University of Cagliari

Ph.D. Program in Mathematics
and Computer Science

Computer Science Track

Polycube Optimization and Applications

From the Digital World to Manufacturing

Author:
Gianmarco Cherchi

Ph.D. Supervisor:
Prof. Riccardo Scateni

2017 - 2018

"It is the time you have wasted for your ***thesis*** *that makes your* ***thesis*** *so important."*

A. De Saint-Exupéry, G. Cherchi
The Little Ph.D. Student. *Unica 27, 2 (2019)*

Special thanks to Ric, Cino and Pierre for patiently guiding me on this stressful and beautiful journey. Thanks to the reviewers for the precious advice. Thanks to everyone who helped, with love, friendship and beer, to make this trip special.

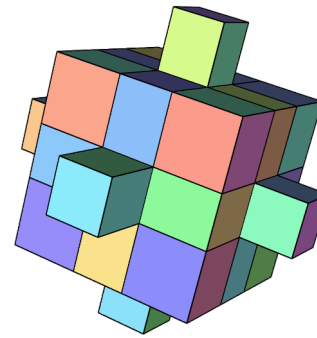
Gianmarco

Abstract

Polycubes are three-dimensional domains formed by connecting axis-aligned cuboids face to face. The simplicity of their structure explains their popularity for several computer graphics problems such as quadrilateral and hexahedral meshing, texture mapping, etc. In this thesis, after a summary of the state-of-the-art methods to compute polycube mappings, we study a possible optimization of the polycube structure and two potential uses of that. We start from polycube corner alignment, to improve the polycube-based mesh structure in terms of coarse surface and volume block decomposition. Then, we present an automatic method to enhance the element quality of a polycube-based hex-mesh, via a selective padding strategy that improves the mesh quality only where it is needed. In closing, we move from the digital world to the manufacturing one, to introduce a method for the decomposition of digital shapes for fabrication purposes, by using a polycube-induced partitioning. The key idea of this thesis is to explore the polycube world from different points of view, by moving from the purely digital world to the more concrete one of fabrication.

Contents

Introduction



Generating structured volumetric and surface representations of 3D objects is today a relevant problem in several computer graphics applications. They are input for methods like Finite Element Methods or IsoGeometric Analysis but also for applications like Animation, Gaming or Biomedical fields. In the last years, important progress in the meshing research community has been made. Several methods and algorithms have been proposed, including the recent parameterization-based techniques. Among these, *polycube-based approaches* are particularly interesting. Since their introduction in 2004 [THCM04], they immediately received the attention of different studies in several computer graphics branches. Starting from texture mapping, a lot of research fields such as spline fitting, morphing, remeshing, etc. have manifested their attention for this simple axis-aligned shapes. The key reason of the spread of polycubes is the bijective mapping function between the input model and its polycube representation, that allows mapping the elements of the starting shape on the polycube domain and vice versa. Thanks to this, it is possible to perform a lot of manipulations on the polycube space (working with a very simple structure) and then mapping them to the input shape. For example, computing "well-structured" meshes via polycubes is a simple and effective task, because a polycube always admits a trivial hex-meshing representation that can be easily computed by gridding it into a regular lattice.

A part from from the purely digital world of three-dimensional representations of models, another emerging field in computer graphics is the manufacturing of digital objects. With the advent of cheap machines for hobby manufacturing (like 3D printers), the need to manipulate digital

models (that did not come out of our screens until recently) has emerged, in order to make them suitable for the fabrication process. The inevitable question is: can polycubes be useful in this process too?

The general idea of this thesis is to analyze the possibility to use polycubes in different fields, in particular for the generation of well-structured meshes and the manipulation of 3D models for the manufacturing process. This manuscript is organized as follows:

Chapter 1 introduces polycubes and the concepts behind them. After an extensive description of their structure and features, a brief recap of the principal polycubization methods is presented in Section 1.2. Then, in Chapter 2, the traditional meshing pipeline via polycube gridding is presented and discussed.

Chapter 3 presents an approach for the polycube structure optimization, in order to produce coarse and structured quadrilateral and hexahedral polycube-based meshes. One of the key ingredients to provide coarse block structures is to achieve a good alignment between the mesh singularities (i.e., the corners of each block of the polycube). In the work presented in this chapter, the polycube-based meshing pipeline is improved by introducing a new step to produce both surface and volumetric coarse block-structured meshes of general shapes. The main goal is to optimize the positions of the polycube corners to produce base complexes that are as coarse as possible. The new locations of the polycube corners are re-mapped on an integer grid and then, by using integer numerical programming to reach the optimal, the singularity misalignment problem is solved directly in polycube space. The proposed corner optimization strategy is efficient and requires a negligible extra running time for the meshing pipeline. The obtained optimized polycubes produce coarser block-structured surface and volumetric meshes if compared to previous approaches. They also induce higher quality hexahedral meshes that better suit for spline fitting because they reduce the number of splines necessary to cover the domain, thus improving both the efficiency and the overall level of smoothness throughout the volume.

The described polycube simplification method has been developed in collaboration with Riccardo Scateni (University of Cagliari) and Marco Livesu (CNR-IMATI Genoa). This work has been published in the article *"Polycube Simplification for Coarse Layouts of Surfaces and Volumes"* [CLS16], in the *Computer Graphic Forum* journal, and it has been presented during the *SGP 2016* conference in Berlin (Germany).

In Chapter 4, the quality of polycube-based hexahedral meshes is analyzed and improved. Hex-meshes generated from polycube mapping often exhibit a low number of singularities, but also low quality elements located near the surface. It is thus necessary to improve the overall mesh quality, in terms of the minimum Scaled Jacobian (MSJ) or average Scaled Jacobian (ASJ). A quality improvement may be obtained via the global padding (or pillowing) operation, which pushes the singularities inside the volume by adding an extra layer of hexahedra on the entire domain boundary. Such global padding suffers from a significant increase of complexity, with unnecessary hexahedra added. In addition, the quality of elements near the boundary may decrease. For this reason, in this chapter a novel optimization method for polycube based hex-meshes is presented. It inserts sheets of hexahedra to perform a "selective padding" where it is most needed, in order to improve the mesh quality. A sheet can pad part of the domain boundary, traverse the domain and form singularities. The proposed global formulation, based on solving a binary problem, allows to control the balance among quality improvement, increase of complexity and number of singularities. A series of experiments shows that this approach increases the MSJ value and preserves (or even improves) the ASJ, while adding fewer hexahedra than global padding.

The described selective padding method has been studied and developed during my period spent in the *Titane* team at the INRIA - Sophia Antipolis Mediterranée (France). This was done in collaboration with Pierre Alliez (INRIA), Riccardo Scateni (University of Cagliari), David Bommes (RWTH Aachen University / Bern University), and with the precious help of Max Lyon (RWTH Aachen University). This work has been published in the article *"Selective Padding for Polycube-based Hexahedral Meshing"* [CAS*19] in the *Computer Graphic Forum* journal.

Then, passing from the digital world to the manufacturing one, in Chapter 5 the digital fabrication is introduced. This chapter shows a brief recap of the main fabrication technologies (additive and subtractive) and related limitations on shape and dimension of the objects to produce.

In Chapter 6, a novel algorithm based on the polycube representation of the original shape is proposed, with the aim to decompose any model into smaller parts, each simpler to fabricate. The input shape is mapped to a polycube and, then, it is split to take advantage of the polycube partitioning. In this way, a partition of the model is quite easily obtained. This chapter also studies and analyzes the pros and cons of this partitioning scheme for fabrication, when using both the additive and subtractive pipelines. The proposed partitioning scheme is computationally light, and it produces

high-grade results, especially when applied to models that can be mapped to polycubes with a high compactness value.

This work has been developed in collaboration with Riccardo Scateni (University of Cagliari), Alessandro Muntoni (CNR-ISTI Pisa), and with the precious help of two master students. Preliminary results have been published in the article *"Polycube-based Decomposition for Fabrication"* [FCS17], in the proceedings of STAG 2017 and presented during the *STAG 2017* conference in Catania (Italy). A second article about this work, entitled *"Fabrication Oriented Shape Decomposition Using Polycube Mapping"* [FCM*18], with new important features and new interesting results, has been published in the *Computer & Graphics* journal.

Finally, Chapter 7 presents the final considerations about the results obtained and discussed in the works presented in this thesis.

Part I

Background

Chapter 1

Polycubes

The *Parameterization* process is always been an essential goal for the computer graphics. The idea of mapping a surface describing a shape into another surface with the same topology (parameter domain) is relevant for a lot of applications. Bijections between general shapes and orthogonal polyhedra, or **PolyCubes**, have been introduced by Tarini et al. [THCM04] in 2004 as a smarter means for the seamless texture mapping generation, and they have received growing attention from the scientific community ever since.

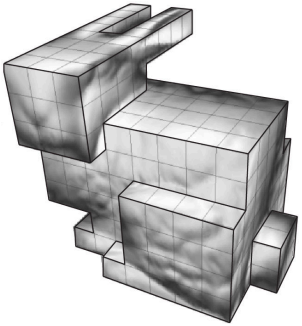


Figure 1.1: *An example of polycube from [THCM04].*

1.1 Structure and properties

The structure of a polycube is characterized by three essential properties: all axis-aligned edges, only 90° angles between adjacent edges, and always flat facets. These features make the polycube manipulation simple

and efficient, and for this reason they are very useful in many graphics applications. Indeed, given a function $f : \mathcal{S} \rightarrow \mathcal{P}$ that maps a generic shape $\mathcal{S}$ into a polycube $\mathcal{P}$, it is easier to perform analysis, processing and different kind of manipulations in the simple structure of $\mathcal{S}$ (all axis aligned and with a limited number of vertices) and then use the bijective reverse mapping function f^{-1} to reflect them on the original shape.

The simplicity of polycubes structure explains their popularity for several computer graphics fields. A polycube always admits a trivial hex-meshing that can be simply computed by fitting it into a regular lattice. Following the parameterization, such grid can, therefore, be mapped into the input shape, generating an hex-mesh version of it (e.g., [HXH10, GSZ11, YZWL14, HJS*14]). Because of their rectangular structure, polycubes can be easily decomposed into a set of regular structures that facilitate the tensor-product surface definition. For this reason, they received great attention from the spline fitting world (e.g., [WJH*08, LLWQ10, WLL*12, LLWQ13, WZLH13]). Also because of their structures, they can be used to efficiently pack textures into rectangular pictures, in order to produce texture mappings without cuts (e.g., [THCM04, CL*10]). Furthermore, the singularity graph induced by polycube corners and edges can be used to easily compute a surface quad-layout and to produce semi-regular quadrangular meshes (e.g., [CLS16, HJS*14]). Other polycubes employments can be listed, for instance morphing, in which polycubes are used as a common base domain to parameterize similar shapes (e.g., [FJFS05]), solid modeling and reconstruction (e.g. [WHL*08]), or fabrication-oriented decompositions (e.g., [CZL*15, FCS17, FCM*18]).

The quality of a polycube is evaluated by considering two important factors: the distortion induced by the mapping and the number of corners (which will become singularities in potentially resulting splines or polycube-based quadrilateral/hexahedral meshes). Notice that not only the number of singularities is essential, but also their position and valence. In addition to these, during the process of polycube construction, two other crucial properties must be taken into account: the *compactness* of the polycube structure and the *fidelity* (independent to the global rotation) to the input model.

Polycubes can be intended as a collection of connected cuboids, glued face to face, approximating a shape. Therefore, on the one hand, the compactness measure indicates the number of cuboids and the consequent number of singularities, influencing the quality of the final mapping. On the other hand, the fidelity, usually expressed as a metric, measures the difference in orientation between the original normals of surface elements in the input shape and the corresponding ones in the polycube space.

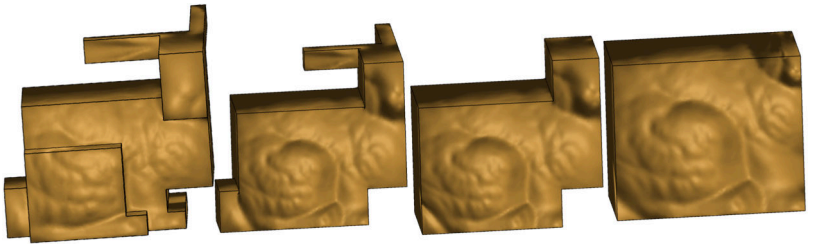


Figure 1.2: *Polycubes of the same model computed with different compactness values. Image courtesy of [LVS*13].*

1.2 Construct polycube maps

The construction of polycubes is certainly not a trivial task. Three main steps usually characterize it: the *segmentation* of the original shape, followed by the *flattening* of the estimated patches and the final *mapping* of the input elements into the polycube structure. In the segmentation step, the surface of the input model is labeled, usually considering the initial normal of their elements and then post-processed to obtain a surface subdivision satisfying the following properties:

- each surface portion has at least four adjacent portions;
- adjacent portions cannot have opposite labels;
- once individuated the four portion corners, the valence of every one of them must always be three.

Once individuated the surface portions, they need to be flattened to obtain the final polycube structure and perform the final mapping process. In their first appearance in [THCM04], polycubes where built by hand. Shortly after, several approaches were introduced to automate the generation. The first attempts to automatically generate polycube maps were not sufficiently robust to process complex shapes and tended to produce either overly coarse (e.g., [LJFW08]) or overly complex (e.g., [HWFQ09, GSZ11]) polycubes, with the former suffering from high distortion and the latter producing unnecessary corners. Nowadays, the polycubes computation is an enough mature technology. Indeed, the most recent algorithms can process complex shapes and consistently provide polycube maps with both low distortion and low corners count [LVS*13, HJS*14].

Below, brief traces of the most important state-of-the-art methods for the polycubes computation are reported, from the first approaches to the more advanced and modern ones.

User-controllable polycube map for manifold spline construction

In 2008, Wang at al. introduced a framework for the user-controllable polycubes generation [WJH*08]. The produced polycubes are ideal parametric domains for constructing spline surfaces. In the first step of the generation pipeline, the users can set the positions and the curvatures of the polycube corners on the input surface. Then, the Riemannian metric of the surface is deformed, such that all the corners have a prescribed Gaussian curvature, and other points are on a flat plane. In the next step, the surface corners are connected by straight lines, in order to partition the surface into a collection of planar quadrilaterals (see Figure 1.3). Each quadrilateral is turned into a planar rectangle by setting the corner angles to $\pi/2$ and running the Ricci flow. Finally, all the planar rectangles are glued together to form the final polycube.

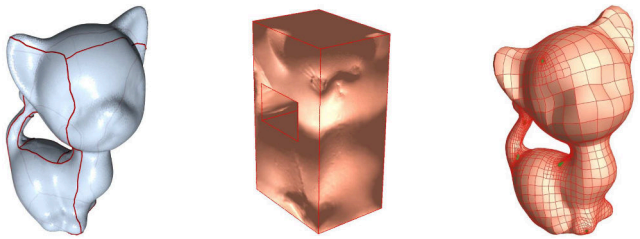


Figure 1.3: *Spline fitting (from [WJH* 08]): user-marked corner points and geodesics between corners, the resulting polycube and the spline fitting result.*

Automatic polycube-maps

Also in 2008, Lin et al. proposed the first method to compute surface polycube-maps automatically [LJFW08]. The method is based on three main steps. In the first one, the input surface is segmented in patches corresponding to relevant features of the model, with the help of a Reeb graph. For each feature-region obtained by the first segmentation step, an appropriate basic polycube primitive is built (cube, L-shape, O-shape, U - shape). Then, each region is subdivided into sub-patches which are assigned to the facets of the basic primitives. In the last step the parameterization is performed. The vertices of every patch of a region are mapped into the corresponding facet of the polycube, by using the mean value coordinates methods. Finally, an iterative optimization step is performed to improve the obtained result. A recap of the pipeline is shown in Figure 1.4.

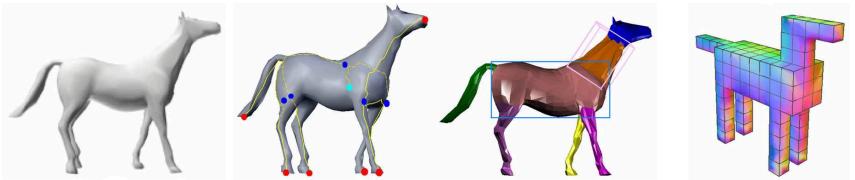


Figure 1.4: *Pipeline recap (from [LJFW08]): the input model, the Reeb graph based features segmentation, the polycube approximation and the final polycube.*

A divide-and-conquer approach for automatic polycube map construction

In 2009, He et al. presented a divide-and-conquer algorithm for the automatic construction of polycubes [HWFQ09]. The input model is segmented into a set of genus-0 shapes by using a harmonic function f. Using the critical points of f, the model surface is sliced with horizontal cutting planes (a user-defined parameter decides the maximum distance between two adjacent cutting planes). Then, the polycube is constructed by extruding all the model slices into axis-aligned polygons (see Figure 1.5). After an iterative set of slicing for both the model and the obtained polycube, an intrinsic mapping between the original surface and the final polycube is computed. This method produces very low distortion polycube mappings but with a high number of singularities in the final base-complex.

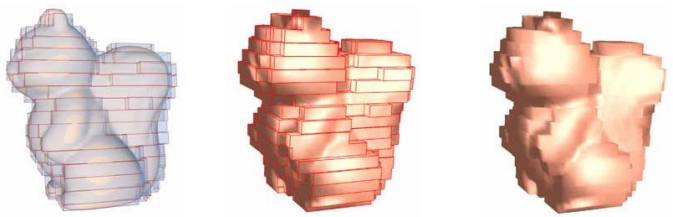


Figure 1.5: *Divide-and-conquer (from [HWFQ09]): the sliced input model, the polycube extraction via polygon extrusion, and the final polycube.*

All-hex mesh generation via volumetric polycube deformation

In 2011, Gregson et al. proposed the first method to compute volumetric polycube for general shapes, with the aim to produce high-quality all-hex-meshes [GSZ11]. Starting from a volumetric input tet-mesh, the

normals of triangles in the surface are gradually aligned with one of the six global axes, preserving as much as possible the original shape and propagating the surface deformation to the interior elements. After iterations of deformation steps, when the model is sufficiently axis-aligned, the polycube structure is determined and, under additional positional constraints, the polycube facets are forced to be planar according to the appropriate axis (see Figure 1.6). The computed polycube is now ready to be gridded and used for the hex-meshing step. Polycubes computed by this method are able to capture the major features of complex models, including off-axis and sharp features of CAD models and intricate details of smooth shapes.

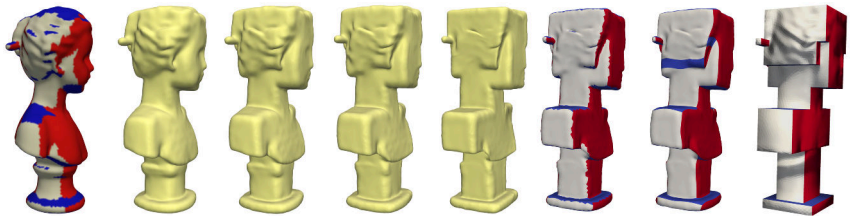


Figure 1.6: *Pipeline recap (from [GSZ11]): from the original model to the polycube, after the deformation iterations.*

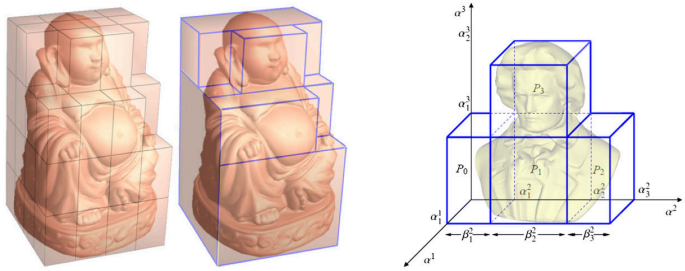


Figure 1.7: *Polycube construction via voxelization (from [WYZ*11]).*

A topology-preserving optimization algorithm for polycube mapping

Also in 2011, Wan et al. introduced a method to compute and optimize the polycube mapping [WYZ*11]. Starting from a coarse voxelization of the input shape (see Figure 1.7), each voxel is marked as inside or outside. All the inner facets are removed, and the interior cells are merged

to the final polycube. The initial polycube mapping is computed via discrete harmonic parameterization. In the following step, the polycube is optimized, by scaling its sub-patches, in order to minimize a mapping energy, but preserving the polycube structure. Finally, without changing the polycube topology, the polycube surface mapping is optimized by searching the optimal corner position to minimize a distortion energy. The proposed optimization is suitable for polycube computed via different methods. Polycubes produced with this method has a low number of cubes and singularities if compared to the previous approaches.

Polycut: monotone graph-cuts for polycube base complex construction

In 2013, Livesu et al. proposed a graph-cut based approach for the polycube computation [LVS*13]. The elements of the input surface are labeled considering their normal. The labeling is then improved with the solution of a multi-label graph-cut problem, taking care about a trade-off between fidelity to the model and compactness of the polycube structure. Then, an iteratively hill-climbing optimization is applied to guarantee the monotonicity of every chart boundaries (see Figure1.8). Once the segmentation is completed, the axis orientation for each chart is computed through a combination of mesh deformation and distortion minimization. The polycube structure is defined and the parameterization is computed. The strong point of this work, compared to the previous ones, is the flexible control of the trade-off between the fidelity and compactness features.

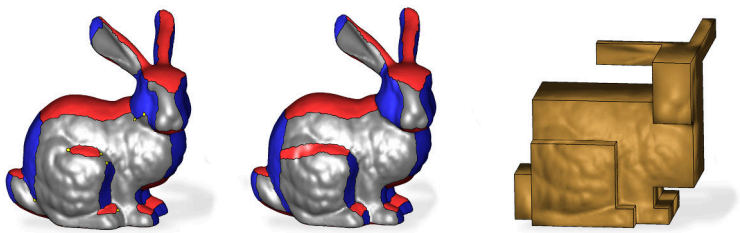


Figure 1.8: *Multi-labeling (from [LVS*13]): the starting labeling, the optimized labeling and the final polycube structure.*

$\ell 1$-based construction of polycube maps from complex shapes

In 2014, Huang et al. presented the first orientation-independent algorithm for polycubes computation [HJS*14]. An input triangle mesh is converted

into a volumetric tetrahedral mesh. Such mesh is then deformed through an iterative constrained minimization of a $\ell 1$-norm based energy containing terms to enforce the axis alignment of the surface mesh normals. At the same time, an as-rigid-as-possible volumetric distortion energy is solved to regularize the distortion. The best orientation of the model is found by adding the solution of an optimal global rotation matrix to the minimization process, to find the best orientation. Furthermore, the user can control parameters like compactness, flat regions and sharp edges. User-guided control over the resulting polycube map is a key point to increase design flexibility. In Figure 1.9, a collection of results obtained with this method is shown.

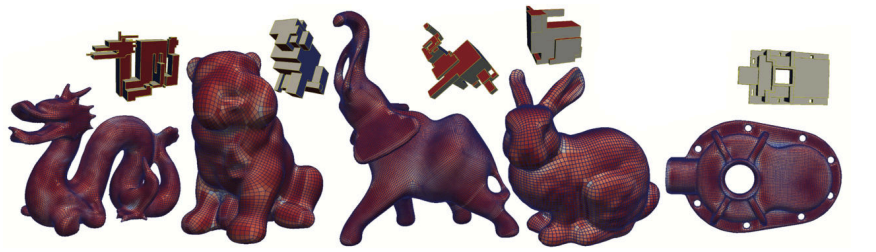


Figure 1.9: *A gallery of results from [HJS*14]: polycubes and extracted hex-meshes.*

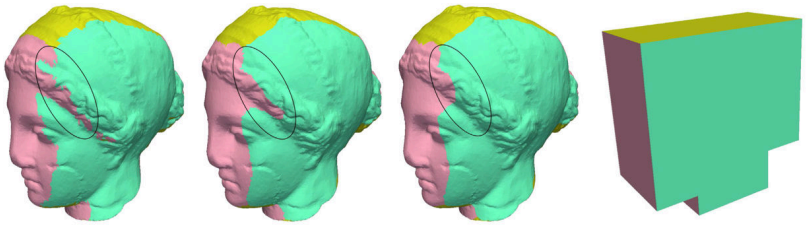


Figure 1.10: *Improving chart boundary monotonicity via centroidal Vornoi tessellation (from [HZ16]).*

Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation

In 2016, Hu et al. proposed an automatic approach to compute polycubes, based on a centroidal Voronoi tessellation [HZ16]. A smooth input triangle mesh is segmented into six clusters according to triangles normal. The polycube structure is then computed by forcing all the corners of each

segment with hard planarity constraints in the corresponding principal axis. Then, a one-to-one mapping between the input model and the polycube is computed. The core of this approach is a new harmonic boundary-enhanced centroidal Voronoi tessellation approach that includes neighboring local information in its energy function. It used to improve the starting segmentation in order to reduce non-monotone chart boundaries, as it is shown in Figure 1.10.

All-hex meshing using closed-form induced polycube

Also in 2016, Fang et al. introduced a new frame-field based approach to compute volumetric polycubes [FXBH16]. The input tetrahedral mesh is cut along a subset of its surface triangles, according to a Reeb graph-based method. In the following step, starting from an initial frame field constant on each tetrahedron, the frame field transitions at the cuts is allowed by optimizing a boundary-aligned smooth cross-frame field with no inner singularities. The resultant frame field and the transitions on cuts are used to deform the cut mesh into a polycube-like shape. Eventually, the final axis-aligned polycube structure is obtained by applying an extension of the $\ell 1$-formulation explained in [HJS*14]. This method is able to reduce unnecessary stairs, corners, and preserve the mesh features more accurately than the previous ones. A recap of the described pipeline is shown in Figure 1.11.

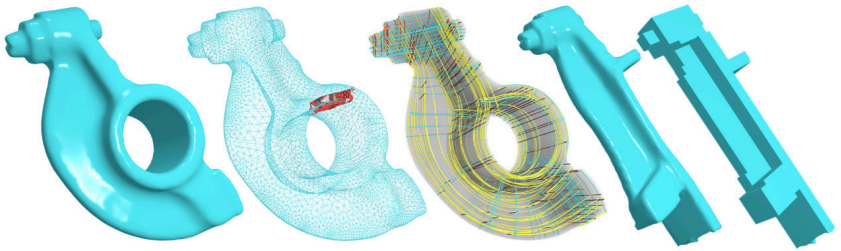


Figure 1.11: *Pipeline recap (from [FXBH16]): the input mesh, the cut faces, the computed frame field, the deformed cut mesh and the final polycube.*

Efficient volumetric polycube-map construction

Again in 2016, Fu et al. proposed a foldover-free and low-distortion method to compute volumetric polycubes [FBL16]. A starting surface mesh is tetrahedralized and analyzed to locate inner edges and facets that, in

the final mapping, could cause degenerate elements. The input model is deformed so that its surface facets are aligned to the projected Gaussian-smoothed normals. The surface is then segmented with already existing approaches, like those of [GSZ11] or [LVS*13], and a check of the existence of a valid polycube topology is carried out. If it exists, a flattening method to force every chart of the segmentation to be axis-aligned is employed. If not, a normal-alignment mesh deformation scheme, based on the minimization of an ad-hoc normal-alignment energy, is applied to solve the segmentation issues. Compared with the previous approaches, this method is at least one order of magnitude faster and it has better mapping qualities. In Figure 1.12, a collection of polycubes obtained with this approach is shown.

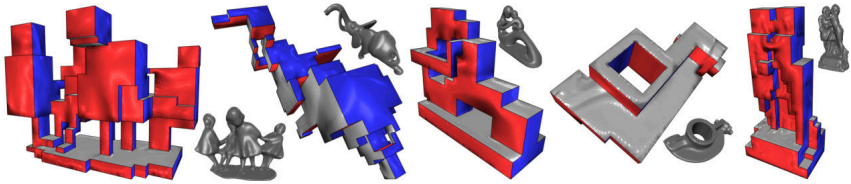


Figure 1.12: *A gallery of results from [FBL16]: the input models and the computed polycubes.*

Figure 1.13: *Pipeline recap (from [HZL17]): the segmentation using the first step, the skeleton-based segmentation and the final polycube.*

Surface segmentation for polycube construction based on generalized centroidal Voronoi tessellation.

