University of Cagliari

PhD Program in Mathematics and Computer Science

Geometry Processing for Subtractive Fabrication

Ph.D. Candidate:
Alessandro Muntoni

Supervisor:
Prof. Riccardo Scateni

Abstract

Subtractive manufacturing technologies, such as 3-axis CNC milling, add a useful tool to the digital manufacturing arsenal. However, each milling pass using such machines can only carve a single height-field surface, defined with respect to the machine tray, limiting the set of geometries that may be produced with this technique.

This thesis presents two methods which enable the fabrication of simplified or decomposed geometries using subtractive techniques. The first method aims to obtain, starting from a given model, a simplified shape that can be easily unfolded. The final objective is to reproduce it by producing the unfolded object using a 3-axis CNC milling machine with special milling tools called V-Routers, to create particular joints which make the assembly process an easy task. The second method enables fabrication of general 3D shapes using 3-axis CNC milling methodology by providing a novel robust algorithm for decomposing general 3D geometries into a small set of overlap-free *height-field blocks*, volumes enclosed by a flat base and a height-field surface defined concerning this base. Such blocks can be manufactured with a single pass of 3-axis milling and then assembled to form the target geometry. Computing the desired decomposition requires solving a highly constrained discrete optimization problem, variants of which are known to be NP-hard. We effectively compute a high-quality decomposition by using a two-step process that leverages the unique characteristics of our setup. Specifically, we notice that if the height-field block directions are constrained to the major axes we can always produce a valid decomposition starting from a suitable surface segmentation. Our method first produces a compact set of large, possibly overlapping, height field blocks that jointly cover the model surface by recasting this discrete constrained optimization problem as an unconstrained optimization of a continuous function, which allows for an efficient solution. We then cast the computation of an overlap-free, final decomposition as an ordering problem on a graph, and solve it via a combination of cycle elimination and topological sorting. The combined algorithm produces a compact set of height-field blocks that jointly describe the input model within a user given tolerance and satisfy all manufacturing constraints. We demonstrate our method on a range of inputs, and showcase some real-life models manufactured using our technique.

Contents

Chapter 1

Introduction

The advent of digital manufacturing has opened the doors to broad-based bespoke 3D object fabrication, while simultaneously introducing numerous new geometry processing challenges. The modern digital fabrication toolbox spans a broad range of additive and subtractive methods. Additive techniques (figure 1.1(a)), such as 3D printing, build 3D objects by adding layer-upon-layer of material, while subtractive ones (figure 1.1(b)), such as 3-axis CNC milling, carve material away from a solid block to produce the desired shape. The different technologies complement one another across some dimensions.

Subtractive technologies offer some advantages compared to their additive counterparts. Most notably they enable manufacturing using un-layerable materials such as wood or stone. Compared to 3D printing, CNC milling exhibits some particularly desirable properties: it can operate across a much wider range of scales and provides higher accuracy, and 3-axis milling machines require less maintenance than 3D printers [Newman et al., 2015].

While other technologies are capable of producing many types of geometries

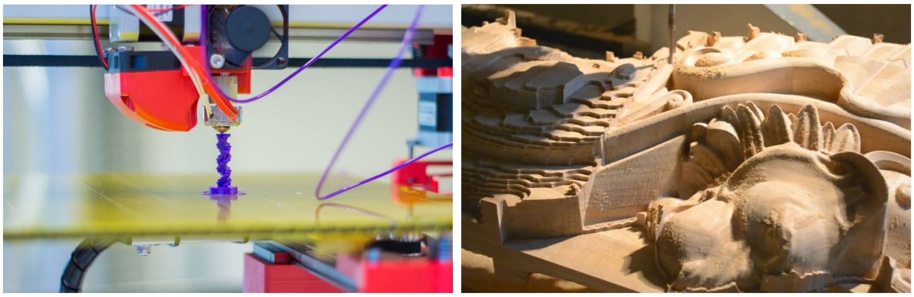

(a) Additive Fabrication (b) Subtractive Fabrication

Figure 1.1: Fabrication Techniques

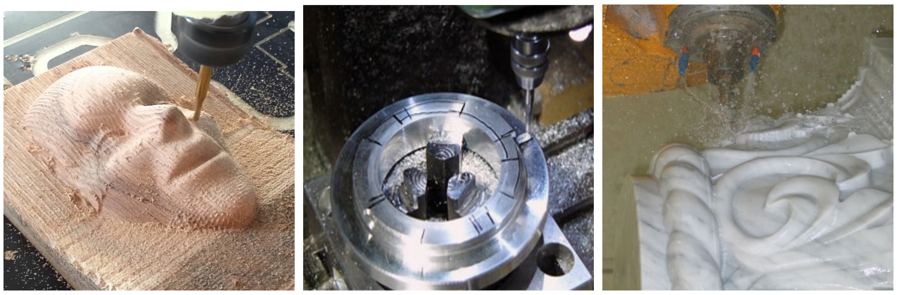

(a) Wood (b) Alluminum (c) Stone

Figure 1.2: Examples of Millable Materials

in a single pass, traditional single-pass 3-axis CNC machining is limited to fabricating 2.5D *height-field blocks*, 3D solids bounded by a flat base and a height field top surface defined along a direction orthogonal to, and located strictly above, this base. 4-axis milling machines have more relaxed constraints, since they allow the target object to rotate along an additional axis. It enables the milling tool to reach every point which can be expressed as a height-field along (at least) one of the planes generated by the rotation along the 4th axis. Furthermore, the 5-axis milling machines allow the rotation along two orthogonal axes, further relaxing the manufacturability constraints. 3-axis CNC machines are cheaper and easier to use than the 4-axis and the 5-axis ones [Jun et al., 2003]. In this thesis, we present two methods which can process 3D digital shapes to produce a physical representation using 3-axis milling machines.

The first method, described in Chapter 3, is still a work in progress and it has the goal to simplify an input 3D surface producing a polygon mesh composed of a small number of flat primitives, which approximates the given surface (Figure 1.3, left). The aim is to reproduce the unfolding of the output polygon mesh in sheets of rigid material (Figure 1.3, center) using CNC machines with special cutters called *V-Routers*, which allow creating high precision V-shaped furrows. The furrows, made along the edges, reproduce the dihedral angle between each couple of adjacent flat primitives. Therefore, once the machining is done, we can easily fold and join the primitives, in order to reproduce the simplified model with a minimum manual effort and a high precision. We show in Chapter 3 what we have already implemented and tested with partial results, and what is still a to do.

The second method, described in Chapter 4, enables the fabrication of general 3D geometries using 3-axis CNC milling machines by algorithmically decomposing general 3D shapes into height-field blocks (Figure 1.4, right). Once fabricated, these blocks can be joined together to produce the desired output shape. This methodology extends customized fabrication of general 3D geometries to additional materials which cannot be used with additive

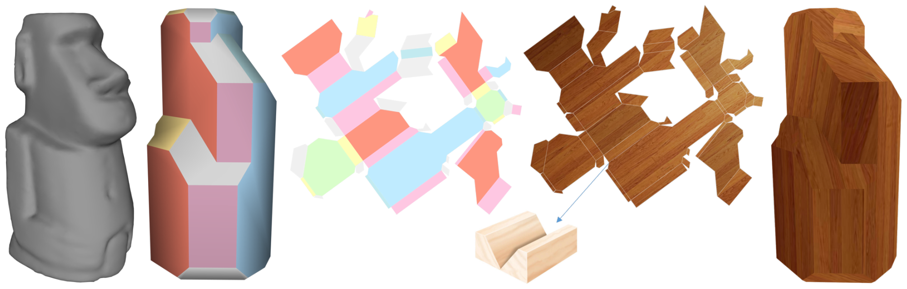


Figure 1.3: We enable fabrication of simplified 3D geometries (left) using V-Router tips with CNC machining along the edges of their unfolding (center), which perfectly join after the manual composition (right).

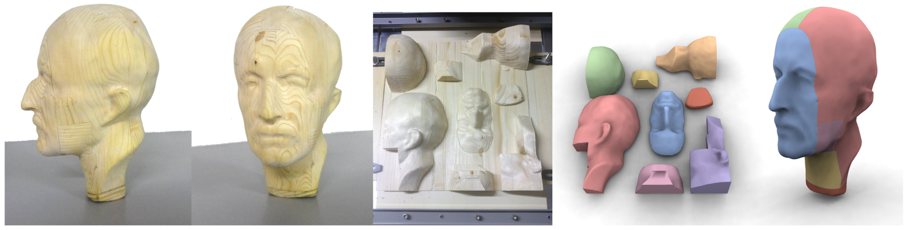


Figure 1.4: We enable fabrication of general 3D geometries (right) using single pass 3-axis CNC machining by algorithmically decomposing 3D geometries (left) into height-field blocks (center).

techniques (Figure 1.4, left), and allows users to benefit from other advantages of 3-axis CNC milling, such as an extensive scales range, when manufacturing general geometries.

In appendix D, it is also shown a method which is not related to the main subject of this thesis, but it has been presented during the Ph.D. studies of the candidate. The technique allows for the automatic generation of structured hexahedral meshes of articulated 3D shapes. The complex problem of generating the connectivity of a hexahedral mesh of a general shape has been recast into the more straightforward problem of creating the connectivity of a tubular structure derived from its curve-skeleton. It has also been provided volumetric subdivision schemes to nicely adapt the topology of the mesh to the local thickness of tubes, while regularizing per-element size. The method is fast, one-click, easy to reproduce, and it generates structured meshes that better align to the branching structure of the input shape if compared to previous methods for hexa mesh generation.

Part of the results obtained during the development of this thesis have been published in the following articles:

Marco Livesu, Alessandro Muntoni, Enrico Puppo, and Riccardo Scateni,

Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes, 2016. Computer Graphics Forum, vol. 35, number 7, pages 237–246.

Alessandro Muntoni, Marco Livesu, Alla Sheffer, Riccardo Scateni and Daniele Panozzo, **Height-Field Block Decomposition for 3-Axis Milling**. Submitted to Transaction on Graphics (TOG).

Chapter 2

Background

Our work fits into the highly active domain of geometry processing for digital fabrication [Livesu et al., 2017, Umetani et al., 2015, Liu et al., 2014]. While much of the recent research focused on additive technologies, we address the complementary challenge of geometry processing for subtractive methodologies, and specifically 3-axis CNC milling. While multiple methods address manufacturing specific aspects of milling such as computing machining paths [Dinh et al., 2015], our work is the first to solve the problem of decomposing generic 3D geometries to enable 3-axis CNC milling based fabrication.

Our work focuses on two different methods: the first one aims to simplify a generic 3D geometry, producing a polygon mesh composed of a small number of flat primitives having a restricted set of dihedral angles between them, i.e., the ones of the V-Router tools. The second one addresses the problem of finding a decomposition of generic 3D geometries, to enable a 3-axis CNC milling based fabrication. Below we review the related simplification and decomposition technologies.

2.1 Simplification to Unfoldable Models

Generally speaking, mesh simplification and approximation have been trend-topics in recent years. [Cohen-Steiner et al., 2004] proposes a method which produces an approximation of the surface using a variational approach.This technique is general purpose, and it has never been used in the fabrication context.

Fabrication of 3D digital shapes using flat sheets of rigid materials has been studied for various applications. [Richter and Alexa, 2015] represent an input geometry using beams with a rectangular cross-section. The authors show some beams fabricable in wood , manufactured with laser cutter machines and assembled with particular joints. [Schwartzburg and Pauly, 2013] and [Cignoni et al., 2014] present methods for the fabrication of 3D models by the interlocking of planar pieces and strips. These three papers propose a solution to problems which are substantially different from ours: they do not aim to create a surface, but to generate a set of strips or planar pieces that are joined together (with interlocks or another type of joints). On the contrary, we want to produce a 3D surface composed of a few polygonal planar faces.

A more related problem has been solved by [Mitani and Suzuki, 2004]. They simplify an input shape, producing a set of triangle strips that can be cut, folded and glued together to obtain a papercraft object. However, the primitives of the output are limited to triangles and the assembly process is quite difficult, mostly suitable for papercraft lovers only.

[Chen et al., 2013] propose a method which solves a problem that is very similar to ours. The authors approximate the input surface in a 3D mesh with a small number of planar polygonal faces for fabrication with CNC cutter machines. However their assembly process is very complicated: it needs proper connectors, and it could take several hours for a single model. In our method, we introduce an additional constraint on the possible dihedral angles between adjacent faces , to simplify the assembly process.

[Chen and Sass, 2016] propose an alternative approach for the assembly process by creating a novel interlocking system with unique joining features. However, their work focuses only on models produced with CAD tools.

In [Zimmer et al., 2014] the authors try to solve the problem of representing an input surface using Zometool, a mathematical modeling system used in various areas. Their problem introduces an angle constraint: every Zometool node has a small set of possible directions, and therefore the angle formed along the edges belongs to a well-known restricted set. However, they do not need to simplify the input mesh with the goal of obtaining a small number of faces on their output mesh.

Polycubes are simplified models in which every primitive is orthogonal to one of the three major axis ([Tarini et al., 2004]). [Livesu et al., 2013] propose an algorithm for the automatic generation of polycubes. However, polycubes are very simple 3D meshes and they rarely look like the original model. Moreover, having only 90 degree dihedral angles on the final results would be very restrictive compared to the V-Router tools for milling available on the market.

2.2 Heightfields Decomposition

Surface Segmentation. Numerous method had been proposed for segmenting a surface model into charts that meet some prescribed requirement [Shamir, 2008]. The general frameworks they employ are focused on surface features, and do not consider volumetric constraints. Thus while they can potentially be modified to use height-field approximation as a desired chart property, they allow for no obvious extension to address our volumetric constraints such as block overlap avoidance.

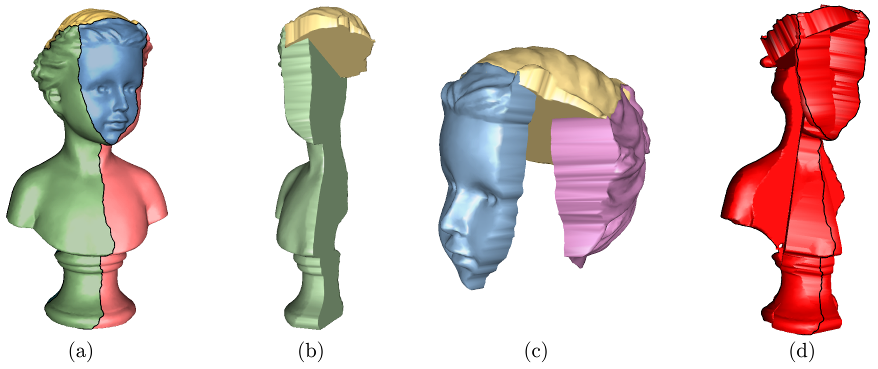


Figure 2.1: The method of [Herholz et al., 2015] produces a set of surface height fields (a), but it does not take into consideration possible intersections between the resulting height blocks (b,c). The extent of the volumetric intersection between all the resulting height blocks can be quite big (d), making impossible to re-assemble the pieces together after fabrication.

Height Field Surface Segmentation. Starting with the early work by [Cook, 1984], a number of height-field based surface segmentation techniques were proposed for efficient support of normal and displacement mapping during rendering. While adequate for this task they are poorly suited for our needs. First, they tend to segment shapes into large numbers of charts, e.g. [Doggett and Hirche, 2000] use over 1500 height fields to represent a human head. Such counts make fabrication infeasible. More important, like general surface segmentation techniques these methods have no obvious extension to the volumetric setup, i.e. no obvious way to avoid or resolve overlaps between resulting blocks (Figure 4.1). The problem is particularly acute in 3D since generic segment boundaries are rarely planar and thus most induced blocks would have large interior overlaps, (Figure 2.1).

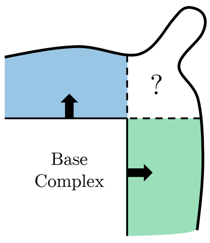
Decomposition for 3D Printing. Multiple methods have been proposed for decomposing shapes into parts to ensure that each component is small enough to fit into the printing chamber during 3D printing [Song et al., 2016, Song et al., 2015, Yao et al., 2015, Alemanno et al., 2014, Luo et al., 2012, Hao et al., 2011, Medellin et al., 2007]. Shape decomposition is also used to ensure quality prints in terms of surface finish [Wang et al., 2016], to minimize the amount of material used [Vanek et al., 2014], and to achieve better mechanical properties [Hildebrand et al., 2013]. Our problem setting is distinct from those addressed by these methods, with only minimal overlap in problem setting or methodology.

Volumetric Decomposition. Computational geometry research has addressed a number of problems which bear strong similarities to our setting. Any convex volumetric decomposition can clearly be converted into a height-field block decomposition by splitting convex parts into two along an equator plane, separating faces with up and down pointing normals. Unfortunately, computing a minimal size exact convex decomposition is known to be NP-hard [Chazelle, 1984, Tor and Middleditch, 1984].

While practical surface-based approximate convex decomposition methods exist [Kraevoy et al., 2007], they do not produce a convex volume decomposition, as they do not prevent the convex hulls of the computed charts from overlapping, and do not prevent the height-field blocks induced by the charts from intersecting. Approximate volumetric convex decomposition, e.g. [Attene et al., 2008, Lien and Amato, 2007], relaxes the convexity requirements to obtain a smaller number of parts; separating these parts along the equator may result in non-height-field blocks. Thus neither method is suitable for our needs as we require strict height-field constraint enforcement.

Pyramidal decomposition aims to decompose an entire volume into height-field blocks, or pyramids. [Fekete and Mitchell, 2001] proved that both the 3D version of this problem and the 2D version on polygons with holes are NP-hard. The approximate pyramidal decomposition method of [Hu et al., 2014], discussed in more detail below, relaxes the pyramidality constraints to obtain approximate compact decompositions within a realistic time frame. Our framework requires exact satisfaction of the height-field or pyramidality requirements and requires all surface points on the model to be covered by height-field block surfaces; contrary to the traditional setting we allow, and even encourage, the existence of an unfilled interior void within the model. To the best of our knowledge this setting has never been explored before.

Height-field Decomposition for Fabrication. A number of papers specifically address height-field decomposition for fabrication.



[Alemanno et al., 2014] propose a user assisted method for decomposing 3D shapes into height-field blocks. Their method is driven by a manually crafted inner structure, which describes the bases and the orientations of each block, fully defining the block decomposition. Overlaps between pairs of blocks, are resolved using an interlocking zipper pattern, where regions shared by multiple blocks are expected to satisfy the height-field requirement for both blocks. This assumption does not hold unless special care is taken in the construction of the inner base structure, (see inset). Our approach algorithmically computes the inner structure and automatically resolves such configurations if and when they occur (Section 4.3.3). Thus our output can be used as an input for this framework, replacing the manual structure design.

[Hu et al., 2014] propose an algorithm for approximate pyramidal decomposition and advocate using it for 3D printing. As they observe, a height-field block, or a pyramid, can be printed standing on its base, thus maximizing its stability and ensuring that no support structures are needed to sustain it during its fabrication. Since as observed earlier, exact pyramidal decomposition is NP-hard to compute, they opt for only weak enforcement of height field constrains and have no direct control on how far the results deviate from a desired approximation accuracy (Figure 4.11). The method is therefore unsuited for 3-axis milling where the height field constraint needs to be strictly satisfied. While this framework seeks to decompose the entire volume into pyramids, we allow for interior voids, while enforcing surface coverage, or the expectation that each point on the model's surface is covered by the height-field surface of some corresponding block. This surface-based formulation is sufficient for both milling and 3D printing settings where users largely care about the look of the resulting model, rather than its interior, and has the extra benefit of reducing the amount of material needed for manufacturing, thus reducing costs. Our algorithm strictly enforces the height-field constraints, but can be relaxed to obtain fixed accuracy height-field approximations. It thus can be used as-is for both additive and subtractive manufacturing (Section 4.4).

[Herholz et al., 2015] decompose free-form shapes into a set of approximate height field surface charts for milling and molding. Candidate height field directions are sampled from the Gaussian sphere with a saliency-based approach. As-rigid-as-possible deformation is used to enforce the height field condition on the charts when violated. The method produces segmentations that induce overlapping height-field blocks (Figure 2.1). In their milling examples the authors resort to a manual process to hollow-out the back sides of each part that results in overlap-free shell parts. By using height-field blocks we remove the need for such manual backside processing and guarantee overlap avoidance.

