some parts of myself. By the way, $\sqrt{4} = 2$ or $\pm\sqrt{4} = \pm 2$. However, $\sqrt{4} = \pm 2$ is wrong! I can prove it to you:
Proof. Let $x \in \mathbb{R}$ be a number such that $x \geq 0$ and $y \in \mathbb{R}$ its square root. Then, we have that $y^2 = x$. Conventionally, we write $y = \pm\sqrt{x}$. Then $+\sqrt{x}$ and $-\sqrt{x}$ are the two square roots of x. Hence, the two square roots of 4 are $+\sqrt{4} = +2$ (or, equivalently, $\sqrt{4} = 2$) and $-\sqrt{4} = -2$. □

Thanks to everyone who cared, even a little bit, about me. Thanks to everyone who made even a single day of my life special!

Finally, my special thanks go to me myself. For the hard work and for never giving up, not even in the darkest moments. I know it sounds a bit awkward, but sometimes it is important to value and be grateful to ourselves.