In 2017, Hu et al. introduced a two-step surface segmentation strategy for polycube construction, through generalized centroidal Voronoi tessellation [HZL17]. Two steps to segment an input surface model are presented. The first one is a new eigenfunction-based centroidal Voronoi tessellation method for the boundary segmentation through the secondary Laplace

operator eigenfunctions, that aims to remove unsmoothed boundaries, over-segmentation errors, and noise effect. The second one is a skeleton-based centroidal Voronoi tessellation algorithm that generalizes the harmonic boundary-enhanced centroidal Voronoi tessellation of the previous step through the introduction of local coordinates to define and update the generators flexibly in the normal space. The result of this step is the drop of unnecessary surface singularities and the production of better results for objects with slim cylindrical components. A pipeline recap of this method is shown in Figure 1.13.

Robust edge-preserving surface mesh polycube deformation.

In 2018, Zhao et al. proposed the most recent method to compute polycubes starting from bounded or unbounded surface meshes [ZLL*18]. In the first step, the input model is segmented by following the approach proposed in [FBL16]. In the second step, the polycube topology is extracted through a rotation of all triangles to their corresponding target normal directions. The core of this work is the last step in which the polycube geometry is fixed, in order to keep as low as possible the mapping distortion, the number of singularities, charts and corners. All these results are obtained via an extended and iterative Poisson system which reconstructs the deformed polycube mesh in order to satisfy the facet normals assigned in the previous step. This method outperforms previous ones in terms of speed, robustness, simplicity, diversity, and quality. A set of polycubes extracted whit this algorithm is presented in Figure 1.14.

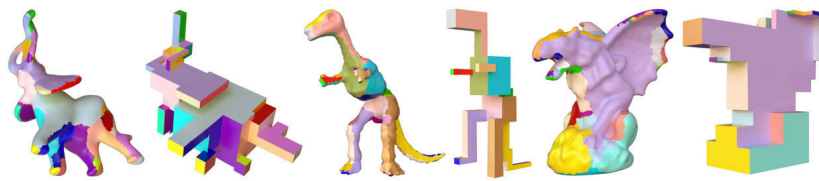


Figure 1.14: *A gallery of results from [ZLL*18]: the input models and the computed polycubes.*

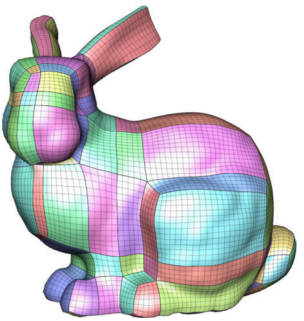
Part II

Polycube-based meshing and optimization

Chapter 2

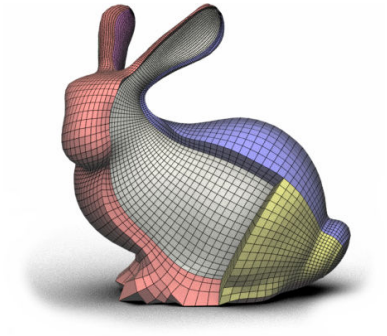
Polycube-based meshes

Representing digital objects with *structured meshes* that embed a coarse block decomposition is a relevant problem in several applications like Computer Animation, FEM (Finite Element Methods), IGA (IsoGeometric Analysis), CAD (Computer Aided Desig), Game Development, Medical fields, etc. Quadrilateral and hexahedral meshes are usually preferred due to their ability to keep a lower resolution and due to their numerical properties. They are also useful because they can be efficiently stored in specialized and efficient data structures.



For the surface case, a block-structured mesh often comes in the form of a quadrilateral mesh obtained by gluing side to side a set of quadrilateral patches (or blocks) in a conforming way [BLP*13]. Each patch is a two-dimensional array of quads. Depending on the application field, these meshes are called *semi-regular meshes* or *multi-block grids*, whereas the graph having as nodes the patches corners and as arcs their edges is usually referred to as *coarse quad-layout* (see the inset on the right). A variety of methods for the automatic computation of coarse block-structured meshes have been proposed in the literature [BLK11, TPP*11]. However, most of these methods are either too demanding from a computational point of view [BCE*13, CBK12] or focus on a specific class of shapes, and do not scale well on general models [ULP*15]. Another research field deals with the generation of user interfaces for the manual construction of quad-meshes and quad-layouts. These methods can produce extremely high-quality layouts [MTP*15, CK14, TP-

SHSH13], but, in order to achieve the maximum result, they need to be controlled by an experienced user. The concepts of structured mesh and block decomposition can be easily extended to the volumetric case.



In [GMD*16], a sweeping method that subdivides the volume enclosed by a triangular mesh into a set of cuboids is proposed (see inset on the left, image courtesy of [GMD*16]). This method can produce extremely coarse layouts, sometimes at the expense of a major deviation from the target shape. It also requires some manual intervention to set up the harmonic field that guides the meshing process. In [GDC15], Gao et al. showed that by coarsening the singularity structure of a given hexahedral mesh, the quality of the mesh elements can be improved without affecting the deviation from the target shape.

In the literature, several approaches with the aim to generate coarse volumes has been proposed. *Skeleton-based* approaches (e.g., [LMPS16]) use tubular structures computed from the model skeleton. These method produces good results but they suitable just for shapes that admit a skeleton representation. *Grid-based* methods (e.g., [LJLJ15]), the most used in the industry, subdivide the volume using regular grids or octrees in order to compute intersections between hexahedra and the surface of the model. These methods are simple, and they work with any type of meshes, but they produce too high-resolution meshes with worst quality elements near the boundary. Furthermore, these methods are not invariant to rotation. *Expanding* approaches (e.g., [TBM96]) generate meshes starting from the surface and moving inwards. These approaches produce meshes with a low quality inside the volume.

Another important class of methods for the creation of hexahedral meshes is the one of the *Parameterisation-based* methods. The input volume is mapped into another parametric space where the final mesh connectivity is generated. With the polycubes spread, new approaches for the meshing have been proposed [GSZ11, LVS*13]. The added value of these approaches is clear: reasoning with a simple structure, together with a mapping function with the original shape, yields several advantages concerning efficiency and simplicity of implementation, as a polycube can be trivially mesh with a regular grid.

2.1 The polycube-based meshing pipeline

Computing quadrilateral or hexahedral meshes via polycube has the advantage to work with a simple structure, bounded by axis-aligned planes, instead of complex models. The state-of-the-art meshing pipeline based on polycube mappings takes advantages of a bijective mapping function $f : \mathcal{S} \rightarrow \mathcal{P}$ that maps each vertex of the input shape $\mathcal{S}$ into a vertex of the polycube space $\mathcal{P}$, and vice versa. The connectivity of the final regular mesh is computed in the polycube space and then the inverse function f^{-1} is used to transfer it to the input space. The set of connected cuboids composing the polycube structure is fitted into a regular and uniform lattice $\mathcal{L}$ of the desired resolution. For each vertex v_i of the lattice $\mathcal{L}$, the polycube elements in which v_i lies must be individuated. In particular, if surface meshing is required, for each $v_i \in \mathcal{L}$ a triangle $t_i \in \mathcal{P}$ is required, while for volumetric meshing a tetrahedron $T_i \in \mathcal{P}$ is needed. Once the belonging element is found, the v_i vertex is expressed via barycentric coordinates $\omega_0, \omega_1, \omega_2$ and potentially ω_3, respect to the considered element (t_i or T_i) vertices. Considering the belonging element t_i, and using the aforementioned f^{-1} function, the equivalent element $t_i^{'}$ in the input model is found. Let A, B, C (end potentially D) the vertices of the $t_i^{'}$ element, the new v_i position is computed as $v_i^{'} = \omega_0 \cdot A + \omega_1 \cdot B + \omega_2 \cdot C$ (using the D term in the same way if a volumetric mapping is required). The connectivity of the new mesh is the same as the lattice in which the polycube has been gridded, while its geometry is computed with the formula mentioned above. Notice that the resolution of the lattice corresponds to the resolution of the final mesh. A recap of the polycube-based meshing pipeline is shown in Figure 2.1.

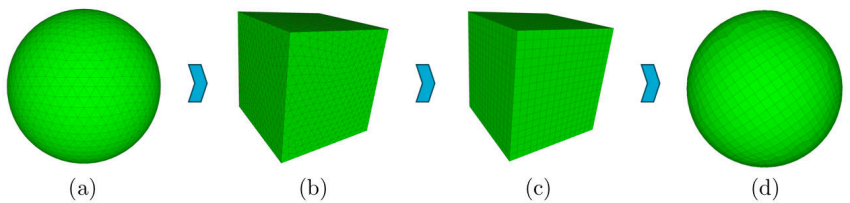


Figure 2.1: *The polycube-based meshing pipeline: the input unstructured model (a), its polycube (b), the policube gridded into a uniform lattice (c), and the final structured quad/hex-mesh (d).*

The structure of the final mesh is therefore defined by the shape of the used polycube. Indeed, the *compactness* term introduced in Section 1.1 is essential in this pipeline. Moreover, the corners of the polycube become singularities in the mesh and chains of edges connecting the pairs of corners

produce the block-decomposition in hex-meshes (or *base complex* [GDC15]) and the quad-layout in quad-meshes.

For surface meshes, the final structure is extraordinarily regular and composed of quads very close to a perfect square, except for areas in proximity of singular vertices (the polycube corners). Notice that, simplifying the block-decomposition in the polycube means to simplify the structure of the final mesh. For this reason, in Chapter 3, a new method to simplify the polycube structure is presented, in order to produce as coarse as possible base complexes for both surfaces and volumes.

For volumetric meshes, the quality of the interior hexahedra is really high, close to the one of a perfect cube. It is not possible to say the same for hexahedra in the proximity of the mesh boundary. In these areas, the distortion is very high, and a post-processing operation is required. The *padding* operation extrudes the surface quads by adding a new layer of hexahedra all over the mesh. In this way, the valence of the singular vertices and edges in the mesh boundary is increased, and the distortion is subdivided in more elements making lower the per-element one. The problem of this operation is that it is performed globally on the hex-mesh boundary, including areas where it is not needed. For this reason, in Chapter 4, a new method to perform selective padding on polycube-based hex-meshes is presented.

Chapter 3

Polycube simplification for coarse layouts of surfaces and volumes

As Section 2.1 shows, a cuboid can be trivially turned into a structured mesh by gridding it. The polycube shape defines the resulting mesh structure: polycube corners become singularities in the mesh, and chains of edges connecting pairs of corners induce a coarse block-decomposition of the domain. In a sense, the base-complex of a mesh $\mathcal{M}$ can be thought of as the coarsest mesh $\mathcal{M}'$ having exactly the same structure of $\mathcal{M}$. In the polycube case, the base-complex is the coarsest mesh that can be generated from a given polycube. Notice that the number of elements in the base-complex does not depend on the number of polycube corners. Depending on how well corners align (or misalign) to each other different base-complexes can be produced (Figure 3.1).

3.1 The singularity misalignment problem

Having a good alignment between the singular vertices of a mesh is a key ingredient in a number of applications. The singularity misalignment problem has been subject of extensive research in recent years, both for surfaces and volumes [MPKZ10, BLP*13, AFTR15, GDC15, RRP15, VS17, PPM*16].

In hexahedral meshing, overly dense base-complexes tend to contain badly shaped cuboids that, when subdivided for the generation of the final hex-mesh, produce poor quality meshes with tiny chances of fur-

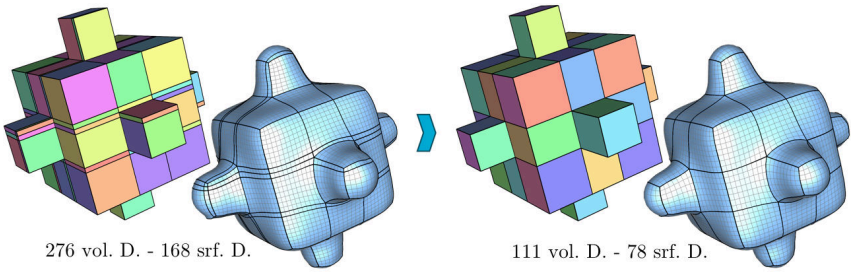


Figure 3.1: *This example shows how, with a few tiny adjustments on the position of the polycube corners, the number of volumetric domains can be reduced from 276 to 111 and the number of surface domains from 168 to 78.*

ther optimization. Coarse base-complexes with well aligned singularities contain better shaped cuboids, therefore tend to produce higher quality hexahedral meshes [GDC15]. Furthermore, coarse base-complexes enable lower resolution meshing, with consequent benefits for applications both in terms of memory requirements and performance speedup.

In higher order-meshing [LZLW15, WZLH13, LLWQ13, WHL*08], a spline basis is fit into each cuboid of the base-complex, with the resulting representation being $\mathcal{C}^2$ continuous within each cuboid and only $\mathcal{C}^0$ at the boundaries between adjacent cuboids.

Coarse base-complexes minimize the extent of the $\mathcal{C}^0$ region, thus providing a higher smoothness throughout the whole domain, enabling both more accurate and more efficient simulations for applications like Isogeometric Analysis (IGA) [HCB05].

Despite the importance that singularity alignment covers for the aforementioned applications, the most of previous methods for polycube computation do not consider this aspect, thus generating sub-optimal base-complexes with far too many cuboids. Given a polycube map, previous methods generate the connectivity of the desired structured mesh by gridding the polycube with an integer lattice. To keep the map bijective, the corners of the polycube must be at integer locations. To ensure this property, a naive snapping is usually performed, rounding each corner to its closest integer location [GSZ11]. Prior to gridding, the polycube is scaled by a factor s in order to control the mesh resolution. Firstly, the naive snapping currently used in state-of-the-art approaches can introduce topological inconsistencies in the polycube structure.

There are many pathological configurations possible. For example, two disjoint corners c and c' may round to the same integer location (Figure 3.2, left); or a corner c may be projected on a polycube facet f if the distance

$d(c, f) \leq 0.5$ (Figure 3.2, right).

Secondly, the scaling factor s not only controls the mesh resolution but also has a non-intuitive, hard to control, effect on the mesh structure. In fact, small scaling factors will produce a better singularity alignment because polycube corners will be more likely to round to the same integer iso-lines. Conversely, big scaling factors will produce worse singularity alignments (and worse coarse layouts) because corners will be more likely to round to different integer iso-lines. In other words, the very same polycube may produce structurally different meshes depending on the sampling density (Figure 3.3).

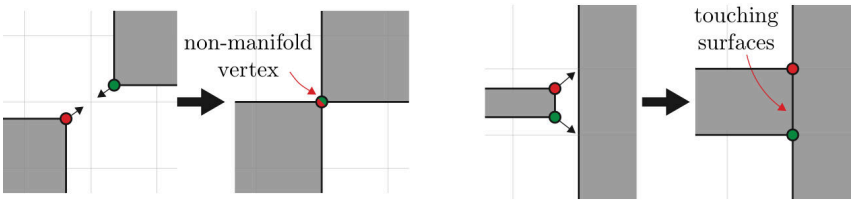


Figure 3.2: *Problems in trivially snapping the polycube corners. Left: two corners (red and green) map to the same integer location, generating a non-manifold vertex (red/green). Right: two vertices snap to the closest iso-line, generating an overlap with another portion of the polycube. In the method proposed in the following sections, explicit constraints are used to avoid such cases.*

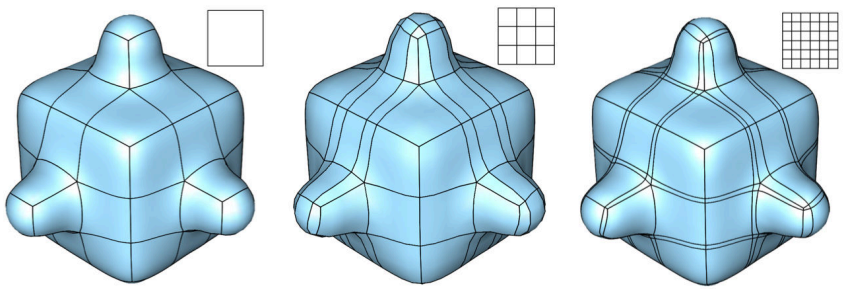


Figure 3.3: *Generation of structurally different meshes depending from different density of the lattice used to sample the polycube. The polycube simplification proposed in the following sections consistently produces meshes with equivalent structure, regardless the density of the sampling.*

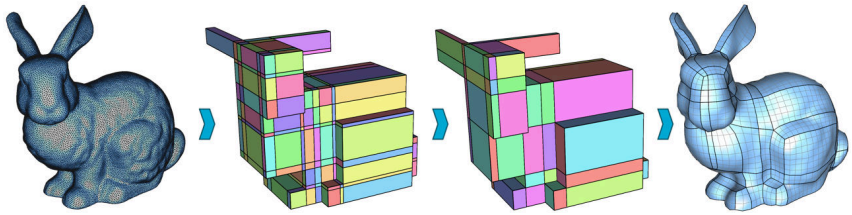
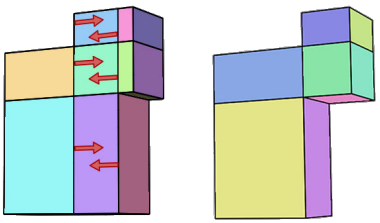


Figure 3.4: *A brief recap of the various stages of the remeshing pipeline: starting from a triangle or tetrahedral mesh (a), the polycube layout is generated (b), the base complex structure is simplified with the proposed method (c), and, finally, a quad or an hexahedral mesh is generated (d).*

3.2 Overview

This Section describe a novel polycube simplification strategy. This is the optimization step of the pipeline as described in Figure 3.4. The blocks of a base-complex are glued face-by-face, so each block separation (i.e., a face shared between two adjacent blocks) defines a separation plane that propagates throughout the whole complex. Therefore, keeping the faces on a limited set of planes reduces the overall number of blocks.



Aligning the block faces on the smallest possible set of planes in each direction, allows producing base-complexes with the lowest possible number of blocks. In the example aside, the initial base complex has 26 surface patches and 33 volumetric domains (included the inner padding). Simply by aligning two faces, it is possible to reduce to 18 surface patches and 22 volumetric domains.

For ease of formulation and implementation, the problem of aligning faces is reduced to the problem of aligning corners. Whenever the position of a corner changes, the position of all the incident edges and faces is coherently updated so as to preserve the axis aligned structure of the polycube. A set of explicit constraints, detailed in Section 3.4.1, allows achieving this result. During the simplification it is possible, therefore, focus only on the relations between polycube corners.

The core of the proposed simplification strategy is a concise iterative method: at each iteration a set of pairs of corners to align is identified

(see Section 3.3). Then, by solving an integer numerical program, they are aligned in the integer lattice (Section 3.4). The alignment process is cumulative, meaning that at each iteration all the corner alignments produced at the previous iterations are preserved. This ensures that the number of aligned pairs grows monotonically. The algorithm converges when no further alignments are possible. Since the set of possible corner pairs is finite, convergence is guaranteed.

Using an iterative method is motivated by observing two things: (i) a corner may want to align with many other corners, so it needs to be paired more than once; (ii) not all the alignments can be discovered *right away*. With an empirical study, it can be observed that iteratively aligning corners and looking for new pairs produces coarser base-complexes than trying to align all the corner pairs in a single global solve.

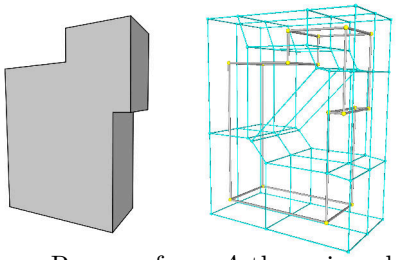
The few lines of pseudo-code in Algorithm 1 summarize the main steps the proposed method, starting from an input polycube (either computed with off-the-shelf algorithms or manually crafted). Both surface and volumetric polycubes, that may come in the form of either a tri-mesh or a tet-mesh, are supported. In the first step, the polycube structure is extracted (i.e., the set of corners and their connectivity). The second step is the iterative alignment, representing the core of the method (see Sections 3.3 and 3.4). At the end of the alignment, each corner in the polycube has new (integer) coordinates. In the final step the input polycube is morphed into its new, optimized, structure (see Section 3.5). The result is a simplified polycube embedding a coarse base-complex that can be used for surface and volumetric meshing, or for spline fitting.

Procedure Polycube Simplification
input : a polycube $\mathcal{P}$
output : a simplified polycube $\mathcal{P}'$
repeat
  Compute corner pairs (Section 3.3)
  Align corner pairs (Section 3.4)
until *convergence*;
Morph $\mathcal{P}$ onto the simplified polycube structure (Section 3.5)
return $\mathcal{P}$

Algorithm 1: *The simplification algorithm in a nutshell. An iteratively interleave corner pairing and alignment is performed until convergence (i.e. until no further alignment can be performed).55*

3.3 Corner pairing

This section describes how to compute $A = \{A_x, A_y, A_z\}$, the sets of corner pairs to align along x, y and z respectively. A heuristic based on the three-dimensional Voronoi diagram of the polycbue corners is employed to find pairs of neighbor corners. Specifically, the dual graph of the Voronoi partitioning is used as the adjacency graph, labeling corners belonging to adjacent cells as *neighbours*. Figure 3.5 shows a simple 2D example of how the Voronoi diagram is used to pair non-adjacent corners.



Once the complete list of neighbor pairs is found, it is possible to select, among them, the candidates for alignment along each coordinate. Then, the graph of adjacencies can be pruned discarding the arcs (pairs of vertices) according to the following rules:

- Remove from A the pairs which are end-points of the same edge, since they are already aligned along one coordinate and it is not possible to align them along another one without changing the edge orientation or without collapsing it.
- Remove from A the pairs of vertices which are end-points of edges incident on the same vertex. In a polycube, if the corners (c, c') and the corners (c, c'') are already aligned along one coordinate, it is not possible to align the pair (c', c'') without losing the axis-align property or collapsing an edge.
- Finally, remove from A external adjacent corners, since it is useless trying align them since their alignment does not produce any reduction in the number of domain of the base complex.

After the pruning, the set A contains only the candidates for alignment. To obtain the sub-sets A_x, A_y and A_z it is needed to determine, for each pair in $(c, c') \in A$,to determinate the coordinate along which the alignment is possible. If more than one possible alignment is found, for a corner c along the same coordinate, the corner c' closest along the considered coordinate is selected as candidate for the alignment.

3.4 Corner alignment

The corner alignment problem is posed as an integer optimization problem, enriched with a set of linear constraints aimed to preserve both the corner

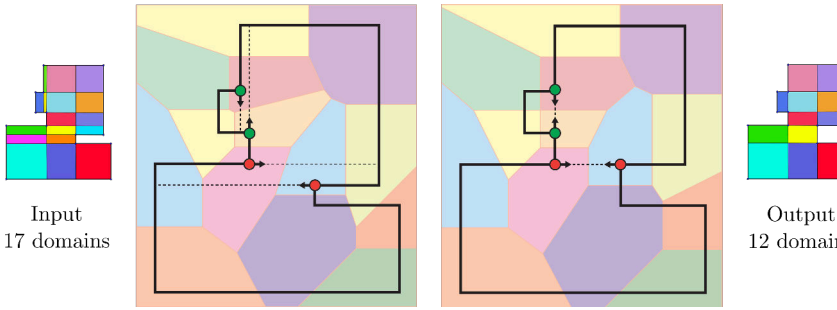


Figure 3.5: *A 2D example that explain how the Voronoi diagram is used to find the candidate pairs of corners to align; the Voronoi diagram in background is computed with the original positions of the vertices (left side); after the alignment the number of domains is reduced from 17 to 12 (right side).*

alignments achieved at the previous iterations and the topological structure of the polycube.

Let A be the set of corner pairs to align at the current iteration, and let A_x^*, A_y^* and A_z^* the sets of corner pairs for which an alignment has already been achieved at previous steps (along the x, y and z coordinates, respectively). The formulation of the proposed optimization problem is the following:

$$\begin{aligned} &\min\ E = E_{align}(A) + \lambda \cdot E_{shape} \\ &\quad \text{s.t.} \\ &c_y = c'_y \quad c_z = c'_z \qquad \forall (c, c') \in A_x^* \\ &c_x = c'_x \quad c_z = c'_z \qquad \forall (c, c') \in A_y^* \\ &c_x = c'_x \quad c_y = c'_y \qquad \forall (c, c') \in A_z^* \\ &\textit{other polycube structural constraints.} \end{aligned} \tag{3.1}$$

The first term, the E_{align} energy, aims to snap each corner pair $(c, c') \in A$ on the same iso-line of the integer lattice. This operation is independently performed on each dimension. Let imagine to split A into three sub-sets, A_x, A_y and A_z, representing the sets of corner pairs to be aligned along the x, y and z coordinate respectively. Then E_{align} can be expressed as follows:

$$E_{align}(A) = \sum_{(c,c') \in A_x} (c_x - c'_x)^2 + \sum_{(c,c') \in A_y} (c_y - c'_y)^2 + \sum_{(c,c') \in A_z} (c_z - c'_z)^2$$

In the carried out experiments, it was instantly clear that the alignment term alone is not capable of producing extremely coarse base complexes.

Indeed, the algorithm tended to align the furthest pairs $(c, c') \in A$ first, leaving all the *"easy"* alignments for the subsequent iterations. This *"complex first, easy after"* behavior is well explained by the fact that E_{align} is quadratic, and thus aligning the furthest pairs first is the best way to rapidly minimize the energy. The problem with this behavior is that, performing the most difficult alignments first, may heavily change the shape of the polycube, generating deadlock configurations in which the (in the beginning) closest corner pairs are no longer possible to align because of the constraints the numerical program is subject to.

To compensate this behavior, a new regularization term is added to the energy: E_{shape}. This energy is a simple corner-wise attraction to the input polycube, that is:

$$E_{shape} = \sum_{c} \|c - \tilde{c}\|^2$$

with c being the current corner position and $\tilde{c}$ the original corner position. The regularization term is particularly useful in the first iterations because prevents dramatic changes in the polycube shape, thus favoring the alignment between the closest corner pairs first. It is however limiting towards the end of the optimization, when these easy alignments are no longer available, and to align the furthest corner pairs would be necessary. In Equation 3.1, the E_{shape} is therefore multiplied by a scaling factor λ. Starting with $\lambda = 1$ at the first iteration, and halving it after each iteration, provides the desired behavior; all the results produced in this chapter have been produced using this scaling strategy. Notice that, different weighting schemes may accommodate better results for certain shapes. Section 3.6 explains how to use λ in order to control the trade-off between polycube simplification and mapping distortion.

3.4.1 Structural constraints

A series of constraints are imposed to preserve the axis aligned structure of the input polycube, also avoiding edge collapsing and self-intersections. This reduces to a set of linear constraints, as detailed below.

Collinearity of the end-points

Polycube edges are forced to keep them axis-aligned. As introduced before, the E_{align} portion of the energy function tries the alignment of corner pairs along one coordinate. It is essential avoiding that this attempt will move the edge off the integer lattice. Let $e(c, c')$ be a polycube edge connecting the corners c and c', and let suppose that e is aligned with the x axis.

In order to keep its original orientation when solving for the c and c' coordinates, the following two linear constraints are imposed:

$$\begin{cases} c_y = c'_y \\ c_z = c'_z \end{cases} \tag{3.2}$$

These linear constraints prevent e to move off the x axis. In a similar fashion edges aligned with the y and z axis can be forced to maintain their original alignment.

Minimum length of edges

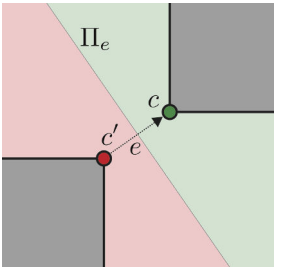
In the proposed formulation the smallest edge length is fixed to 1, to avoid edge collapses. This is ensured by combining (3.2) with one more linear constraint per edge $e(c, c')$:

$$(c - c') \cdot u \geq 1 \tag{3.3}$$

Where $u = \frac{c-c'}{\|c-c'\|}$ is a pre-computed unit length vector aligned with e. Remind that the polycube edges cannot change orientation during the optimization, therefore, the vectors u can be computed once by using the original corner coordinates and then use them throughout the whole iterative simplification. Given the axis aligned structure of a polycube, u can be either $(\pm 1, 0, 0)$ or $(0, \pm 1, 0)$ or $(0, 0, \pm 1)$. Furthermore, notice that this constraint not only prevents the edge to be shorter than 1, but also preserves its original orientation, avoiding an edge flip.