[Gao et al., 2015a] propose a multi-directional 3D printing system that allows to fabricate an object around a cuboidal shell, using its six facets as printing beds. The method is only suitable for genus zero objects which can be segmented into six axis-aligned approximate height field blocks. While the algorithm seeks for a solution that minimizes the overhang angle it cannot guarantee that the resulting angles will be below any specific threshold. Our framework can provide both strictly height-field blocks and blocks with strictly constrained overhang angles regardless of topology (Figures 4.9, 4.12).

Optimizing Construction Sequences. Our formulation of overlap resolution (Section 4.3.3) is inspired by earlier works on construction and assembly sequences, e.g. [Wu et al., 2016, Attene, 2015, Hildebrand et al., 2012, Schwartzburg and Pauly, 2013, Cignoni et al., 2014, Skouras et al., 2015, Lo et al., 2009, Xin et al., 2011, Song et al., 2012, Deuss et al., 2014, Zhang et al., 2016]. However, the constraints and optimization goals we address are distinctly different, with none of these approaches directly applicable to our needs.

Chapter 3

Simplification to Unfoldable Models

3.1 Overview

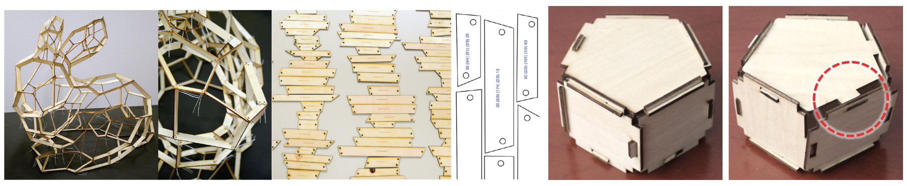
The first method presented in this thesis is still a work in progress, and it aims to simplify an input 3D geometry , producing a polygon mesh composed of a few polygonal flat facets which can be unfolded and which are easy to be assembled, once the unfolded object has been fabricated. We show what it has been developed until now, some partial results and what is still a to do.

As argued in section 2.1, many works aim to simplify a model to get a polygon mesh for fabrication purposes. Unlike all these works , we want to simplify a model in a way that there is no need to address the problem of designing joints necessary for the manual assembly process after the manufacturing.

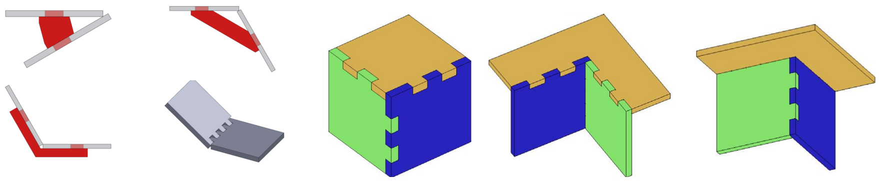
More precisely, if two adjacent primitives (which are mainly cut with laser-cutter machines) need to be manually joined along one edge and the angle between them is arbitrary, then it is necessary to study a joint system that allows the user to easily obtain the desired arbitrary angle avoiding errors which can be propagated and that could cause a bad representation of the desired shape. Some examples of joints used in some previous works are shown in figure 3.1.

The goal of this project is to make the manual assembly process more manageable and less error-prone. To make it possible, the idea is to use CNC machines with V-Router milling tools to carve the primitives in a sheet of rigid material (e.g., plywood, stiff paper, glass).

V-Router (or V-Groove) cutters are accessories for milling which allows engraving furrows on blocks made of millable materials. These V-shaped milling tools enable to mill exact angles according to the tool angle. There are a lot of available tools with different cutting angles on the market, but the most common are multiples of 30 and 45 degrees (Figure 3.2).



(a) [Richter and Alexa, 2015] (b) [Song et al., 2016]



(c) [Chen et al., 2013] (d) [Chen and Sass, 2016]

Figure 3.1: Different type of joints used to assemble the primitives. [Richter and Alexa, 2015] requires a delicate and precise assembly in order to manually reproduce the desired angle; [Song et al., 2016] and [Chen and Sass, 2016] propose joints which are specifically designed for the reproduction of the desired angle; [Chen et al., 2013] propose special joints which are suitable only for CAD models (especially 90°).

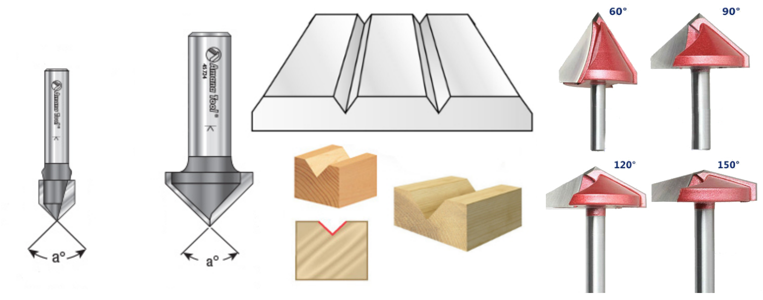
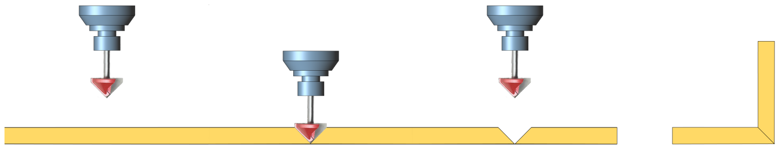
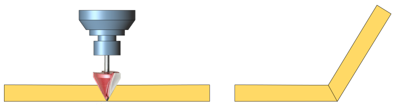


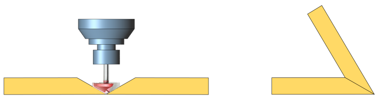
Figure 3.2: V-Router milling cutters: they can mill furrows with correct angles on rigid materials, and they are available with different milling angles on the market. Images courtesy of www.toolstoday.com and www.aliexpress.com.



(a) 90° dihedral angle made with a 90° tool



(b) 120° angle made with a 60° tool



(c) 60° angle made with a 120° tool

Figure 3.3

It is possible to easily fold the primitives together, to obtain the desired dihedral angle along the edge, using these cutters to carve from top to bottom the sheet along the edge of each pair of adjacent primitives (Figure 3.3).

To use this carve-and-fold strategy, the primary constraint that the simplified model needs to satisfy is that all its internal dihedral angles belong to a restricted and well known set that contains only the angles which can be carved using V-Router milling cutters. As a direct consequence of this constraint, we have that also each facet normal of the simplified model can only belong to a restricted set of values.

An initial attempt to provide a solution to this problem appears in [Muntoni and Scateni, 2014]. They present a method to compute the simplification in two steps. They first calculate a topology by assigning a label representing a target normal to every triangle. In the second phase, they try to compute the geometry by rotating every triangle to match the target normal. The main issue with this method is that it does not guaranteed that a corresponding geometry exists for the computed topology. The generation of a water-tight manifold geometry, starting from a topology having the constraint that only a limited number of possible facet normals exist, is still an open problem.

3.2 Marching Cubes

The approach we propose is strictly related to the Marching Cubes algorithm [Lorensen and Cline, 1987]. Marching Cubes, on its naive version of the algorithm, generates a triangle surface mesh starting from a scalar field of boolean values that are discretized to a lattice composed of cubes. The surface is generated using look-up tables where every permutation of values on each cube of the scalar field is associated to a set of triangles. All the triangles of the resulting surface are generated from a restricted and well-known set of combination of cubes. Guaranteeing the regularity of the lattice, the possible

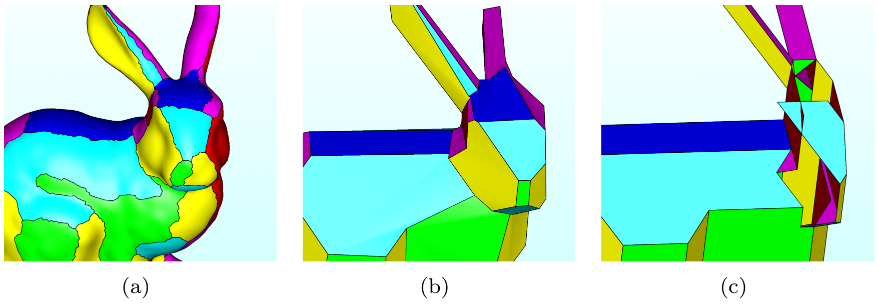


Figure 3.4: Problems on [Muntoni and Scateni, 2014] approach, where the topology obtained on the bunny model (a) doesn't correspond to geometry with facet normals restricted to a limited number of fixed values (c).

triangle normals of a mesh computed with the algorithm belong to a well-known set. As a direct consequence, all the possible dihedral angles between triangles are finite and well-known. We will describe the set N of all the triangle normals which can be generated by the Marching Cubes algorithm on the next paragraph.

A face normal which belongs to the set N is a 3D vector with only three allowed values per component:

- $+s$;
- 0;
- $-s$;

where s is a non-zero scalar value. We define the set of normals N as the set composed of the normals having as components all the possible permutations with repetition of the three values listed above, except the vector $(0, 0, 0)$. We can divide the elements of this set into three different categories:

- Category 1: One component different from 0: $\begin{bmatrix}\pm s & 0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & \pm s & 0\end{bmatrix}$, $\begin{bmatrix}0 & 0 & \pm s\end{bmatrix}$;
- Category 2: Two components different from 0: $\begin{bmatrix}\pm s & \pm s & 0\end{bmatrix}$, $\begin{bmatrix}0 & \pm s & \pm s\end{bmatrix}$, $\begin{bmatrix}\pm s & 0 & \pm s\end{bmatrix}$;
- Category 3: Three components different from 0: $\begin{bmatrix}\pm s & \pm s & \pm s\end{bmatrix}$.

The adjacency of two polygonal faces f_1 and f_2 having normals n_{f_1} and n_{f_2} such that $n_{f_1}, n_{f_2} \in N$, $n_{f_1} \neq n_{f_2}$ and $n_{f_1} \neq -n_{f_2}$ can be summarized as follow:

- A facet with generic normal $\pm u$ (cat. 1):
  - adjacent to a facet having normal with only one component different from 0 (cat. 1):

        - ◇ always generates a 90° angle;
    - ○ adjacent with a facet having normal with two components different from 0 (cat. 2) generates a:
        - ◇ 135° angle if the u component is different from 0 and it has the same sign;
        - ◇ 45° angle if the u component is different from 0 and it has opposite sign;
        - ◇ 90° angle if the u component is 0;
    - ○ adjacent to a facet having normal with three components different from 0 (cat. 3) generates a:
        - ◇ $\approx 126°$ angle if the u component has the same sign;
        - ◇ $\approx 54°$ angle if the u component has opposite sign;

- A facet with generic normal $\pm u \pm v$ (vector with two components different from 0, cat. 2):
    - ○ adjacent to a facet having normal with two components different from 0 (cat. 2) generates a:
        - ◇ 120° angle if it has a component (u or v) different from 0 and with the same sign, and the other one is equal to 0;
        - ◇ 60° angle if it has a component (u or v) different from 0 and with opposite sign, and the other one is equal to 0;
        - ◇ 90° angle if both the u and v components are different from 0 and only one has the same sign;
    - ○ adjacent to a facet having normal with three components different from 0 (cat. 3) generates a:
        - ◇ $\approx 145°$ angle if it has both u and v components with the same sign;
        - ◇ $\approx 35°$ angle if it has both u and v components with opposite sign;
        - ◇ 90° angle if one component (u or v) has the same sign and the other one has opposite sign;

- A facet with generic normal $\pm u \pm v \pm w$ (vector with three components different from 0, cat. 3):
    - ○ adjacent to a facet having normal with three components different from 0 (cat. 3) generates a:
        - ◇ $\approx 70°$ angle if the number of components with the same sign is even;
        - ◇ $\approx 110°$ angle if the number of components with the same sign is odd;

3.3 Method

3.3.1 Automatic Simplification

Initialization. We start generating a regular lattice on the Bounding Box of the input mesh, and for every lattice point we check whether the point is inside or outside the shape, assigning a sign accordingly. Applying the Marching Cubes algorithm to the lattice, we obtain a surface mesh which is a possible solution to our problem because of the restricted facet normals of the mesh. However, as shown in Figure 3.5, the resulting mesh is composed of a significant number of polygonal facets (we actually consider adjacent triangles with the same normal as a unique polygonal facet). We set the grid spacing as the average edge length of the mesh multiplied by an user parameter, which defines the "granularity" of the final simplified mesh.

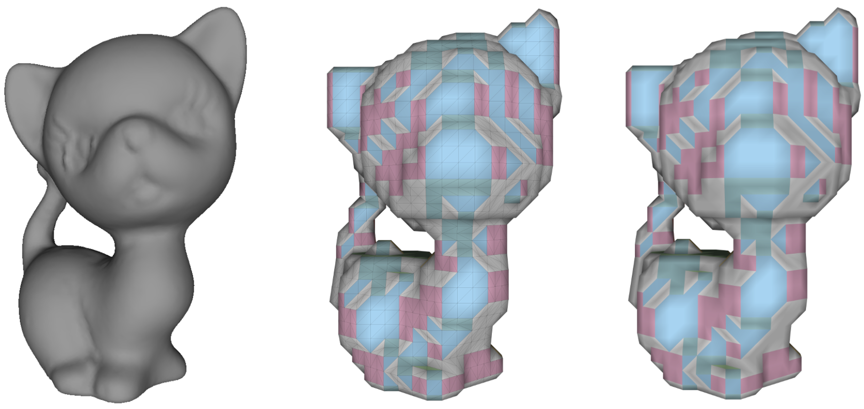


Figure 3.5: Input model (left), a Triangle Mesh (center) and a Polygon Mesh (right) generated by the naive Marching Cubes Algorithm. The meshes are respectively composed of 3628 triangles and 408 polygonal facets. The triangle normals set colors.

Geometry. The idea behind our method is to make changes to some signs of the regular lattice on which the Marching Cubes algorithm is applied. To identify the lattice vertices for which we want to change their sign, we introduce the concept of *Mask*. A Mask is a set of adjacent cubes having a specified combination of signs on their vertices (and, therefore, a combination of adjacent triangles generated by Marching Cubes) that we do not want to have on our output model. Every mask has a set of *Points of interest* to change this configuration of adjacent signs. These points, when switched, can resolve the local bad configuration or move it to another place, improving the simplified global model.

An example of a Mask is shown in Figure 3.6. The shown mask (left) is composed of four adjacent cubes with specified signs on its points, and is composed of two sets of *Points of Interest* (respectively circled in orange and cyan). Only one of these sets is selected and switched in order to change the triangulation (center or right). We designed a set composed of 18 types of masks that, applying rotations, generate a total number of 340 masks. We enumerate the set of all the types of masks on Appendix A.

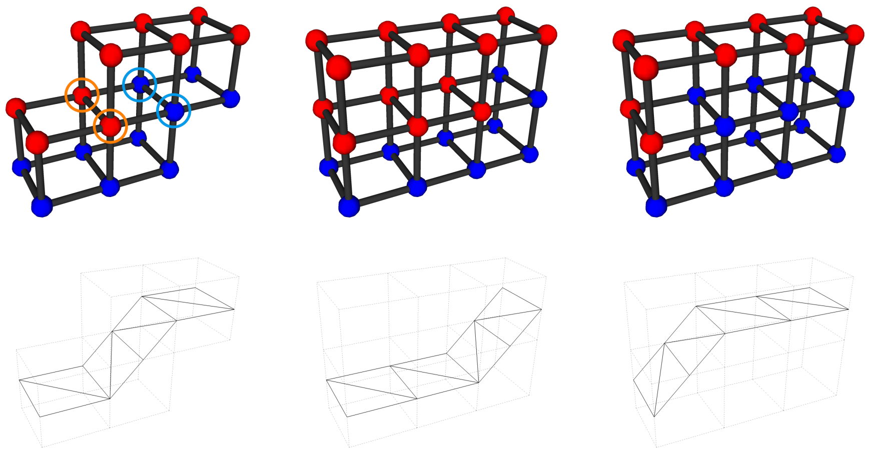


Figure 3.6: We show an example of a mask (left) which generates a step between two facets with normal of cat. 1 and a facet with normal of cat. 2. This configuration can be solved or moved switching the signs circled in orange or in cyan. If we switch the cyan vertices, we obtain the configuration presented at the center. Otherwise, if we change the orange vertices, we obtain the configuration figured on the right. We choose the set of vertices which favors the expansion of the bigger involved chart, deleting or moving the undesired local configuration.

For some masks, there are different ways to solve or move the undesired local geometry using different sets of Points of Interest, and only one set has to to be chosen. We give priority to bigger polygonal facets, but to do that it is necessary to compute a segmentation on the triangle mesh to compute all the areas. This is an massive operation to do for every switch of sign (which changes the triangle mesh and, therefore, the segmentation). However, every switch of signs on the lattice vertices is a local modification on the Marching Cubes triangle mesh (a switch of a point sign reflects a change of the triangles generated by its eight incident cubes of the lattice), which means that also the segmentation can be locally updated. Therefore, to make the switch operation fast we link three data structures to each other: the Lattice, the Marching Cubes triangle mesh, and the Segmentation. Each cube of the lattice generates triangles of the Marching Cubes mesh, and it keeps the references to them. Every triangle links to the chart (that is a polygonal flat facet) which belongs to the Segmentation. When we find a matching Mask on the lattice, we give priority to the Points of Interest that link to the bigger charts of the Segmentation, and we modify only the involved charts whenever a sign switches.

We actually modify the geometry of the model by putting all the cubes of the lattice in a queue and checking if every cube and its neighbors match with a mask of cubes that belongs to our set of Masks. If they match, we switch its best set of points of interest, we update all the data structures, and we push back all the modified cubes in the queue. The process ends when the queue is empty, or a loop is detected (making sure that only cubes which generate loops are left in the queue). We show in Figure 3.8 some results using the presented method. These models have still some features (like small spurious facets between big orthogonal facets) that are uneasy for a fabrication task. However, they are bound to the Marching Cubes Algorithm, as shown in Figure 3.7. We will deal with this problem by proposing a graphic user interface, which is presented in the next paragraph.

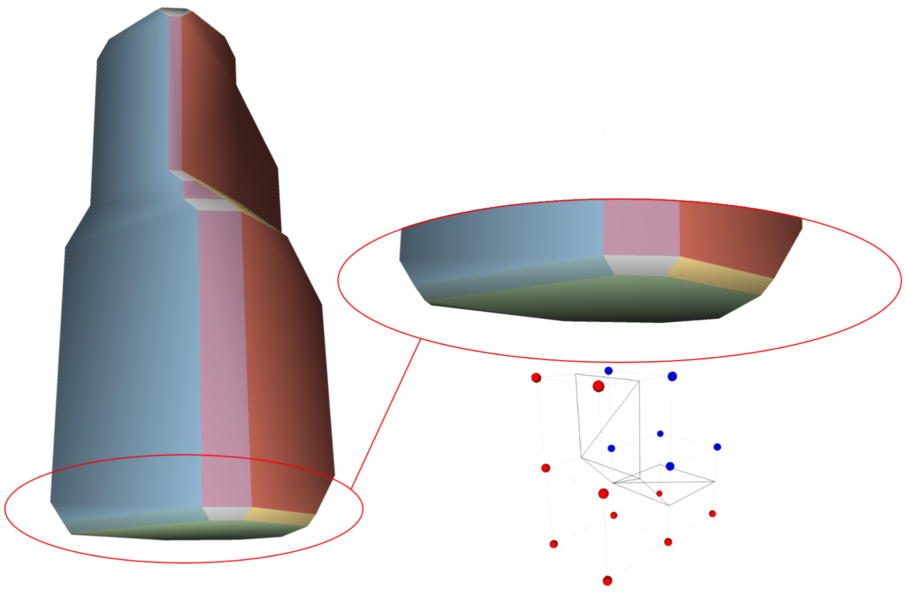


Figure 3.7: Using the Marching Cubes lookup table, it is impossible to obtain a mesh with 90 degrees dihedral angles. This means that there will always be a small facet which acts as a junction between two facets with a 90 degreed dihedral angle.