Corners collapsing

All the constraints described before are quite natural to impose. One condition more subtle to check is the avoidance of the collapse of corners that are not end-points of the same edge. In particular, it is essential to avoid that corners pairs $(c, c') \in A$ reach the alignment by occupying the same position in the integer lattice. In this, a simple criterion like the edge collapse is not enough to avoid the move. Recall that, in such a case, the resultant polycube would be not manifold and would lose its original topology (Figure 3.2, left). To tackle this problem, it is necessary to add a particular constraint to the model , in order to create a separation plane between the two corners that are attracting each other. Let $e(c, c')$ be an invisible edge connecting the non adjacent corners c and c', and Π_e the plane passing through the middle point of e and having



$\frac{c-c'}{\|c-c'\|}$ as normal orientation. This defines a partition of the space, with c belonging to the positive half-space and c' belonging to the negative half-space of Π_e. The planes Π_e are computed at each iteration according to the current coordinates of the polycube. When the model is solved for the new polycube coordinates, two new constraints to keep c and c' in the half-space they belong to are added, specifically:

$$\begin{cases} \Pi_e(c) > 0 \\ \Pi_e(c') < 0 \end{cases} \tag{3.4}$$

This is done for each corner pair $(c, c') \in A$.

Dummy vertices and edges

The last condition to control is even more subtle than the previous. The aim is to avoid that: (i) parts of the polycube which are not adjacent compenetrate; (ii) the outside boundary and any of the holes of a non-simple face touch, thus changing the face topology. To this extent, *dummy vertices* and *dummy edges* are necessary. A dummy vertex is defined as the intersection between the supporting line of a polycube edge and the polycube faces closest to it on each side (if any). A dummy edge, on the other hand, connects an end-point of a polycube edge to its corresponding dummy vertex. All the dummy vertices and edges are computed in the polycube, and treated as if they were real polycube edges, imposing that their length must be equal or bigger than 1 (as in Equation 3.3). With this simple new set of constraints the collapse between vertices and edge, vertices and faces, edges and edges or inner and outer boundaries of the same face are avoided. For consistency, special constraints to ensure that dummy vertices will stay within the edge or face they belong to are imposed. In Figure 3.6, the importance of our consistency constraints is emphasized, showing an 2D example of polycube optimization with and without dummy edges. As can be noticed, dummy edges ensure topological consistency, without penalizing the quality of the alignment.

Dummy edges computation

Also for the computation of dummy edges, it is possible to work on each dimension separately. The proposed procedure is explained for the x axis; everything applies also to the y and z axis. Let $f_x^0 \leq f_x^1 \leq ... \leq f_x^n$ be the list of polycube facets having the x axis as normal orientation, ordered according to their x coordinate. Let $e(c, c')$ be a polycube edge aligned with the x axis, such that $c_x < c'_x$. Firstly, e is extended from the c side, finding the first non-empty intersection with the closest facet f_x having x coordinate lower than c_x. If such facet exists, the intersection point is

considered as a dummy vertex and a dummy edge connecting such point with c is added. The same process is repeated for c'. This operation is performed for any edge in the polycube, using similar lists for the y and z axis as well. Notice that, given the coarseness of polycubes, this procedure is very fast (i.e., a fraction of a second).

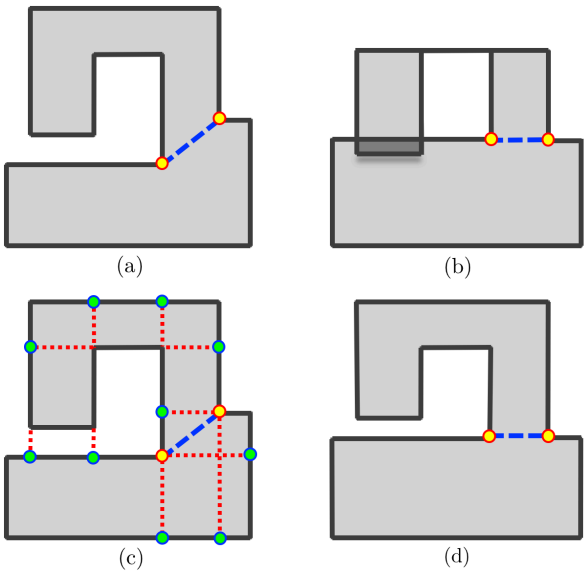


Figure 3.6: *A simple 2D example of a pair of corners to align (a). In (b), the alignment without using dummy constraints. In (c) the red edges are the dummy edges and the green vertices the dummy vertices. In (d) the final polycube after the optimization with the dummy constraints.*

3.5 Finalization

At the end of the corner optimization, a new polycube structure is obtained, optimized in the sense that it has the least number of blocks. This polycube may be dramatically different from the input one because the alignment process produces a lot of compression and stretching of the block volumes, with bad consequences for the distortion of the associated polycube map.

In order to preserve the quality of the original polycube map as much as possible, it is necessary to solve the problem formulated in Equation 3.1 once more, without the E_{align} part of the energy and with $\lambda = 1$. In this final optimization, *all* the corner pairs alignments found at the previous stage of the algorithm are constrained. This generates a polycube structure

that is as close as possible to the input polycube but, at the same time, has integer coordinates and optimal structure. The resulting polycube is so similar to the input one that the distortion induced by our optimization in the polycube map is often negligible (see Figure 3.7, the OUTPUT bunny).

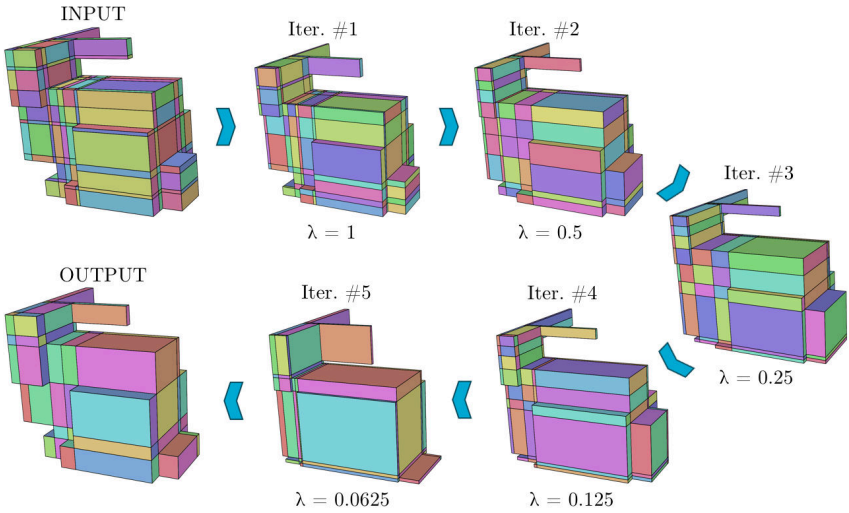


Figure 3.7: *From left to right: the input polycube, the five iterations necessary to complete the simplification process, and the output polycube.*

The output is then finalized by fitting the input polycube (which is either a triangle or a tetrahedral mesh) into this structure. Let $\mathcal{P}$ be the input polycube: to fit the polycube in the optimized structure a simple *Laplacian* problem $\nabla\mathcal{P} = 0$ is solved. Each vertex $p \in \mathcal{P}$ is constrained as follows: (i) all its three coordinates if it is a polycube corner; (ii) two, if p lies on a polycube edge; (iii) one if p is onto a polycube facet; (iv) no coordinates at all if p is inside the polycube. The last condition applies only if $\mathcal{P}$ is a tetrahedral mesh. For surface meshes, the ∇ operator can be implemented by using the cotangent weights, whereas for volumetric meshes the 3D mean value coordinates introduced by Floater et al. in [FKR05] can be used. The result of this process is a polycube with the same connectivity of the input but optimized structure. This polycube may contain flipped or inverted elements and also has overlaps at concave features. Depending on the applications, an optimization strategy may be used to improve the mesh quality (e.g., [AL13]).

3.6 Results

For the generation of the meshes, the standard polycube-based meshing pipeline as described in Chapter 2 as been implemented. Then, the Edge-Cone Rectification method [LSVT15] has been applied to optimize the resulting hexahedral meshes and remove all the inverted elements possibly present. A gallery of results achieved with the proposed method is depicted in Figures 3.8 (only base-complexes) and Figures 3.11, 3.9, and 3.10 (base-complexes and hex-meshes). The standard polycube meshing pipeline has been compared with a modified pipeline in which we added the described simplification system prior to the hexahedral mesh generation step.

In Table 3.1, the obtained numerical results are compared with the standard meshing pipeline. For each model it is shown: elements count, per-element quality (minimum and average Scaled Jacobian), and number of domains in the base-complex (see Section 4.4.1 for a brief introduction to the Scaled Jacobian measure). For a fair comparison, hexahedral meshes with similar elements count have been computed, and the hex-mesh optimizer with standard parameters has been run.

In *all* cases the optimized polycubes produced higher quality hexahedral meshes compared to the meshes produced from the non-optimized counterparts. This confirms what Gao et al. had already shown in their work [GDC15]. Overall, with the proposed approach it is possible to reduce the complexity of the initial polycube with factors ranging from 25% to 70%. In the last column of the table, the time necessary to perform the polycube simplification is reported. Even in the most complex cases a few seconds are enough for the algorithm to converge. Unfortunately, performing a precise and extensive comparisons with the most similar work [GDC15] has been difficult, because of the absence of the polycubes they used to generate the hexahedral meshes shown in their paper. However, to give the reader an idea of the performances of the two algorithms, two attempts of comparisons are shown here. For the RockerArm, they started with a polycube-generated hexahedral mesh having 664 domains, and they simplified its structure reducing it to 335 domains (50% gain). With the presented method, the number of domains has been reduced from 648 to 335 domains (49% gain). For the Bunny, they passed from 580 to 194 domains (67% gain), while with this method the reduction is from 636 to 197 domains (gain 69%). These numbers suggest that the proposed method produces comparable results both in terms of reduction factor and minimum number of domains in the complex. The convergence of this algorithm is reached in a few seconds, while the method presented in [GDC15], in some cases, requires heavier computational effort.

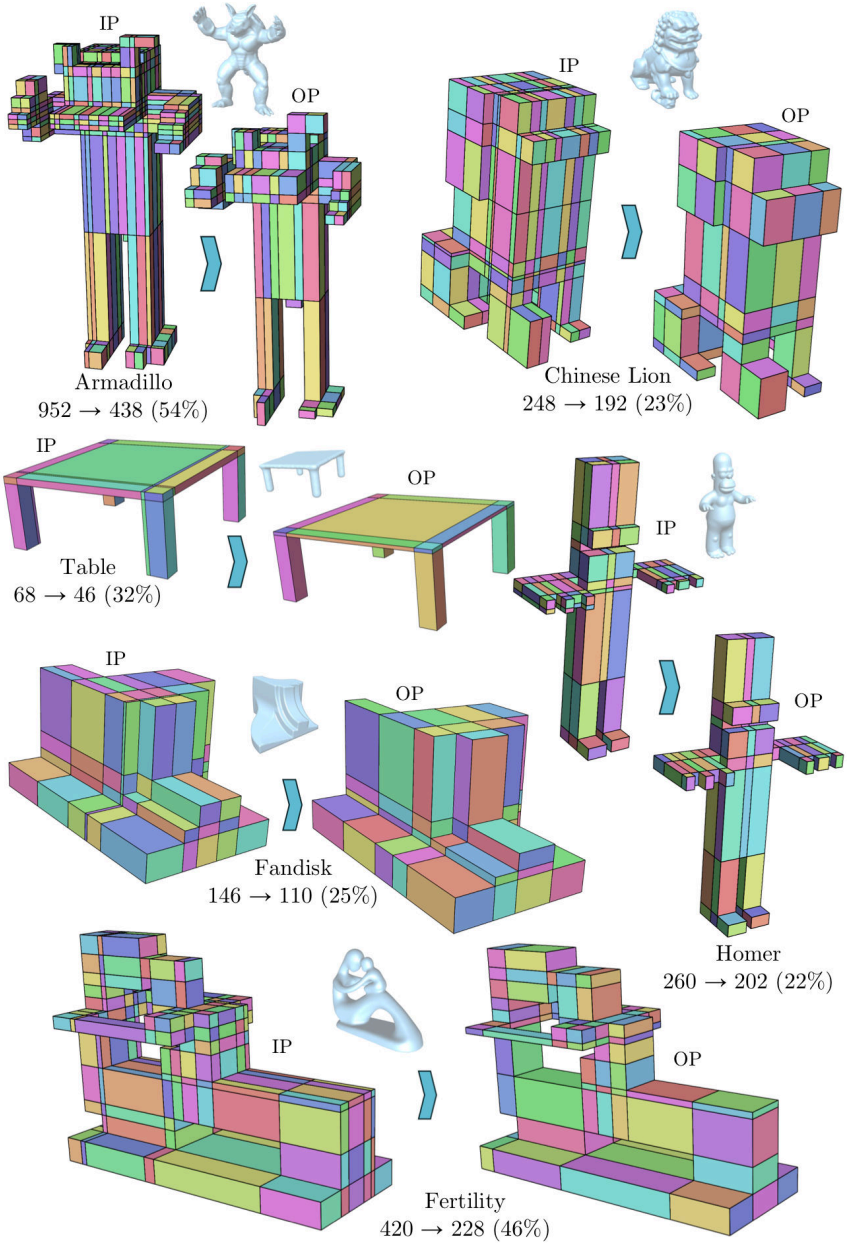
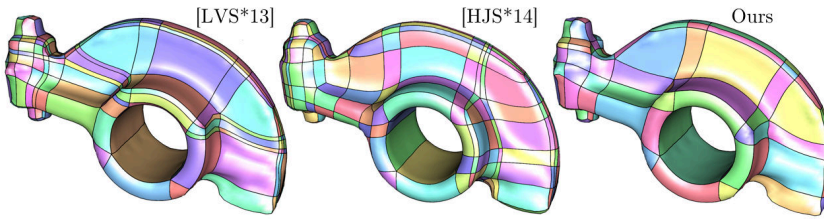


Figure 3.8: *A gallery of optimized polycubes, shown with the color coded quad layout. For every polycube, the number of surface blocks in the input (IP), in the output (OP) and the gain ratio are reported.*

Model	Without simplification			With simplification			Gain	Time
	#H	mSJ/aSJ	#D	#H	mSJ/aSJ	#D		
RockerArm	22K	.22 / .94	648	22K	.27 / .94	332	49%	0.82s
Bunny	48K	.18 / .97	636	8K	.21 / .94	197	70%	2.43s
ASM	30K	.22 / .97	184	30K	.29 / .97	114	38%	1.33s
CubeSpikes	32K	.66 / .97	276	23K	.69 / .98	111	60%	1.36s
Block	18K	.18 / .95	158	18K	.18 / .96	100	37%	1.96s
Femur	15K	.50 / .96	145	15K	.54 / .96	110	24%	0.43s
Hand	32K	.46 / .98	172	5K	.49 / .94	107	38%	1.33s
Table	8K	.30 / .94	195	10K	.59 / .94	149	24%	1.20s
Teapot	35K	.45 / .98	323	34K	.57 / .98	193	40%	4.28s

Table 3.1: *The results are shown in terms of number of domains (#D) obtained with the proposed simplification. In almost all the experiments, the hexahedral meshes computed on top of optimized polycubes have both higher minimum and average Scaled Jacobian (mSJ and aSJ). The #H column represents the number of hexahedra of the model.*



Model	[LVS*13]		[HJS*14]		Ours	
	#sv	#dom	#sv	#dom	#sv	#dom
Cube Spikes	56	168	–	–	56	78
Bunny	64	352	76	176	64	136
Block	48	112	–	–	48	76
Rocker Arm	62	352	64	426	62	208

Table 3.2: *Both visual and numerical results for the generation of coarse quad-layouts are reported here. Comparisons with two famous polycube-based methods are shown. For each algorithm, both the number of singular vertices (#sv) in the layout and the number of surface domains (#dom) are reported.*

3.6.1 Coarse quad-layouts

The proposed method can also be used to generate quadrilateral meshes embedding a coarse quad-layout. To do so, at the meshing stage the only surface of the base-complex is sampled, ignoring the vertices of the integer lattice located in the interior of the polycube.

In Table 3.2 the quad-layouts obtained with the proposed method are compared with the ones produced by [HJS*14] and [LVS*13]. For some of the tested shapes, the performances of some of the best algorithms specifically designed to generate coarse quad-layouts have been matched. In particular, the layout computed for the block model has 48 singular vertices and 76 blocks and it is equivalent to the ones generated from [BCE*13, CBK12]; the layout generated for the Cube Spikes model has 56 singularities and 78 domains and is equivalent to the one generated from [TPP*11]. However, since this optimization works in the polycube space, it may not be flexible enough to match the performances of these algorithms for shapes whose features fail to align to the XYZ frame. For example, the method proposed in [ULP*15] is capable of producing a layout with 24 singularities and 28 domains for the RockerArm, whereas the proposed result on that model contains 62 singularities and 208 domains.

3.6.2 Simplicity vs distortion

Optimizing the structure of the a polycube base-complex is always a matter of finding the right balance between simplification and mapping distortion. Too aggressive simplifications may result in distorted polycubes that degrade the quality of the associated map. In Figure 3.7, all the iterations (and associated λ values) for the simplification of the Bunny's polycube are shown. By properly setting a lower bound for the λ parameter the user can control the simplification process, making the iterative simplification quit before the natural convergence, thus obtaining a partially simplified polycube with lower distortion map. This is an easy and intuitive way, for the user, to control the trade-off between simplicity and distortion.

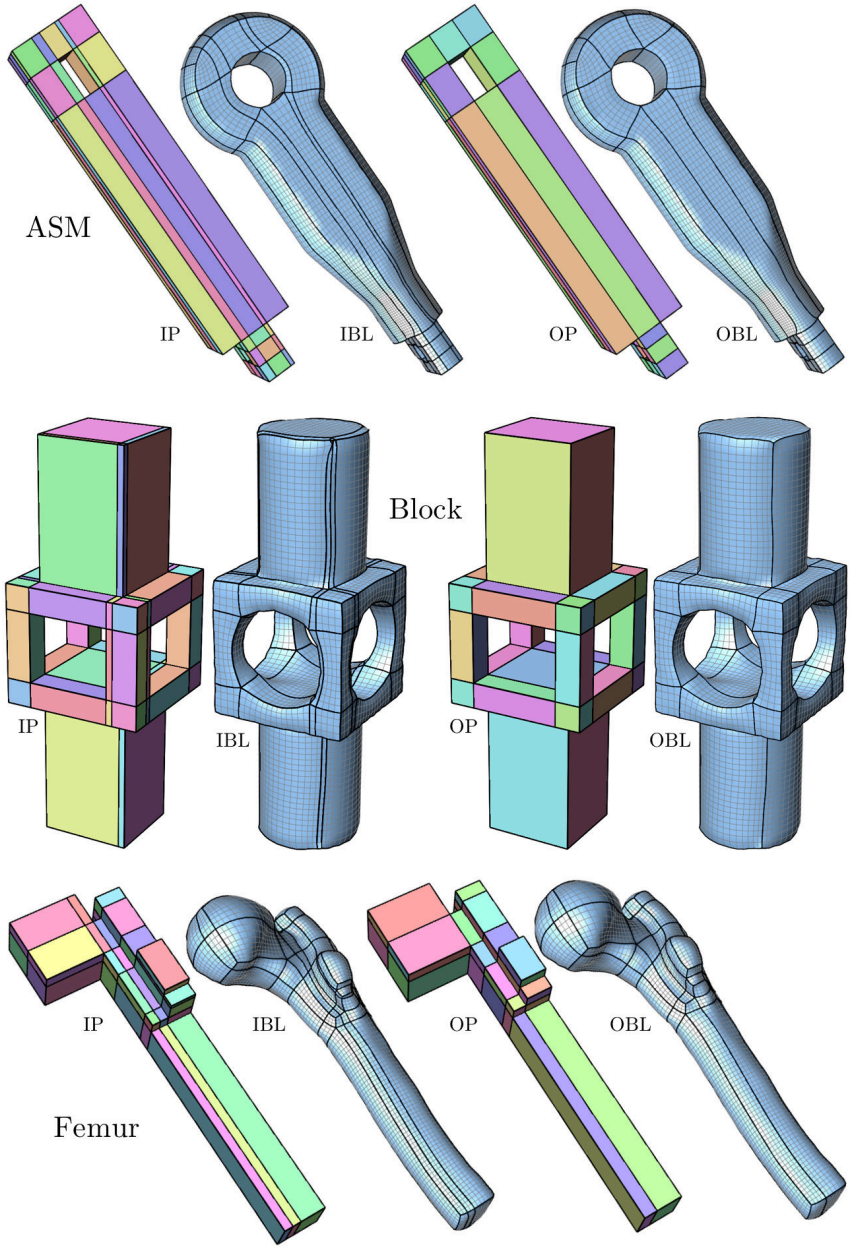


Figure 3.9: *Volumetric results, pt. 1. For every model: the input polycube (IP), the block layout derived from it (IBL), the optimized polycube (OP) and the block layout derived from it (OBL).*

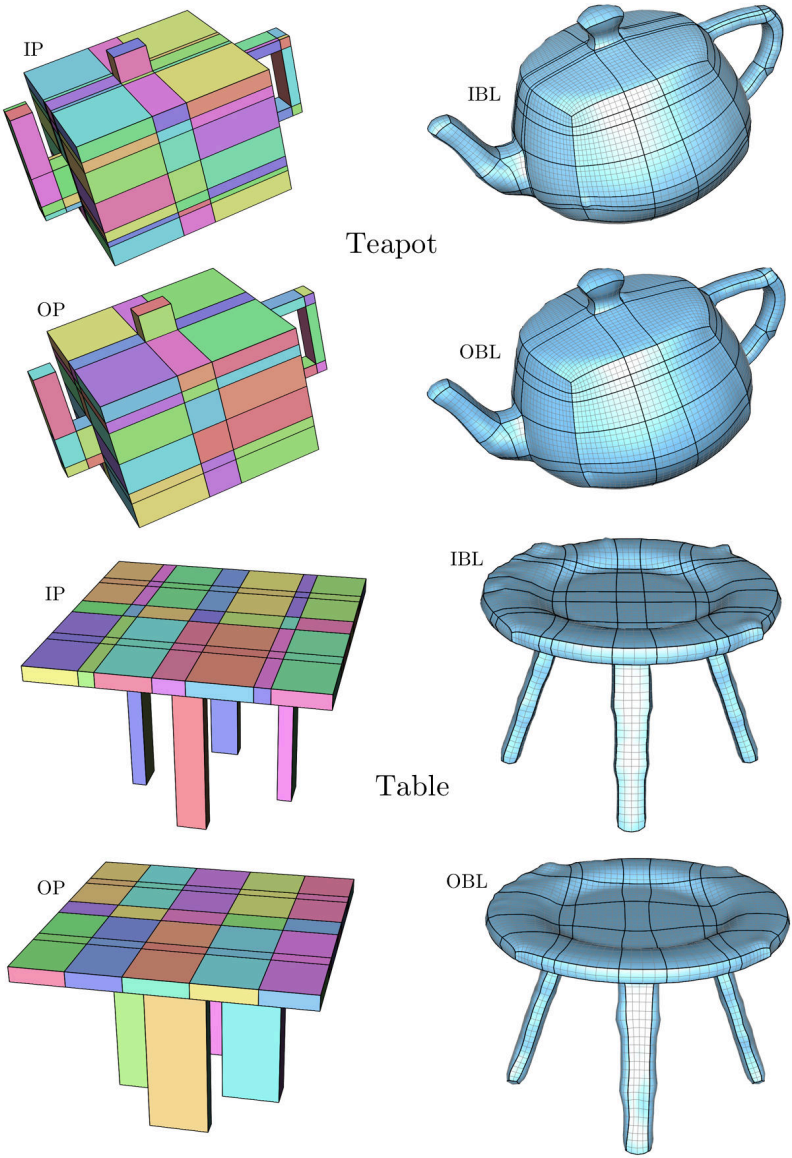


Figure 3.10: *Volumetric results, pt. 2. For every model: the input polycube (IP), the block layout derived from it (IBL), the optimized polycube (OP) and the block layout derived from it (OBL).*

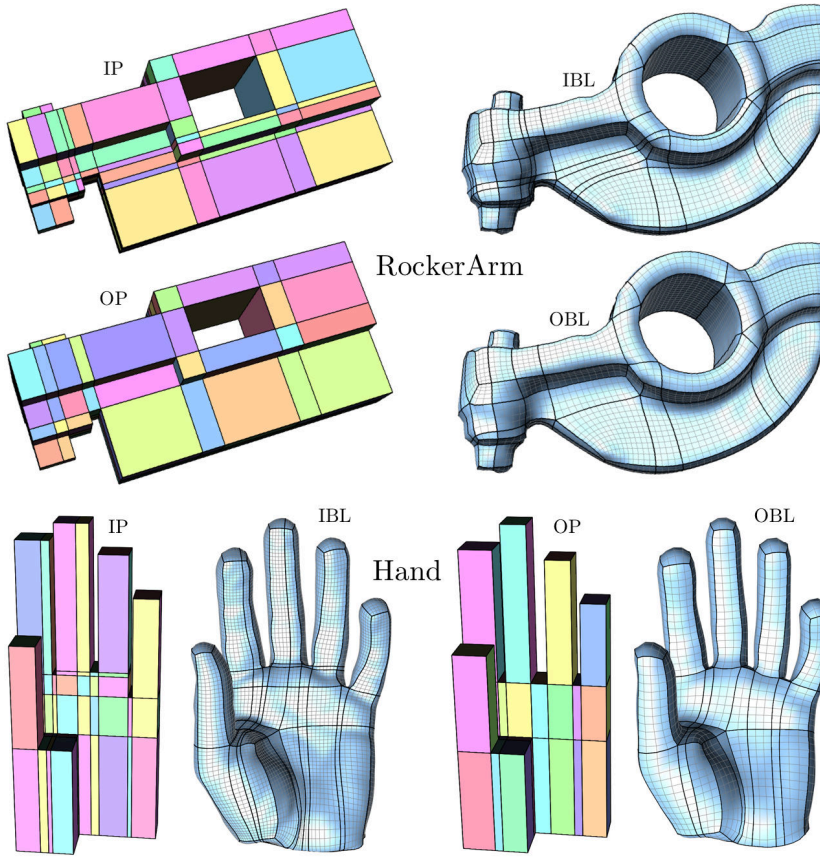


Figure 3.11: *Volumetric results, pt. 3. For every model: the input polycube (IP), the block layout derived from it (IBL), the optimized polycube (OP) and the block layout derived from it (OBL).*

3.7 Limitations

The method proposed in this paper is a heuristic and, as such, it is subject to some limitations. Following, a recap of the major shortcomings of the proposed approach.

Corner pairing

The Voronoi-based corner pairing strategy described in Section 3.3 is not guaranteed to generate all possible corner couples. Furthermore, when

there are multiple possible pairings for a given corner, the closest one along the considered coordinate is selected. This may not be the optimal choice in some cases, as depicted in Figure 3.12. Notice that, it is possible to change or improve the corner pairing step without changing the mathematical model set for the alignment.

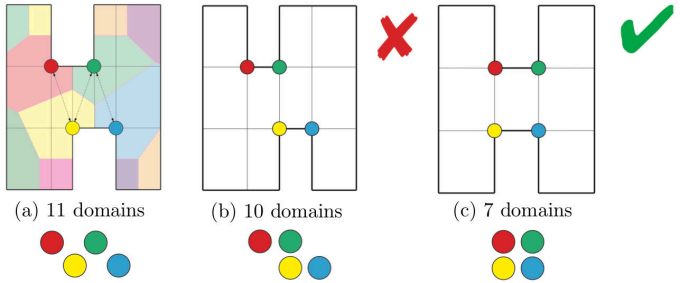


Figure 3.12: *When a polycube corner can align to more, nearly equidistant corners, the alignment scheme may take the wrong decision, generating sub-optimal results.*

Map distortion

In the finalization step (Section 3.5), the similarity between the input and output polycubes is maximized, assuming that this is a good proxy to bound the distortion of the polycube map. Although this heuristic produces good results for most of the tested models, there might be pathological cases in which this assumption is not true. An example of this is given in Figure 3.13, where some of the domains underwent severe stretching after simplification, thus producing a low quality coarse layout. In this case the user can trade simplicity for a lower map distortion with the mechanism described in Section 3.6.2.

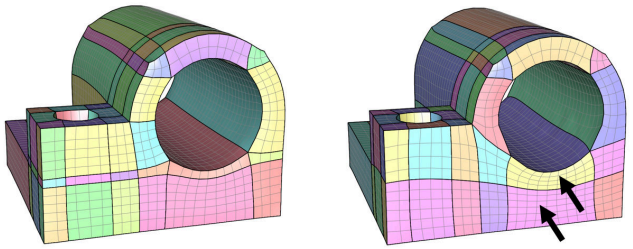


Figure 3.13: *An example of excessive distortion produced by the corner alignment is indicated by the black arrows in the right figure.*

Domains

The mesh structure is optimized in the polycube space, therefore the limitations of the minimum domain number is inherited. In Figure 3.14, two different structures for the triple torus model are shown. The one on the left has been obtained with [TPP*11] and the one below has been obtained with the proposed simplification approach. The curved domains at the extremities of the shape cannot be represented in the polycube space, therefore it is split into three sub-domains each, thus generating a higher number of cuboids.

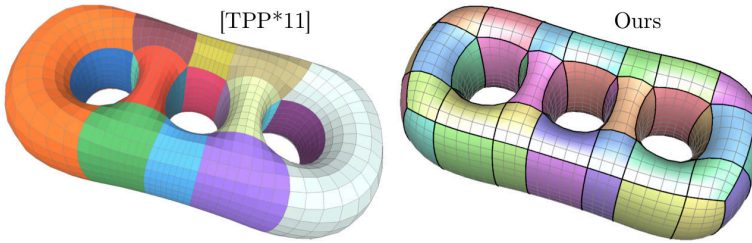


Figure 3.14: *The minimum number of domains in the decomposition is lower bounded by the polycube structure. In the proposed decomposition is not possible to obtain less than 18 domains, while the same model decomposed with [TPP* 11] has only 10 domains.*

Technical information

A beta version of the Cinolib library [Liv17] has been used as data structure, to store both models and polycubes. The Voro++ [Ryc09] library has been used for the computation of the Voronoi partitioning and Gurobi [Gur] as numerical solver. Tetgen generator [Si15] has been used to turn tri-meshes into tet-meshes and the Polycut algorithm [LVS*13] to generate all the polycubes shown in this chapter. See Appendix A for more details.

Chapter 4

Selective padding for polycube-based hexahedral meshing

When volumetric domains to partition in hexahedra are regular enough (e.g., with limited range of levels of details like mechanical and CAD models), polycubes are an ideal tool for generating hexahedral meshes, by computing volumetric mappings with their corresponding domains. However, the quality of the hexahedra directly depends on the mapping distortion. While the distortion is typically negligible inside the domain (because of the regularity of the lattice used to grid the polycubes), it can be excessive near the boundary. For instance, when a convex boundary edge of the polycube with a folding angle of 90 degrees, maps to a flat part of the domain boundary, with 180 degrees.

A common solution consists in *Padding* (or *Pillowing* [MT95]) the entire surface by adding an extra layer of hexahedra, in a way that pushes inside all 90-degree edges and replace them with edges incident to two hexahedra. As detailed in [She07, SJ08], the padding operation starts with an initial mesh from which a subset of hexahedra is defined to create a *shrink set*. The shrink set is separated from the original mesh and shrunk. The void left by such a shrinking process is filled by adding a new layer of hexahedra. Padding offers a means to trade deformation for number of elements, and increase the overall quality of the domain.

This chapter presents an automated method able to insert padding elements only where they are needed to increase the overall quality. The

motivation for this approach stems from the observation that, for specific shapes, the padding operation is not necessary for the whole boundary. In addition, it may in some cases worsen the mesh quality. The proposed algorithm works selectively on the volumetric domain, keeping untouched the parts that are directly derived from the polycube mapping having already a good quality. It reaches this goal by operating selective insertion of sheets of hexahedra in the domain. The solution is posed in terms of complexity and quality of the final hexahedral mesh.

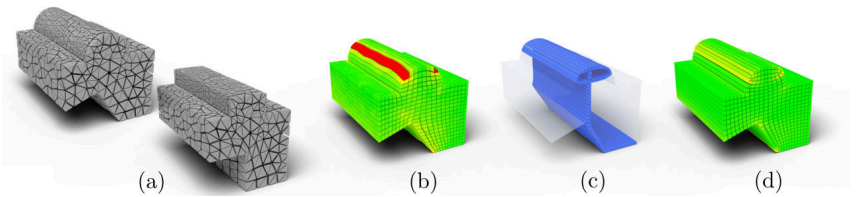


Figure 4.1: *The proposed pipeline takes as input a model and its polycube mapping (a); the relative hex-mesh is computed and the surface areas in need of padding are located by analyzing the mapping quality (b); a binary problem is set and solved to find a set of facets to extrude, in order to create a selective padding layer (c); the mapping with the new hex-mesh structure is computed (d).*

4.1 General hex-mesh refinement

A wide range of local refinement algorithms have been proposed both for quadrilateral and hexahedral meshes [SDW*10]. Common objective are to change the mesh resolution, to decrease the valence of inner vertices or to adapt the mesh density in specific areas as required by FEM simulations.

Zhu et al. [ZCWG14] proposed a method to improve the quality of CAD-based hex-meshes. While the user deforms the CAD model, the associated hex-mesh is automatically improved by adding or removing hexahedral sheets by using dual operations, in order to keep the resolution and the quality of the mesh constant. In the proposed approach, the regularity of the mesh is guaranteed by the polycube properties. For this reason, it is always possible to limit the topology changes to the mesh boundary.

Chen et al. [CGWW16] introduced an approach to achieve complex

sheet inflation under various constraints, in order to improve the mesh quality. The method takes as input a set of user-defined boundary mesh edges and a set of hexahedra. The edges specify the boundary position where the new sheets should be inserted, and then the algorithm computes, through an iterative solving of a Max-Flow and optimization steps, the whole layer position. In the meshes used in the work presented in this chapter, even if the sheet insertion is allowed in all the mesh, the quality analysis is restricted to the mesh surface, because polycube-based hex-meshes always have regular and good-quality inner elements. For this reason, it is easier in this case to automatically detect the positions where the insertion of new hexahedral sheets can improve the mesh structure.

Wang et al. [WSC*17] proposed an automated block decomposition method based on sheet operations, which generates a block decomposition from which a high-quality hex-mesh stems. They start from a B-rep solid model, compute the relative tet-mesh and finally extract a hex-mesh. They insert and collapse whole sheets of hexahedra to improve the obtained hex-mesh by solving an integral linear problem. Based on the good polycube-based inner structure, it is possible to insert sheets of hexahedra to improve the quality of the near-surface elements in the final hex-mesh. It is interesting enough that they agree on the importance to develop an algorithm for a robust insertion of boundary hexahedral layers, which is what it is proposed in this chapter.

Owen et al. [OSE17] contributed a template-based approach for generating locally refined all-hex meshes. Using a restitched set of split configurations, a local refinement of the hex-mesh structure is performed, yielding elements with minimum Scaled Jacobian of 0.3. The approach proposed below hinges upon a similar set of templates. Since the sheets insertions is applied in polycube based hex-meshes, using a restricted set of dual operators is always possible to achieve the desired result. Indeed, since only edges belonging to 1 to 4 facets are present in polycube-based hex-meshes, it is possible using only the set of operators that increase by one the valence of the hex-mesh edges, where it is needed.

Wang et al. [WGZC18] presented a method to improve the topology of hex-meshes via frame field optimization and sheet operations. Starting from a hex-mesh in which they build an initial frame field, they optimize the field in order to obtain a high-quality one, that can be used to identify the most problematic areas in the mesh. Then, they adjust the structure of the mesh via a set of sheet operations. Even if they work with different topologies and perform other types of analysis, Section 4.5 compares the

novel proposed approach with this work, as both of them are adopting different methods to achieve similar goals. Indeed, working with polycube base hex-meshes allows to focus the analysis on the mesh boundary, while they need to analyze the whole structure to perform the optimization.

4.2 Overview

This chapter presents an algorithm to perform selective and localized padding into polycube-based hex-meshes. The introduction of distortion when mapping an object with curved surfaces or non-right angles into an axis-aligned shape like a polycube (e.g., a sphere mapped to a cube) is inevitable. The aim is, thus, to improve the quality of the mapping by adding hexahedral elements only in selected and limited areas of the hex-mesh topology.

State-of-the-art padding operations, *global padding* in the whole hex-mesh surface, can sometimes have the opposite effect of locally worsening the quality of the mesh instead of improving it. The idea of this work is to analyze the quality of the mapping between the polycube space and the object space, and then to identify where a selective insertion of sheets of hexahedra can improve the quality of the final mesh. The topology of the mesh is changed only where it is needed, and it is left untouched in areas where the quality is already acceptable.

4.2.1 The hex-mesh quality

The starting point of this approach is the *mapping quality* of the hexahedral mesh elements. The Scaled Jacobian (SJ) will be used as a quality metric. It computes, for each hexahedron, a value between -1 and 1. When $SJ = 1$ the hexahedron is a perfect cube with highest possible quality. When $SJ \leq 0$ the hexahedron is flipped and thus unfit to further processing. The quality of a hexahedron is judged low when SJ is smaller than a user-specified threshold.

With a simple example, the intuition behind this approach is shown. Assume a box-cylinder object formed by a right box and a cylinder on top, as depicted by Figure 4.2. When computing the hex-mesh of such an object by gridding its polycube, the distortion is localized in the cylinder portion of the object, where the polycube forms right angles.

Performing global padding on this model decreases the quality of elements which are of high quality in the input model, as it is shown by second row of Figure 4.3. By analyzing the local mapping distortion and performing selective padding instead, it is possible to insert a sheet of hexahedra only around the cylinder, and straight through the box to avoid

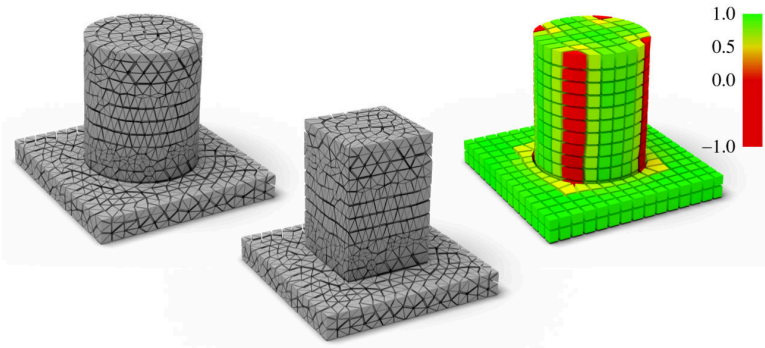


Figure 4.2: *Introducing distortion. From left to right: the tet-mesh of the input model, its polycube (tet-mesh) and the final hex-mesh with distortion highlighted. Here and in the next figures in this chapter, the color ramp on the right is used as a quality indicator where red stands for highly distorted elements, yellow for medium quality, and green stands for good quality elements.*

creating too many unnecessary elements and singularities (edge and vertex turns, see Figure 4.3). The additional elements provided by sheet insertion offer a means to improve the SJ-distortion of the low-quality elements, after vertices relocation. It is forthwith evident the local negative effect of global padding where it is not necessary. Performing global padding increases the quality around the cylinder surface but also decreases the quality of the right box, while adding 850 elements. The proposed selective padding instead increases the quality only where it is needed (improving both the *minimum* SJ and the *average* SJ) and adds only 336 elements.

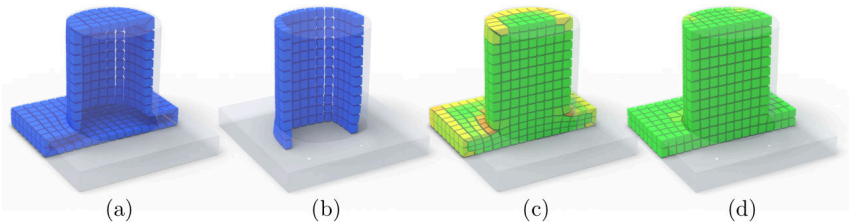


Figure 4.3: *Global vs selective padding applied to the box-cylinder model (column). Left to right: global (a) and selective (b) padding. In (c) the quality of the elements induced by (a) and in (d) the one induced by (b).*

4.2.2 The sheet insertion

Figure 4.4 depicts a sub-set of three template configurations used to extrude the facets of a hexahedron. To minimize the number of added hexahedra, the new layers of hexahedra are allowed to turn, around an edge (edge turn) or a vertex (vertex turn). Notice that, an edge turn introduces 2 singular edges, while a vertex turn introduces 7 singular edges. For this reason, in the following sections, a way to balance between introduced elements and introduced singularities is proposed.

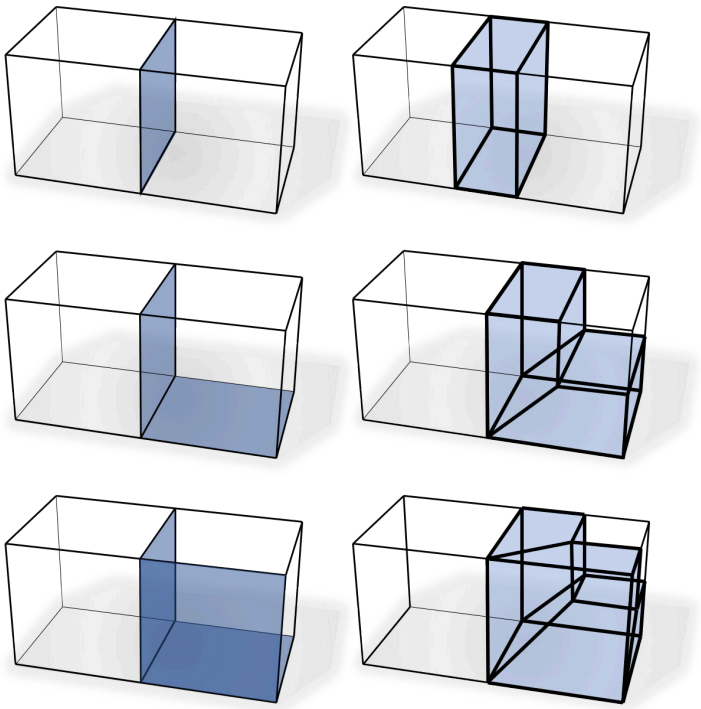


Figure 4.4: *Padding via facet extrusion. Top to bottom: padding a single facet (one added hexahedron), padding two facets (two added hexahedra and one edge turn) and padding three facets (three added hexahedra and a vertex turn).*

The described approach takes as input a volumetric mesh and its polycubes generated with the Polycut algorithm [LVS*13]. It proceeds in two main steps: mapping analysis (Section 4.4) then selective padding (Section 4.3). The selective padding step refines the mesh structure to provide additional elements, and hence degrees of freedom for existing

mesh optimization approaches.

More specifically, the role of the mapping analysis step is to identify a set of facets delineating hexahedra with high mapping distortion. According to an analysis of the dihedral angles between facets of bad quality hexahedra, the set of facets used as padding constraints for the global solver are determined.

Starting from these facets, the goal is to selectively pad the mesh with just-enough hexahedra to reduce distortion where needed. The proposed solution to preserve the structure of the hexahedral mesh is to proceed by sheet insertion, the sheets being decomposed into a series of consistent facet extrusion operators.

A model with a constrained objective function and binary variables is formulated: one variable per facet; one variable per edge (to count edge turns) and one variable per vertex (to count vertex turns). The goal of the optimization is to find a satisfactory balance between quality, number of elements and number of singularities.

4.3 Selective padding

The local padding problem is posed as a binary all-linear problem with a set of constraints that preserve the consistency of the topological hex-mesh structure. Given a set of Hard-constrained Facets (HF) of facets which must be padded, the solution of the binary problem yields a set of facets which allow a consistent padding that includes at least the facets from HF. The proposed formulation enables the user to trade the number of additionally padded facets for the number of singularities introduced via padding.

4.3.1 Simple binary problem

Let $M = (V, E, F, H)$ be the polycube-based input hex-mesh composed of vertices, edges, facets and hexahedra. A binary variable $xf_i \; \forall \; f_i \in F$ is created for each facet specifying whether this facet should be padded (through extrusion) or not. By definition, every inner edge of a polycube-based hex-mesh has four incident facets. If changing the mesh by extruding facets is needed (as explained in Section 4.3.3), it is possible to extrude, for an inner edge, only 2 or 4 facets without creating topological inconsistencies. The outer edges, on the other hand, can be incident to 1, 2 or 3 hexahedra. These sub-sets of edges are denoted by $E1H$, $E2H$ and $E3H$. The first goal is to pad as few facets as possible:

$$E_{padding} = |H|^{-\frac{2}{3}} \sum_{f_i \in F \setminus HF} xf_i \tag{4.1}$$

where the term $|H|^{-\frac{2}{3}}$ is added to achieve resolution independence.

Two constraints are enforced during optimization. Firstly, all facets in HF must be padded:

$$xf_i = 1 \qquad \forall f_i \in HF \tag{4.2}$$

Secondly, in order to achieve a valid, hex-topology preserving padding, we require the number of padded facets around each edge, but the ones belonging to $E1H$ and $E2H$, to be even:

$$\sum_{f_i \in F(e_j)} xf_i = 2k_j \qquad \forall e_j \in E \setminus (E1H \cup E2H) \tag{4.3}$$

where k_j is an integer variable defined for each edge involved in this constraint, and $F(e_j)$ is the set of the facets incident to edge e_j. They must be treated in different ways, with respect to the constraint:

1. Edges in $E1H$ can have 0, 1 or 2 selected incident facets. All these configurations are suitable.

2. Edges in $E2H$ can only have three incident facets, two on the surface and one inside. Almost all the possible paddings are legal: (i) the padding of the inner facet, (ii) the padding of both surface facets, (iii) the padding of one of the surface facets along with the inner one, causing an edge turn, and (iv) the padding of all three facets. A custom constraint is inserted in the model to avoid the case of padding only one of the surface facets that would cause a topological inconsistency.

3. Edges in $E3H$ are covered in the general constraints of Eq 4.3. It is clear that, in this way, some legal cases are excluded but, considering how the insertion of new layers via facet extrusion is defined, they would not generate valid solutions. Moreover, the possibility to reach an optimal solution is not limited.

Figure 4.5 illustrates an example solution of this simple binary problem. In order to reduce the number of inserted singularities, the base formulation is then extended.

4.3.2 Binary problem extension

Minimizing $E_{padding}$ alone under constraints 4.2 and 4.3 yields a valid solution with the lowest number of extra elements. In practice, however, this may not always be the desired solution. As depicted by Figure 4.5(d), the padding may introduce singularities inside the mesh. While adding few more hexahedra is often better than introducing extra singularities, turns of the padding layer provides a means to avoid generating many extra elements. A trade-off between extra elements and extra singularities is required.

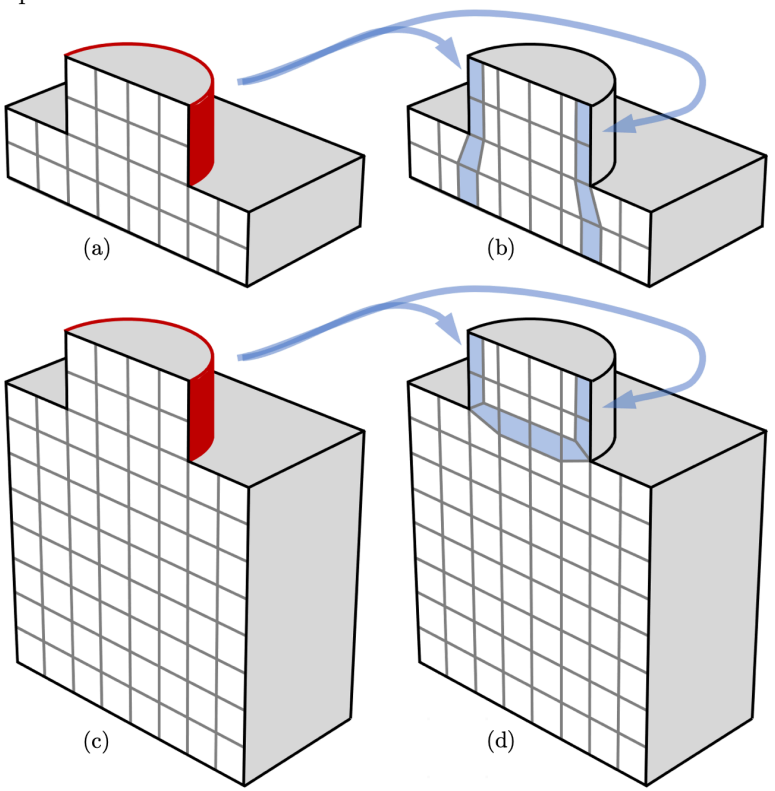


Figure 4.5: *An example of the described padding strategy. In (a) the red set of facets has to be extruded; in (b) the solution obtained with the basic formulation; the mesh in (c) is similar but, in this case, padding straight to the bottom would insert many hexahedra. The proposed formulation finds the solution in (d), with fewer added hexahedra, by introducing edge and vertex turns inside the mesh.*

The problem formulation is thus extended with a binary variable te_i

for every edge $e_i \in E$, recording a turn configuration at the location of e_i (cf. Figure 4.4 middle) and thus the introduction of a pair of valence 3 and 5 edges. Additionally, a binary variable tv_l for every vertex $v_l \in V$ is added to the model, recording a vertex turn configuration (cf. Figure 4.4 bottom). Using these edge and vertex variables, it is possible to formulate an additional objective of keeping the number of introduced singularities low (e.g., minimizing the number of layer turns) via:

$$E_{complexity} = |H|^{-\frac{1}{3}} \sum_{e_j \in E^* \setminus E1H} te_j \quad + \quad \sum_{v_l \in V^* \setminus V1H} tv_l \tag{4.4}$$

where E^* and V^* are the sub-sets of edges and vertices that are incident to two orthogonal facets of HF since these layer's turns are unavoidable and will be in the final solution. As for edges, $V1H$ denotes the sub-sets of vertices incident to only one hexahedron. As before, the term $|H|^{-\frac{1}{3}}$ is added to the left sum to render the formulation independent of the hex-mesh resolution.

In order to ensure the indicator variables te_i to be 1 if and only if an edge turn configuration is present, the following constraint is added:

$$\begin{gathered} te_j = |xf_i - xf_k| \quad \forall e_j \in E^* \setminus E1H, \\ \vec{f_i} = \vec{f_k} \quad and \quad f_i, f_k \in F(e_j) \end{gathered} \tag{4.5}$$

where $F(e_j)$ denotes the set of facets incident to edge e_j. According to Equation 4.5, to detect a possible edge turn in the edge e_j, a pair of facets $f_i, f_k \in F(e_j)$ having the same orientation ($\vec{f_i} = \vec{f_k}$) is considered. As can be observed in Figure 4.6, the value of the subtraction $|xf_i - xf_k|$ determines if an edge turn is present in e_j.

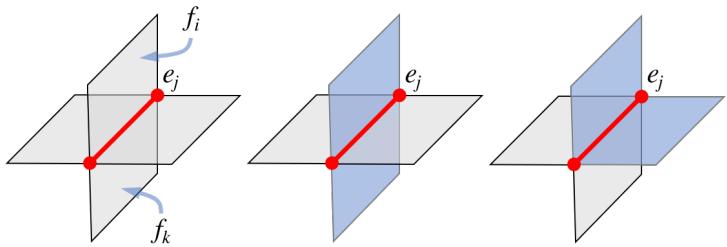


Figure 4.6: *Detection of an edge turn. Left: two selected facets with similar orientation. Middle:* $|xf_i - xf_k| = 0$ *hence no edge turn is detected. Right:* $|xf_i - xf_k| = 1$ *hence an edge turn is detected.*

Similarly, to ensure the indicator variables tv_l to be 1 exactly when a vertex turn configuration is present, the following constraint is added:

$$\begin{gathered} tv_l = |te_i - te_k| \quad \forall v_l \in V^* \setminus V1H, \\ \vec{e_i} = \vec{e_k} \quad and \quad e_i, e_k \in E(v_l) \end{gathered} \tag{4.6}$$

where $E(v_l)$ denotes the set of edges incident to vertex v_l. According to Equation 4.6, to find a vertex turn, a pair of edges e_i, e_k in the set of edges incident to v_l in $E(v_l)$ having the same orientation ($\vec{e_i} = \vec{e_k}$) is considered. The te variables of the selected edges are subtracted, in absolute value, in order to detect whether a vertex turn is present in v_l.

Finally, under constraints 4.2, 4.3, 4.5 and 4.6, a linear combination of $E_{padding}$ and $E_{complexity}$ is optimized:

$$\min\ E = E_{padding} + \lambda \cdot E_{complexity} \tag{4.7}$$

Adjusting the coefficient λ provides the user with a means to trade the number of extra elements for the number of extra singularities. Figure 4.14 illustrates the output solution for three different values of λ.

Notice also that, satisfying Equation 4.2 for all surface facets is a feasible solution, so it is easy to affirm that a solution satisfying the aforementioned mentioned constraints always exists.

4.3.3 The new layer insertion

The output of the solver is a set of facets PF representing the areas where one or several hexahedral sheets must be inserted. The padding layer is generated by extruding each facet $f_i \in PF$ and transforming it into a hexahedron. The set of extruded facets forms the new layer. Each facet is extruded in both directions, considering a fraction of the edge lengths of incident hexahedra as a reference, except for the surface facets which are extruded only towards the inside. The final structure of the mesh, including the new elements, is deduced from analyzing the global configuration of facets in PF. As shown by Figure 4.7, the extra singularities are decided in accordance to the adjacent facets.

4.4 Mapping analysis

The previous section describes a method which, given a set of facets HF that need to be padded, finds a complete set of facets which can be extruded while preserving the hex-mesh structure. The following one, instead, details how the suitable set HF can be found, such that the completed padding improves the quality of the hex-mesh.

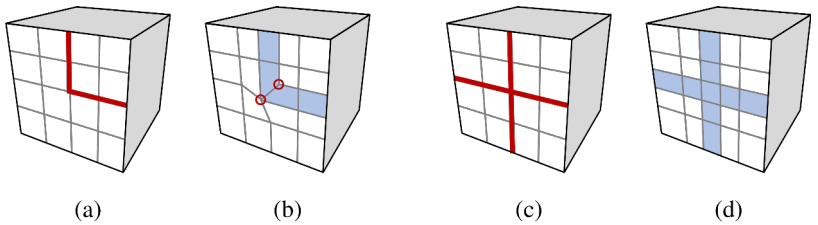


Figure 4.7: *Example of padding configurations and associated sheet insertions. When four facets are selected (a) the resulting new layer of hexahedra introduces new singularities, depicted with red circles (b). If the same four facets are selected, together with four more ones (c) the final layer is different (d), and there are no extra singularities added.*

4.4.1 Distortion per facet

As polycube-based hex-meshes are commonly well shaped inside the domain, the main idea is to analyze the boundary hexahedra in order to identify the ones which would benefit from being padded. The quality of a hex-mesh is commonly measured via the Scaled Jacobian (SJ). For each hexahedron in the hex-mesh, the minimum determinant of the Jacobian matrix is computed, evaluated at each of its 8 corners, and the center of the element divided by the corresponding edge lengths. According to the Verdict manual [SEK*07], a "good" quality hex-mesh should have only hexahedra h_i such that $SJ(h_i) \geq 0.5$. By observing the made experiments, improving the quality of the mesh beyond this value is sometimes possible. Therefore, it has been decided to limit the analysis to the boundary hexahedra whose Scaled Jacobian is lower than $\mathcal{T} = 0.6$, considering the remaining ones as already good. This value may be, of course, adjusted to fit application-specific requirements.