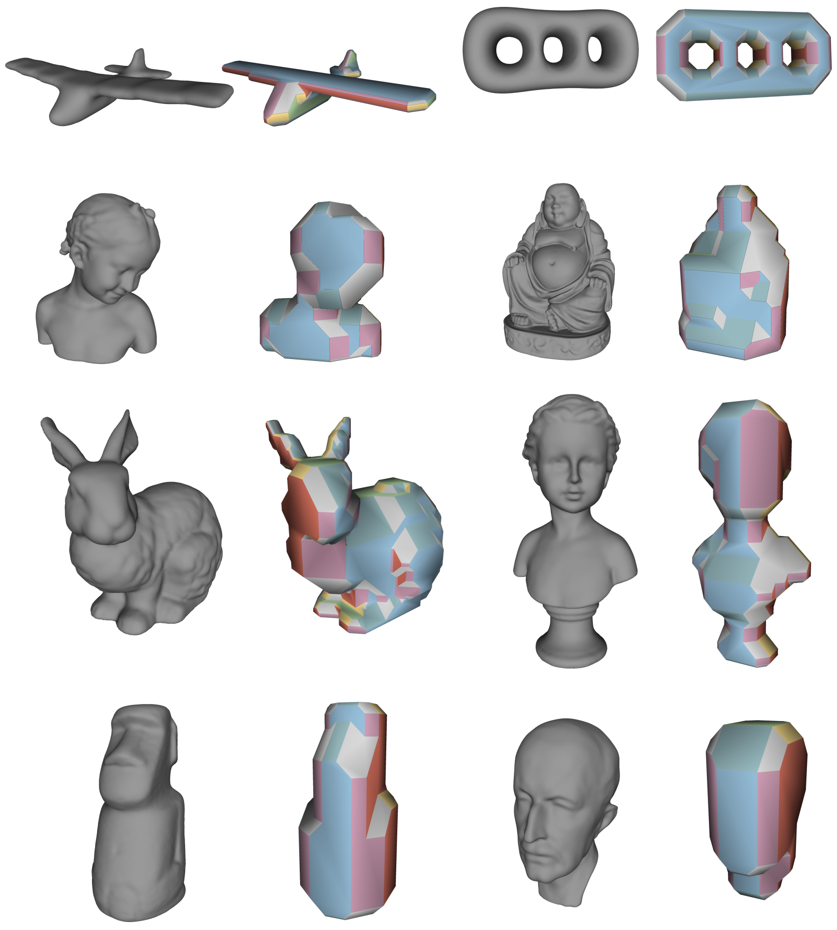


Figure 3.8: Some results obtained using the method described in section 3.3.1.

3.3.2 User-Driven Simplification

We deal with the problem shown in figure 3.7 by providing a Graphic User-Driven tool that allows the user to select and remove some unwanted small polygonal facets, keeping the shape manifold and watertight. We will briefly explain the method, and we invite the reader to see [Scalas and Scateni, 2016] for more in-depth study.

The main idea is to remove the polygonal facet that is selected by the user, and then close the surface by extending all the adjacent facets along the hole. Once the undesired facet is selected, there are two different outputs that are immediately visible to the user:

- the closed surface without the undesired facet, or
- an error message which communicates to the user that it is not possible to close the surface due to local configuration of the adjacent faces.

The process behind the deletion of a single facet is represented in figure 3.9. The first step removes merely the facet selected by the user, its edges, and its vertices. All its adjacent facets become then non-closed facets. We collect all the non-closed facets in a circular buffer, following the adjacency order. Every pair of adjacent facets generates a straight half-line, which is the intersection between the two planes defined by the planar facets, having as origin the survived vertex of the edge lying on the line. We first look for a couple of adjacent straight lines (which is a triplet of adjacent facets) for which there is an intersection, and we choose the closest intersection to one of the vertices of the deleted facet. If this intersection doesn't exists it means that the surface cannot be closed if the initial facet would be removed. If the intersection exists, the intersection point will belong to the new mesh, and the facet at the center of the triplet will be closed with two edges that will meet each other at the intersection point. The facet at the center will now exit from the circular buffer, and the two external facets of the previous triplet will become adjacent, generating a new straight half-line with origin in the early created vertex. The procedure iterates on the triplet which involves the earlier involved two facets and where the intersection between the straight half-lines is closest to the created vertex. The process terminates when:

- there are only two facets in the buffer;
- the last three or more facets in the buffer have the intersection between all the straight half-lines that lies on the same point;
- two adjacent triplets (sharing two adjacent facets) don't have an intersection.

This approach is correctly working on facets having a fully convex or fully concave neighborhood, and when the intersections do not involve external facets that are not adjacent to the undesired facet. This particular case is complicated

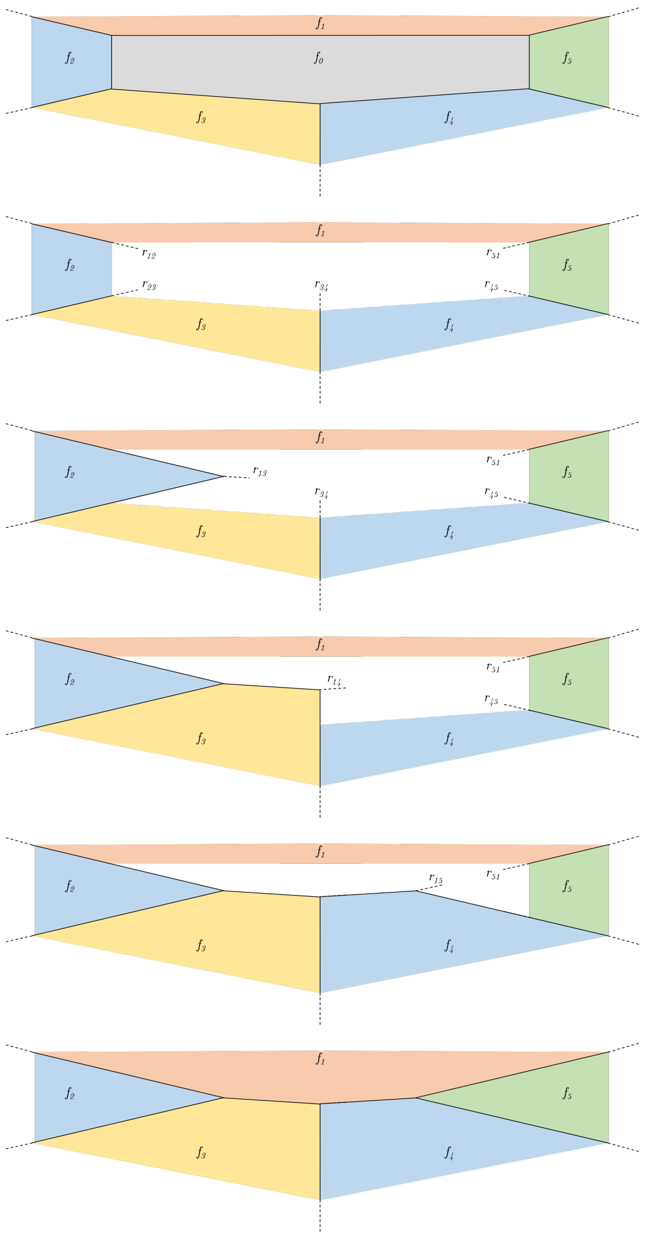


Figure 3.9: Deletion of a facet. After removing f_0, a circular buffer containing all its adjacent facets is created. The triplet f_1, f_2, f_3 is selected at the beginning, the intersection between the two rays r_{12} and r_{23} is computed, and the facet at the center of the triplet (f_2) is removed from the buffer (c). The process is iterated for the next nearest intersection until the surface is closed (f).

to manage due to the high variety of cases which can happen. This case is still an open problem and we plan to solve it on future works. For the presented application, we only show an error message. An example of some facets deleted on the bottom of the moai model using this method is shown in figure 3.10.

The user can also select multiple adjacent facets and delete them together. This feature allows the simplification of shapes having local configurations in which a facet has two adjacent facets with the same normal but a different lying plane. In this case, deleting the facet only would be impossible. The deletion algorithm used is the same as shown above. An example is shown in figure 3.11.

In figure 3.12 we show the Moai model before and after the simplification by the user. The initial model was made of 45 polygonal facets, and the final result is composed of 21 polygonal facets. The total time required for the simplification is about one minute.

3.3.3 Unfolding

We also provide an edge-unfolding heuristic algorithm (Figure 3.13) designed on purpose for our meshes. We studied different unfolding strategies; we refer to [Nuvoli and Scateni, 2017] for deepening. One of the best heuristics we studied uses a normal ordering towards a given number of directions for the facets. Furthermore, it makes different post-processings to minimize the final number of connected components of unfolded faces, hence maximizing the average area of every connected component.

Starting from our polygon mesh, we try to unfold the highest number of facets in a single subtree, such that each face in the unfolding is not overlapping with any other. At each iteration we add a new reachable facet, checking if it causes an overlap. At the end of these iterations, some facets could not have been included in the starting subtree, and that means they can't be attached to any facet of the current unfolding with no overlaps. These remaining faces are often single disjoint faces, and we want to avoid these situations. Therefore, we try to create a new unfolding starting from the smallest subtree and repeating the iterations with the goal of unfolding the set of the remaining facets. If the starting unfolding produces more than one subtree, then the resulting unfolding will be a spanning forest. The method tries to maximize the average area covered by all the spanning trees.

3.4 Conclusions and Future Works

The proposed method is still a work in progress, but the results are encouraging. However, the results shown are improvable in different respects. The main issue is given by the set of masks, which is handmade. We need to find a set which can be automatically generated using a set of well-defined rules and which still solves the problem in a proper way. We also need to improve the method on symmetric models, and we need to make the algorithm independent from the processing order of the masks. We also need to focus on the post-processing to transform the Graphic User-Driven tool for the deletion of small faces in a full-automatic approach. Another idea is to introduce a texturing-manager to the method, to have a mapping from a texture attached to the input model to a texture which could be printed and glued on the final simplified model. Once all these problems are solved, we will finally be able to fabricate our tangible results.

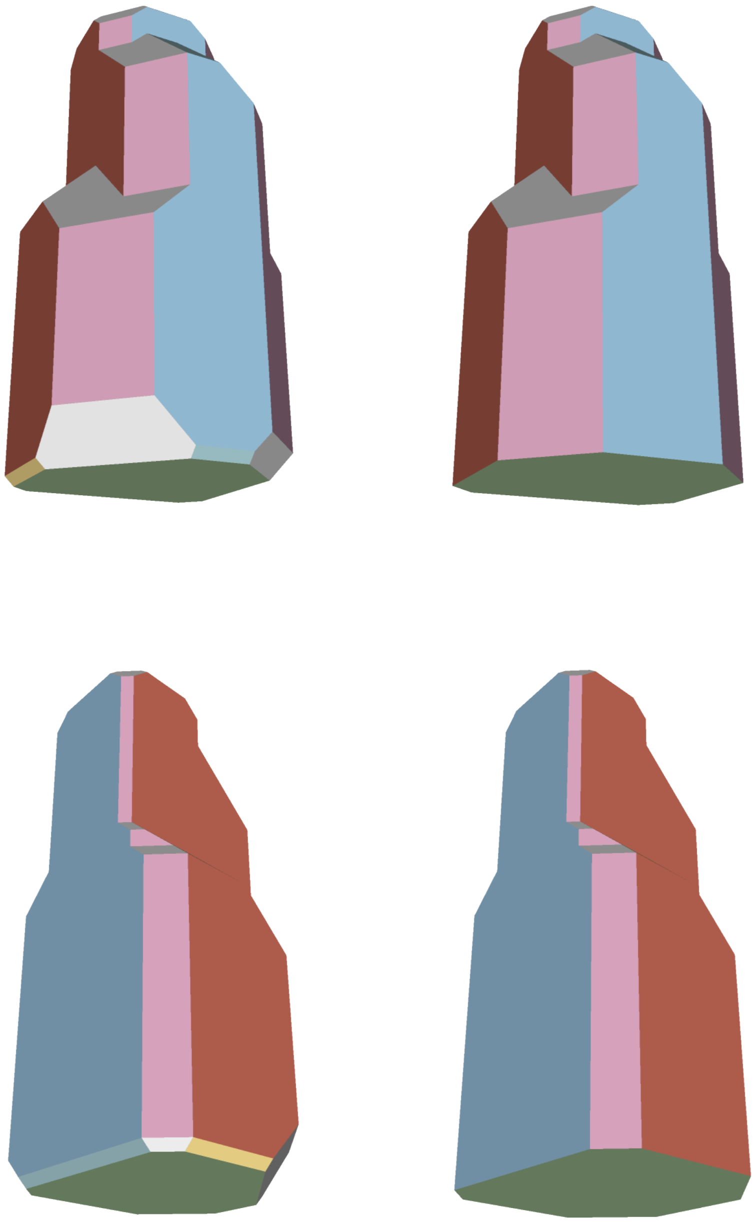


Figure 3.10: The bottom of the moai before (left) and after (right) removing some spurious facets with our method.

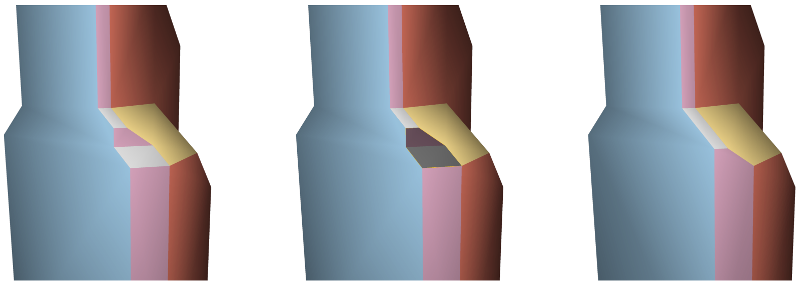


Figure 3.11: Selection and deletion of two adjacent facets. The facets cannot be deleted singularly due to the adjacency of two facets with the same normal but a different lying plane. If selected together, the surface can be successfully closed thanks to the extension of the adjacent facets on the border of the two deleted facets.

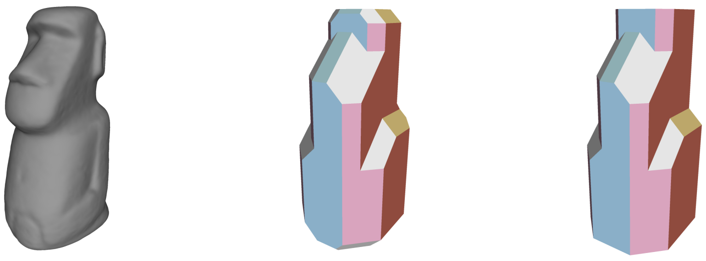


Figure 3.12: The pipeline for the simplification of the moai model: the input mesh (left) is first automatically simplified (center) using the method described on 3.3.1, then is finalized by the user with our tool described on section 3.3.2. After the user-driven simplification, we obtained a model composed of 21 polygonal facets versus a starting number of 45.

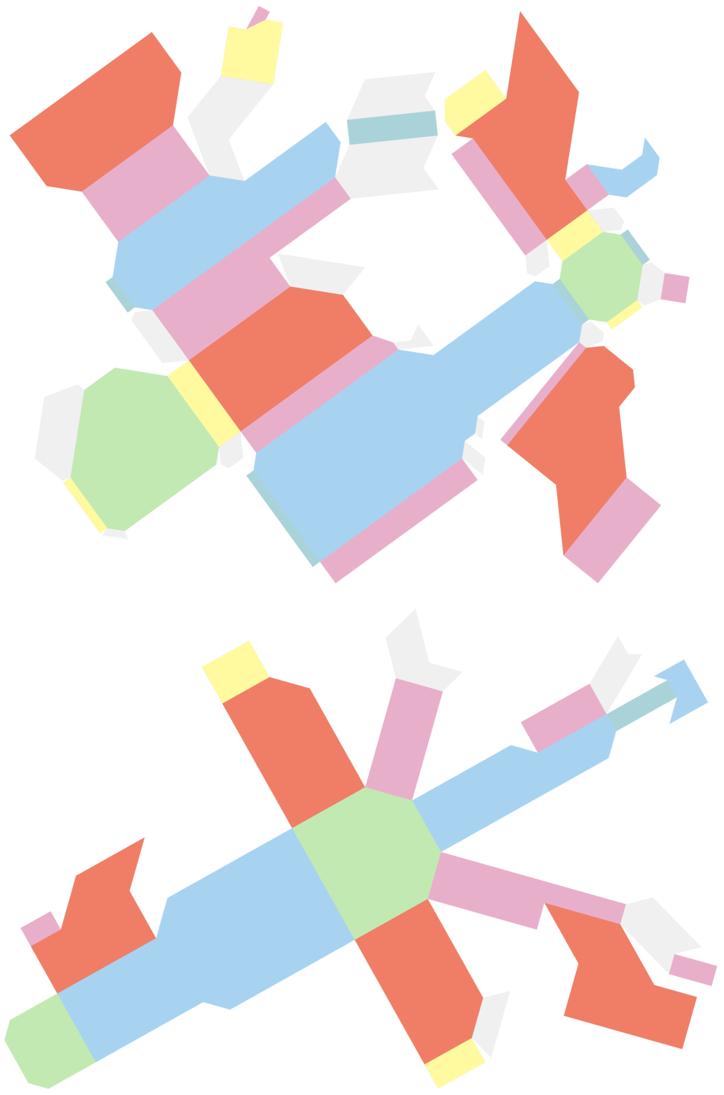


Figure 3.13: The unfolding of the moai model (bottom-left of figure 3.8) before (up) and after (down) the user-driven simplification.

Chapter 4

Heightfields Decomposition

4.1 Overview

Requirements and Desiderata. A decomposition of an input mesh into blocks suitable for 3-axis CNC milling must account for the following criteria. It should assign each point on the surface to a corresponding height-field block, *covering* the input surface. To facilitate manufacturing, the blocks should obviously never overlap and should remain strictly inside the input surface (Figure 4.1). To reduce manufacturing time and to facilitate easy assembly, we need to keep the number of blocks small and maintain comparable block sizes, avoiding tiny blocks. At the same time, for manufacturing purposes, these blocks do not need to cover the interior of the processed model; in fact, increasing the interior void leads to cost and time savings as fabrication requires less material.

Based on these requirements, our algorithm's goal can be formulated as computing a decomposition that satisfies the constraints above while simultaneously minimizing the number of blocks and maximizing the size of the smallest block. While this specific problem setting has not been investigated before, closely related problems such as minimal pyramidal decomposition, or covering a volume by non-overlapping height-field blocks [Hu et al., 2014, Fekete and Mitchell, 2001], have been shown to be NP-hard and have no known exact or approximate polynomial time solutions. To obtain a pyramidal decomposition within an acceptable computation time [Hu et al., 2014] significantly relax the height-field constraints and consider only a finite set of possible height-field orientations. In our setting such relaxation is not possible, since the identified constraints are critical for manufacturing. We note that, as discussed in Section 4.3, we can always guarantee a valid solution by quantizing the set of possible block base and side face orientations to the major axis directions. We use this observation to develop an algorithm that is guaranteed to obtain valid decompositions that satisfy the constraints exactly and can be computed within a feasible time frame. As shown by our results, our algorithm robustly produces

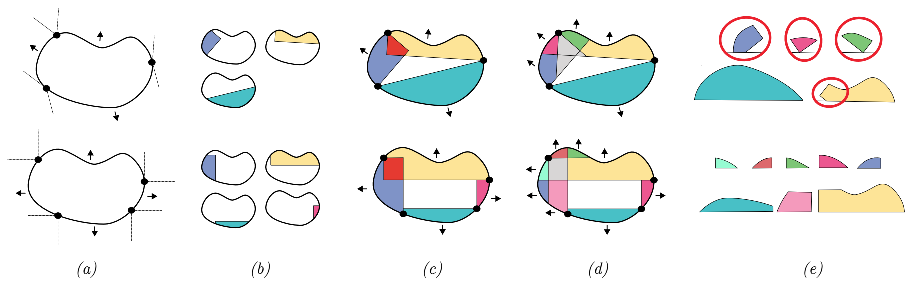


Figure 4.1: From height field segmentations to height field solid blocks: (a) two alternative height field segmentations defined on the boundary of a simple shape. Top one uses arbitrary directions, the bottom is constrained to the global axes; (b) the resulting minimal solid blocks with flat base defined by each height field segment. Notice that raising any of the bases would leave a portion of the boundary uncovered; (c) some blocks overlap in the interior of the shape (red extent); (c) overlaps are resolved by splitting the blocks along their supporting lines, thus increasing the number of blocks; (e) the split operation produces 4 invalid blocks out of 5 for the non axis aligned case (top, red ovals). Constraining the height field directions to the global axis we can always guarantee a valid solution (proof in Appendix C)

decompositions with acceptable, even if sub-optimal, block counts for a range of complex geometries.

Optional Human-Guidance. Our method is fully automatic but allows for optional human guidance for a number of output properties: user can trade input fidelity for lower block count (Section 4.3.5); they can limit block size to ensure that each block fits into the printing bed of a user-specific CNC milling machine and to provide tradeoff between number of blocks and amount of material used for fabrication (which is important for certain materials and could lead to significant cost savings, Section 4.3.2).