The quality of a boundary hexahedron can often be improved by padding. However, padding all its facets is not suitable (c.f., Figure 4.3). A transition from a distortion measure per hexahedron to a distortion measure per facet is therefore necessary, where a high facet distortion indicates that padding is required. The set HF of facets which must be padded can then be simply defined as those facets whose distortion measure exceeds a user-specified threshold $\mathcal{T}$ derived from the quality requirement of a specific application.

The transition takes place in two steps. First, a distortion measure per boundary edges is defined, based on the dihedral angle between incident facets. The distortion of a facet is then defined as the maximum of the distortion of the four facet edges. In Section 4.3.1, the sub-sets of edges

$E1H$, $E2H$ and $E3H$ have been introduced as the edges respectively incident to 1, 2 and 3 hexahedra. Since padding the facets incident on the edges of $E2H$ introduces two new hexahedra which again share a $E2H$ edge, distortion is unlikely to improve. Therefore, only the boundary edges in $E1H$ and $E3H$ are considered, both shown in Figure 4.8.

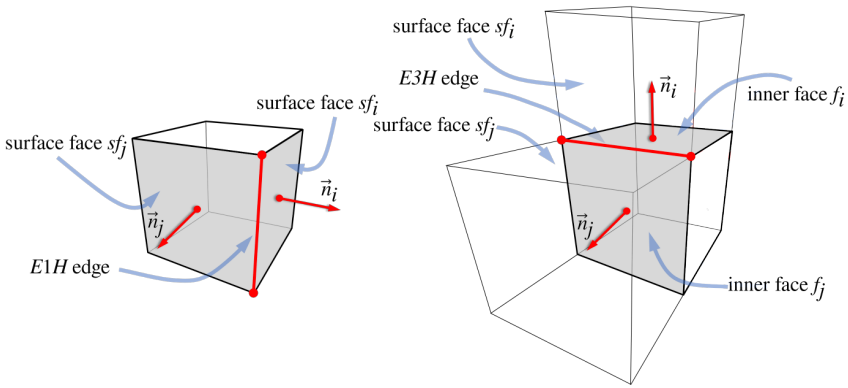
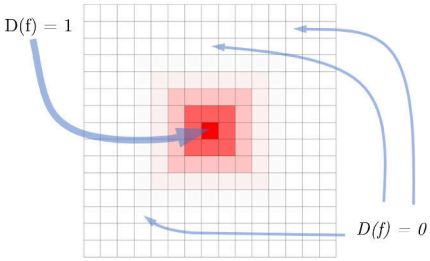


Figure 4.8: *Example of an $E1H$ edge (left) and of an $E3H$ edge (right).*

For each boundary singular edge, belonging to either $E1H$ or $E3H$, the dihedral angle between its incident boundary facets is computed as $\theta = \overrightarrow{n_i} \cdot \overrightarrow{n_j}$ (see Figure 4.8 to identify n_i and n_j), and the measure of how much it deviates from its ideal value (90° for the $E1H$ edges or 270° for the $E3H$ ones) is analyzed. For $E1H$ surface edges, $D(e) = \theta$ if $\theta \geq 0.5$ and 0 otherwise. For $E3H$ surface edges, θ is computed based on the two incident inner facets f_i and f_j, because they determine the angle to split in case of high distortion. The $D(e)$ valued is defined as $D(e) = |\theta|$ if $\theta \leq -0.5$ and 0 otherwise. Finally, a value $D(f)$ between 0 and 1 is assigned to the surface facets, to record how much it is necessary to extrude them in order to improve the quality of the mesh. The $D(f)$ value of a face is defined as the maximum distortion value of the four incident edges: $D(f) = \max_{e \in f} D(e)$. In this step, the areas of the mesh composed of only hexahedra with $SJ \geq \mathcal{T}$ are ignored.

4.4.2 Padded facets

Each facet of the polycube hex-mesh surface is now assigned a distortion value. During the experiments phase, it has been observed that the direct use of these values to determine the set of constrained padding facets HF is not ideal as they may form a fragmented set with isolated low distortion facets in the middle of high distortion patches, and vice-versa



(see Figure 4.9). To obtain a more consistent set of uniform patches of facets HF, a smoothly propagation of the distortion values is needed. Then, the HF set can be defined via the solution of a *Max Flow - Min Cut* problem [BK04]. To propagate the distortion values associated with the surface facets, a simple iterative flooding algorithm is applied, starting from facets with $D(f) \neq 0$ and filling the adjacent empty ones. The process is iterated for a maximum number of steps denoted by k in the following formula:

$$D(f_i)^{n+1} = D(f_j)^n \cdot e^{\frac{-n}{2}} \quad \forall f_i \in N_b(f_j), \quad 0 \leq n \leq k \tag{4.8}$$

where n is the current iteration, $e^{\frac{-n}{2}}$ is a term to favor a soft propagation of the deformation values (see Figure on the left), and $N_b(f_i)$ is the set of neighboring boundary facets of f_i. The described formula is applied only to facets with $D(f) = 0$, while the facets with a value different from 0 are not changed.

The *Max Flow - Min Cut* graph is defined by two nodes for the two used labels (L_1 for facets $f \in HF$ and L_0 for the other facets), a node for each surface facet, an arc between adjacent surface facets and an arc between facet nodes and label nodes. Then, the Max Flow - Min Cut problem is formulated as follows:

$$E(L) = \sum_{f \in SH} P_f(L_f) + \sum_{\langle f_p, f_q \rangle} P_{pq}(L_p, L_q) \tag{4.9}$$

where SH is the set of hexahedra with at least one face on the boundary. $P_f(L_f)$ represents a penalty for cutting an arc between a facet f_i and the label L_{f_i}. P_{f_i} is defined as:

$$P_{f_i}(L_i) = 1 - |l_i - D(f_i)| \tag{4.10}$$

where $l_i = 0$ for $P_{f_i}(L_0)$ and $l_i = 1$ for $P_{f_i}(L_1)$. Assume $D(f_i)$ is close to 1: $P_{f_i}(L_0) \simeq 0$ and $P_{f_i}(L_1) \simeq 1$. Therefore, it is convenient to cut the arc between f_i and L_0 and to assign to f_i the label L_1. Notice that, assigning the facet f_i to the label L_1 corresponds to inserting f_i in the HF set of hard constraints.

The right sum represents the penalty for cutting an arc between two adjacent facets f_p and f_q. In other words, it assigns a price to assigning

two labels L_p and L_q to two adjacent facets f_p and f_q and $L_p \neq L_q$. We define P_{pq} as:

$$P_{pq}(L_p, L_q) = [1 - |D(f_p) - D(f_q)|] \cdot [D(f_p) + D(f_q)] \tag{4.11}$$

where $[1 - |D(f_p) - D(f_q)|]$ measures the difference between $D(f_p)$ and $D(f_q)$, and $[D(f_p) + D(f_q)]$ favors cuts between arcs in low distortion areas.

The solution of the *Max Flow - Min Cut* returns a collection of facets ready to be used as hard constraints (HF) in the binary problem formulation described in Section 4.3. As it is shown by Figure 4.9, they are organized in consistent patches with neither holes nor isolated facets.

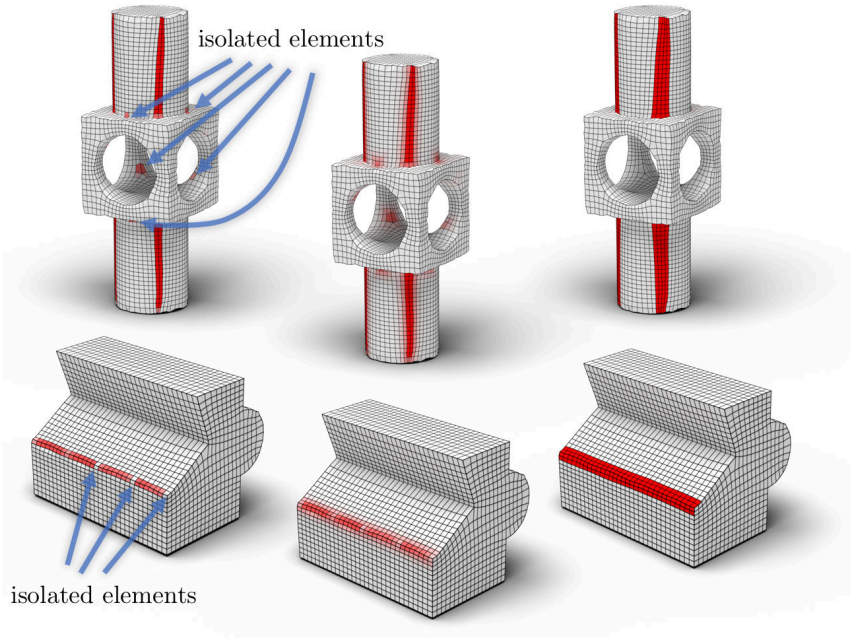


Figure 4.9: *Computing hard constraints. Left: $D(f)$ values associated to each facet after the first step; notice, on the top model the isolated high-distortion elements, on the bottom model the isolated low-distortion elements (pointed by the arrows). Middle: $D(f)$ values after the soft propagation. Right: the final constraints resulting from solving the Max Flow - Min Cut formulation, which have filled the interruptions.*

While the above steps may appear ad-hoc, they offer a robust way to produce homogeneous patches of hard constraints. The alternative

solution of computing one distortion value per facet and then use such values as soft constraints as been also considered. However, considering the negligible time required by the graph-cut solve step (fractions of a second), it has been decided to keep this method in order to provide the solver with a set of hard constraints. By using hard constraints obtained as described above, better results have been achieved (concerning both quality and singularity count) in considerably lower time. Moreover, by visual inspection of the hard constraints, the user is provided with the guarantee that they will be part of the final solution.

4.5 Results

The meshing pipeline described in Section 2.1 has been implemented to produce all the polycube-based hex-meshes, and the optimizer described in [LSVT15] has been used to optimize the results. For a fair comparison, the optimizer has been applied to the no-padded version and in the ones with global and selective padding. During the optimization, the *surface attraction* parameter has been set as high as possible, to preserve the original shape of the model. In this way, for each one of the models in this chapter, a maximum *Hausdorff distance* lower than 0.009 w.r.t. the bounded box diagonal has been obtained. In all the figures depicting results, the following color-code is used: the mesh separatrices are depicted in red, and the quality of the mesh elements ranges from green (good) to red (bad). The padding layer is shown in blue. For each model, the hex-mesh directly derived from the polycube is compared to the hex-meshes with global and selective padding applied.

Table 4.1 reports the most relevant data for the tested models. For each model, the following information are reported: the number of hexahedra ($\#H$), the number of singular vertices and edges ($\#S_v$ and $\#S_e$), and the minimum and average Scaled Jacobian (mSJ and aSJ). In particular, for the Chamfer and Lego models, the results obtained with different λ values are recorded, to measure the impact of λ over the final mesh. Moreover, for the "Double hinge" model, the version without padding the holes (NH) and the one with padded holes (WH) are reported. It shows that padding concave angles, sometimes, does not significantly improve the final quality. The proposed selective padding is also applied to the Test 1, 2, 4 and 5 models,taken from [WGZC18]. The sheet insertion operation takes less than a few seconds and, thus, it is not reported, since it is negligible.

Model	Original model					Global Padding					Ours (Selective Padding)					
	$\#H$	$\#S_v$	$\#S_e$	mSJ	aSJ	$\#H$	$\#S_v$	$\#S_e$	mSJ	aSJ	$\#H$	$\#S_v$	$\#S_e$	mSJ	aSJ	Time
Bearing	7362	64	988	.07	.95	12132	128	1052	.11	.86	8062	72	1020	.42	.97	0.9 s
Block	12408	48	944	.08	.93	17896	96	992	.15	.95	15216	56	1008	.69	.98	2.5 s
Chamfer ($\lambda = 0$)	4347	20	358	.10	.96	6197	40	378	.10	.94	4945	40	588	.60	.95	7.6 s
Chamfer ($\lambda = 4$)											5773	20	366	.74	.98	31.6 s
Chamfer (Fig4.1)	10354	20	486	.02	.96	13750	40	506	.13	.95	12121	28	614	.61	.98	40.6 s
Column	940	16	224	.12	.94	1790	32	240	.09	.91	1276	24	240	.81	.97	0.2 s
Double hinge (NH)	3120	24	424	.03	.94	5342	48	448	.31	.89	3770	40	536	.65	.95	3.1 s
Double hinge (WH)											4550	40	536	.63	.95	4.2 s
Gear	6816	72	796	.03	.96	10136	144	868	.03	.93	8640	72	812	.70	.98	8.2 s
Joint	9032	32	680	.09	.97	13868	64	712	.16	.95	9872	40	804	.67	.97	5.3 s
Lego ($\lambda = 0$)	8676	112	1828	.10	.94	18810	224	1940	.13	.77	9372	520	2596	.39	.94	2.3 s
Lego ($\lambda = 2$)											9876	160	1876	.54	.97	3.9 s
Lego ($\lambda = 4$)											9940	176	1900	.31	.95	8.9 s
Wrench	1576	32	472	.06	.95	3576	64	508	.14	.89	1796	40	492	.71	.97	2.7 s
Test 1	4272	72	832	.07	.95	7688	144	904	.03	.90	5080	72	852	.66	.98	4.06s
Test 2	4752	40	520	.06	.92	7026	80	560	.10	.87	6672	56	556	.73	.96	20.7s
Test 4	9779	56	812	.14	.97	14391	112	868	.17	.95	11730	56	824	.68	.99	2.6s
Test 5	50830	84	2266	.10	.95	73666	168	2350	.19	.94	53686	84	2370	.41	.98	49.8s

Table 4.1: *Statistics. Three versions of the same model are compared: input model, with global padding and with the proposed selective padding method.*

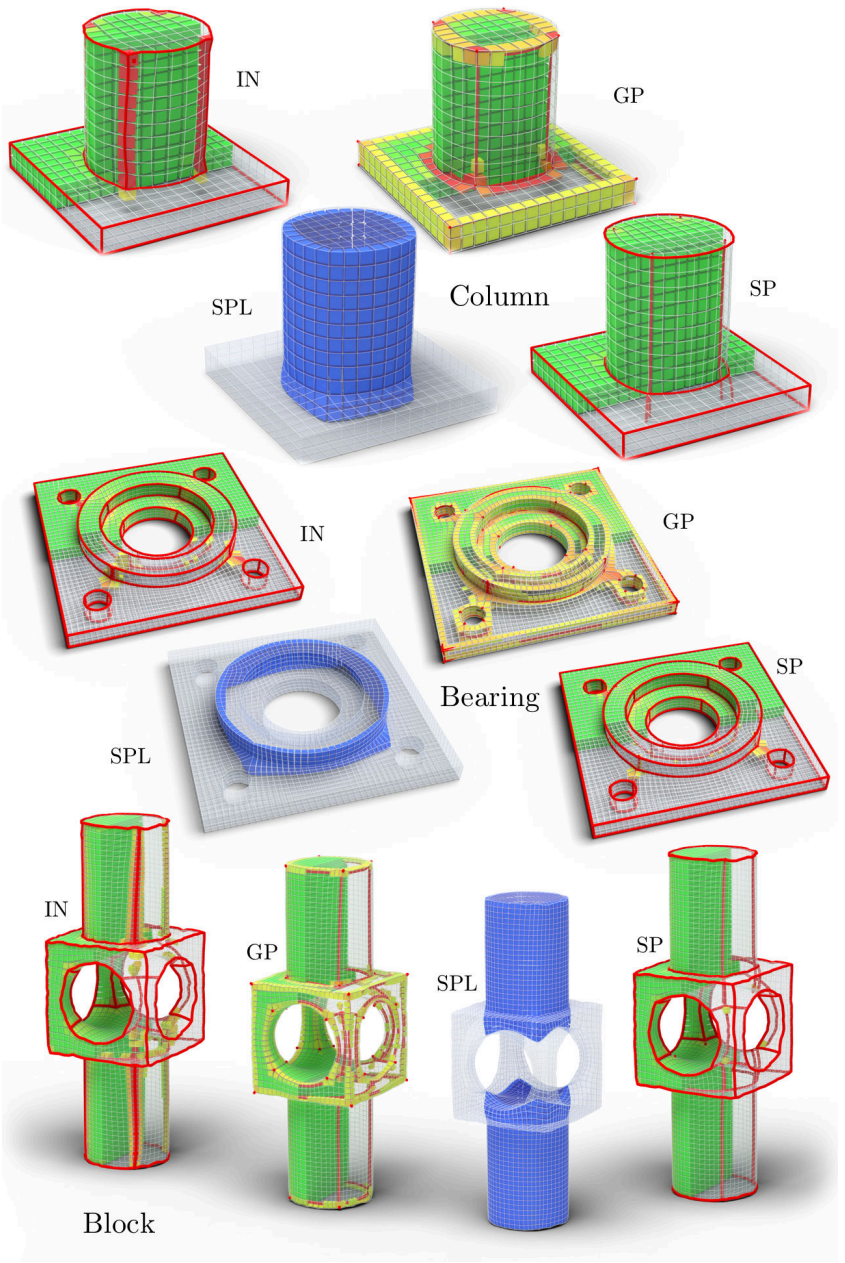


Figure 4.10: *Results, pt. 1. For each shape: the input model (IN), the global padding (GP), the new layer, in blue, of the selective padding (SPL) and the final model (SP). Colors indicates quality as described in Figure 4.2.*

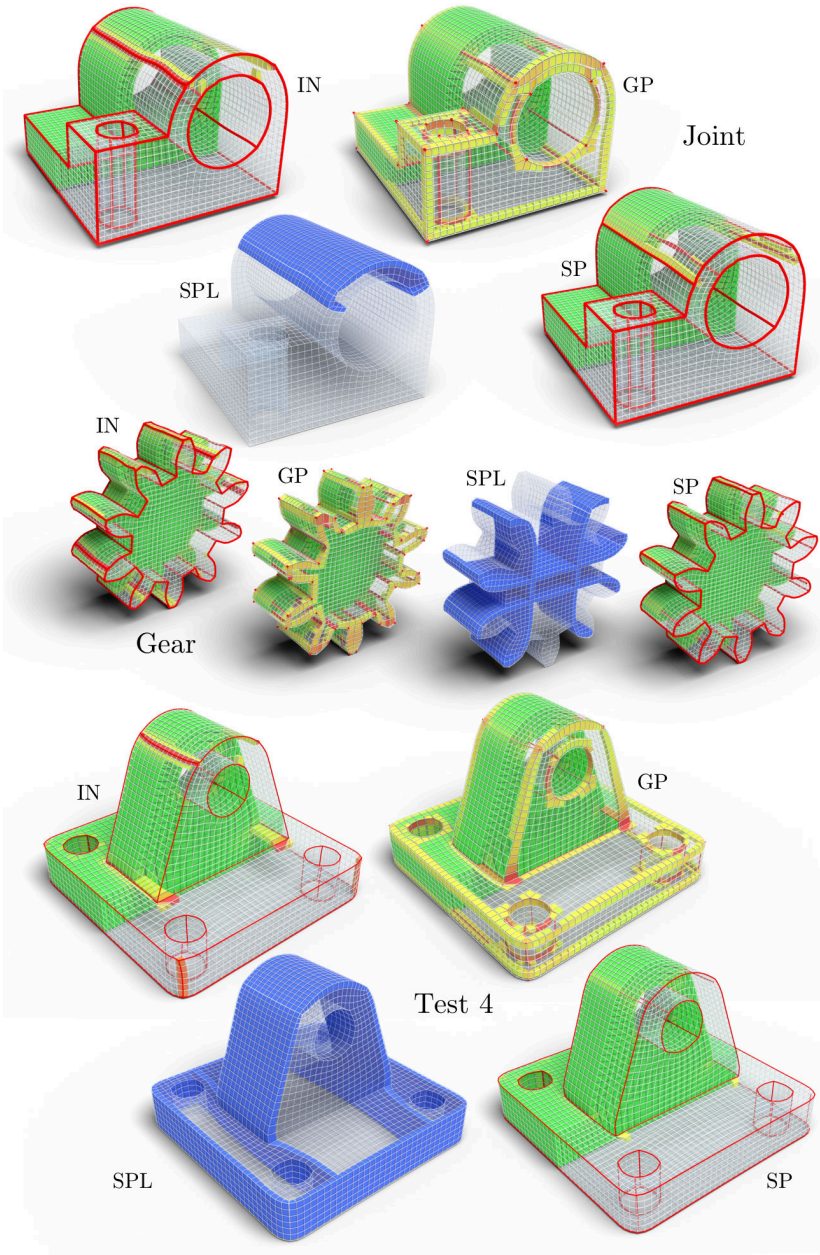


Figure 4.11: *Results, pt. 2. For each shape: the input model (IN), the global padding (GP), the new layer, in blue, of the selective padding (SPL) and the final model (SP). Colors indicates quality as described in Figure 4.2.*

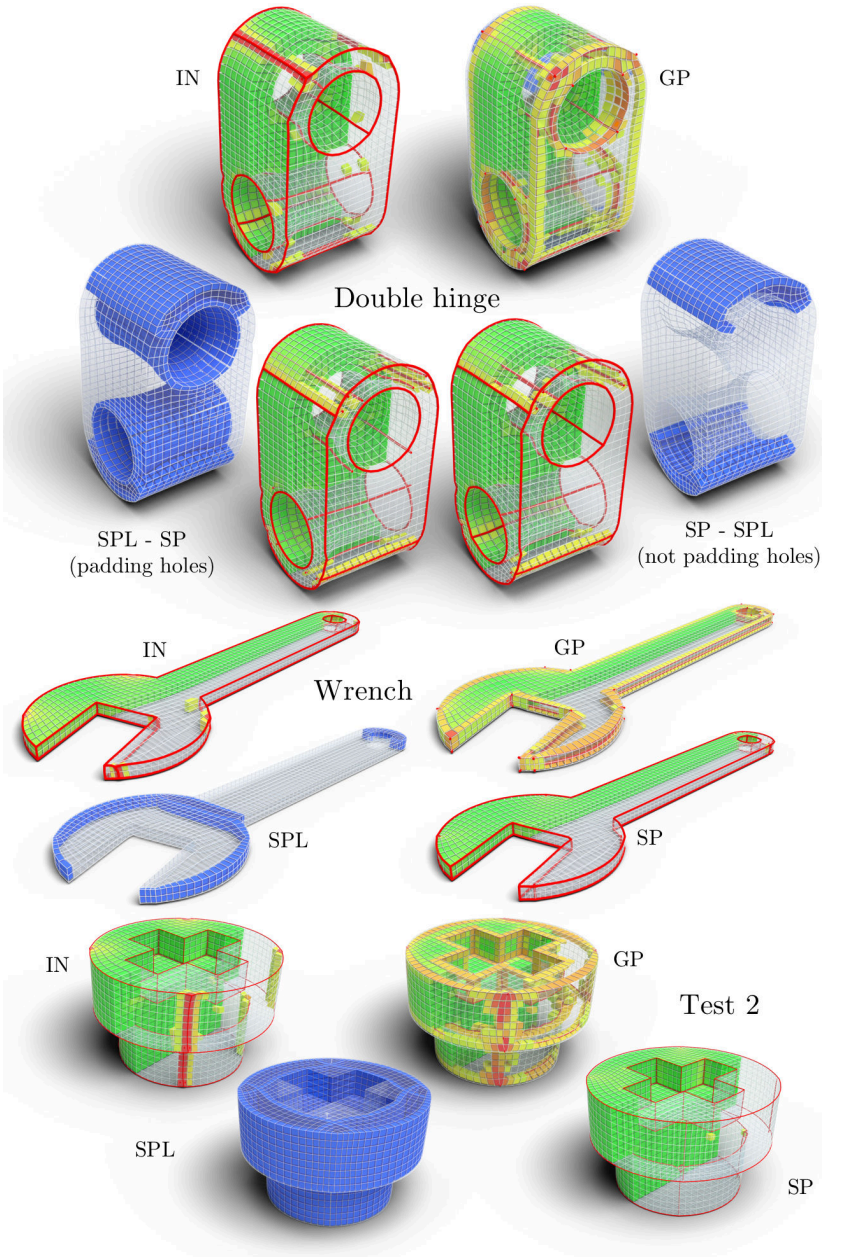


Figure 4.12: *Results, pt. 3. For each shape: the input model (IN), the global padding (GP), the new layer, in blue, of the selective padding (SPL) and the final model (SP). Colors indicates quality as described in Figure 4.2.*

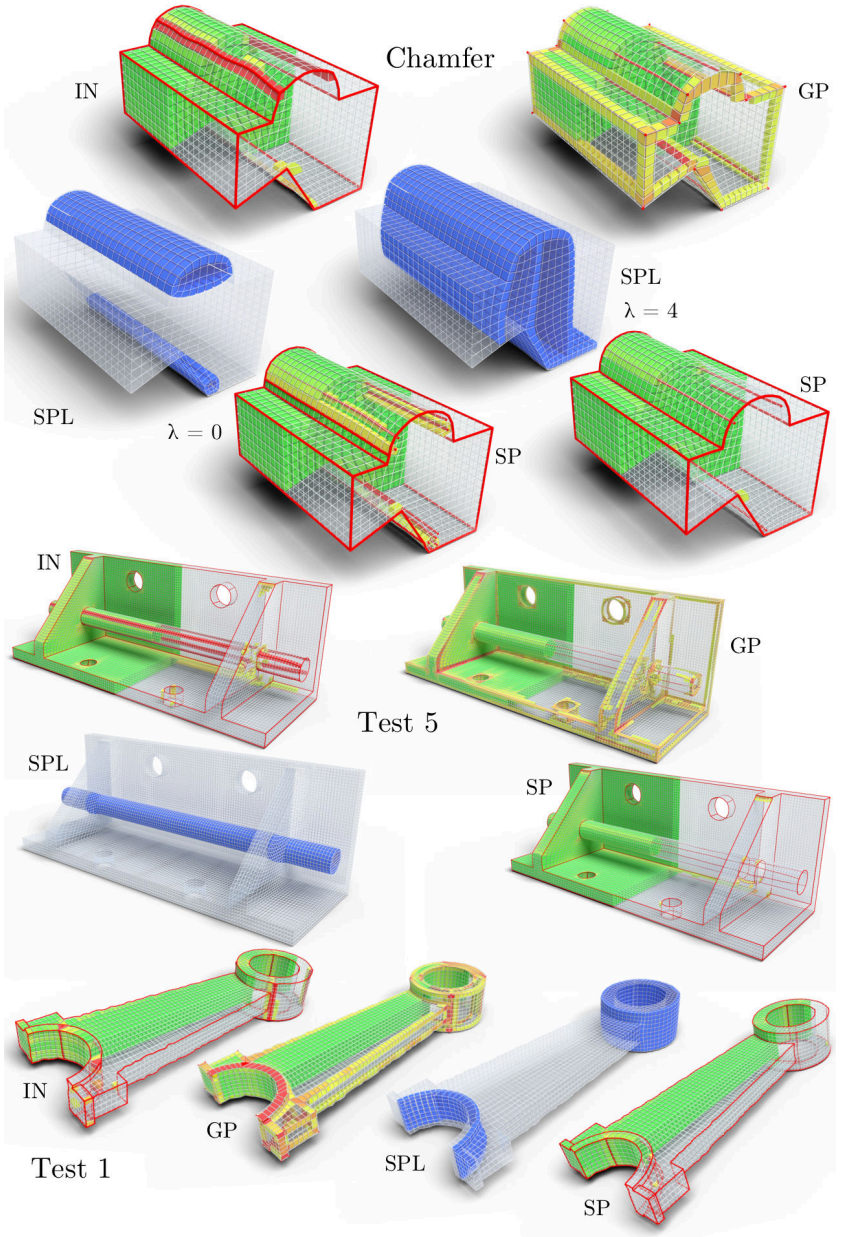


Figure 4.13: *Results, pt. 4. For each shape: the input model (IN), the global padding (GP), the new layer, in blue, of the selective padding (SPL) and the final model (SP). Colors indicates quality as described in Figure 4.2.*

As commented at the beginning of this chapter, the padding operation usually improves the global quality of a volumetric mesh. The results shows that the proposed selective padding allows to obtain a substantial quality improvement over the global padding. In Table 4.1, it is also possible to observe that, on the top of obtaining a better quality, the proposed method adds fewer extra elements than global padding. The only trade-off to pay is, sometimes, the increase number of singularities.

Table 4.2 records a brief comparison between the results of the proposed algorithm and the one described in [WGZC18], applied to the same domains. As we have not been granted access to the original software, the comparisons is performed on a series of results produced by their algorithm. Therefore, starting from the same shape, the proposed approach extracts and optimizes a polycube-based structured hex-mesh, while they analyze and optimize an unstructured one. This allows the described pipeline to start from a regular structure of good quality inside the shape, and apply the algorithm just on the surface. Comparable or better results are obtained for both the minimum and average Scaled Jacobian.

Model	[WGZC18]		Ours	
	mSJ	*aSJ*	*mSJ*	*aSJ*
Test 1	.35	.94	.66	.98
Test 2	.34	.88	.73	.96
Test 4	.64	.96	.68	.99
Test 5	.39	.89	.41	.98

Table 4.2: *A comparison between results obtained with the proposed selective padding and with the mehod proposed in [WGZC18].*

4.5.1 Extra elements vs extra singularities

The improvement of the hex-mesh structure requires to find the right trade-off between the number of extra elements and the number of extra singularities. The user-specified λ parameter controls the number of turns of the inserted sheets and therefore the addition of singular vertices to the mesh structure. The Lego model in Figure 4.14 and the Chamfer model in Figure 4.13 show the differences in the final result depending on the used λ value. With a low λ value, the solver can insert turns everywhere in the mesh structure, with the goal of padding as few facets as possible.

With a high λ value, the number of turns, and new singularities, is limited at the price of adding more extra elements. Finding the well-balanced λ value allows reaching a reasonable compromise between extra elements, extra singularities and final quality, as it is shown in the middle example in Figure 4.14 (with $\lambda = 2$).

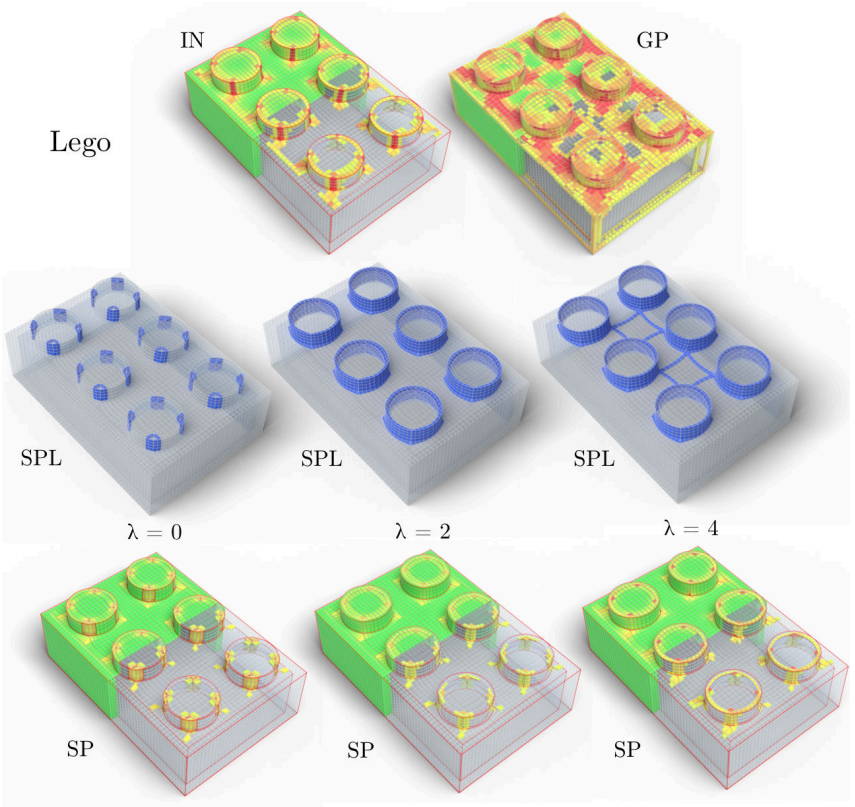


Figure 4.14: *Three versions of the proposed selective padding applied to the Lego model, with three different λ values.*

4.5.2 Timing

The solver's timing depends on the number of elements in the mesh and its shape. The solution space varies depending on the input structure of the mesh. A "coarse" hex-mesh is usually used to solve the padding problem, since a refinement step is always possible in post-processing (e.g., splitting each hexahedron in eight sub-hexahedra). When a model is

complex enough not to allow to start from coarse hex-meshes, the solver can require up to several minutes of computation to produce the set of facets to extrude.

4.5.3 Mechanical parts vs. organic shapes

As mentioned at the beginning of this chapter, polycubes are an ideal tool for generating hexahedral meshes of quite regular shapes with a limited set of details. In this class of objects, mechanical pieces and CAD models are a relevant subset. As it is clear from the shown results, the focus of this work is on this class of objects. Indeed, the application of the proposed selective padding on organic and free-form polycube-based shapes produces the same result of global padding. In Figure 4.15, the use of this algorithm is applied on the Bunny model. It is evident that, to improve the mesh quality, it is needed to push inside all the singular edges of the mesh.

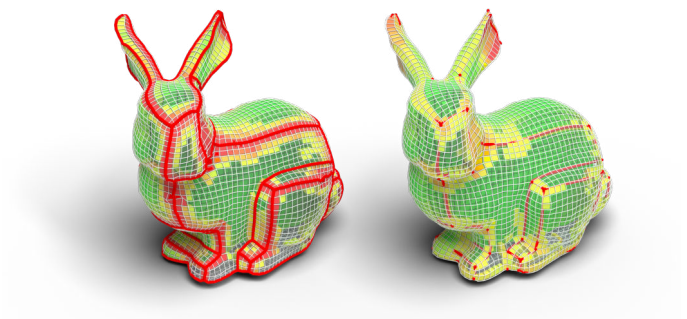


Figure 4.15: *Selective padding in organic shapes. On the left the model without padding, on the right the selective padding (equivalent to the global one) applied on the same model.*

4.6 Limitations

While the proposed experiments show excellent results, improving the quality of both the original and the global padding models, it is not possible to prove that this approach reaches the maximum possible quality for the chosen application field. Searching for the best trade-off between complexity and distortion would require either trying all possible λ values and then selecting the one yielding the best results, or proceeding by dichotomy. According to the carried out experiments, the value of the λ parameter which leads to the best results depends on the shape of the

model. It is, thus, not possible to suggest the silver bullet value of λ that could work for any model. The choice of λ can be a fine-tuning task and the user is let to set it interactively.

The selective padding approach can detect distortion in the proximity of all concave tunnels in models with genus greater than 0. For simplicity, these particular shapes ar refereed as "Holes". However, padding the holes is not always relevant for improving the overall quality. Experiments carried out on the *Double hinge* model show that the quality can even decrease.

Technical information

The Cinolib library [Liv17] has been used as data structure to store both models and polycubes. Gurobi [Gur] has been used as numerical solver. Both the Tetgen [Si15] and Tetwild [HZG*18] tet-mesh generators have been used to turn tri-meshes into tet-meshes and the Polycut algorithm [LVS*13] to generate all the polycubes shown in the chapter.See Appendix A for more details. A special thanks goes to the authors of [WGZC18], for sharing the models they used in their article.

Part III

Polycubes for fabrication

Chapter 5

Digital fabrication

Fabrication is a fast emerging field of research in geometry processing. It collects the study and the implementation of all the processes and techniques that can be used to produce real objects from digital models. Novel algorithms and techniques have flourished, to let almost everybody reliably 3D print accurate and cheap reproductions of digital objects. At the base of this explosion, there are many manufacturers which are selling low-priced entry-level 3D printers, leading to a sound diffusion between hobbyists.

The most relevant technologies are, but are not limited to, 3D printers and CNC milling machines (see Figure 5.1). These represent two different approaches to fabrication, usually referenced as *additive manufacturing* and *subtractive manufacturing*. Research in the fabrication field is mostly related to additive manufacturing, while a few works address subtractive manufacturing. This chapter briefly introduces the principal features and limitations of additive and substractive manufacturing, and the most relevant literature regarding both topics.

5.1 Additive manufacturing

Additive manufacturing (or additive fabrication) makes use of machines that build the final object layer by layer. These machines are the *3D printers*, and they can use multiple materials. The most common printers use thermoplastic polymers and deposit the fused filament to build the layer. However, other technologies are available to use other materials, such as liquid resins, metals, and various powders.

This kind of manufacturing does not impose any constraint on the model's shape. However, some models require external *support structures*,

Figure 5.1: *An example of a 3D printer and a CNC milling machine. The models in the picture are the machines we have in our laboratory and we use to test our works in the fabrication field.*

as 3D printers can not directly print steep overhangs or islands (see Figure 5.2). One has to remove these structures manually after printing, and in an industrial context, they represent a significant waste of material and time. To avoid this waste, Hu et al. proposed in [HLZCO14] an algorithm to subdivide the model in *approximate pyramidal shape*, printable without supports. Herholz et al. in [HMA15] suggest a similar approach, by exploiting the surface deformation to reduce the number of pieces.

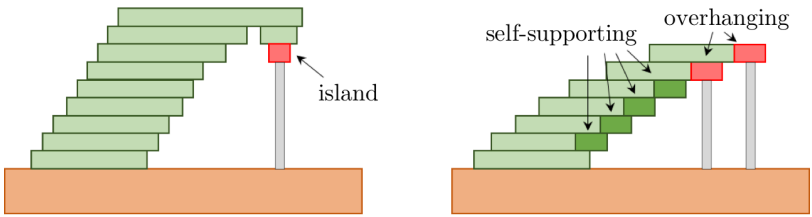


Figure 5.2: *Islands and overhangs need external support structures.*

The hardest constraint imposed by 3D printers regards the size of the object, as it is undoubtedly impossible to print anything greater than the printing chamber. The solution to this problem is, again, to partition the model into smaller portions, print them, and reassemble them back.

Many works that face this problem appeared in the last years, and the most remarkable ones are in [LBRM12], [SFLF15] and [HFW11]. The algorithm proposed by Song et al. in [SFLF15] creates self-interlocking structures, to avoid the use of glue or connectors, and it obtains a stable structure that can be disassembled and reassembled multiple times. The algorithm proposed by Hao et al. in [HFW11] tries to minimize the aesthetic impact of seams. Lastly, the algorithm proposed by Luo et al. in [LBRM12] generates a partitioning of the input model that optimizes

a set of objective functions, including *printability* of every block in the working volume, *assemblability*, avoiding small blocks and optimal position of the seams (both for aesthetics or structure). They subdivide the model using cutting planes, and a BSPTree gives the order of cut. This approach does not allow keeping apart semantically separate portions of the object. Furthermore, they don't consider the supports that are necessary to print every block of their partition, which can complicate the assemblability due to the presence of connectors in the planar portions of the blocks.

An extensive discussion on pros and cons of additive fabrication, partitioning and related issues can be found in the survey of Livesu et al. [LEM*17].

5.2 Subtractive manufacturing

Subtractive manufacturing (also known as machining or subtractive fabrication) consists in removing material from a starting block until only the desired shape is left. CNC milling machines have a crucial role in mechanical manufacturing since decades, but only recently experiments on automatic free-form shape production started. Milling, unlike 3D printing, enables manufacturing objects with a large variety of materials, like wood, metal or stone. Despite this significant plus, the usage of subtractive techniques for free-form production still struggles in the digital fabrication field, due to the hard constraints that they impose on the geometry of the objects. There are three main categories of milling machines, which differ for the degrees of freedom of the milling tool.

5.2.1 3-axis

The most diffused, inexpensive and easy to use milling machines can move their tool on the three axes of the Cartesian system and, thus, they have three degrees of freedom. These characteristics limit the class of objects they can produce: millable shapes can be only *height-fields* with flat bases. In other words, each line parallel to the z axis can cross the shape only once, as shown in Figure 5.3. Even if there is a vast bibliography on the production of mechanical parts with CNC manufacturing, there is still limited literature on the subject of decomposing generic free-form shapes into a set of millable parts. Alemanno et al. [ACP*14] define a user-assisted method for decomposing 3D shapes into height-field in the domain of cultural heritage. Their method is manually driven and overlaps between pairs of blocks are resolved by using an interlocking zipper pattern. Herholz et al. [HMA15] use 3-axis milling machines to create millable molds. These molds can be used to obtain the final model by

solidifying a liquid material which decants inside the glued molds. Muntoni et al. [MLS*18] perform a decomposition in height-field blocks that can be manufactured in a single pass with a 3-axis milling machine, using a fully automatic algorithm in two steps. They first identify all the bounding boxes containing height-fields and then select a subset determining a partition of the input shape. As they explain, a geometry manufacturable with a 3-axis machine must respect several constraints (such as height-field geometry w.r.t. a given direction, flat polygonal base, etc.).

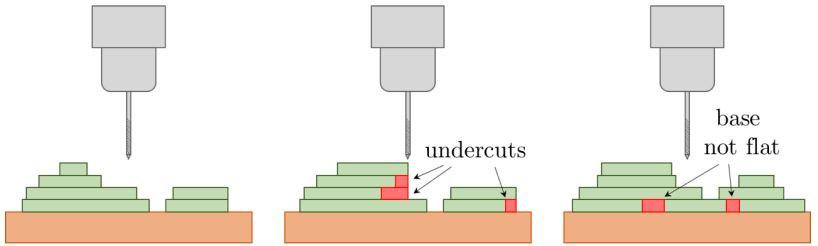


Figure 5.3: *The model on the left is millable (it is a height-field); the model in the middle is not millable (it is not a height-field and, thus, it has undercuts), the model on the right is not millable even if it is a height-field because its base is not flat.*

5.2.2 4-axis or more

More complex machines have higher degrees of freedom, typically moving the tools over four, five, and six axes. These devices impose looser constraints over the machined shape, but, at the same time, they are more expensive than the 3-axis ones, and they require more sophisticated software and analysis which enables the automatic generation of tool-paths. Typically, in fact, the user generates the tool-paths manually based on his or her own experience. It is also possible to add accessories to a 3-axis machine having a 4^{th} degree of freedom given by the rotation axis. This add-on is quite useful since a 4-axis machine can produce all the models that, given a rotation axis, exposes every point of the surface in at least one rotation. This constraint is weaker than the one imposed by the 3-axis machines. Recently Hou et al. [HF17] improved the results obtained previously by Frank et al. [FWJ06], and using the global visibility map *(GVM)* of the shape can determine the best rotational axes for machining it. The authors show results obtained on mechanical parts and the computational effort reported is in the order of tens of minutes for shapes just more complex than a cube with pockets.

In the next chapter, a novel method to compute a shape partitioning, with fabrication purposes, is proposed. The decomposition of a 3D digital model is induced by its polycube, and the fabricability of every piece of the decomposition is analyzed, for both additive and subtractive technologies. This analysis also includes a checking tool, developed to verify the manufacturability of the parts with a 3-axis milling machine.

Chapter 6

Fabrication oriented shape decomposition using polycube mapping

The introduction of cheap and small 3D printers and milling machines has boosted the research in the field of digital model representation for fabrication. As smaller and cheaper these machines are, as fewer functionalities and features, compared to the high-level ones, they have. Noticeable difference are the *size* of the printing chamber and the milling volume (the maximum fabricable volume). The only possibility to print big objects is, thus, to decompose them into multiple portions, fabricate them separately and, later on, reassemble the object.

In both 3D printing and machining, other essential constraints apply to the *shape* of the object, and a way to bypass them is to subdivide the object into pieces that satisfy the required features:

- a 3D printer cannot produce, without introducing extra-structures, *the supports*, parts with an overhanging larger than a fixed amount, usually set at 45°;
- a 3-axis milling machine can produce only parts which are height-fields with a flat base;
- a 4-axis milling machine can produce parts which are radially height-fields but machining a piece at a time.

A solution to this class of problems is a shape decomposition guided by the above constraints and size constraints. One can obtain a straightforward decomposition by using cutting planes to fit each part in size, but

it would probably be meaningless in shape. Moreover the cutting planes are keen to cut other portions of the shape in an uncontrolled way. On the other hand, it could be difficult to control the size of parts obtained using fancy and efficient decompositions which take into full account the semantics of the shape.

In this chapter, a simple and low-cost, computational wise, approach for the decomposition of a three-dimensional shape is proposed, which passes through its parametrization in the polycube space. As explained in Chapter 1, polycubes have to respect three main constraints: axis-aligned faces, only 90° dihedral angles and integer coordinates. These restrictions cause the polycubes to be a simplified representation of the original mesh which can catch the low-frequency semantics of the shape. Using the state-of-the-art polycube generation software Polycut [LVS*13], it is possible to control the decomposition of the input shape fine-tuning the parameters which determine the compactness and fidelity of the result. This choice allows producing decompositions in a time varying from seconds to few minutes.

The main contributions of the work presented in this chapter are:

- A pipeline to obtain an object decomposition that is guaranteed to be *printable* with a 3D printer of a given chamber volume without or with reduced use of supports.
- A checker to verify if the obtained decomposition could be *milled* using a 3-axis or a 4-axis milling machine.

It is worth to mention again that the proposed pipeline strictly relies on the preliminary decomposition induced by the polycube mapping. Without an efficient polycube generator, the whole process does not hold.

6.1 Overview

As introduced in Chapter 5, several works have the aim to decompose a 3D shape for the fabrication process. The aim of the work proposed in this chapter is to decompose complex models into simpler parts that better suit limitations in current fabrication processes, both additive and subtractive, by using a polycube-based decomposition. As well stated in [LEM*17], when planning the production of an object in additive manufacturing, it is possible to decide to partition the object into multiple pieces. This partition can be due to multiple reasons, but one of the most common situations is when the object is greater than the printing chamber.

In this work, an alternative way of partitioning the shape is proposed. This method is conceptually simple if a polycube map is available for the

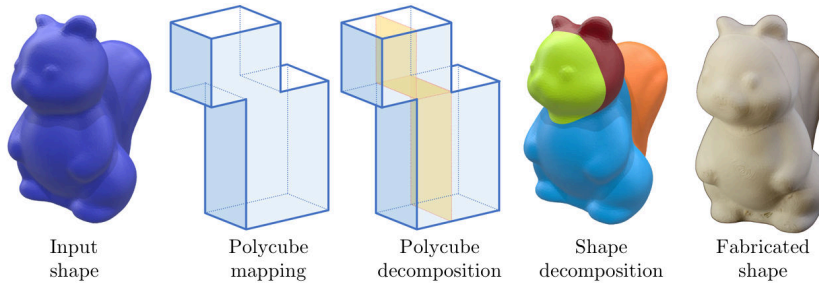


Figure 6.1: *The decomposition pipeline, from left to right: the mesh representing the input shape; the polycube computed from the input model; the polycube decomposed in orthogonal parallelepipeds with a sweeping algorithm; the decomposition mapped back onto the input shape and the final fabricated real object.*

input shape. It then induces a partition on the shape by using this map, smartly and efficiently. In the reported results, it is clear that the proposed method worked satisfactorily for a variety of examples (see Section 6.6).

The proposed decomposition algorithm requires only one parameter: the *compactness* term of the Polycut algorithm for building the polycube (see Section 6.6). Furthermore, the shape is partitioned keeping in mind the requirements of both additive and subtractive manufacturing.

The proposed production pipeline can be summarized as follows:

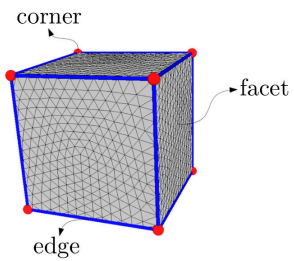
1. It start from a 3D input shape (a triangle mesh representing the surface of the model and a tetrahedral mesh for the interior);
2. The polycube representation of the input shape is computed;
3. The polycube is partitioned in orthogonal parallelepipeds;
4. The partition computed at step 3. is used to subdivide the original model in the shape space using *boolean operations*.

All the steps listed above are fully automatic. If the results do not respect the constraints (45° maximum overhang for printing and height-field for milling), it is possible readily repeat the last step after manually splitting one or more parallelepipeds in polycube space with a plane orthogonal to the Cartesian axes. The splitting is trivial working in the polycube space.

The following sections explain the third and fourth steps of this pipeline which are the primary focus of this work. Figure 6.1 shows is a sketched

representation of the proposed pipeline. The third step of the pipeline, explained in details in Section 6.2, takes as input the topology of the polycube and, using a queue-based sweeping algorithm, outputs its partition in orthogonal parallelepipeds. The fourth step (Section 6.3) maps back each parallelepiped found in the previous step to parts of the original model. It is not possible to use a simple mapping since it would not produce the flat surfaces we need. Thus, the boxes are used as parameters for intersections with the original shape. The final result is manually evaluated to verify if one or more parallelepipeds needs further partitioning step.

6.2 Polycube partitioning



As shown in the inset on the left, in the polycube, a *corner* is a vertex with at least three adjacent triangles (or faces) having three different normals and an *edge* is the shortest rectilinear path of triangle edges that connect two corners. A *facet* is a closed chain of edges and corners containing a set of triangles with the same normal. For the following steps, the triangle mesh structure can be ignored, focusing only on the elements as mentioned above.

The primary step of the pipeline is the decomposition of the polycube in orthogonal parallelepipeds. The idea that every concave edge in the polycube defines a partial decomposition of the model is followed. Since every edge is axis-aligned, it lies at the intersection two planes parallel to xy, yz or zx. By construction, the two planes are orthogonal. The intersections between the two planes and the polycube induce the partial decomposition mentioned before. Iterating the decompositions obtained visiting all the concave edges allows obtaining a decomposition in orthogonal parallelepipeds of the input polycube.

Firstly, it necessary to make sure that each corner coordinates is rounded to integer values. In this way, by fitting the polycube into an integer lattice, it is simple to create a uniform discrete grid inside the polycube. By applying a sweep line algorithm along all the three axes the lattice is split at every concave edge. In Figure 6.2, for the sake of compactness, both steps are represented as they were only one pass; notice that the concave edges are evidenced by marking the explementary convex angles.

This method works fine for all polycubes except for self-intersecting ones. In the carried out experiments, however, it never happened any case

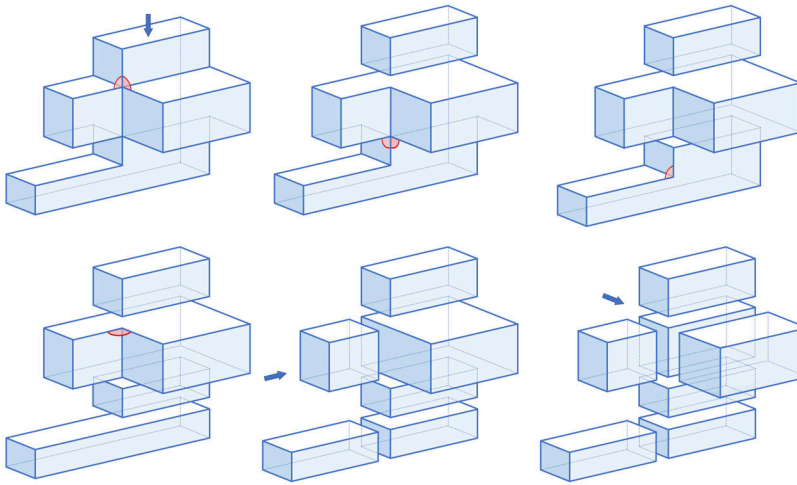


Figure 6.2: *The space sweeping partitioning of the polycube. A step in the direction marked by the arrow (top-down) is depicted: any time one or more edges delimiting an internal concave angle (in red the explementary convex angles) are encountered the polycube complex is split.*

of self-intersection, confirming the intuition that those cases should be extremely rare in the class of polycubes that is relevant in this practical scenario.

The cutting planes can now be computed. The reverse mapping, from polycube back to the original shape, is applied only to each part's corner. The tetrahedral version of the original shape and the polycube are used for the mapping. For each corner $P(x, y, z)$ on the surface of the part, it is required to determine in which tetrahedron of the polycube it lies. This operation is performed by indexing them by using an octree, and expressing its position in barycentric coordinates: $\omega_0, \omega_1, \omega_2, \omega_3$. The new corner position P' is now computed by applying these barycentric coordinates to the tetrahedron in the original model. If A, B, C, D are the vertices of the chosen tetrahedron, then $P' = \omega_0 \cdot A + \omega_1 \cdot B + \omega_2 \cdot C + \omega_3 \cdot D$. The computational complexity is $O(n_c \cdot log(n_t))$, where n_c is the number of corners in the polycube portions and n_t is the number of tetrahedra in the original model.

In this way, a set of eight vertices for each internal parallelepiped of the polycube is identified. It is now possible, for each quadruple of vertices on a face of the parallelepiped, to compute the plane that better approximate them and, repeating it for all the internal parts, obtain the set of cutting

planes. The following section explains this step.

6.3 Cutting planes

To be able to fabricate each part of the decomposed model, as it will be evident in the next section, it is beneficial for 3D printing, and mandatory for 3-axis milling having at least one *side* planar. The word *side* here means the set of triangles mapped from one facet of a parallelepiped. It's worth to remind that in polycube space, since each component is an orthogonal parallelepiped, each face is a planar rectangle.

To map back the parallelepiped in $\mathbb{R}^3$, the inverse function of the projection in polycube space is used. This inverse function only rarely maps a rectangle onto a planar portion of the surface: almost always the four vertices of the rectangle do not lie on a plane. Thus, the position of these four vertices are modified, so that they will lie on a plane. This step is called *flattening*.

It is not possible to change the position of the mesh vertices on the original surface of the input shape, of course, since a deformation of the input shape is not an appreciated result. To reach this goal, an iterative method that works in $\mathbb{R}^3$ is used, taking advantage of the polycube topology and, thus, in the following, the term *side* is used instead of facet for making this clear. It works as follows, using only a queue $\mathcal{Q}$ as the data structure to support the process:

1. Pick an external side of the model;
2. Check if one or more of the other five sides of the same parallelepiped are internal sides ;
3. Put all the internal sides found in $\mathcal{Q}$;
4. Take the first side in $\mathcal{Q}$, say f, flatten it and remove it from $\mathcal{Q}$;
5. Move to the parallelepiped incident on f not already visited and go to step 2;

The process ends when all the cubes have been visited and $\mathcal{Q}$ is empty.

6.3.1 Side flattening

Different approaches are used to flatten a side, depending on how many vertices are free to move.