Support-Free Additive Manufacturing. Height-block decomposition can benefit applications beyond 3-axis CNC machining. Deposition modeling printers require supports when printing models with large overhangs. Eliminating supports reduces printing time and increases surface quality, motivating methods that seek to decompose models into blocks with limited overhang extent or angle, e.g. [Hu et al., 2014]. Our framework supports coarse block decomposition with relaxed height-field constraints, allowing each block's top surface to have overhangs while constraining the overhang angle (Figure 4.12). Contrary to previous attempts it allows strict constraints on the overhang angle based on printer properties, leading to perfect geometric fidelity.

Contribution. Our core contribution is a computational solution for manufacturing generic geometries using single pass 3-axis CNC machining. We achieve this goal by providing an algorithm for height-field block decomposition that produces compact sets of blocks that satisfy all manufacturing constraints. We demonstrate the practical applicability of our algorithm on 5 fabricated results, 4 of which are milled (Figures 1.4 and 4.10) and 1 3D printed (Figure 4.12), and compare our method against potential alternatives (Figures 2.1 and 4.11). To ensure replicability of our results and to accelerate adoption of our technique, we attach a reference, open-source implementation of our algorithm in the additional material.

4.2 Problem Setting

Formal Problem Statement. We can formulate height-block decomposition as a semi-volumetric partition of an input geometry into blocks that satisfy the following requirements:

1. *Axis*: each block B has an assigned *milling direction*, referred to as the *axis*;
2. *Base*: each block has a flat polygonal base b orthogonal to its axis n.
3. *Height-Field*: each block has height field geometry with respect to its milling direction - i.e. for any point p inside the block B, the line segment between p and the perpendicular projection p' of p onto the base b lies entirely inside B. Moreover the block is located strictly to one side of its base. The block is bounded by its top surface, the base and optional *side* faces orthogonal to the base.
4. *Maximal Size*: the size of each block is not larger than the milling chamber of the CNC machine;
5. *Coverage*: the top, or height-field, surfaces of the blocks jointly cover the input surface;
6. *Non-overlapping*: blocks do not overlap;
7. *Complexity*: the overall number of blocks is small, and each block contains as few thin features as possible.

Conditions (1) through (4) are necessary to fabricate each block using a 3-axis milling machine in a single machining pass. We jointly refer to these four conditions as *block-fabrication* constraints. The block size is often further restricted along the height-field block axis, as the block's *height* affects the amount of material used, and reducing it shortens fabrication time and saves costs. The coverage and non-overlap conditions are necessary to assemble the target model from these blocks. We refer to a block decomposition which satisfies all six conditions as *valid*. The last criterion, while not mandatory, is important for real-life fabrication since an excessive number of blocks would make assembly too cumbersome to attempt, and thin features make the blocks fragile.

Algorithm. To obtaine the desired decomposition we need to solve a highly constrained discrete-continuous optimization problem over a very large search space, the variables of which are the number of blocks, the direction associated with each block, and the location and geometry of each block's base. We require a method that is capable of producing the desired compact solutions on any given input within a reasonable time-frame. The key observation behind our framework is that if we initialize our computation using a set of height-field blocks defined via intersections of the input model with axis-aligned boxes then we can always produce a valid, non overlapping decomposition via a finite set of boolean operations (proof in Appendix C). Specifically, given such set of input blocks that jointly cover the input surface, we can split all blocks along the bounding planes of the bounding boxes associated with all the other blocks (Figure 4.1, bottom). This splitting process eliminates all partial block overlaps and allows for trivial overlap elimination via duplicate block removal. Similarly, all resulting rectangular box sub-blocks that do not touch the surface can be trivially deleted. The resulting set of sub-blocks satisfies all our requirements. In particular, each resulting sub-block is an intersection of a rectangular box with a section of the input surface, a priori constrained to be a height-field surface, i.e. another height-field block. Note that the same argument applies if the initial blocks are intersections of the model with a prism whose faces are axis aligned. Note also that using the same process on non-axis aligned blocks would not produce the desired result (Figure 4.1, top). This hypothetical splitting process provides a robust height-field decomposition of the input, but clearly will generate a large number of blocks. In practice, rather than performing all such splits at once, we perform a more restricted set of Boolean operations that use a subset of the box bounding planes and seek to minimize the number of blocks produced. Our process preserves the height-field property of each individual block and terminates once all overlaps are removed. Thus, in the worst case it produces the same block set as the basic splitting algorithm above, but in practice its outputs while equally valid are drastically more compact. The overlap removal process identified above can be applied to input height-field blocks that overlap not only in their interior but have overlapping top surfaces as well (Figure 4.2) and there is no practical advantage to starting with blocks with non-overlapping top surfaces. Thus rather than computing a disjoint surface segmentation first, we can obtain a desired height-block decomposition as, or more, efficiently by using as a starting point a compact set of, possibly overlapping, height-field blocks that jointly cover the surface of the input model. To ensure output validity and for algorithmic simplicity we constrain these initial blocks to intersections of axis-aligned boxes and the input model (an axis aligned prism constraint while possible is unwieldy to enforce). Lastly, we note that individual blocks induced by purely surface-based segmentation may not be entirely inside the input model. We explicitly constrain the induced height-blocks to be inside the model by using a volume-aware block growth process. We first compute a dense set of maximal size valid height-field blocks that jointly cover the surface of the input model without intersecting it (Figure 4.2b). We avoid redundant intersection tests by reformulating the computation of each block as an unconstrained continuous optimization problem that can be solved efficiently using standard tools. We then compact this set by computing a minimal subset that covers the entire input surface while keeping the overlaps between the selected blocks small (Figure 4.2c).

Given this set of blocks, we perform a sequence of Boolean operations that remove all overlaps and jointly ensure that the resulting blocks retain the height-field property and can be assembled to form the desired output (Section 4.3.3). In computing the sequence we seek to minimize the number of block produced and to maximize the smallest feature size as much as possible, to avoid the creation of fragile components that might break during fabrication. The combined algorithm strictly enforces all manufacturing constraints, while producing decompositions into small number of blocks and preserving the input surface geometry.

Fabrication Advantages. Restricting the sides and bases of all blocks to axis aligned surfaces not only guarantees our ability to generate a valid solution within a feasible amount of time, but has several practical fabrication advantages. Once height-field blocks are manufactured, they need to be joined together to form the target object. Generating suitable joints is an easier task when the angles between interior block surfaces are restricted to a fixed set, and is easiest when all surfaces are axis aligned with respect to a common coordinate system. For small models we can then connect blocks together using standard L-shaped (90°) or flat metal supports to avoid the need for customized joints. For large scale models necessitating additional internal scaffolds for structural robustness, we can similarly use a fixed, standard, set of joints throughout the scaffold. Our orientation restriction thus makes our approach straightforward to integrate into existing manufacturing workflows.

4.3 Method

4.3.1 Initialization

The orientation of the input is arbitrary and may impact the height block decomposition. We fix this degree of freedom by finding the optimal orthonormal coordinate frame to which align our mesh $\mathcal{S}$. Specifically, we compute a global rotation matrix R that minimizes:

$$\arg\min_{R} \frac{\sum_{f \in \mathcal{F}} \mathcal{A}_f \|R\mathbf{n}_f\|_1}{\sum_{f \in \mathcal{F}} \mathcal{A}_f},$$

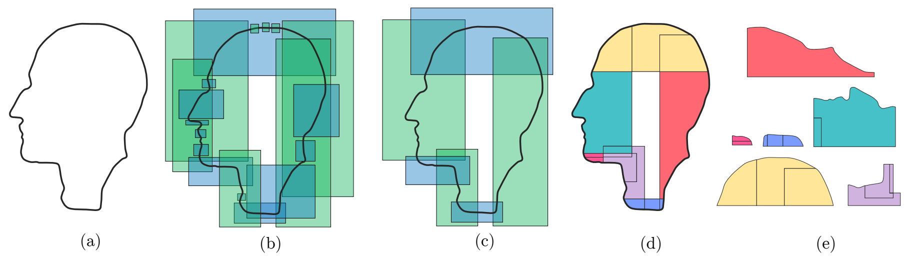


Figure 4.2: Overview: (a) input; (b) dense set of axis aligned maximal height height-field blocks (height-field blocks along the horizontal direction in green, blocks along the vertical direction in blue); (c) minimal block covering of the shape; (d) final blocks after overlap removal (with boundaries of original intersections demarcated); (e) resulting blocks, laid for fabrication.

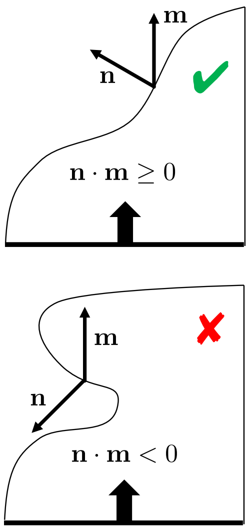
with $\mathbf{n}_f$ denoting face normals, and $\mathcal{A}_f$ face areas. Similarly to [Gao et al., 2015a], we solve this problem with a RANSAC approach, sampling the Gauss sphere using spherical Fibonacci and selecting the orientation that performs best.

4.3.2 Partition into Overlapping Height-Field Blocks

The goal of this stage is to compute a set of individually valid height-field blocks, that jointly cover the input surface, where each block is constrained to be a intersection of a the input model and an axis-aligned rectangular box. To produce a compact final decomposition we seek to minimize the number of blocks in this covering set. A greedy approach to generating such a set would be to start from a seed block, maximally grow it until it cannot be further extended without violating our constraints, and then add more blocks using a similar process until coverage is achieved. This approach is heavily dependent on the strategy used to compute seed blocks, and can result in drastically larger numbers of blocks than necessary. Instead we use a more conservative, if more time consuming, strategy where we first compute a large set of maximal blocks that cannot be further extended without violating our block-fabrication constraints, and then select a minimal subset of them that satisfies our coverage constraint. We avoid time-consuming brute-force evaluation of fabrication constraints by precomputing valid solution spaces for block extension and constraining block computation to these spaces. This two stage process provides a suitable starting point for our overlap resolution stage (Figure 4.2b). An advantage of this two stage approach is that the initial maximal blocks can be computed entirely in parallel, allowing for a trivial speedup. While other strategies could be used to grow the initial maximal blocks, we found our solution to be simple to implement and robust.

Maximal Height-Field Block Set

We desire a set of maximal size blocks that provide a good starting point for selecting a compact subset that covers the entire model. While in general the number of sides such blocks or their base polygons can have is unlimited, searching for all possible block geometries is impractical. To efficiently find a solution we reduce the search space by limiting the set of possible blocks to the intersections between axis aligned boxes and the input shape. To obtain a compact set of blocks that satisfied coverage we start by computing a dense set of axis-aligned maximal size blocks that describe portions of the input surface. We seed these blocks using bounding boxes of individual triangles as a starting point and grow them until they reach a maximal size, where any further expansion would violate validity constraints.


m
n
$\mathbf{n} \cdot \mathbf{m} \geq 0$
m
n
$\mathbf{n} \cdot \mathbf{m} < 0$

Problem Setting. The input to this stage is a closed, intersection-free, triangle mesh $\mathcal{S} = (\mathcal{V}, \mathcal{F})$, where $\mathcal{V}$ is the set of its vertices and $\mathcal{F}$ the set of faces. The output of this stage is a set $\mathcal{B}$ of axis aligned boxes, where the geometry of each $\mathbf{b} \in \mathcal{B}$ is encoded via the positions of its extrema corners (ones with the smallest and largest coordinate values). A box $\mathbf{b}$ is *valid* if its intersection with the input shape is a valid height-field block. We note that for closed volumes, this requirement can be recast as requiring the angle between the outward pointing normal of any input shape triangle fully or partially inside the box and the milling direction associated with the block to be acute. Evaluating this condition explicitly and repeatedly during maximal box computation can be prohibitively expensive. Below we describe an efficient way to sidestep such explicit evaluations.

Initialization. To ensure complete coverage, we initialize the set $\mathcal{B}$ with the bounding boxes of all the mesh triangles. Since each triangle can be part of at most three outward oriented height-field surfaces, we create seed bounding boxes associated with only these three orientations. In general normal directions on a surface change gradually, thus we expect each of these boxes to be valid, i.e. only overlap triangles which satisfy the acute angle constrain vis a vis our three initial axis directions. Section 4.3.5 discuses a pre-process which can be applied to the models to enforce this condition, if not satisfied *a priori.*

Expansion. We seek to maximize the coverage provided by each box while satisfying validity conditions. The validity testing can be reduced to two conditions. First, and most important, we need to test whether the expanded box overlaps with any triangles whose normals point in the opposite direction

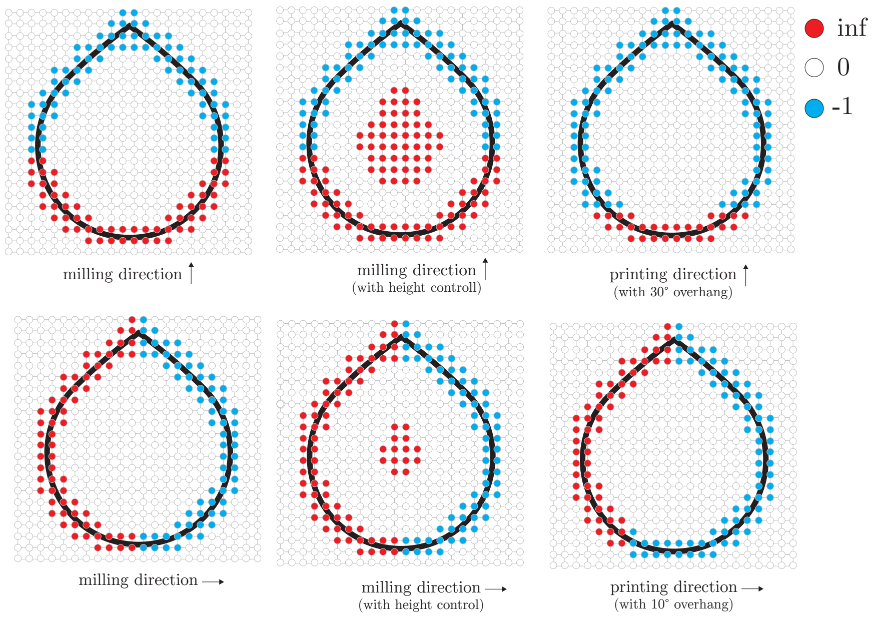


Figure 4.3: Example scalar fields for maximal height field block computation. Left column: The field is infinite next to triangles with normal opposite to the selected milling direction, negative in the areas whose coverage we seek to maximize, and zero elsewhere. Middle column: Defining maximal block height (distance from surface to block base) translates into specifying an infeasible (infinite field value) region inside the model, where the distance to the surface along the axis direction is above this maximal height value. Right column: by relaxing the opposite direction criterion (testing against $90° + \epsilon$ instead of $90°$ we can generate quasi height field blocks useful for 3D printing setup.

to the box's axis. Second, we must at all times ensure that the dimensions of the box, and specifically its height, do not exceed the dimensions of the milling machine processing volume. A naive approach to block computation would be to grow each box using small steps, e.g. adding one triangle at a time, terminating growth if and when the validity constraints are violated. However, the first test in particular can be quite time consuming, and repeatedly performing it for each box at each expansion step can be prohibitively computationally expensive for large models. Instead of testing constraints directly, we define a valid solution space for box expansion and constrain our maximization problem to this space. The solution spaces are defined independently for each of our six orientations and are reused for all blocks which share this orientation. They are formulated so as to simultaneously prevent constraint violation and enable easy continuous optimization of coverage maximization.

Solution space. We define a set of volumetric scalar fields, one for each milling direction $\mathbf{m}$, that smoothly encode an energy function we seek to minimize for all corresponding boxes. We represent each field using a regular grid defined over a bounding cube of the input model (we set grid spacing to the average mesh edge length). The field is designed to be infinite outside the valid region with respect to the axis of interest, negative in the areas whose coverage we seek to maximize, and zero elsewhere (see Figure 4.3), and is specified as follows:

- ∞ on vertices of grid cells that contain triangles whose normals form obtuse angles with the milling direction;
- -1 on vertices of grid cells that contain only triangles whose normals form acute angles with milling direction;
- ∞ on grid vertices that are further from the boundary than our maximal height threshold (this part is optional and used only when we seek to control the height if each block);
- 0 on all other vertices.

We assign continuous values within each cell by using tricubic interpolation. The energy of a box is then defined as the integral of the scalar field inside the box:

$$\mathbf{E}(\mathbf{b}) = \iiint\limits_{\mathbf{b}} s\, dV$$

which can be represented as the sum of the integrals over all cells of the regular grids that intersect the box. Each one of these integrals can be evaluated in closed form; we provide the derivation of this integral and of its derivatives in Appendix B. Note that any box $\mathbf{b}$ with $\mathbf{E}(\mathbf{b}) \neq \infty$ by construction satisfies our block-fabrication constants, with the exception of maximal size which is discussed in the next paragraph.

Optimization. We simultaneously grow all boxes to cover as much of the input surface as possible, while still keeping them valid. The boxes are expanded by minimizing $\mathbf{E}$, subject to additional hard constraints that limit the size of the boxes to prevent them from growing beyond our maximal size threshold, and constrain the initial seed triangle associated with each box to remain inside this box. Both sets of constraints can be expressed as linear inequalities with respect to the position and dimension of the box $\mathbf{Cb} \leq \mathbf{d}$, leading to the following non-linear optimization with linear inequality constraints:

$$\arg\min_{\mathbf{b}} \mathbf{E}(\mathbf{b}) \tag{4.1}$$

$$s.t.\, \mathbf{Cb} \leq \mathbf{d} \tag{4.2}$$

We convert this constrained optimization into an unconstrained one using logarithmic barriers, and minimize it using BFGS with bisection line search.

The optimization is stopped when the residual is smaller than 10^{-6} of the model bounding box diagonal.

Heuristic Pruning. The collection of maximal height blocks extracted using this basic procedure is highly redundant. To improve performance we reduce the set of processed boxes as follows. First, instead of considering all three valid directions for each seed triangle, we only use the milling direction closest to its normal. Second, we seed (and grow) boxes at random triangles, evenly distributed over the surface, and stop seeding new boxes as soon as the entire surface is completely covered, i.e. as soon as all the triangles of the surface have been assigned to at least one height block. While these heuristic could in theory lead to inferior results, they work well in practice, as we demonstrate in Figure 4.4 (bottom part), where we compare the results obtained with and without pruning. The difference in quality is negligible, but the heuristics reduce the computation cost from 47 to 2 minutes.

Minimal Covering

After maximally expanding all block boxes, we compute a minimal subset of them which entirely covers the surface. Computing such a set amounts to solving the classical minimal set cover problem, known to be NP-complete [Cormen, 2009]. We obtain a solution by casting it as an integer linear programming problem:

$$\arg\min \mathbf{1}^T \mathbf{x} \tag{4.3}$$

$$s.t. \sum_i x_i \, \mathbf{a}_i \geq \mathbf{1} \tag{4.4}$$

$$x_i \in \{0, 1\} \tag{4.5}$$

where x_i is the i-th entry of $\mathbf{x}$, a vector of binary variables that indicates if the box $\mathbf{a}_i$ is kept in the minimal covering. $\mathbf{a}_i$ is a binary vector with as many entries as the faces of the input, and where a value of 1 indicates that the corresponding face is contained in the box. We use an off-the-shelf solver (`http://www.gurobi.com/`) to obtain a solution. While in theory the runtime for this step can be exponential, in practice the solver converges to a solution in minutes (Table 4.1).

4.3.3 Overlap Resolution

Given the set of height-field blocks produced by the minimal covering, we seek to resolve the overlaps between them with a minimal increase in the number of blocks and without introducing thin fragile features.