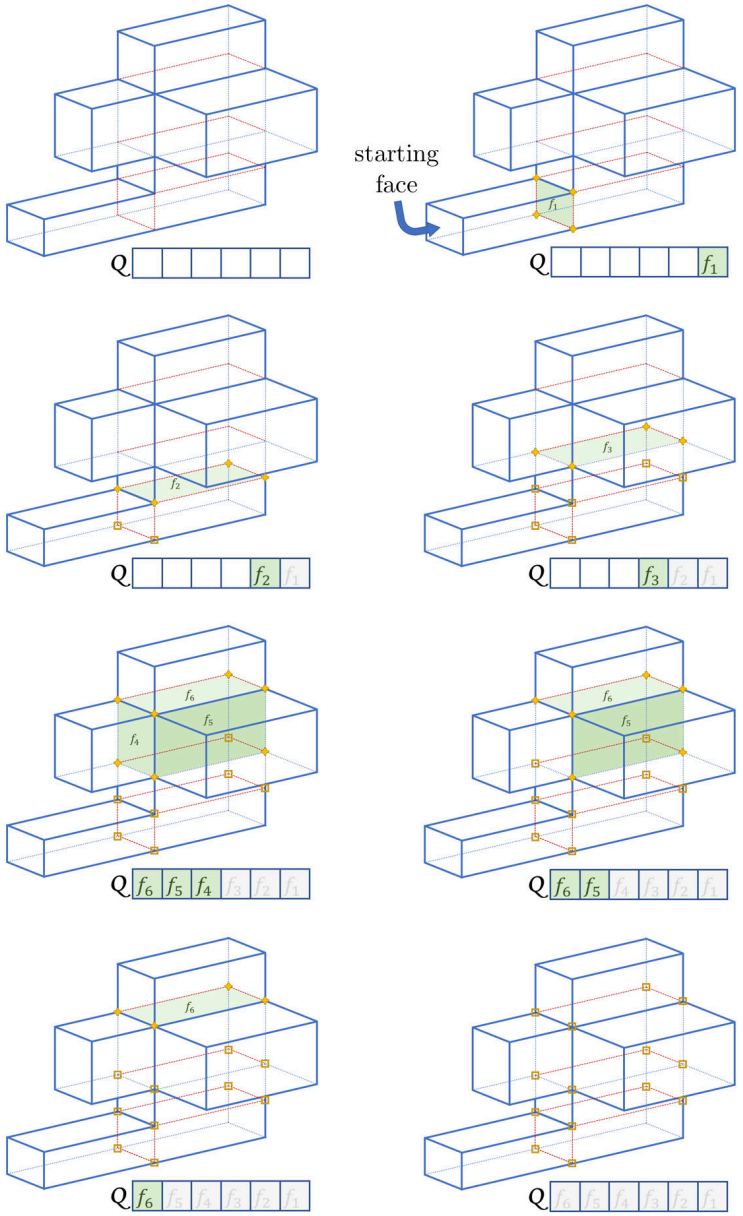


Figure 6.3: *From left to right and top to bottom, the whole process of flattening is shown. The vertices to be moved are the ones marked in yellow rhombus, once fixed they are marked with a yellow square.*

If all the *four* vertices are free to move (e.g., for the very first side to flatten) a least square method is applied to find the best fitting plane, and the four vertices are projected on it. There are two more possible cases: (i) *two* vertices can move only on a given plane; (ii) *one* vertex can move only on a given line. In both cases, the solution is straightforward. Finally, consider that when sides adjacent to already planar ones are flattened, since it is not possible to move the edge in common, it is used as a pivot and allows the other two vertices only lay on a plane passing through the pivot. In Figure 6.3, the whole process on a simple example is illustrated. For simplicity, it is in polycube space.

6.4 Final decomposition

The last step is the definition of the geometry of each part of the partition, in order to proceed to the fabrication feasibility analysis. To perform this refinement, the exact boolean operations has been used, as described by Zhou et al. in [ZGZJ16], which permit to obtain the surface mesh resulting from a boolean operation (intersection, union, difference) of two surface meshes. The final geometry of the part is obtained by performing an intersection between the surface extracted from the input triangle mesh and a box enclosing the part. The boundaries of the enclosing box are given by the cutting planes of the part and the bounding box of the whole mesh.

If the cut results in more than one connected component, only the component containing the four vertices generating the cutting plane is selected, and the other cuts are ignored. Each portion stems from a single orthogonal parallelepiped of the polycube, and therefore it can be trivially split, if necessary, by using an appropriate axis-aligned plane that will map back onto the original shape. The result of this final step is a partition consisting of a set of triangle meshes.

6.5 Feasibility checking for fabrication

6.5.1 3D printing support control

Each piece of the resulting partition can be fabricated with a 3D printer, since the chamber size is used as a control in the last step. The focus of the feasibility check is on the usage of supports.

The proposed algorithm is guaranteed to generate a set of pieces having from one to six flat polygonal facets. For each piece, all the possible printing directions (one to six), given by its bounding cutting planes, are checked, and the printing orientation that gives the best results in terms of needed

supports is selected. The best orientation is the one with less surface triangles exceeding the overhang angle of the 3D printer. Since the number of cutting planes bounding a piece is at most six, the overall complexity of the check is linear with respect to the triangles number of the piece. The next section analyzes the results.

Notice that, as stated in Section 6.4, with the proposed algorithm, a piece can be split with a new cutting plane if the volume of the part exceeds the chamber size. This operation permits also to avoid any outer supports iterating this split step until having only parts not needing supports. Whit this splitting, a new flat base is added to each of the two new pieces (one of them has already a flat face), allowing to choose a new printing direction for both pieces.

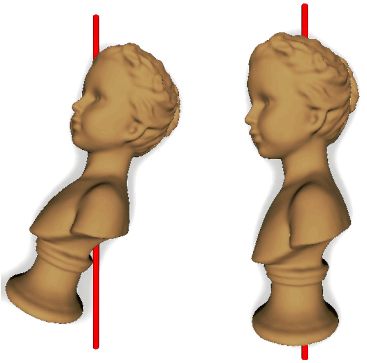
6.5.2 3-axis milling checking

While for 3D printing it is possible to guarantee results, for milling it is only possible to perform a check on the obtained decomposition to verify the fabrication feasibility. Two different checkers are devised, which allows to state if a block can be manufactured with both 3- and 4-axis machinery. The 3-axis milling checker is very simple: it just checks if a piece of our decomposition is a height-field. The milling direction is orthogonal to one of the cutting planes, and, again, at most, this check is performed six times per piece. The piece is oriented in all its possible milling directions and, for every orientation it is checked if it has triangles having normal with an angle greater than 90° for the milling direction. All the triangles which belong to the polygonal base generated by the selected cutting plane are excluded from the check step.

6.5.3 4-axis milling checking

This section introduces a method for checking if an individual part of an object can be feasibly machined with a 4-axis milling machine.

One of the main challenges for the fabrication of an object with a 4-axis milling machine is the identification of the rotation axis. This problem has been achieved by [HF17], but they focus the study mostly on mechanical and very regular objects with a low number of triangles, emphasizing that the computational effort dramatically increases if the visibility resolution raises. The proposed subdivision



approach focus on free-form geometries and, thus, the more reasonable solution is to leave the user to choose the rotation axis, assisted by an automatic initial orientation of the model. The suggested orientation is derived, as explained in [MLS*18], maximizing the alignment between the global axes and the face normals of the shape. As shown by [MLS*18], since this method is a heuristic it can fail. As it is evident in the inset on the right, the rotation axes automatically detected for *BU* (left) is skewed while the manual choice of taking the vertical line passing through the center of the base (right) is the correct one. Therefore, a tool that allows the user to adjust the orientation, if he deems it is necessary, is provided. This is the only user-controlled step of this checker.

A surface can be (theoretically, see the end of section) manufactured using a 4-axis milling machine if the milling tool can reach every surface point considering all the possible rotations of the model along the selected rotation axis. Since all the possible rotations along the 4^{th} axis are infinite, once the rotation axis is chosen, it is possible to sample with a small set of angles that generates a family of planes which intersect each other along the selected axis. Then, for each triangle and at every rotation of the model, a ray orthogonal to the plane associated to the i-th rotation and passing through the barycenter of the triangle is traced. If the ray intersects more than one triangle, the farthest away from the plane is marked as visible from the milling tool and, therefore, millable in the present orientation.

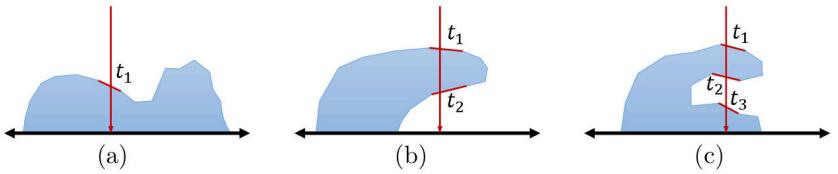


Figure 6.4: *Three main possible cases of ray-triangle intersection.*

As it is clear from Figure 6.4, three primary cases can present:

- In case (a) the ray traverses only one triangle.
- In case (b) the ray traverses a whole portion of the shape and two triangles (one front-facing and one back-facing the milling tool).
- Case (c) is an example of summation of both previous cases, that can sum up even more; in all these cases the ray traverses three or more triangles.

In the two cases sketched in Figure 6.4(b) and in Figure 6.4(c), a triangle belonging to the surface and not directly reachable by the milling tool is present. If this happens, it is needed to verify if for some rotation angle the triangle could be visible. Is is not possible to mill all the triangles never visible. In this case it is possible to change the axis, or remove the triangles from the surface, filling the holes with a mesh repair tool.

Checking the visibility of the barycenter is an approximation. If the barycenter is visible, it does not guarantee that the entire triangle is visible from the milling tool. However, this approximation is good enough for the purposes of this work, because in the worst case precision for portions of triangles is lost. In the carried out experiments, these problems never appear. The checker has been tested with oversampled meshes (keeping the same geometry but doubling or tripling the number of triangles), resulting with the same percentages of samples visible/not visible, with minimal differences (less than 0.1%).

As mentioned before, the selection of the rotation axis is user-assisted. The sampling number is an input parameter. In the reported experiments (the results are in Table 6.2) forty rotation planes uniformly distributed in the interval $[0, 2\pi)$ have been used.

The whole checking process does not take into account some practical details like the thickness and the height of the milling tool. They depend on the machine used for the manufacturing process. Even if these are essential aspects for the real feasibility of the fabrication of the pieces, they can be integrated into the checker using a configuration left to future extensions.

6.6 Results and analysis

The proposed pipeline has been applied to an extensive set of models, having different characteristics in term of details and complexity. A gallery of results is shown in Figure 6.5. Other peculiar or fabricated results are reported and commented separately in this section.

The polycube of each model is an input for the proposed pipeline and, thus, to make their production process clear, in Table 6.1 are reported, for each model, the compactness factor used to compute these polycubes with Polycut, with the number of resulting orthogonal parallelepipeds. The number of parallelepipeds in the polycube is the number of parts of the decomposition since any kind of merging post-processing step is not used. The timing for the cutting planes calculation is not reported because they are negligible (always less than a second).

Model	Polycube compactness	Parts
Angel	5	4
Bird	4	5
Bu	9	9
Dea	7	4
Duck	5	5
Fandisk	4	50
Hole3	3	18
Max Plank	7	3
Moai	3	9
Ramses	9	9
Sphinx	10	6
Squirrel	3	4

Table 6.1: *Results of the polycube computation on the models shown in in this chapter. The polycubes structure is optimizes with the algorithm presented in Chapter 3. The second column lists the compactness values of the polycubes.*

Also the timing for polycube computation and for the generation of the pieces using exact boolean operations are not reported, since external tools are used for these steps. To give an idea of the order of magnitude, the computation of the polycube map never exceeds three minutes, and boolean operations always stay under one minute, for all models except for *Fandisk*. The longest step of the whole pipeline is the polycube computation, that is a pre-processing step. The entire timing is not a problem when compared to the fabrication time.

It is not possible to assure that the resultant partition is optimal in term of the number of parts, but it is unquestionably easy and fast to compute. In this chapter, of course, it is not proposed an absolute improvement of the results in [LBRM12], but just a suggestion of how it can be possible to partition a mesh for fabrication purposes with a simple method controlled by the user, and using just one parameter in the whole pipeline (the polycube compactness). The proposed subdivision pipeline gives excellent results in efficiency.

The compactness of the polycube influences the final decomposition. The polycube simplification algorithm described in Chapter 3 has been used to reduce the number of small and not semantically relevant pieces.

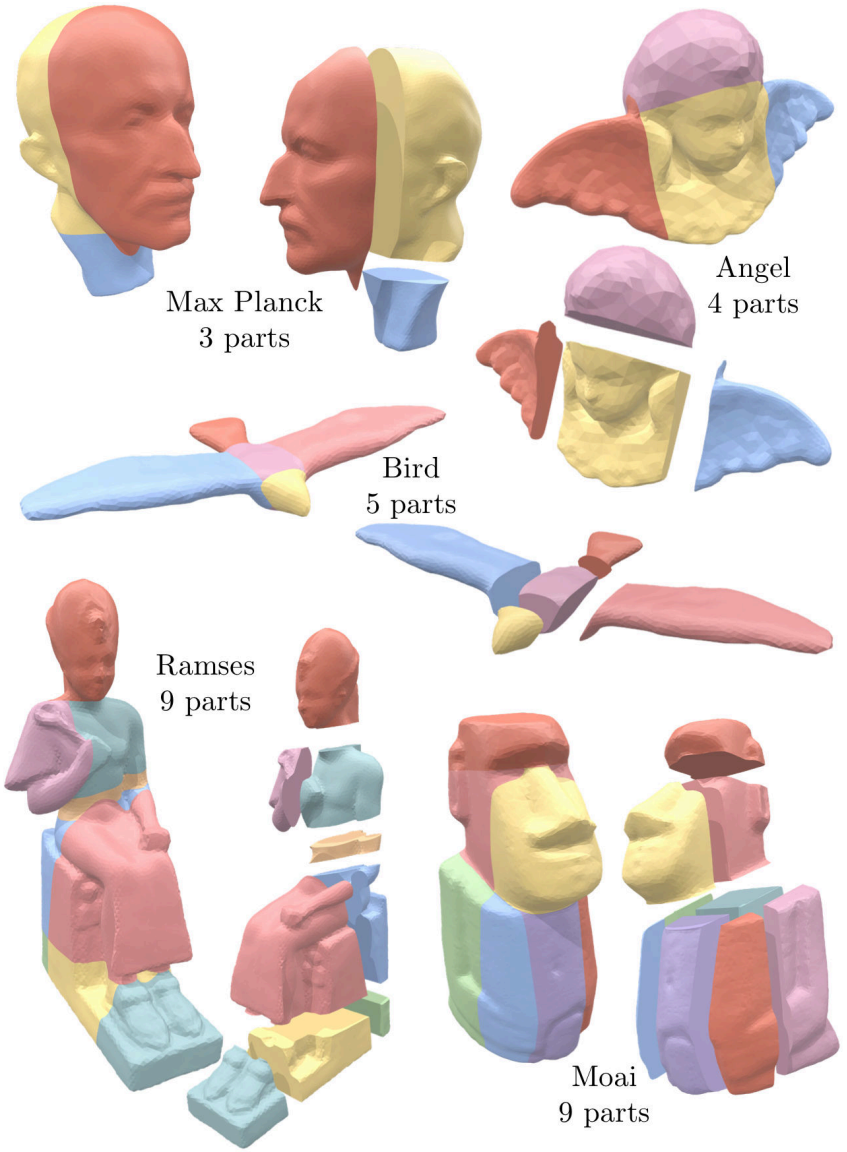


Figure 6.5: *A gallery of decompositions obtained with the proposed polycube-based pipeline. For each model, on the left the whole partitioned model, and, on the right, its exploded view. Captions in the figure and Table 6.1 show the number of pieces in the decomposition.*

Model	%3DPST	BID	%3DPPS	%3DPPST	%3AM	%4AM
Angel	4.6	0	0.0	0.0	13.7	0.0
		1	0.0		41.2	0.1
		2	0.0		17.7	0.0
		3	0.0		22.0	0.0
Bird	32.0	0	0.0	0.0	45.3	0.0
		1	0.0		2.9	0.0
		2	0.0		44.7	0.0
		3	0.0		16.6	0.0
		4	0.0		44.8	0.0
Bu	11.8	0	17.8	10.1	47.5	0.0
		1	5.0		16.9	0.0
		2	5.3		32.7	0.1
		3	5.9		28.8	0.0
		4	1.8		36.1	0.0
		5	1.5		48.0	0.1
		6	2.9		12.7	0.1
		7	4.6		9.4	0.0
		8	3.9		11.6	0.0
Dea	11.7	0	0.0	0.06	39.6	0.1
		1	0.0		0.9	0.0
		2	0.1		36.7	0.3
		3	0.0		0.3	0.0
Duck	14.5	0	3.0	0.7	37.9	0.0
		1	0.0		32.5	0.0
		2	0.0		0.4	0.0
		3	0.0		14.1	0.0
		4	0.0		23.4	0.0
Max Plank	21.7	0	0.0	0.0	6.2	0.0
		1	0.0		32.6	0.0
		2	0.0		31.2	0.0
Moai	6.0	0	0.0	0.1	11.6	0.0
		1	0.0		21.3	0.0
		2	0.0		1.8	0.0
		3	0.0		28.8	0.0
		4	0.5		23.9	0.0
		5	0.0		26.2	0.0
		6	0.0		19.8	0.0
		7	0.0		7.1	0.5
		8	0.0		30.0	2.4
Ramses	3.0	0	1.7	1.8	48.2	0.0
		1	0.0		16.6	0.0
		2	0.6		16.4	0.0
		3	4.3		23.4	0.0
		4	0.0		23.4	0.0
		5	1.4		34.0	0.0
		6	0.6		36.9	0.0
		7	3.0		28.8	0.0
		8	0.0		20.7	0.0
Sphynx	9.8	0	0.0	0.2	29.4	0.9
		1	0.0		29.9	0.0
		2	0.5		30.9	0.1
		3	0.0		37.7	0.4
		4	0.0		12.6	0.0
		5	0.0		24.3	0.0
Squirrel	9.8	0	0.0	0.0	30.1	0.6
		1	0.0		29.6	1.2
		2	0.0		6.9	0.0
		3	0.0		23.7	0.0

Table 6.2: *Fabricability of most of the models listed in Table 6.1. Column labels have the following meanings: percentage of surface covered by supports if the entire model is printed (%3DPST); block identifier (BID); percentage of surface covered by supports for each piece (%3DPPS); percentage of surface covered by supports on the entire subdivided model (%3DPPST); percentage of surface not visible from the machine tool during the 3-axis machining (%3AM); percentage of samples not visible from the machine tool during the 4-axis machining (%4AM).*

6.6.1 Partitioning in additive manufacturing

Since the partitioning allows the complete model to be larger than the printing chamber, with the proposed pipeline, it is possible to print almost any free-form shape, using supports to handle overhanging features. For every part of the partition, the surface percentage needing support during the fabrication is analyzed. The base for the printing is the flat face giving the lowest percentage. The results are canalized by comparing the percentage of the surface of the whole mesh needing supports with the percentage of surface needing supports in the decomposition (Table 6.2, columns *%3DPST* and *%3DPPST*). This decomposition allows fabricating the final model, guaranteeing to have less percentage of surface needing support. Additionally, since the seams between pieces are planar, it is possible to guarantee that the matching areas are as regular as the production process allows, and, therefore, the parts accurately match during the assembly step.

6.6.2 Partitioning in subtractive manufacturing

Subtractive technologies impose stricter constraints on the model shape. For 3-axis fabrication, the proposed polycube-based decomposition does not always produce a suitable partition. Even in presence of flat faces, it is not possible to guarantee that the pieces are height-fields. Testing the 3-axis milling checker introduced in Section 6.5.2 on the shown results, it is clear that the height-field constraint is too hard to respect without taking into account specific precautions during the decomposition. As pointed out in Table 6.2 (column *%3AM*), only five pieces of the decompositions (two belonging to the same model) have tiny percentages of surface that cannot be reached by the milling tool because occluded (less than 3%). These pieces can be manufactured using 3-axis milling machines, at the cost of losing the details of the occluded parts and, in some cases, introducing discontinuities between adjacent blocks if the occlusion involves one of the flat faces of the block. However, no pieces of the decompositions are strictly height-fields, and none of the presented results is composed only by pieces with negligible percentages of non-millable surface. This is a clear limitation of the presented method.

Two results obtained with this pipeline are compared to [MLS*18] in Figure 6.6. The proposed method guarantees a regular decomposition inherited from the polycube partitioning, where every block can be produced with 4 axis milling machines or 3D-printed without supporting structures. On the other hand, the method proposed in [MLS*18] guarantees blocks that can be milled using 3 axis milling machines.

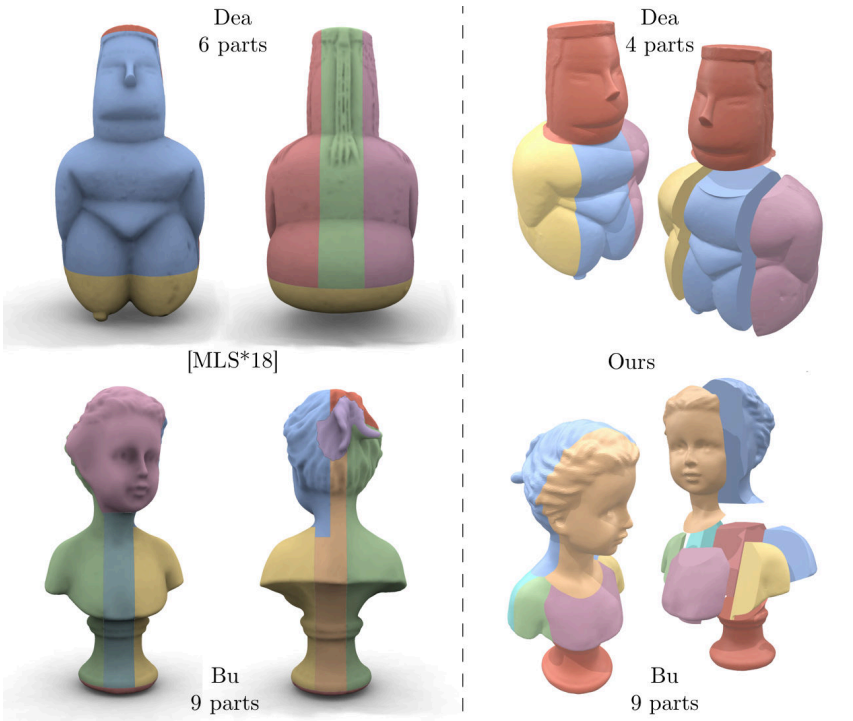


Figure 6.6: *Decompositions of Dea and Bu models obtained with the the method of [MLS*18] (left) and with the method proposed in this chapter (right).*

The use of 4-axis machines allows for more flexible constraints, but it requires to identify first the rotation axis. Using the checking procedure described in Section 6.5 it is possible to demonstrate the feasibility of almost all the parts obtained from the experimented models. The results show that the machining tool cannot reach only a limited percentage of the surface (Table 6.2, column *%4AM*). In Figure 6.7, the chosen axes that would guarantee the fabrication of each piece of the *Duck* model are shown (notice that this model cannot be fabricated with the 3-axis technology).

All the shown results use the hypothesis of an ideal machining tool of indefinite length and infinite narrow size. Should one manufacture the parts, it would require to revise the checking procedure to take in account size, length, and shape of the tool. The change would not substantially modify the results.

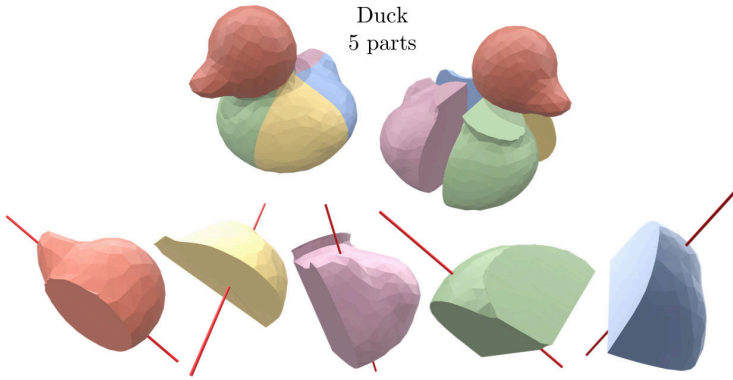


Figure 6.7: *The five parts of the Duck model decomposition, all with at least one flat face. The red cylinder is the rotation axis used for the 4-axis milling.*

6.6.3 3D printing examples

Five of the computed decompositions have been fabricated by using additive manufacturing: *Duck*, *Sphinx*, *Angel*, *Max Plank*, and *Squirrel*. These models have different polycube mappings, all very simple, and they decompose, respectively into five, six, four, four, and three pieces (see Fires 6.1, 6.5 and 6.8, for the decompositions). Figure 6.9 shows photos of *Max Planck* with the total height (24.5 cm). The chamber of the used 3D printer, a Flashforge Creator Pro, is 227 × 148 × 150 mm, and thus it could not be possible to print the model in this size without decomposing it. Furthermore, note that a large number of external supports would have been necessary to fabricate this model without partition it. Photos of the models of the *Duck* and *Angel* are in Figure 6.10.

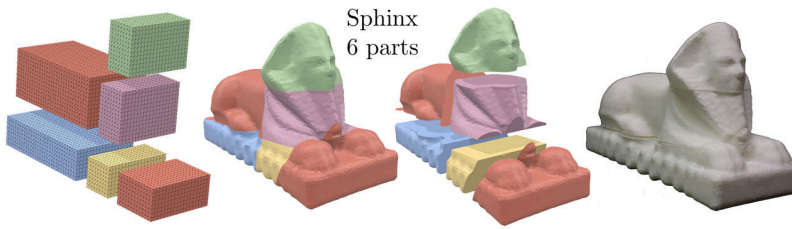


Figure 6.8: *The Sphinx model, from left to right: the polycube mapping; the partitioned model; the exploded set of parts; the ABS model.*

Figure 6.9: *The Max Planck model fabricated. The three separated parts on the left, and the assembled model on the right.*

Figure 6.10: *The models of Duck and Angel. On the top left the five printed parts of the Duck, and on the top right two views of the assembled model. On the bottom left the four parts of the Angel, and on the bottom right the assembled model.*

6.7 Limitations

The polycube-based partitioning is not well suited when used to decompose models with large almost flat not orthogonal surfaces. The *Fandisk* model, for instance, is quite regular and straightforward but, due to its geometric features, decomposes in 50 portions (see Figure 6.11 left). This characteristic is a drawback when compared to pure semantical approaches. As an example, a manual segmentation can easily partition the fandisk in much fewer portions. But the fully automatic pipeline still makes the proposed approach advantageous on models without these characteristics. Since optimized polycubes are used, it is possible to decompose in as few parts as possible, but some small pieces can still be produced, as one can notice in the Ramses model of Figure 6.5.

Another limitation of this pipeline is its application to models like the *Hole3* one. This model has genus three and is an elementary CSG object that a human could easily split into two parts with flat bases which would print with no supports. The proposed algorithm decomposes it in eighteen pieces (see Figure 6.11 right) because each hole induces additional partitions in polycube space. Theoretically, the compactness of the polycube can be reduced to one single cube but only for objects of genus zero. When the genus is higher than zero, the theoretical limitations do not allow a compact subdivision. Note that this is the same problem highlighted in the limitations of the polycube simplification approach presented in Chapter 3.

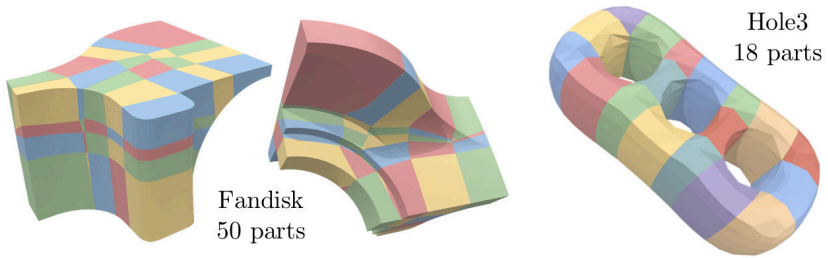


Figure 6.11: *The Fandisk model decomposed in fifty parts (left) and the Hole3 model decomposed in eighteen parts (right).*

Technical information

The Cinolib library [Liv17] has been used to store and process the 3D models and polycubes. The CGAL library [CGA18] has been used to implement some functions of the tool described in 6.5, LibIGL [JP17]

for performing the boolean operations, and CG3Lib [MN*18] for basic algorithms and data structures. The Tetwild [HZG*18] generators has been used to turn tri-meshes into tet-meshes and the Polycut algorithm [LVS*13] to generate all the polycubes shown in the chapter. See Appendix A for more details.

Part IV

Conclusions and Future works

Chapter 7

Conclusions

After a brief introduction to the polycube world (see Chapter 1), three main works have been presented in this thesis. The goal of the projects described in this manuscript was to optimize the polycube structure in order for it to be used in different application fields, from the digital world of 3D models to the more concrete one of fabrication.