Single Pair. Before addressing the general case, we consider overlap resolution on an individual pair of blocks $\mathbf{b}_1$ and $\mathbf{b}_2$ (Figure 4.5). The overlap between

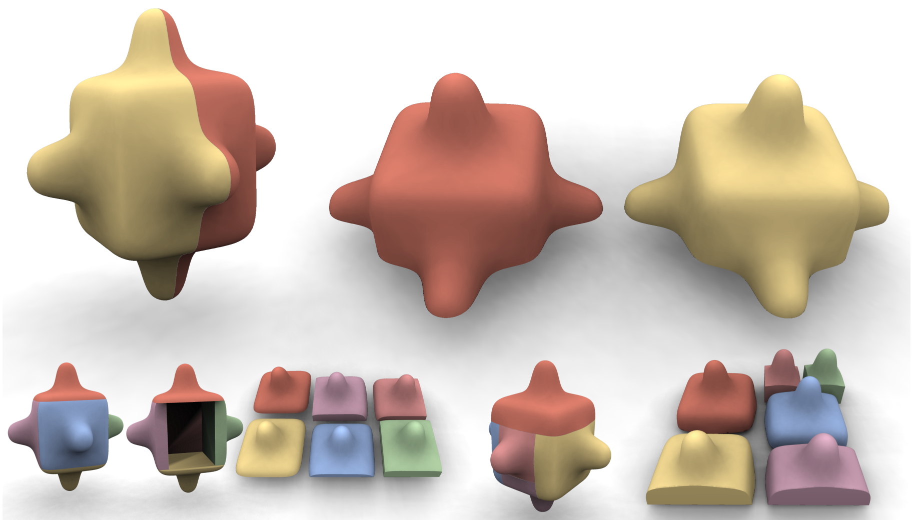


Figure 4.4: We enable users to balance the number of height-field blocks against the amount of material necessary to fabricate the object. Top: the block height is unconstrained, thus the blocks are free to expand in the interior and cover the whole volume. Bottom-Left: the block height is highly constrained, thus leaving a big void in the interior of the model and producing thinner blocks which would require much less time and material to be fabricated. Specifically, the model on top is composed of 2 height-field blocks and its fabrication requires 35% more material than the model on bottom left, which is composed of 6 blocks. Bottom-Right: height-field block decomposition with height constraints and with no heuristic pruning. The difference in quality is negligible, but the pruning reduces the computation cost from 47 to 2 minutes.

the blocks can always be eliminated by subtracting one block from another, e.g. $\mathbf{b}_2$ from $\mathbf{b}_1$. In all but some special cases, which we will discuss later, such a subtraction keeps the number of blocks constant. Post-subtraction, the block $\mathbf{b}_1 \setminus \mathbf{b}_2$ may no longer satisfy the validity constraints (Figure 4.5a). Depending on the configuration, reversing the order and computing $\mathbf{b}_2 \setminus \mathbf{b}_1$ may result in two valid blocks, but it is not guaranteed. In many instances, no order can produce a valid result (Figure 4.5c). Invalid blocks created by subtraction have multiple bases - i.e. polygons orthogonal to the milling direction. Each such block $\mathbf{b}_i \setminus \mathbf{b}_j$ can be therefore converted into a set of valid blocks by splitting it along one or more of the planes it shares with block b_j, such that each such base defines a separate block. While this solution is guaranteed to work, we want to minimize the number of such splitting operations. The basic overlap resolution method for two blocks $\mathbf{b}_1$ and $\mathbf{b}_2$ can be hence formulated as follows: (1) if only one of the two differences is valid use this difference to form a solution (Figure 4.5a); (2) if both $\mathbf{b}_1 \setminus \mathbf{b}_2$ and $\mathbf{b}_1 \setminus \mathbf{b}_2$ are valid blocks,

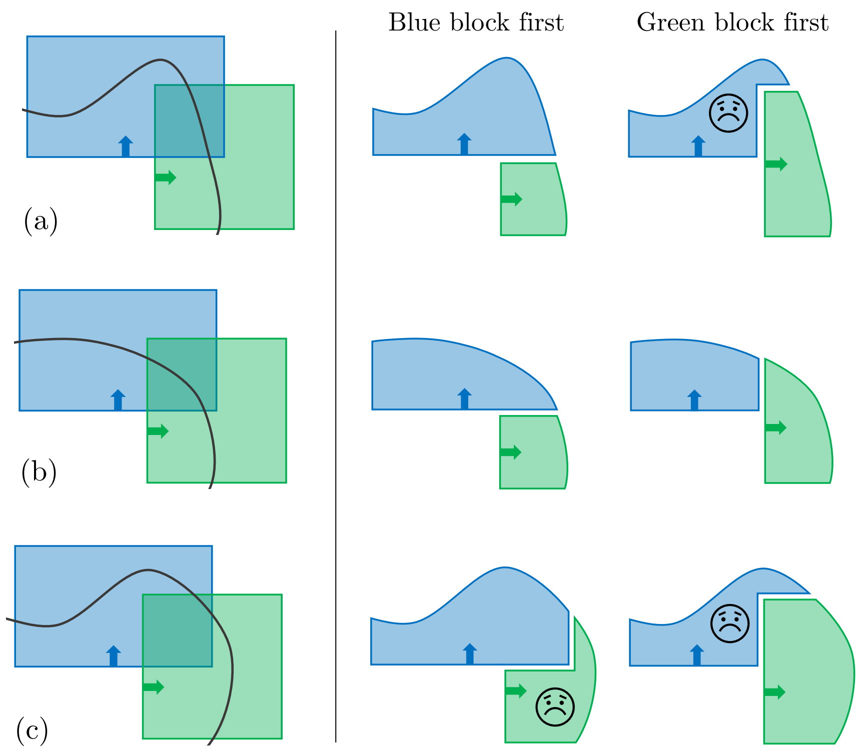


Figure 4.5: Different block processing orders result in different decompositions: (a) processing the blue block before the green one produces two valid height fields — inverting the order the blue block ceases to be a height field; (b) independently on the processing order, two valid height blocks are produced; (c) no valid solution exists, independently on the processing order. In the latter case, splitting one of the boxes allows for a valid processing order.

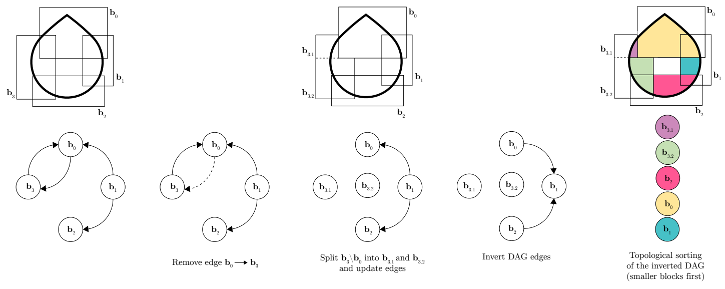

$\mathbf{b}_0$
$\mathbf{b}_1$
$\mathbf{b}_2$
$\mathbf{b}_3$
$\mathbf{b}_{3.1}$
$\mathbf{b}_{3.2}$
Remove edge $\mathbf{b}_0 \rightarrow \mathbf{b}_3$
Split $\mathbf{b}_3 \setminus \mathbf{b}_0$ into $\mathbf{b}_{3.1}$ and $\mathbf{b}_{3.2}$ and update edges
Invert DAG edges
Topological sorting of the inverted DAG (smaller blocks first)

Figure 4.6: Conflicts between intersecting blocks are encoded in a directed graph. An edge $\mathbf{b}_i \rightarrow \mathbf{b}_j$ means that $\mathbf{b}_j \setminus \mathbf{b}_i$ is not a height field. If the graph is acyclic we are guaranteed that, without splitting any box, subtracting blocks using an inverse topological order would produce a valid height-field block decomposition. If the graph contains cycles, we reduce it to a DAG by iteratively removing edges (and splitting the blocks accordingly). Among all the possible inverse topological orders, we select the one that maximizes the size of the smallest height-field block.

perform the subtraction operation that maximizes the smallest output block (Figure 4.5b); (3) otherwise, split one of the difference blocks to obtain valid sub-blocks, selecting the refinement that maximizes the smallest output block (Figure 4.5c).

Multi-Block. We extend this framework to the multi-block scenario by casting overlap resolution as finding a sequence of subtraction operations that minimizes the number of splits required to ensure output validity. When splitting is unavoidable, we prioritize split operations that avoid producing very small blocks.

We represent the relation between adjacent height-field blocks using a directed graph whose vertices represent blocks; each graph edge represents a subtraction order dependency between its end vertices. More formally, we introduce an edge from $\mathbf{b}_1$ to $\mathbf{b}_2$ if and only if the difference $\mathbf{b}_2 \setminus \mathbf{b}_1$ is not a valid height block. Note that it is possible to have two opposite edges connecting the same pair of vertices if both subtraction orders produce invalid blocks (Figure 4.5c). Cycles in this graph exactly correspond to scenarios where splitting cannot be avoided.

To obtain the desired subtraction order, if the initial graph contains cycles, we first transform it into a directed acyclic graph (DAG) by breaking all the cycles (and splitting the associated blocks). We then produce a valid subtraction order by computing an optimal topological order on this acyclic graph. The overall complexity of this step is $O(|s|(n+e)(c+1))$, where $|s|$ is the number of splits, n is the number of vertices, e is the number of edges, and c is the number of cycles.

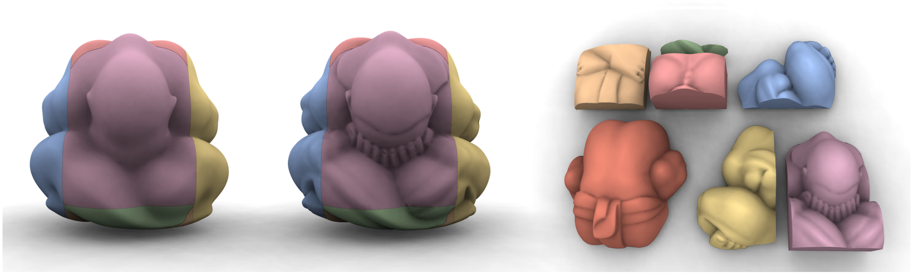


Figure 4.7: To keep the number of height-field blocks low we trade faithfulness for complexity. Specifically, given a detailed model we run our method on a pre-filtered version with no high frequencies (left). We then maximally restore the details while preserving the height filed property everywhere (middle, right).

Reduction to a DAG To break cycles in the graph we iteratively select an edge $\mathbf{b}_1 \rightarrow \mathbf{b}_2$ on each cycle and split $\mathbf{b}_2 \setminus \mathbf{b}_1$ along the height field direction of $\mathbf{b}_2$, generating two or more non-overlapping valid blocks. We are sure that they are valid relying on a simple assumption: cutting a block with any plane containing the milling direction keeps requirements *Axis*, *Base*, and *Height-Field* in both sub-blocks. The vertices corresponding to the generated sub-blocks, denoted $\mathbf{b}_{2,1}, \ldots, \mathbf{b}_{2,n}$, are then added to the graph. Note that each sub-block's vertex can at most inherit the edges of its parent, excluding $\mathbf{b}_1 \rightarrow \mathbf{b}_2$ (since there is no intersection between $\mathbf{b}_{2,1}$ and $\mathbf{b}_{2,2}$ and neither of them intersects $\mathbf{b}_1$). As a consequence, if the block $\mathbf{b}_2$ participated in n cycles, its sub-blocks can participate at most in $n-1$ cycles. This observation guarantees that each refinement step reduces the total number of cycles that graph vertices participate in, and consequently ensures that the reduction process terminates in a finite number of steps, producing a DAG.

We detect all the cycles in the graph using Johnson's algorithm [Johnson, 1975]. Among all the edges participating in a loop, we give priority to the one that maximizes the size of the smallest height block created. The splitting algorithm stops when a DAG is obtained.

Topological Sorting. We produce a valid subtraction sequence by computing an optimal topological order on the resulting DAG. Producing a linear ordering of a DAG's vertices such that for every directed edge $\mathbf{b}_i \rightarrow \mathbf{b}_j$, $\mathbf{b}_i$ comes before $\mathbf{b}_j$ is a classical problem in graph theory. Notice that our directed edges encode pathological splitting orders, we therefore aim to find an *inverse* topological sorting of the DAG vertices (i.e., for every directed edge $\mathbf{b}_i \rightarrow \mathbf{b}_j$, $\mathbf{b}_j$ should come before $\mathbf{b}_i$). We pre-process the DAG by inverting the orientation of each directed edge (i.e., transforming the roots in leaves, and vice versa) and run a standard algorithm for topological sorting [Kahn, 1962]. Among all the possible orderings, we favor the one that maximizes the size of the smallest height-field block. To do so, we use Kahn's iterative algorithm [Kahn, 1962], prioritizing the vertices associated with the smallest blocks. In short, the algorithm works as follows: at each iteration we find the roots of the graph (vertices with no incoming edges — if the graph is a DAG at least one root always exists); we order them from the smallest to the biggest block, and use this order to perform the subtraction, removing their corresponding vertices from the DAG. We repeat the process, iteratively looking for new roots until all the vertices in the DAG have been processed (Figure 4.6).

4.3.4 Improving Blocks Size and Shape

A shortcoming of the method described so far is that it does not explicitly prevent the generation of tiny blocks or blocks with narrow protruding features, which could potentially break during fabrication (due to the stress induced by the milling tip) or during assembly. Such features are usually generated when performing Boolean operations between blocks with close by faces with similar orientation. We describe here a greedy twofold strategy that has no theoretical guarantees but that in our experiments successfully removes narrow features, leading to the formation of well shaped height blocks.

Block Snapping and Shrinking. We process the blocks selected by the minimal covering (Section 4.3.2), aiming to minimize the number of intersections between blocks before starting the overlap resolution (Section 4.3.3). First, we consider all pairs of face-adjacent blocks, that is blocks with same orientation faces for which the distance between these faces is less than a fixed amount (the default is one grid unit, the user can choose to change it). We sort all the candidate pairs according to these distances, and adjust their dimensions reducing the distance to zero, making them perfectly face-adjacent. We then consider the remaining set of intersecting blocks, and try to shrink them in order to avoid overlaps. Notice that shrinking blocks may leave some portion of the surface uncovered, we therefore apply block shrinking if and only if complete surface covering is preserved. These steps result in a conflict graph with less arcs and typically less cycles, thus reducing the number of splits necessary to reduce it to a DAG. On average, using this strategy we decreased the number of box splits by 50%.

Modified Processing Order. If, by processing the blocks in the order computed in Section 4.3.3, we generate a tiny height-block or a narrow feature, we rollback the operation and modify the processing order, giving the current block higher priority w.r.t. all the blocks it overlaps (Figure 4.8). This maximizes the size of the sub-blocks derived from such height-block and typically reduces the number of narrow features produced. We automatically detect small blocks by measuring their volume [Zhang and Chen, 2001], and detect tiny features by measuring the distance between pairs of block side and base edges. More advanced approaches [Zhou et al., 2013] could be used instead but in our

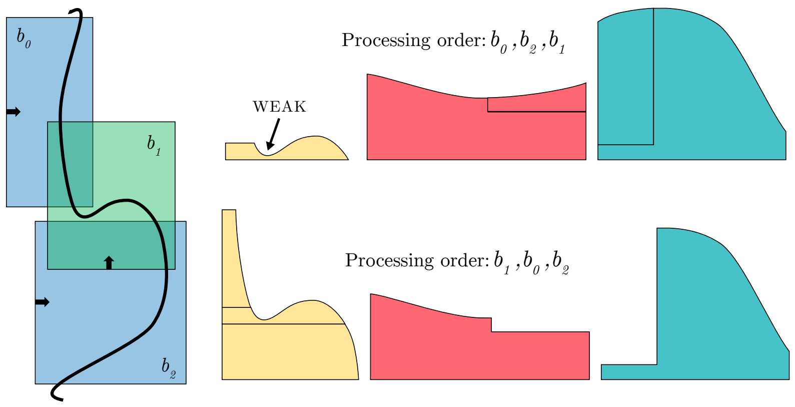

b_0
b_1
b_2
WEAK
Processing order: b_0, b_2, b_1
Processing order: b_1, b_0, b_2

Figure 4.8: Left: a portion of surface is covered with three blocks: b_0, b_1, b_2. In this configuration any processing order would produce three valid height blocks. Right: processing b_1 after b_0, b_2 produces an undesirable narrow feature (top). We detect such configurations and locally modify the block processing order: giving higher priority to b_1 produces a better height block decomposition (bottom).

experiments this was not necessary since all the thin features were detected by the two criteria above.

Post Processing. The boolean processing may result in adjacent blocks with same height field direction. To reduce block count we merge them into a single block, either by raising the base of the lower block along the height field direction, or by lowering the base of the higher block. The condition for performing the first operation is that there are no surface triangles in between the old and the new base. The condition for performing the second operation is more complex: we need to make sure that the new block doesn't intersect any other block in the decomposition, and that it still satisfies the height field constraint. After moving one of the two bases, we merge the two blocks. This post-processing typically merges one or two pairs of blocks.

4.3.5 Faithfulness vs Complexity

Enforcing strict fabrication constraints on highly detailed models often results in an excessive number of height-field blocks. For practical applications, exact fidelity to the input can often be sacrificed to reduce block count and facilitate easier fabrication. We provide an optional mechanism that allows users to reduce reconstruction accuracy, or faithfulness, in exchange for a lower block count. We note that smoother, or less detailed, models typically require significantly fewer height-field blocks to reconstruct than their detailed counterparts. We consequently achieve our target using a two step procedure that removes high-frequency surface details before the decomposition, and reintroduces the removed details into each block subject to preserving the fabrication constraints (Figure 4.7).

To remove high-frequency details from the input shape we use the low-pass filter proposed in [Taubin, 1995]. After computing the height-field block decomposition we reintroduce the high-frequency details using a variation of the Laplacian surface reconstruction framework [Sorkine, 2006], enriched with height field constraints that ensure that the vertices assigned to each block remain above its base with respect to the milling direction and that no triangle flips its orientation. Specifically, after the block decomposition of the smooth geometry is computed, every vertex $\mathbf{v}_i$ is assigned to a block $\mathbf{b}_{\mathbf{v}_i}$, which has a milling direction $\mathbf{m}_{\mathbf{v}_i}$. We then reintroduce the details by minimizing the following energy:

$$\arg\min \|\Delta\mathbf{v} - \delta\|^2 s.t. \tag{4.6}$$

$$\mathbf{v}_i \in \mathbf{b}_{\mathbf{v}_i} \tag{4.7}$$

$$\mathbf{n}_t(\mathbf{v}) \cdot \mathbf{m}_t \geq 0 \tag{4.8}$$

where $\delta_i = \frac{1}{|N_i|}\sum_{v_j \in N_i}(v_i - v_j)$ are the differential coordinates of the original mesh [Sorkine, 2006], and $\mathbf{n}_t$ is the normal of the triangle t. Equation 4.7 ensures that every vertex is constrained to stay above the block's base and can be modeled with a set of linear inequality constraints. Equation 4.8 prevents triangle normals from flipping and is a quadratic condition on the vertex positions. We minimize this energy using coordinate descent, by optimizing one vertex at a time and freezing the others. The per-vertex optimization is solved with Newton iterations and it typically converges within 5 iterations, recovering most of the details of the input meshes (Figure 4.7). Specifically, the average distance between the two models, measured with Metro [Cignoni et al., 1998], is less than 1×10^{-4} times the bounding box diagonal.

Initialization Constraints. Given an input mesh, we assume that bounding boxes of individual triangles define valid height blocks. This condition can be violated in two scenarios. In the first case the top or outer surface defined by the intersection of the box and the input model may not be a height field. This situation can only occur on meshes with high-frequency features, and it is solved using our high-frequency removal preprocess. Initial blocks can, in theory, intersect the input surface. This situation can only occur if the mesh triangle size is larger than the local feature size. Such situations can always be avoided by refining the mesh using one or more rounds of one-to-four triangle subdivision.

4.4 Results

Throughout the paper we demonstrate a range of models decomposed into height-field blocks using our approach, ranging from relatively simple (Spiky cube) to highly complex (Chinese lion, Lion vase) and from relatively smooth (Kitten) to highly detailed (Buddha, Bimba). All our output decompositions are fabrication ready and strictly satisfy all validity conditions. The number of blocks in our decompositions varies from single digits (Moai, Max Planck) to 63 for Fertility.

4.4.1 Milled Results.

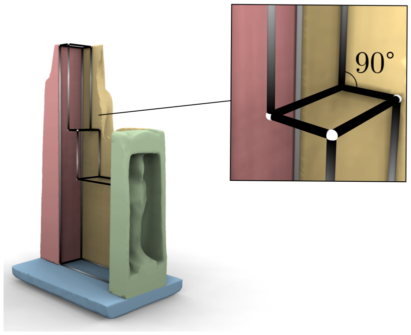
We milled four objects from solid blocks of two different woods: pinewood and beechwood; the former is softer and easier to mill while producing less detailed results, the latter is harder and needs a longer milling time but leads to more detailed models.

We milled *Moai* (Figure 4.10) and *MaxPlanck* (Figure 1.4) from the pinewood blocks; *Buddha* and *Egyptian Statue* from the beechwood blocks (Figure 4.10). *Moai* was milled with a Roland Modela MDX40, while *MaxPlanck*, *Buddha* and *Egyptian statue* have been milled with a Stepcraft-2/840 Desktop CNC System.

Moai is assembled from 12 blocks and is 27 centimeters tall; *MaxPlanck* from 8 blocks and is 22 centimeters tall; *Buddha* from 8 blocks and is 19 centimeters tall (we skipped the curved base since the model does not stay straight with it); the *Egyptian Statue* from 11 blocks and is 32 centimeters tall. They have been assembled using wood glue. After assembly, the seams on each model have been covered with wood putty and sanded with fine grain sandpaper. This procedures hides the seams, which are only visible in the discontinuities of the wood pattern. We tested an alternative procedure on *Moai* statuette: we covered it with water-based enamel and then polished it. The final appearance is shown in the inset.

4.4.2 Internal Framework.

Restricting our milling directions to the six major axis does not only simplify the assembly of small models, where the large overlaps can be easily covered with glue or kept together using metal L-shaped connectors, but it also drastically simplifies the construction of large architectural-scale objects. In this scenario, the interior of the shape can be realized with a supporting framework made of metal beams, which follows the internal edges, and is kept together with a single type of joints: since the height blocks are axis aligned, the only possible intersections of the beams are multiples of 90 degrees. The required



joints are simple to fabricate and reusable. A virtual example for the *Egyptian Statue* in the inset.

4.4.3 Height Control.

By controlling the maximal height of the produced blocks users can control the amount of material necessary to fabricate the object (Figure 4.4), trading material savings for increased block count.

4.4.4 High-Frequency Models.

While our method can directly generate brute-force decompositions for high-frequency inputs, such decomposition would inevitably lead to high block counts, due to the very restrictive fabrication constraints inherent in 3-axis CNC milling. Our optional high-frequency filtering algorithm (Section 4.3.5) leads to much simpler decompositions, while losing minimal surface details, as shown in Figure 4.7.

4.4.5 Comparison with [Hu et al., 2014].

As noted earlier, in contrast to our framework the method of [Hu et al., 2014] has no direct control on how far its outputs deviate from the height-field or pyramidality constraints, and demonstrate results where these constraints are far from satisfied. To compare the two approaches, we repeat the experiment proposed in Figure 23 of [Hu et al., 2014] and show the results in Figure 4.11. Our method introduces less blocks to decompose the model, and, more importantly, our blocks are millable, while the outputs of Hu et al. do not satisfy the height-field property (highlighted with red ovals in Figure 4.11) and thus can only be fabricated with a 3D printer.

4.4.6 3D Printing.

Our pipeline is designed to produce height-field blocks, but, with a minor modification, becomes a powerful tool to produce decompositions tailored for FDM 3D printers. These printers require support material to sustain overhanging parts with angles larger than 35-40 degrees. The support does not only increases printing time and wastes material, but also lowers surface quality [Zhang et al., 2015]. Our method can be used to decompose an object into blocks whose top surface contains overhang that are strictly smaller than a printer-specific threshold by simply relaxing the height field condition in Section 4.3.2, assigning an infinite value to the scalar field only when triangles have a larger overhang. We show an example of a decomposition with a maximal overhang of 30 degrees in Figure 4.12.

Model	Timing				# Blocks	Height
	MHFBC	MC	OR	B		
Airplane	1′41″	1″	1″	9″	11	∞
Batman	33′44″	20″	1″	54″	8	0.15
Bimba	25′23″	3″	0.2″	39″	16	∞
BU (orientation)	17′40″	9″	2″	27′24″	16	0
BU (no orientation)	71′3″	9″	1″	23″	9	0
Buddha	61′37″	15″	1″	33″	8 (7 milled)	0.125
Chinese Lion	7′38″	3″	2″	59″	27	0.125
	2′35″	14″	0.1″	6″	2	∞
Cube Spike	1′19″	4″	0.1″	9″	6	0.4
	47′21″†	6′48″	0.1″	13″	6	0.4
David	26′27″	9″	0.2″	18″	7	0.075
Dea	31′33″	20″	0.2″	21″	7	0.075
Egea	15′56″	6″	0.1″	7″	6	0.1
Egyptian Statue	16′7″	6″	5″	21″	11	0.05
Eros	39′33″	10″	1″	1′1″	10	0.125
Fertility	15′32″	11″	32″	10′20″	63	∞
Gentildonna	36′23″	9″	0.1″	20″	10	0.125
Kitten	21′25″	6″	2″	59″	25	0.05
Lincoln	8′23″	6″	0.6″	1′41″	14	0.125
Lion Vase	29′24″	8″	0.4″	14″	10	0.175
Max Plank	15′59″	10″	0.3″	14″	8	∞
Moai	12′47″	4″	0.2″	9″	12	0.225
Pensatore	11′4″	9″	0.1″	29″	7	0.1

† No pruning

Table 4.1: Model statistics: computation time (split into Maximal Height-Field Block Computation (MHFBC), Minimal Covering (MC), Overlap Resolution (OR), CSG Operations (B)); number of blocks; and the block height maximum used (as percentage of model diagonal, ∞ means no height limit).

4.4.7 Implementation Details.

We implemented our algorithm in C++, using *Eigen* [Guennebaud et al., 2010] for linear algebra routines, *Gurobi* [Gurobi,] for branch and bound, and *libigl* for mesh booleans [Zhou et al., 2016, Jacobson et al.,]. We run all our experiments on a workstation with a 4-cores Intel i7-4790K processor clocked at 4.0 Ghz and 16 Gb of memory. Our method takes under one hour on even the most complex model (*Buddha*) with the runtime dominated by the initial maximal size block computation. A summary of the timings and number of height-field blocks for all our experiments is shown in Table 4.1.

4.5 Limitations and Concluding Remarks

We presented an automatic and robust pipeline to decompose a triangle mesh into a collection of non-overlapping, valid height-field blocks, which can be directly manufactured using a 3-axis CNC milling machine. Our method allows for producing real-world replicas of complex 3D geometries from materials such as stone, wood, or styrofoam, none of which can be processed by additive manufacturing methods. Our method is compatible with existing milling systems and mixed manufacturing techniques, and can be used to produce high quality real-life replicas of complex virtual geometries in a range of sizes.

Our pipeline is automatic and robust: the only hard requirement on the input is that it should be a closed surface. The number of blocks we produce is dependent on the input model complexity, and can increase dramatically for models with narrow prominent features or high genus, resulting in objects that may be hard to assemble. Manual decomposition and independent processing of the different parts can help reduce block count. Our pipeline is also heavily affected by the orientation of the model, which might lead to decompositions with more pieces than necessary. While our canonical orientation algorithm strive to ameliorate this problem, it is not guaranteed to succeed. An example is shown in Figure 4.14, where a manual orientation of the model leads to a simpler decomposition. A second limitation is that while we optimize to avoid blocks with small features, we cannot guarantee to find the optimal decomposition with the smallest number of such blocks. This is an interesting theoretical avenue for future work, since in our experiments, our greedy algorithm is sufficient to avoid the most problematic cases that lead to fabrication failures.

We expect our algorithm to have a large impact in digital fabrication labs, since it provides an automatic and effective way to exploit 3-axis milling machines to fabricate generic shapes. We will release a complete open-source implementation of our algorithm with an open source license to encourage its adoption.

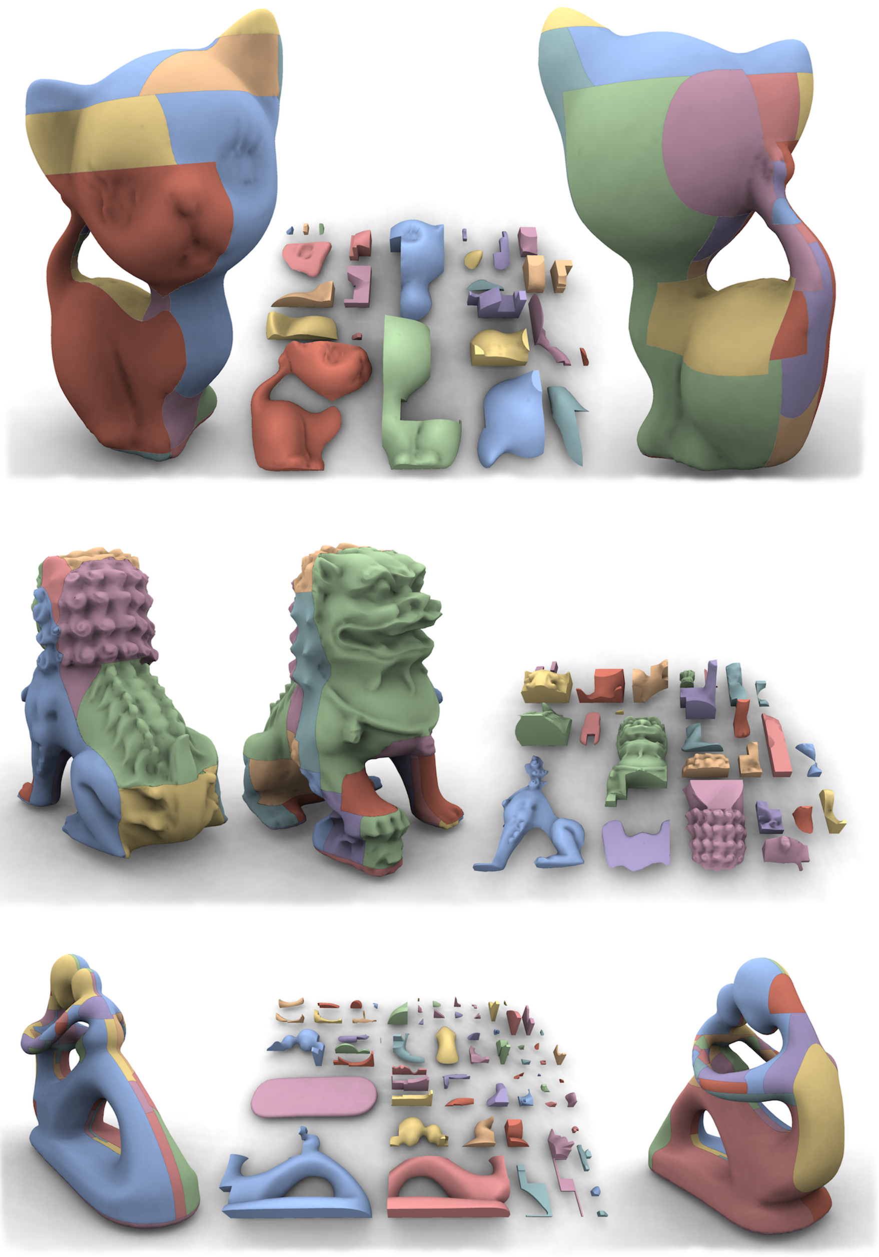


Figure 4.9: Our method produces valid, compact decompositions for complex models containing: non-trivial topology (e.g., Fertility), thin features (e.g., the ears of the Kitten) and large portions not aligned with the global frame (e.g., the Chinese Lion's body) for which manual decomposition is highly challenging to compute, resulting in 27 blocks for the Chinese Lion, 25 for Kitten and 63 for Fertility.

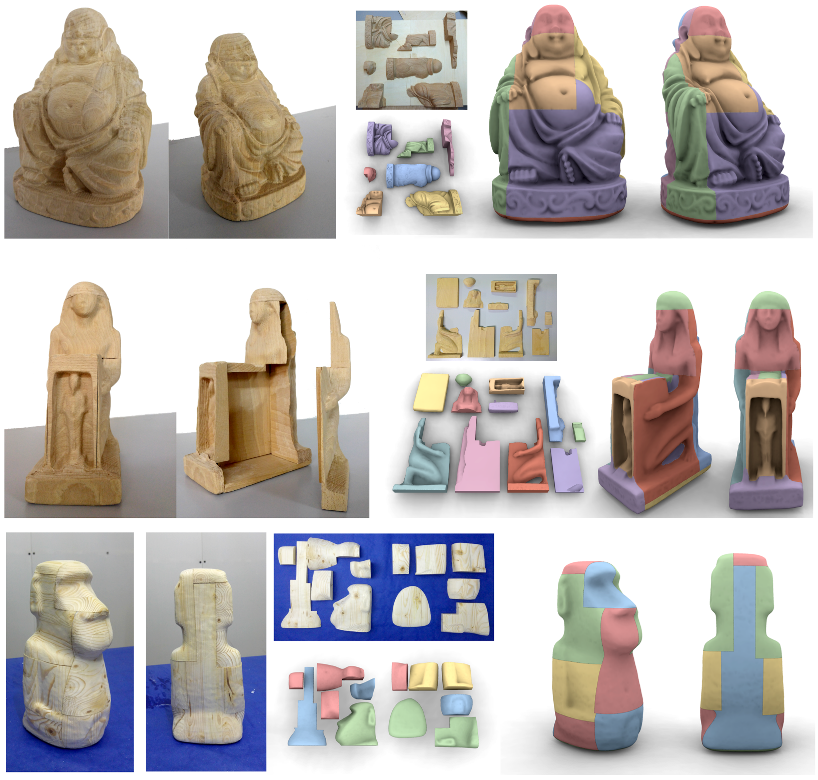


Figure 4.10: A gallery of decompositions computed with our algorithm and fabricated in wood using a 3-axis milling machine.

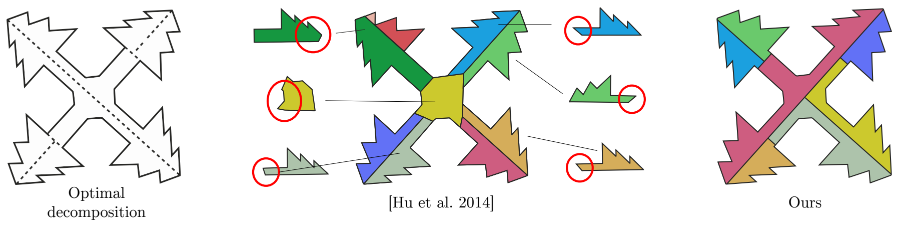


Figure 4.11: Comparison with [Hu et al., 2014]: pyramidal decomposition (middle) produces ten blocks, six of which violate the height-field condition (see red ovals). Our method (right) decomposes the shape into seven valid height-field blocks, one more than the optimal decomposition (left). Notice that we re-oriented the model before running our method, according to the strategy described in Section 4.3. For the sake of better visual comparison all the models are shown in the same position.

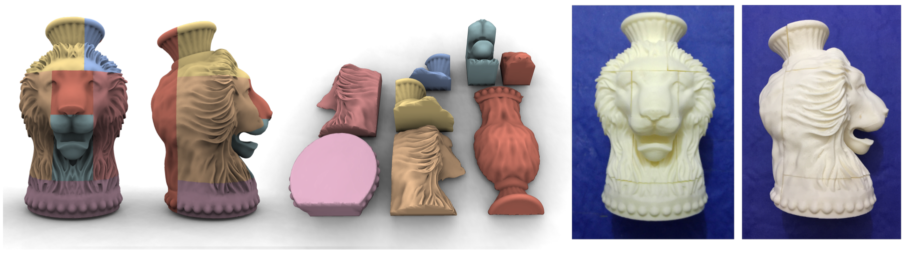


Figure 4.12: By relaxing the height field constraint we can decompose an object into blocks whose top surface contains overhangs that are strictly smaller than a printer-specific threshold. Here we show an example of a decomposition with a maximal overhang of 30 degrees for the Lion Vase dataset (left, middle). Notice from the side view that the lion mouth contains a considerable amount of overhangs. Since the threshold we set is compatible with that of common FDM printers, we safely printed each block without using support structures, saving time and material, and achieving better surface quality (right).

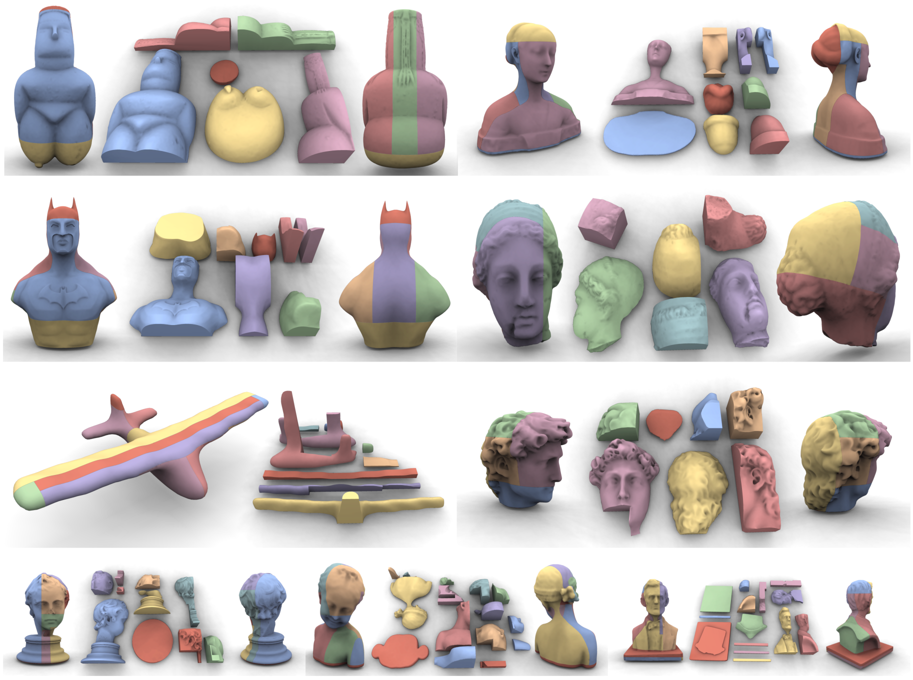


Figure 4.13: Additional results generated using our method.

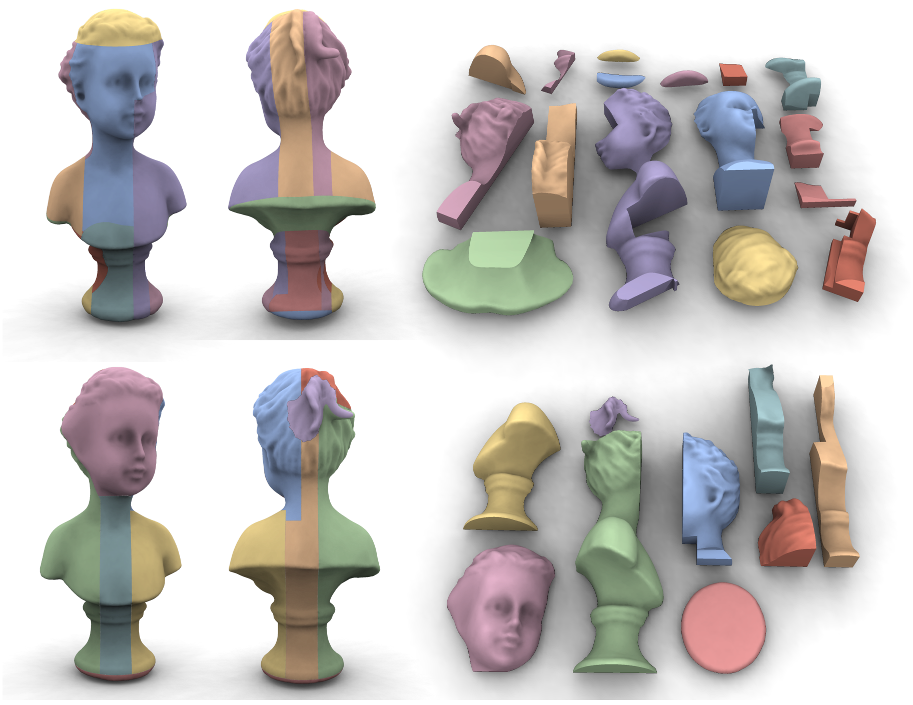


Figure 4.14: Our canonical orientation algorithm is not guaranteed to produce decompositions with minimal number of height blocks. A failure example is illustrated here: with automatic orientation the BU statue is decomposed in 16 blocks (top); with manual orientation the number of blocks goes down to 9 (bottom). Finding the orientation that minimizes the number of blocks in the decomposition is a challenging problem that we plan to tackle in future work.

Appendix A

Simplification Masks

We describe here the sets of masks used for the automatic simplification algorithm described in section 3.3.1. We used 18 different initial masks. Considering all their rotations around some well-known axis, the total number of masks is 340. There are two main types of masks:

- Masks with more than one set of Points of Interest;
- Masks with only one set of Points of Interest.

The masks of the second type are very simple: if we find a set of adjacent cubes corresponding to the signs of the mask, we just switch the signs of its Points of Interests. The first type of masks are harder to be examined, because they have more than one set of Points of Interest. We need to choose one of these sets using area of the involved charts. We list all the masks in the next sections.