In Chapter 3, a novel method for the simplification of a polycube structure has been presented. The proposed method is very fast, it can be easily inserted in a re-meshing pipeline, and it needs a single parameter (λ) that allows to choose the balance between alignment and shape fidelity. This approach is meant to be plugged into the state-of-the-art pipeline used to re-mesh shapes represented by triangle meshes into quadrilateral meshes (in case of surfaces), or ones represented by tetrahedral meshes into hexahedral meshes (in case of volumes). The simplification is performed in polycube space, by optimal alignment of polycube corners in an integer lattice. The proposed approach overcomes previous limitations in polycube-based meshing, making the process independent from the sampling resolution and generating structured meshes with a lower number of domains. The results shown in Section 3.6 demonstrate that, with the proposed simplification approach, it is possible to reduce the complexity of the input polycube with a factor ranging from 25% to 70%. Furthermore, the proposed extra step in the classical meshing pipeline requires negligible time to be performed.

In Chapter 4, a novel pipeline for the generation of high-quality hexahedral meshes has been introduced, in which the quality measure of a hexahedron refers to a deviation from the perfect cube. The proposed pipeline utilizes a polycube mapping for decomposing the input domain

into portions which are simple to discretize into a hexahedral mesh. It needs a single parameter to be set: the λ value to balance between the number of new elements and the number of new singularities. In this pipeline, the main contribution is a selective padding strategy which automatically adds sheets of hexahedra only where they are needed, in order to increase the global quality of the output hexahedral meshes. These sheets can form turns inside the domain, which induce extra edge and vertex singularities. Compared to global padding or greedy straight sheet insertion, the proposed approach improves the global quality while generating fewer hexahedra. It works at its best when applied to input domains bounded by a piecewise planar surface, which is typical of mechanical parts. As it is clear from the results shown in Section 4.5, the proposed selective padding allows to obtain a substantial quality improvement of the hex-mesh quality, concerning both minimum and average Scaled Jacobian.

In Chapter 6, a simple and effective polycube-based decomposition scheme has been presented. It is able to manipulate digital shapes, and partition them in view of their fabrication. The proposed method allows fabricating any shape using any kind of 3D printer. It also includes two checking procedures that allow to verify if the decomposition is suitable for 3-axis and 4-axis milling machining. As Section 6.6 shows, this simple subdivision scheme allows the models to be printed while also guaranteeing to have less percentage of surface needing supports. From the machining point of view, the proposed decomposition is suitable for 4-axis milling machines, as it is clear from Table 6.2, while it is not for 3-axis ones, as the height-fields condition requires stricter constraints. A set of fabricated models, in the same section, shows the quality of the proposed subdivision.

7.1 Future works

Several improvements are possible for the works presented in this thesis. In the field of the simplification of polycube structures, we are already working on an advanced semantic simplification scheme of the polycube, with the aim to simplify it by removing "useless" cuboids in the subdivision. In particular, looking at Figure 3.14 in the last paragraph of Section 3.7, it is evident that the domains number induced by the polycube structure is sometimes higher than the one obtained with other meshing approaches. With this new work, we aim to remove the four volumetric domains at the four ends of the subdivision shown in Figure 3.14, without giving up the use of polycubes in the hex-meshing pipeline. Another interesting topic to further study and investigate is a new strategy to compute the pairs of corners to align. Notice that, even if the Voronoi-based heuristic

is replaced with another method, the full mathematical model set in this thesis continues to work.

Talking about the selective padding proposed approach, we plan to explore an automatic parameter selection approach, in order to find the best balance between quality and complexity. We also wish to explore a global optimization approach to select the optimal set of constrained facets that yields the maximum quality. We then intend to analyze whether a combination of padding and inverse padding (removing hexahedral layers from the mesh) can provide better results. Furthermore, we plan to extend our approach to more general polycube-based hex-mesh structures, like those used by Fang et al. [FXBH16].

Moving on to the fabrication field, we plan to improve the presented subdivision scheme in many ways. The first improvement is related to fabrication with 3-axis machines. We always produce parts with a flat base but only this property is not sufficient to guarantee that the parts are height-fields. A solution to this problem could be splitting the not height-field portions into sub-portions, using cutting planes. The choice of appropriate planes would lead to splitting a part into height-fields. The iterative application of the splitting step would produce an entirely fabricable set of portions. The optimal choice of cutting planes and the demonstration of the termination of the iterative method, apart from trivial solutions, are open issues. Another interesting topic to further investigate is a post-processing step to reduce the number of portions. A strategy to face this problem passes through the merging of adjacent pieces into clusters. This step would not be trivial, as we would have to apply the right constraints to maintain the partition suitable for fabrication. The constraints are: size, since it is necessary to avoid generating clusters greater than the printing chamber, and shape, to prevent the increase of supports and to make sure that milling constraints are still satisfied.

Appendix A

Technologies

This appendix introduces the technologies used for the works described in Chapters 3, 4 and 6.

The **Cinolib** library [Liv17], used as data structure to store models and polycubes, is a generic programming header-only C++ library for processing polygonal and polyhedral meshes. It contains different data structures for different kind of meshes (structured, unstructured and mixed surfaces and volumes) and several algorithms for their manipulation.

Other geometry processing libraries have been used to manipulate models and polycubes. The first one, **CGAL** [CGA18], is one of the most famous software projects that provides efficient and reliable geometric algorithms in the form of a C++ library. The second one, **Libigl** [JP17], is a C++ library of geometry processing algorithms designed for and by researchers. It includes construction of common sparse discrete differential geometry operators, simple facet- and edge-based topology data structures, mesh-viewing utilities for OpenGL and GLSL, and many core functions for matrix manipulation. In particular, this library has been used in Chapter 6 for the boolean operations between meshes. Finally, for minor operations, the **Cg3Lib** library [MN*18] has been used. It is a C++ geometry processing library developed by our team (the CG3HCI Group of the University of Cagliari). It is composed of different data structures and a collection of geometry processing and computational geometry algorithms, and it provides simple interfaces with the most famous geometry processing libraries like CGAL, Libigl, Cinolib, etc.

The **Gurobi** Optimizer [Gur], used as numerical solver for the described mathematical models, is a state-of-the-art solver for mathematical programming. The solvers in the Gurobi Optimizer has been designed by using

the most advanced implementations of the latest algorithms. It includes the following solvers: linear programming solver (LP), mixed-integer linear programming solver (MILP), mixed-integer quadratic programming solver (MIQP), quadratic programming solver (QP), quadratically constrained programming solver (QCP) and mixed-integer quadratically constrained programming solver (MIQCP). It supports interfaces for a variety of programming and modeling languages, like C++, java, python, C, Matlab, etc., and it provides a simple way to construct a mathematical model. According to the set objective function and the imposed constraints, it decides the most appropriate algorithm to solve the model.

The **Voro++** library [Ryc09], used in Chapter 3, is a software library for the three-dimensional computation of the Voronoi tessellation. It is written in object-oriented C++, allowing it to be easily modified and incorporated into other programs. It can carry out calculations by using a mix of periodic and non-periodic boundary conditions, and it has a general class mechanism for handling different types of walls.

Two tet-mesh generators have been used in the works composing this thesis. The first one is **Tetgen** [Si15]. It is a software to generate tetrahedral meshes of any 3D polyhedral domains, generating the exact constrained Delaunay tetrahedralization, boundary conforming Delaunay mesh, and Voronoi partitions of an input object. Tetgen is written in C++ and it provides various features to generate good quality and adaptive tetrahedral meshes suitable for numerical methods, such as finite element or finite volume methods.

In the second part of this thesis, another tet-mesh generator has been used together with Tetgen: **Tetwild** [HZG*18]. It is a new tetrahedral mesh generator that is extremely robust, without user interaction required, that allows the directly conversion of a triangle soup into an analysis-ready volumetric mesh. Compared to the previous one, it allows obtaining more homogeneous tet-meshes (in terms of tetrahedra dimension) without setting any parameters.

The **PolyCut** algorithm [LVS*13] has been used for the polycubes generation (see Section 1.2 for more information). Moreover, the **hex-mesh optimizer** proposed in [LSVT15] has been used for the optimization of all the hex-meshes presented in this thesis. It is a robust framework for optimizing the hex-mesh quality, capable of generating high-quality meshes, without inverted elements, from poor-quality initial input shapes.

Bibliography

[ACP*14] Alemanno G., Cignoni P., Pietroni N., Ponchio F., Scopigno R.: Interlocking Pieces for Printing Tangible Cultural Heritage Replicas. In *Proceedings of the Eurographics Workshop on Graphics and Cultural Heritage* (2014), The Eurographics Association, pp. 145–154. doi:10.2312/gch.20141312. 73

[AFTR15] Armstrong C. G., Fogg H. J., Tierney C. M., Robinson T. T.: Common Themes in Multi-block Structured Quad/Hex Mesh Generation. *Procedia Engineering 124* (2015), 70 – 82. doi:10.1016/j.proeng.2015.10.123. 21

[AL13] Aigerman N., Lipman Y.: Injective and Bounded Distortion Mappings in 3D. *ACM Transactions on Graphics 32*, 4 (2013), 106:1–106:14. doi:10.1145/2461912.2461931. 32

[BCE*13] Bommes D., Campen M., Ebke H.-C., Alliez P., Kobbelt L.: Integer-grid Maps for Reliable Quad Meshing. *ACM Transactions on Graphics 32*, 4 (2013), 98:1–98:12. doi:10.1145/2461912.2462014. 17, 36

[BK04] Boykov Y., Kolmogorov V.: An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 9 (2004), 1124–1137. doi:10.1109/TPAMI.2004.60. 56

[BLK11] Bommes D., Lempfer T., Kobbelt L.: Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum 30*, 2 (2011). doi:10.1111/j.1467-8659.2011.01868.x. 17

[BLP*13] Bommes D., Lévy B., Pietroni N., Puppo E., Silva C., Tarini M., Zorin D.: Quad-Mesh Generation and

Processing: A Survey. *Computer Graphics Forum 32*, 6 (2013), 51–76. doi:10.1111/cgf.12014. 17, 21

[CAS*19] Cherchi G., Alliez P., Scateni R., Lyon M., Bommes D.: Selective Padding for Polycube-Based Hexahedral Meshing. *Computer Graphics Forum* (2019). doi:10.1111/cgf.13593. vii

[CBK12] Campen M., Bommes D., Kobbelt L.: Dual Loops Meshing: Quality Quad Layouts on Manifolds. *ACM Transactions on Graphics 31*, 4 (2012), 110:1–110:11. doi:10.1145/2185520.2185606. 17, 36

[CGA18] CGAL Project: *CGAL User and Reference Manual*, 4.11 ed. CGAL Editorial Board, 2018. URL: http://doc.cgal.org/4.11/Manual/packages.html. 95, 103

[CGWW16] Chen J., Gao S., Wang R., Wu H.: An approach to achieving optimized complex sheet inflation under constraints. *Computers & Graphics 59*, C (2016), 39 – 56. doi:10.1016/j.cag.2016.05.001. 44

[CK14] Campen M., Kobbelt L.: Quad Layout Embedding via Aligned Parameterization. *Computer Graphics Forum 33*, 8 (2014), 69–81. doi:10.1111/cgf.12401. 17

[CL*10] Chang C.-C., Lin C.-Y., et al.: Texture tiling on 3d models using automatic polycube-maps and wang tiles. *Journal of Information Science and Engineering 26*, 1 (2010), 291–305. 4

[CLS16] Cherchi G., Livesu M., Scateni R.: Polycube Simplification for Coarse Layouts of Surfaces and Volumes. *Computer Graphics Forum 35*, 5 (2016), 11–20. doi:10.1111/cgf.12959. vi, 4

[CZL*15] Chen X., Zhang H., Lin J., Hu R., Lu L., Huang Q., Benes B., Cohen-Or D., Chen B.: Dapper: Decompose-and-pack for 3D Printing. *ACM Transactions on Graphics 34*, 6 (2015), 213:1 – 213:12. doi:10.1145/2816795.2818087. 4

[FBL16] Fu X.-M., Bai C.-Y., Liu Y.: Efficient Volumetric PolyCube-Map Construction. *Computer Graphics Forum 35*, 7 (2016), 97–106. doi:10.1111/cgf.13007. 11, 12, 13, 115

[FCM*18] Fanni F. A., Cherchi G., Muntoni A., Scateni R., Tola A.: Fabrication Oriented Shape Decomposition Using Polycube Mapping. *Computers & Graphics 77* (2018), 183 – 193. doi:https://doi.org/10.1016/j.cag.2018.10.010. viii, 4

[FCS17] Fanni F. A., Cherchi G., Scateni R.: Polycube-based Decomposition for Fabrication. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference* (2017), Giachetti A., Pingi P., Stanco F., (Eds.), The Eurographics Association. doi:10.2312/stag.20171220. viii, 4

[FJFS05] Fan Z., Jin X., Feng J., Sun H.: Mesh morphing using polycube-based cross-parameterization. *Computer Animation and Virtual Worlds 16*, 3-4 (2005), 499–508. doi:10.1002/cav.92. 4

[FKR05] Floater M. S., Kós G., Reimers M.: Mean value coordinates in 3D. *Computer-Aided Geometric Design 22*, 7 (2005), 623–631. doi:10.1016/j.cagd.2005.06.004. 32

[FWJ06] Frank M. C., Wysk R. A., Joshi S. B.: Determining setup orientations from the visibility of slice geometry for rapid computer numerically controlled machining. *Journal of manufacturing science and engineering 128*, 1 (2006), 228–238. doi:10.1115/1.2039100. 74

[FXBH16] Fang X., Xu W., Bao H., Huang J.: All-hex Meshing Using Closed-form Induced Polycube. *ACM Transactions on Graphics 35*, 4 (2016), 124:1–124:9. doi:10.1145/2897824.2925957. 11, 101, 115

[GDC15] Gao X., Deng Z., Chen G.: Hexahedral Mesh Re-parameterization from Aligned Base-complex. *ACM Transactions on Graphics 34*, 4 (2015), 142:1–142:10. doi:10.1145/2766941. 18, 20, 21, 22, 33

[GMD*16] Gao X., Martin T., Deng S., Cohen E., Deng Z., Chen G.: Structured Volume Decomposition via Generalized Sweeping. *IEEE Transactions on Visualization and Computer Graphics 22*, 7 (2016), 1899–1911. doi:10.1109/TVCG.2015.2473835. 18

[GSZ11] Gregson J., Sheffer A., Zhang E.: All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Computer Graphics Forum 30*, 5 (2011), 1407–1416. doi:

10.1111/j.1467-8659.2011.02015.x. 4, 5, 7, 8, 12, 18, 22, 115

[Gur] Gurobi: Optimizer 6.5. http://www.gurobi.com/. 41, 67, 103

[HCB05] Hughes T., Cottrell J., Bazilevs Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering 194*, 39 (2005), 4135 – 4195. doi:10.1016/j.cma.2004.10.008. 22

[HF17] Hou G., Frank M. C.: Computing the Global Visibility Map Using Slice Geometry for Setup Planning. *Journal of Manufacturing Science and Engineering 139*, 8 (2017), 081006–081006:11. doi:10.1115/1.4036423. 74, 85

[HFW11] Hao J., Fang L., Williams R. E.: An efficient curvature-based partitioning of large-scale STL models. *Rapid Prototyping Journal 17*, 2 (2011), 116–127. doi:10.1108/13552541111113862. 72

[HJS*14] Huang J., Jiang T., Shi Z., Tong Y., Bao H., Desbrun M.: ℓ1-based Construction of Polycube Maps from Complex Shapes. *ACM Transactions on Graphics 33*, 3 (2014), 25:1–25:11. doi:10.1145/2602141. 4, 5, 9, 10, 11, 35, 36, 115

[HLZCO14] Hu R., Li H., Zhang H., Cohen-Or D.: Approximate Pyramidal Shape Decomposition. *ACM Transactions on Graphics 33*, 6 (2014), 213:1–213:12. doi:10.1145/2661229.2661244. 72

[HMA15] Herholz P., Matusik W., Alexa M.: Approximating Free-form Geometry with Height Fields for Manufacturing. *Computer Graphics Forum 34*, 2 (2015), 239–251. doi:10.1111/cgf.12556. 72, 73

[HWFQ09] He Y., Wang H., Fu C.-W., Qin H.: A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics 33*, 3 (2009), 369 – 380. doi:10.1016/j.cag.2009.03.024. 5, 7, 115

[HXH10] Han S., Xia J., He Y.: Hexahedral Shell Mesh Construction via Volumetric Polycube Map. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling* (2010), ACM, pp. 127–136. doi:10.1145/1839778.1839796. 4

[HZ16] Hu K., Zhang Y. J.: Centroidal Voronoi tessellation based polycube construction for adaptive all-hexahedral mesh generation. *Computer Methods in Applied Mechanics and Engineering 305* (2016), 405 – 421. doi:10.1016/j.cma.2016.03.021. 10, 115

[HZG*18] Hu Y., Zhou Q., Gao X., Jacobson A., Zorin D., Panozzo D.: Tetrahedral Meshing in the Wild. *ACM Trans. Graph. 37*, 4 (2018), 60:1–60:14. doi:10.1145/3197517.3201353. 67, 96, 104

[HZL17] Hu K., Zhang Y. J., Liao T.: Surface segmentation for polycube construction based on generalized centroidal Voronoi tessellation. *Computer Methods in Applied Mechanics and Engineering 316* (2017), 280 – 296. doi:10.1016/j.cma.2016.07.005. 12, 115

[JP17] Jacobson A., Panozzo D.: Libigl: Prototyping Geometry Processing Research in C++, 2017. doi:10.1145/3134472.3134497. 95, 103

[LBRM12] Luo L., Baran I., Rusinkiewicz S., Matusik W.: Chopper: Partitioning Models into 3D-printable Parts. *ACM Transactions on Graphics 31*, 6 (2012), 129:1–129:9. doi:10.1145/2366145.2366148. 72, 88

[LEM*17] Livesu M., Ellero S., Martínez J., Lefebvre S., Attene M.: From 3D models to 3D prints: an overview of the processing pipeline. *Computer Graphics Forum 36*, 2 (2017), 537–564. doi:10.1111/cgf.13147. 73, 78

[Liv17] Livesu M.: cinolib: a generic programming header only C++ library for processing polygonal and polyhedral meshes, 2017. https://github.com/mlivesu/cinolib/. 41, 67, 95, 103

[LJFW08] Lin J., Jin X., Fan Z., Wang C. C. L.: Automatic PolyCube-Maps. In *Advances in Geometric Modeling and Processing* (2008), Chen F., Jüttler B., (Eds.), Springer Berlin Heidelberg, pp. 3–16. doi:10.1007/978-3-540-79246-8_1. 5, 6, 7, 115

[LJLJ15] Lin H., Jin S., Liao H., Jian Q.: Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm. *Computer-Aided Design 67-68* (2015), 107–117. doi:10.1016/j.cad.2015.05.004. 18

[LLWQ10] Li B., Li X., Wang K., Qin H.: Generalized PolyCube Trivariate Splines. In *2010 Shape Modeling International Conference* (2010), pp. 261 – 265. doi:10.1109/SMI.2010.40. 4

[LLWQ13] Li B., Li X., Wang K., Qin H.: Surface Mesh to Volumetric Spline Conversion with Generalized Polycubes. *IEEE Transactions on Visualization and Computer Graphics 19*, 9 (2013), 1539–1551. doi:10.1109/TVCG.2012.177. 4, 22

[LMPS16] Livesu M., Muntoni A., Puppo E., Scateni R.: Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes. *Computer Graphics Forum 35*, 7 (2016), 237–246. doi:10.1111/cgf.13021. 18

[LSVT15] Livesu M., Sheffer A., Vining N., Tarini M.: Practical Hex-mesh Optimization via Edge-cone Rectification. *ACM Transactions on Graphics 34*, 4 (2015), 141:1–141:11. doi:10.1145/2766905. 33, 58, 104

[LVS*13] Livesu M., Vining N., Sheffer A., Gregson J., Scateni R.: PolyCut: Monotone Graph-Cuts for PolyCube Base-Complex Construction. *ACM Transactions on Graphics 32*, 6 (2013). doi:10.1145/2508363.2508388. 5, 9, 12, 18, 35, 36, 41, 48, 67, 78, 96, 104, 115

[LZLW15] Liu L., Zhang Y., Liu Y., Wang W.: Feature-preserving T-mesh construction using skeleton-based polycubes. *Computer-Aided Design 58* (2015), 162–172. doi:10.1016/j.cad.2014.08.020. 22

[MLS*18] Muntoni A., Livesu M., Scateni R., Sheffer A., Panozzo D.: Axis-Aligned Height-Field Block Decomposition of 3D Shapes. *ACM Transactions on Graphics 37*, 5 (2018), 169:1–169:15. doi:10.1145/3204458. 74, 86, 91, 92, 116

[MN*18] Muntoni A., Nuvoli S., et al.: CG3Lib: A structured C++ geometry processing library., 2018. https://github.com/cg3hci/cg3lib. 96, 103

[MPKZ10] Myles A., Pietroni N., Kovacs D., Zorin D.: Feature-aligned T-meshes. *ACM Transactions on Graphics 29*, 4 (2010), 117:1–117:11. doi:10.1145/1778765.1778854. 21

[MT95] Mitchell S. A., Tautges T. J.: Pillowing doublets: refining a mesh to ensure that faces share at most one edge. In *4th International Meshing Roundtable* (1995), Citeseer, pp. 231–240. doi:10.1.1.38.7681. 43

[MTP*15] Marcias G., Takayama K., Pietroni N., Panozzo D., Sorkine-Hornung O., Puppo E., Cignoni P.: Data-driven Interactive Quadrangulation. *ACM Transactions on Graphics 34*, 4 (2015), 65:1–65:10. doi:10.1145/2766964. 17

[OSE17] Owen S. J., Shih R. M., Ernst C. D.: A template-based approach for parallel hexahedral two-refinement. *Computer-Aided Design 85* (2017), 34 – 52. doi:10.1016/j.cad.2016.09.005. 45

[PPM*16] Pietroni N., Puppo E., Marcias G., Scopigno R., Cignoni P.: Tracing Field-Coherent Quad Layouts. *Computer Graphics Forum 35*, 7 (2016), 485 – 496. doi:10.1111/cgf.13045. 21

[RRP15] Razafindrazaka F. H., Reitebuch U., Polthier K.: Perfect matching quad layouts for manifold meshes. *Computer Graphics Forum 34*, 5 (2015), 219 – 228. doi:10.1111/cgf.12710. 21

[Ryc09] Rycroft C.: Voro++: A three-dimensional Voronoi cell library in C++. *Lawrence Berkeley National Laboratory* (2009). doi:10.1063/1.3215722. 41, 104

[SDW*10] Shepherd J. F., Dewey M. W., Woodbury A. C., Benzley S. E., Staten M. L., Owen S. J.: Adaptive mesh coarsening for quadrilateral and hexahedral meshes. *Finite Elements in Analysis and Design 46*, 1 (2010), 17–32. doi:10.1016/j.finel.2009.06.024. 44

[SEK*07] Stimpson C., Ernst C., Knupp P., Pébay P., Thompson D.: The Verdict library reference manual. *Sandia National Laboratories Technical Report 9* (2007). 54

[SFLF15] Song P., Fu Z., Liu L., Fu C.-W.: Printing 3D objects with interlocking parts. *Computer-Aided Geometric Design 35* (2015), 137 – 148. doi:10.1016/j.cagd.2015.03.020. 72

[She07] Shepherd J. F.: *Topologic and Geometric Constraint-based Hexahedral Mesh Generation*. PhD thesis, University of Utah, 2007. AAI3256106. 43

[Si15] Si H.: TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Transactions on Mathematical Software 41*, 2 (2015), 11:1–11:36. doi:10.1145/2629697. 41, 67, 104

[SJ08] Shepherd J. F., Johnson C. R.: Hexahedral mesh generation constraints. *Engineering with Computers 24*, 3 (2008), 195 – 213. doi:10.1007/s00366-008-0091-4. 43

[TBM96] Tautges T. J., Blacker T., Mitchell S. A.: The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering 39*, 19 (1996), 3327–3349. doi:10.1002/(SICI)1097-0207(19961015)39. 18

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: Polycube-maps. *ACM Transactions on Graphics 23*, 3 (2004), 853–860. doi:10.1.1.413.3944. v, 3, 4, 5, 115

[TPP*11] Tarini M., Puppo E., Panozzo D., Pietroni N., Cignoni P.: Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Transaction on Graphics 30*, 6 (2011), 142:1–142:12. doi:10.1145/2070781.2024176. 17, 36, 41

[TPSHSH13] Takayama K., Panozzo D., Sorkine-Hornung A., Sorkine-Hornung O.: Sketch-based Generation and Editing of Quad Meshes. *ACM Transactions on Graphics 32*, 4 (2013), 97:1–97:8. doi:10.1145/2461912.2461955. 17

[ULP*15] Usai F., Livesu M., Puppo E., Tarini M., Scateni R.: Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. *ACM Transactions on Graphics 35*, 1 (2015), 6:1–6:13. doi:10.1145/2809785. 17, 36

[VS17] Verma C. S., Suresh K.: A Robust Combinatorial Approach to Reduce Singularities in Quadrilateral Meshes. *Computer-Aided Design 85*, C (2017), 99–110. doi:10.1016/j.cad.2016.07.008. 21

[WGZC18] Wang R., Gao S., Zheng Z., Chen J.: Hex mesh topological improvement based on frame field and sheet

adjustment. *Computer-Aided Design 103* (2018), 103 – 117. doi:10.1016/j.cad.2017.11.007. 45, 58, 64, 67

[WHL*08] Wang H., He Y., Li X., Gu X., Qin H.: Polycube splines. *Computer-Aided Design 40*, 6 (2008), 721–733. doi:10.1016/j.cad.2008.01.012. 4, 22

[WJH*08] Wang H., Jin M., He Y., Gu X., Qin H.: User-controllable Polycube Map for Manifold Spline Construction. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling* (2008), ACM, pp. 397 – 404. doi:10.1145/1364901.1364958. 4, 6, 115

[WLL*12] Wang K., Li X., Li B., Xu H., Qin H.: Restricted Trivariate Polycube Splines for Volumetric Data Modeling. *IEEE Transactions on Visualization and Computer Graphics 18*, 5 (2012), 703–716. doi:10.1109/TVCG.2011.102. 4

[WSC*17] Wang R., Shen C., Chen J., Wu H., Gao S.: Sheet operation based block decomposition of solid models for hex meshing. *Computer-Aided Design 85* (2017), 123 – 137. doi:10.1016/j.cad.2016.07.016. 45

[WYZ*11] Wan S., Yin Z., Zhang K., Zhang H., Li X.: A topology-preserving optimization algorithm for polycube mapping. *Computers & Graphics 35*, 3 (2011), 639 – 649. doi:10.1016/j.cag.2011.03.018. 8, 115

[WZLH13] Wang W., Zhang Y., Liu L., Hughes T. J. R.: Trivariate Solid T-spline Construction from Boundary Triangulations with Arbitrary Genus Topology. *Computer-Aided Design 45*, 2 (2013), 351–360. doi:10.1016/j.cad.2012.10.018. 4, 22

[YZWL14] Yu W., Zhang K., Wan S., Li X.: Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design 46* (2014), 58–68. doi:10.1016/j.cad.2013.08.018. 4

[ZCWG14] Zhu H., Chen J., Wu H., Gao S.: Direct Editing on Hexahedral Mesh through Dual Operations. *Procedia Engineering 82* (2014), 149 – 161. doi:10.1016/j.proeng.2014.10.380. 44

[ZGZJ16] Zhou Q., Grinspun E., Zorin D., Jacobson A.: Mesh arrangements for solid geometry. *ACM Transactions on*

Graphics 35, 4 (2016), 39. doi:10.1145/2897824.2925901. 84

[ZLL*18] Zhao H., Lei N., Li X., Zeng P., Xu K., Gu X.: Robust edge-preserving surface mesh polycube deformation. *Computational Visual Media 4*, 1 (2018), 33 – 42. doi:10.1007/s41095-017-0100-x. 13, 115

List of Figures