A.1 Masks with more than one set of Points of Interest

We designed 10 different types of masks with more than one set of Points of Interest. They are listed from figure A.1 to figure A.7.

A.2 Masks with only one set of Points of Interest

We designed 10 different types of masks with more than one set of Points of Interest. They are listed from figure A.8 to figure A.11.

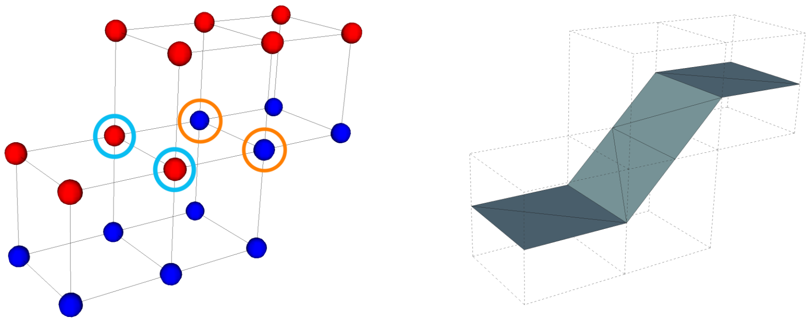


Figure A.1: This mask represents a step between two facets with normal of cat. 1 and a facet with normal of cat. 2. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. This mask can be rotated, generating a total number of 24 masks.

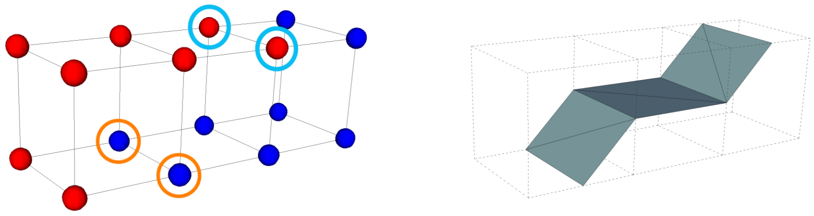


Figure A.2: This mask represents a step between two facets with normal of cat. 2 and a facet with normal of cat. 1. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. This mask can be rotated, generating a total number of 24 masks.

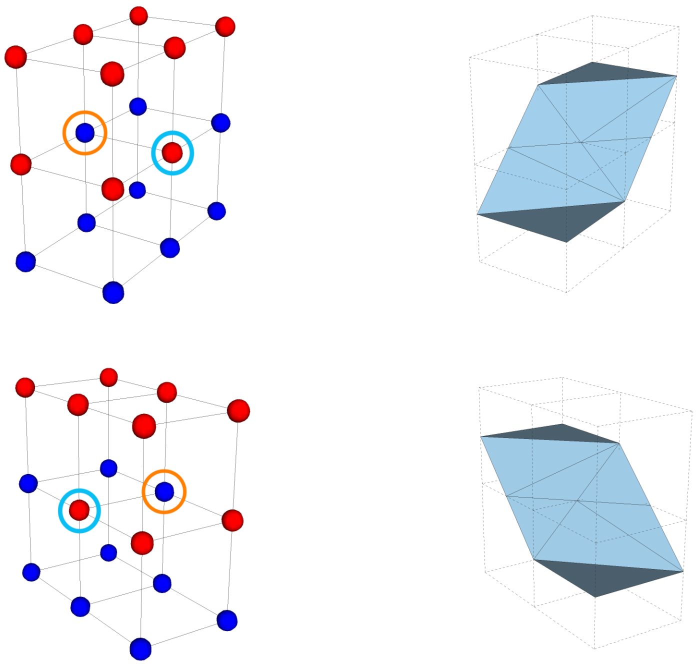


Figure A.3: The masks shown in the figure are the reflection of each other, and they represent a step between two facets with normal of cat. 1 and a facet with normal of cat. 3. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. These masks can be rotated, generating a total number of 48 masks.

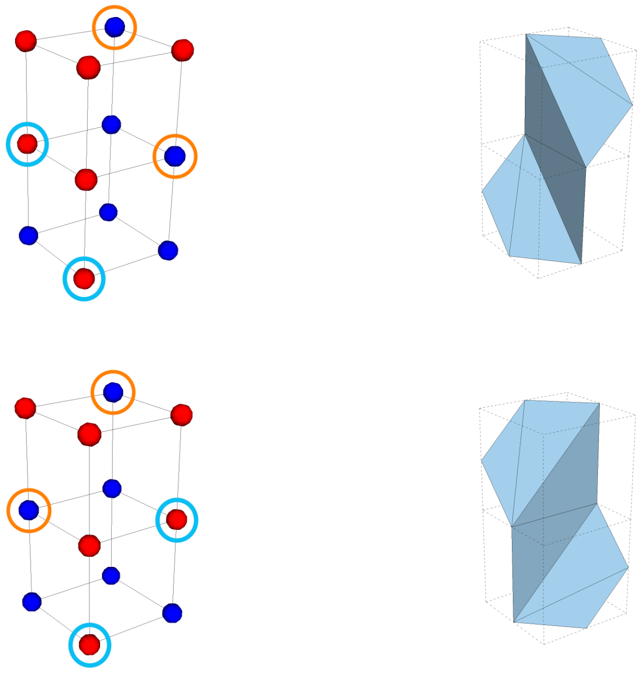


Figure A.4: The masks shown in the figure are the reflection of each other, and they represent a step between two facets with normal of cat. 3 and a facet with normal of cat. 1. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. These masks can be rotated, generating a total number of 48 masks.

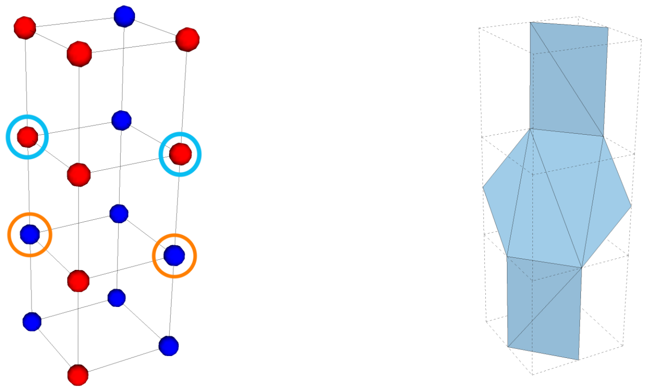


Figure A.5: This mask represents a step between two facets with normal of cat. 2 and a facet with normal of cat. 3. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. This mask can be rotated, generating a total number of 24 masks.

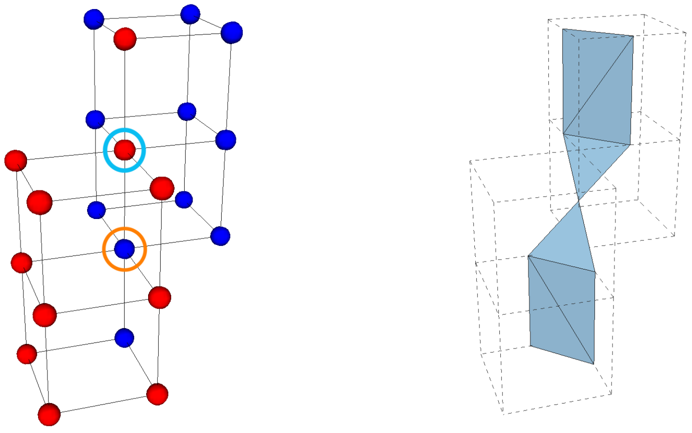


Figure A.6: This mask represents a step between two facets with normal of cat. 2 and a facet with normal of cat. 3. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. This mask can be rotated, generating a total number of 24 masks.

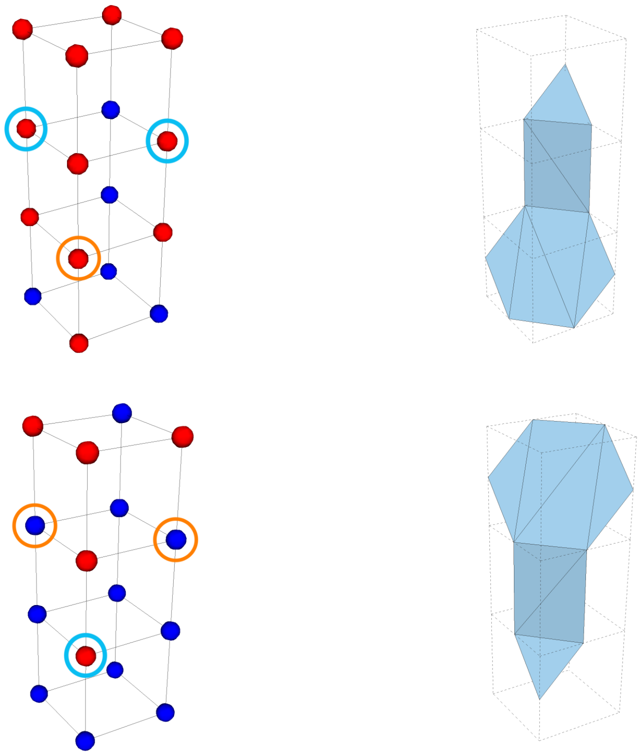


Figure A.7: The masks shown in the figure are the reflection of each other, and they represent a step between two facets with normal of cat. 3 and a facet with normal of cat. 2. The two sets of Points of Interest are circled in cyan and orange respectively, and one of the two sets is selected according to the area of the two facets. These masks can be rotated, generating a total number of 48 masks.

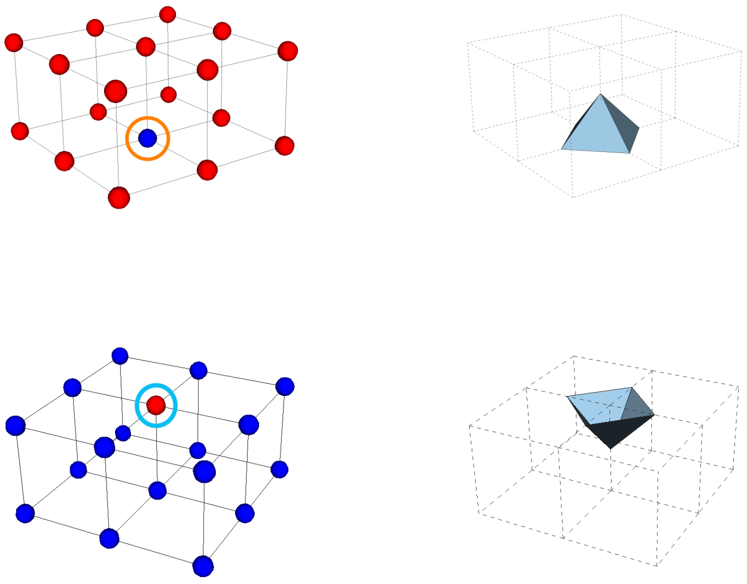


Figure A.8: The masks shown in the figure are the reflection of each other, and they represent a "pyramid" on a local surface with facets with normals which don't belong to cat. 3. The single set of Points of Interest is circled and the points are switched if the configuration of points is found on the regular lattice. These masks can be rotated, generating a total number of 12 masks.

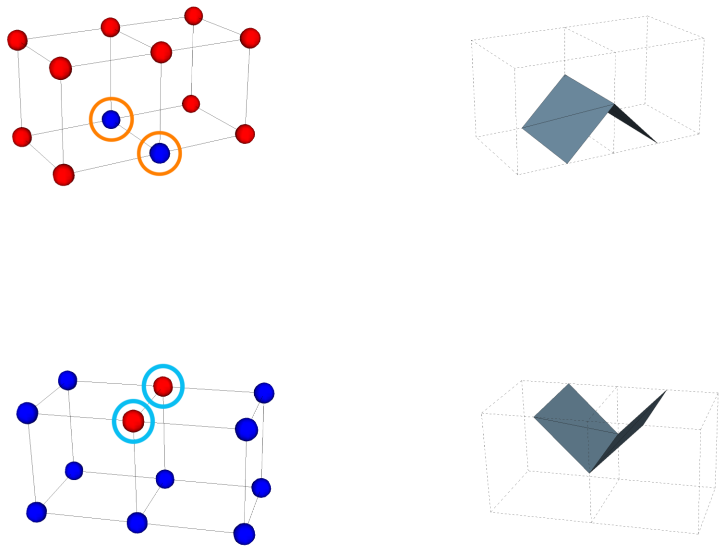


Figure A.9: The masks shown in the figure are the reflection of each other, and they represent a ”pyramid” on a local surface with facets with normals which don’t belong to cat. 2. The single set of Points of Interest is circled and the points are switched if the configuration of points is found on the regular lattice. These masks can be rotated, generating a total number of 24 masks.

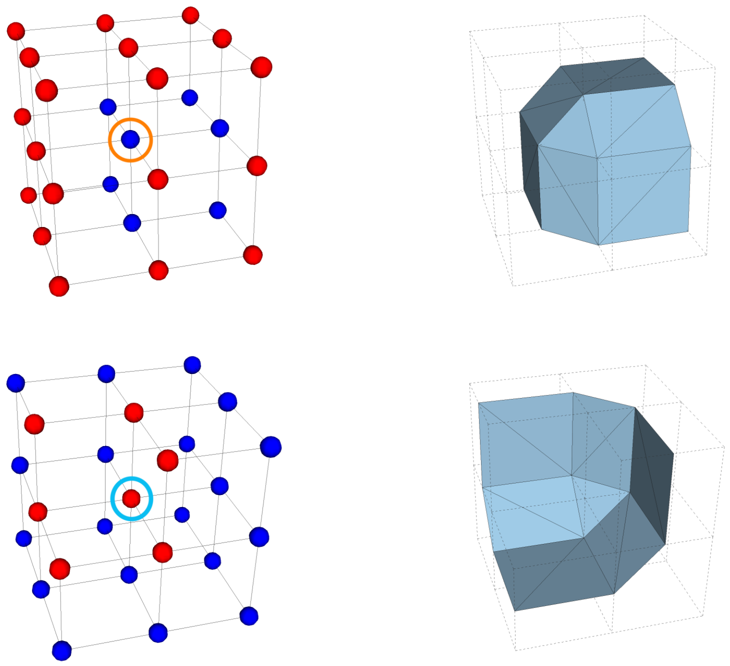


Figure A.10: The masks shown in the figure are the reflection of each other, and they represent a local configuration where the surface has facets with normals which belong to cat. 3. The single set of Points of Interest is circled and the points are switched if the configuration of points is found on the regular lattice. These masks can be rotated, generating a total number of 16 masks.

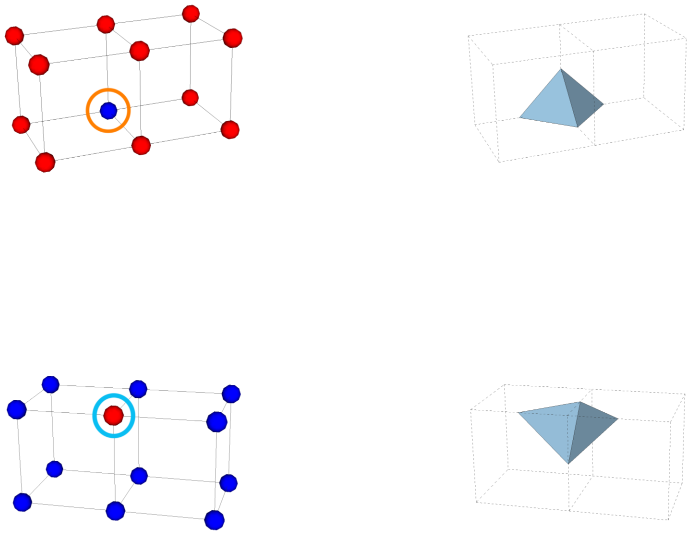


Figure A.11: The masks shown in the figure are the reflection of each other, and they represent a "pyramid" on a local surface with facets with normals which don't belong to cat. 3. The single set of Points of Interest is circled and the points are switched if the configuration of points is found on the regular lattice. These masks can be rotated, generating a total number of 48 masks.

Appendix B

Box-Integration of a Tricubic Scalar Field.

Let B be a an axis-aligned box defined by its two extreme points $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ and L a regular lattice. We define S as the set of all the cubes $(s_{u,v,w})$ in the lattice L partially or completely contained in B:

$$S : \{s_{u,v,w} \in L | s_{u,v,w} \cap B \neq \emptyset\}$$

and we define D as the set of triplets of indexes in S:

$$D : \{(u, v, w) | s_{u,v,w} \in S\}$$

Each cube $s_{u,v,w}$ in L is defined by its two extreme points (x_m, y_n, z_p) and $(x_{m+1}, y_{n+1}, z_{p+1})$ and to each cube we associate a set of coefficients $a_{i,j,k}^{(u,v,w)}$ for performing the tricubic interpolation inside $s_{u,v,w}$.

The interpolated value f in a generic (x, y, z) point inside $s_{u,v,w}$ is:

$$f(x, y, z) = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} \left(a_{i,j,k}^{(u,v,w)} x^i y^j z^k \right)$$

The energy to minimize is given by the integral over B, hence the sum of all the integrals over the cubes inside B:

$$\min \sum_{u,v,w \in D} \int_{x_{\min}, y_{\min}, z_{\min}}^{x_{\max}, y_{\max}, z_{\max}} f(x, y, z)$$

where:

$$x_{\min} = \begin{cases} x_m & \text{if } x_m > p_x \\ p_x & \text{otherwise} \end{cases}$$

$$x_{\max} = \begin{cases} x_{m+1} & \text{if } x_{m+1} < q_x \\ q_x & \text{otherwise} \end{cases}$$

and similarly for y and z.

The parameters are the coordinates of the points defining B. To be sure that the starting box will always cover the first primitive, we sum to the energy a barrier function tending to $+\infty$ when one of the coordinates of B is too near to one of the coordinates of the *points to cover* (i.e, the endpoints of a segment), and is 0 when the coordinates are enough far from these points as in [Schüller et al., 2013].

Suppose to have a set C of points $c = (c_x, c_y, c_z)$ to be covered by our box B. For each $c \in C$ we will have a function for p and for q:

$$\Phi_{c,t}(p_x) = \begin{cases} +\infty & \text{if } p_x \geq c_x \\ \frac{1}{g(p_x)} & \text{if } c_x - t < p_x < c_x \\ 0 & \text{if } p_x \leq c_x - t \end{cases}$$

$$\Phi_{c,t}(q_x) = \begin{cases} 0 & \text{if } q_x \geq c_x + t \\ \frac{1}{g(q_x)} & \text{if } c_x < q_x < c_x + t \\ +\infty & \text{if } q_x \leq c_x \end{cases}$$

and similarly for y and z, where t is $\frac{1}{10}$ of the lattice's edge, and g is:

$$g(x) = \frac{1}{t^3}x^3 - \frac{3}{t^2}x^2 + \frac{3}{t}x$$

Defining

$$\Phi_{c,t}(p) = \Phi_{c,t}(p_x) + \Phi_{c,t}(p_y) + \Phi_{c,t}(p_z),$$
$$\Phi_{c,t}(q) = \Phi_{c,t}(q_x) + \Phi_{c,t}(q_y) + \Phi_{c,t}(q_z)$$

we can add the barriers to the energy function to minimize:

$$\min \sum_{u,v,w \in D} \int_{x_{\min}, y_{\min}, z_{\min}}^{x_{\max}, y_{\max}, z_{\max}} f(x,y,z)s \qquad + \qquad \sum_{c \in C} \left(\Phi_{c,t}(p) + \Phi_{c,t}(q)\right).$$

Appendix C

Valid Height-Block Decomposition via AA Box Splitting.

Let B_1 and B_2 be a pair of intersecting axis aligned height boxes associate to the milling directions m_1 and m_2, which are in the set $(\pm X, \pm Y, \pm Z)$. Since B_1 and B_2 are axis aligned, their intersection $BI = B_1 \cap B_2$ is an axis aligned box. The planes on which the six facets of BI lie, partition both B_1 and B_2 into eight sub-boxes each, namely $B_{1.1}, \dots, B_{1.8}$ and $B_{2.1}, \dots, B_{2.8}$. Notice that, since B_1 and B_2 intersect, there always exist two indices $i, j \in [1, 8]$ such that $B_{1.i} \equiv B_{2.j} \equiv BI$.

Let us now consider which milling directions, between m_1 and m_2, could be chosen for these sub-boxes. We can observe that: (i) B_1, B_2 and all the sub-boxes are axis aligned, therefore the angle between the box facets and the milling direction is either $0°$ or $90°$; (ii) each sub-box of B_1 $(B_{1.1}, \dots, B_{1.8})$ has m_1 as candidate milling direction, and each sub-box of B_2 $(B_{2.1}, \dots, B_{2.8})$ has m_2 as candidate milling direction. The only exception is BI, which has both m_1 and m_2 as candidate milling directions.

From (i) and (ii) descends that any sub-box has *at least* one valid milling direction with an axis aligned facet as supporting base. In other words a valid height block decomposition obtained by splitting the original boxes using axis aligned planes always exists. $\square$

Notice that this is true only for the special case of axis aligned boxes and milling directions; in any other case condition (i) would not be satisfied, as the angle between the box facets and the milling direction may be greater than $90°$, thus violating the height field condition.

Appendix D

Skeleton-driven Adaptive Hexahedral Meshing of Tubular Shapes

D.1 Introduction

The quest for volumetric meshes for physically based simulations has dramatically increased in recent years. While classical applications deal with mechanical objects (e.g., in the aeronautical, mechanical and medical industries) new applications have emerged (e.g., in the movie and gaming industry, as well as in the biomedical field) which deal with natural shapes, often coming in the form of articulated objects. Special effects involving fluids and deformable objects are ubiquitous in many of the most recent movie productions and videogames. Likewise, the accurate simulation of human tissues and organs widely extends the diagnostic power of CT data as well as of virtual surgery. No matter whether models are taken from reality or imagination, they are requested to behave and interact with the virtual world as if they were real.

Physically-plausible simulations [Nealen et al., 2006] require a volumetric discretization of all elements of a scene interacting with each other. This need raises the bar for meshing algorithms, often shaped around precise target shapes [Blacker, 2001] and now required to provide new tools to automatically generate industry-ready meshes from shapes that are definitely different: no more mechanical parts exposing sharp edges and regular patterns, but shapes mimicking living beings.

To this aim, although high quality tetrahedral meshes can be reliably generated with existing methods [Alliez et al., 2005], hexahedral meshes are usually preferred due to their superior numerical properties and ability to keep the resolution lower [Kremer et al., 2014, Blacker, 2001]. Structured hexahedral meshes are often preferred to semi-structured or unstructured ones because they can be more efficiently stored using specialized data structures, yet because they are the models of choice for many simulations where a strict alignment of elements is required [Gao et al., 2015b, Ruiz-Girones, 2011]. A hexahedral mesh is *structured* when it is composed of a single regular volume, or it can be decomposed into a few sub-volumes, each one with the connectivity of a regular grid [Tautges, 2004].

On the one hand, manually creating high-quality structured hexahedral meshes is a laborious task that can take days of work. On the other hand, automatic meshing is a challenging open problem, with much work happening in recent years.

In this work we restrict our attention to articulated shapes whose general structure is well captured by a curve-skeleton [Tagliasacchi et al., 2016, Cornea et al., 2007]. Shapes in this category include, but are not limited to: humans and animals; articulated (possibly imaginary) creatures; tree-like structures like vessels; and plants. This restrictive choice to shapes for which a curve-skeleton can be extracted is based on the fact that physically-based simulations on such shapes are common not only in medicine and biology but also in the animation industry, where most characters created by digital artists belong to this category.

We propose an automatic algorithm that, taking in input a surface mesh and its curve-skeleton, produces a structured hexahedral mesh covering the volume bounded by the input surface. Our method is fast, one-click, easy to reproduce and it does not require any parameter tuning by the user. The hexahedral meshes we produce have high quality and nicely align with the main features of the target surface, a key component for accurate simulations [Blacker, 2001].

The main contribution of our paper is twofold:

- We extend the work of [Usai et al., 2015] from surfaces to volumes, automatically generating hexahedral meshes that directly encode the structure of the input shape, given by its curve-skeleton (Section D.3);

- We propose a sampling technique for the curve-skeleton to control density *along* the skeleton arcs, and a set of volumetric subdivision schemes to control density *across* the skeleton arcs. Our density control system nicely adapts to the local thickness of the shape, minimizing the resolution of the model and reducing the variance of the element sizes (Section D.5).

D.2 Related work

The generation of high quality hex meshes filling a target surface has been object of research since decades. An exhaustive survey of the literature in the field is beyond the scope of this paper. Here, we just recap only the most relevant approaches, grouped by meshing technique. We refer the reader to [Blacker, 2001, Tautges, 2001] for further details on classical approaches.

Skeleton-based approaches are the most related to our work. When the meshing process is driven by curve-skeleton, the critical part is the discretization of junctions, where incoming arcs from different directions meet. In [Lin et al., 2012] a split-and-merge meshing method is presented. Each part of the skeleton is meshed separately, then all the components are grafted together. The method is validated on a set of simple models, where the most complex junction has valence four. For complex shapes like the octopus in Figure D.10, where eight limbs meet the core of the shape at the same point, it is not clear whether this method would produce a valid result. In the best case it would fail to produce the right topological structure, converting the corresponding valence nine junction into a set bifurcations. Our method produces a hex mesh that encodes the correct structure of the octopus without introducing high valence vertices, thus providing the right balance between structure and mesh complexity. Overall, we generate hexahedral meshes with higher quality (see Section D.7).

In [Zhang et al., 2007] a sweeping method to mesh tubular shapes is proposed. Their method focuses on blood vessels and uses a set of templated solutions to mesh junctions, with the curve-skeleton used as a proxy to fit the best template according to the morphology of the vessel. Hexahedra are radially arranged around the skeleton, thus generating badly shaped elements near the spine. Moreover, this method works best for bifurcations (which are typical on blood vessels), while it tends to generate high valence vertices and badly shaped hexahedra when junctions with higher valence are present. Our method avoids high valence irregular vertices thus favoring better per-element quality [Livesu et al., 2015].

In [Liu et al., 2015] a skeleton-based method for T-spline fitting is proposed, which can also be used for hexahedral meshing. Half-planes are employed to mesh bi-furcations and tri-furcations. This method suffers the same drawbacks as [Zhang et al., 2007]: it tends to produce overly complex meshes with high valence vertices if the skeleton contains high-valence junctions, setting a tight upper bound to the quality of the elements directly incident at them.

In [Yao et al., 2009] Yao and colleagues propose to drive the meshing process with a manually sketched curve-skeleton. Junctions are handled with a neat subdivision process. Similarly to [Usai et al., 2015], this work focuses on the generation of base domains for quadrilateral meshing and, therefore, it can be considered an alternative starting point for our method.

Grid-based methods subdivide the volume using either a regular grid or an octree and, subsequently, they move the vertices of the hexahedra intersecting the surface onto the surface itself so as to better approximate the original shape [Lin et al., 2015,Maréchal, 2009,Schneiders, 1996]. Due to their simplicity and ability to mesh any object, grid-based methods are still dominant in industry. However, these methods suffer from several drawbacks: they tend to produce unnecessarily high resolution meshes, they are not invariant to rotation (i.e., the same volume can be meshed differently, when rotated) and they tend to push the elements of worst quality near the boundary. Our method is rotationally

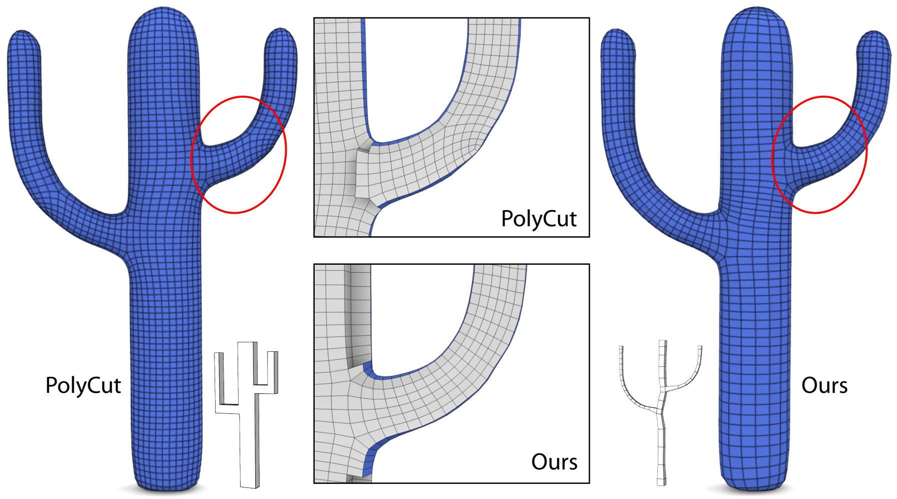


Figure D.1: Polycube mappings may force the introduction of unnecessary singularities, penalizing the alignment with the boundary of the shape (top closeup); our meshing better aligns to the limbs of the CACTUS (bottom closeup).

invariant and generates *boundary conforming* hexmeshes (Figure D.1) with much less elements, also promoting high quality hexahedra near the boundary, an important requirement to ensure accurate simulations [Ruiz-Gironés et al., 2015].

Advancing front techniques start the mesh generation process on the surface and then move inwards [Tautges et al., 1996]. This approach tends to place singularities and lower quality hexahedra inside the volume. A recent example of expanding front method and a review of similar methods is provided in [Kremer et al., 2014]. These methods generate high quality meshes near the boundary regions, which is a desired property for many applications. Unfortunately, they are prone to generate low quality meshes in the interior (where the fronts merge) and cannot be applied to all classes of shapes (only genus zero shapes are supported). Our method can handle complex topologies, like FERTILITY in Figure D.10 and the BLOCK model in Figure D.13.

Parameterization based methods map the input volume to some parametric space, where the connectivity of the mesh is generated. The inverse mapping is then used to place the corresponding vertices in the original domain. PolyCubes [Tarini et al., 2004] (i.e., orthogonal polyhedra) have been widely used as parametric domains because they can be trivially hex-meshed with a regular grid, generating a structured mesh [Fang et al., 2016, Cherchi et al., 2016, Huang et al., 2014, Livesu et al., 2013, Gregson et al., 2011]. However, the structure of the mapping domain often causes the introduction of unnecessary singularities

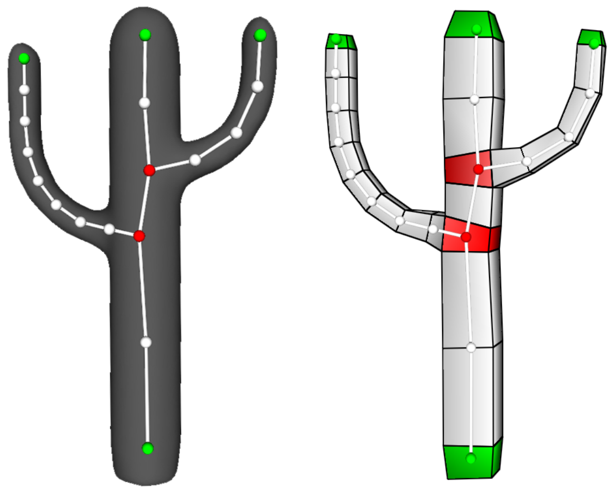


Figure D.2: We derive from the curve-skeleton of a triangle mesh (left) a volumetric decomposition in tubes (white), branching cubes (red) and terminal cubes (green). Each element in the tubular structure is a hexahedron.

that penalize the alignment with the boundary (Figure D.1). Other popular parameterization-based techniques associate to each point in the interior of the shape a 3D frame such that the resulting frame field is aligned to the surface of the shape. The mesh connectivity is then generated by sampling the field, with singularities occurring at its vanishing points [Sokolov et al., 2016, Kowalski et al., 2014, Huang et al., 2011, Nieser et al., 2011]. These algorithms generate meshes that nicely adapt to the input surface, however, they do not provide any control on the structure of the mesh and tend to introduce misaligned singular vertices, resulting in a complex singular structure [Li et al., 2012]. Our method avoids the singularity misalignment problem, resulting in coarse singularity layouts that embed the high level structure of the input shapes, a key factor to ensure high quality hexahedral meshes [Gao et al., 2015b]. Furthermore, there are no theoretical guarantees that a volume parameterization from a frame field admits an all-hex structure. Current methods may fail to produce a mesh depending on the type of singularities in the input field [Li et al., 2012].

D.3 Pipeline overview

We propose a method to generate a full hexahedral mesh out of a tubular shape. We input a triangle mesh $\mathcal{M}$ and its curve-skeleton $\mathcal{S}$, that we use as a proxy to infer the structural properties of $\mathcal{M}$ and drive the meshing process. The result is a full hexahedral mesh $\mathcal{H}$ that embeds in its connectivity the structure of $\mathcal{S}$ and has $\mathcal{M}$ as outer boundary. We optionally apply templated schemes to

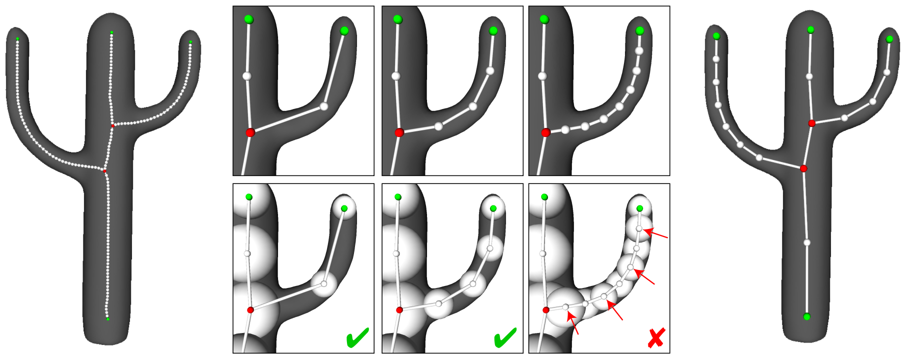


Figure D.3: We start with a dense sampling of the curve-skeleton (left) and we re-sample it by iteratively splitting half-way each of its portions. The splitting process ends when the maximal sphere centered at the new sample intersects both the spheres centered at the two end-points of the current segment (middle). The resulting coarse sampling of the skeleton (right) determines the connectivity of the final hexahedral mesh.

control the meshing density and adapt it to the morphology of the shape, thus keeping element size variance as low as possible.

Our work builds upon the coarse quad layout generation algorithm described in [Usai et al., 2015], which we extend adding a volumetric interpretation of the tubular structure described in their paper. The method presented in [Usai et al., 2015] generates a quadrilateral mesh of tubular shapes. This mesh is obtained from a decomposition into *tubes* - cylinders with quadrilateral section that surround the skeleton curves - and *cubes* centered at both the branching and terminal nodes of the curve-skeleton (Figure D.2). The quads composing each tube and each cube are derived and assembled so that the resulting mesh is *conforming* (i.e., free from T-junctions); models with loops and complex topologies are supported.

This structure lends itself to straightforward hexahedral meshing, by gridding each cube and each tube with a number of subdivisions that keep the conformity in the hex-mesh as well. Note that the resulting hexahedral mesh is *structured* by construction, because its connectivity embeds the decomposition in tubes and cubes encoded in the tubular structure, yet because each tube and each cube is meshed as a regular grid [Tautges, 2004].

We extend the pipeline presented in [Usai et al., 2015] and adapt it to the volumetric case. Our hex-meshing strategy consists of the following steps:

1. **Skeleton resampling:** we propose a fully automatic strategy to sample the skeleton $\mathcal{S}$, in order to avoid excessively dense meshes and badly shaped elements (Section D.4);

2. **Tubular structure:** we initialize mesh $\mathcal{H}$ as the tubular structure enclos-

ing $\mathcal{S}$, placing a hexahedron at each skeleton branching node, extruding its facets along the skeleton curves and subdividing each element in order to avoid T-junctions (see [Usai et al., 2015] for details);

3. **Resolution control:** we identify changes in the local thickness of the skeleton and use a set of templates to locally adapt the meshing density to the morphology of the shape (Section D.5.1);
4. **Projection:** we project the boundary of $\mathcal{H}$ onto the input shape $\mathcal{M}$, using ray casting to generate an initial poor quality map, and then we refine such map using the *abstract domains* approach described in [Usai et al., 2015, Tarini et al., 2011];
5. **Finalization:** we optimize the interior of $\mathcal{H}$, carefully positioning its vertices so as to maximize per-element quality (Section D.6).

In the remainder of the paper we discuss technical details regarding the sampling of $\mathcal{S}$ (1), the resolution control (3) and the hexmesh finalization (5). The initialization of the tubular structure (2) and the projection of its boundary over $\mathcal{M}$ (4) are equivalent to the ones presented in [Usai et al., 2015] and, as such, they will not be discussed here. We point the reader to the original paper for technical details regarding their implementation.

D.4 Skeleton resampling

Although simply gridding the tubular structure would produce a hexahedral mesh (Figure D.2 right), a naive use of this technique may lead to excessively dense meshes with poorly shaped elements (see the left side of Figure D.4). Indeed, the density of the mesh in the longitudinal direction is directly related with the sampling density of the underlying curve-skeleton. Expanding and subdividing the tubular structure to flood the interior of the input shape would easily result in a hex-mesh containing many inverted elements and no practical usefulness. In order to improve the mesh quality and produce well shaped elements we re-sample the curve-skeleton. For each skeleton point we assume to have the radius of its maximal sphere available. Some skeletonization algorithms already provide this information (e.g. [Livesu and Scateni, 2013, Livesu et al., 2012]); otherwise, the radius at a skeleton point can be easily estimated by measuring its distance to the input surface.

We start from a dense sampling of the curve-skeleton and, then, we subsample it applying arc-length parameterization to each curve of the skeleton, mapping its length in the interval $[0, 1]$. We then split the curve in the parametric space, adding a new sample point half-way, and iteratively repeat the process for the resulting sub-intervals. The stop criterion is as follows: we do not split an interval when the maximal sphere centered at a candidate sample would intersect both spheres centered at the endpoints of the current segment (Figure D.3). The resulting sampling adapts to the local thickness of the shape

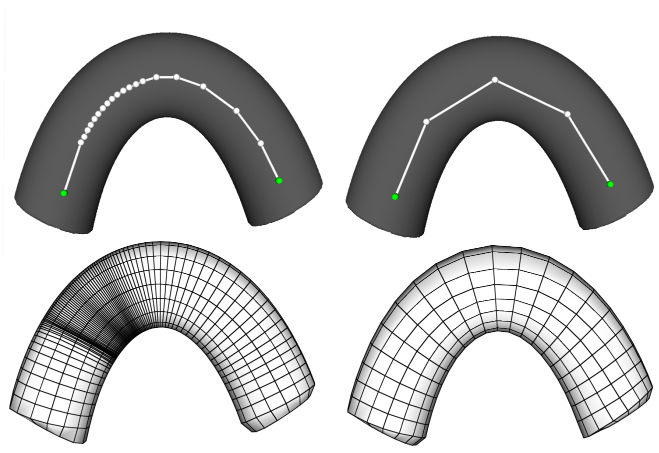
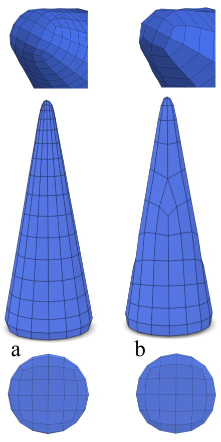


Figure D.4: The sampling of the curve-skeleton determines the density of the hexahedral meshing. Arbitrarily sampling the curve-skeleton may lead to poor meshes with badly shaped elements (left). Our sampling strategy generates good meshes and promotes isotropy (right).

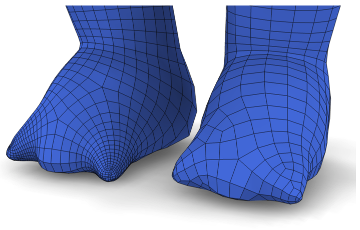
and induces a meshing with a good isotropy (Figures D.4 and D.10). After re-sampling the skeleton we generate a tubular structure fully enclosing the skeleton with the method described in [Usai et al., 2015]. The result is a coarse structured hexahedral mesh $\mathcal{H}$, ready to be further subdivided and inflated to adhere to the surface of the target shape (Figure D.2).

D.5 Resolution control



The approach described so far is capable of producing hex-meshes for any shape in our class of interest (i.e. tubular shapes); very accurate and high quality models can be obtained via the projection and finalization step described later (see Figure D.10). However, depending on the morphology of the model, the elements sizes may be uneven, with the presence of high density areas that unnecessarily increase the resolution of the model. To give an example let us consider the cone-like shapes aside: the area covered by the base of the cone is much larger than the area covered by its tip, hence, if the cone is meshed with a regular grid the density of the elements on the tip will be much higher than the elements at the base (a). In order to avoid this behavior we introduce a mechanism to adjust the resolution of the model and better adapt to the morphology of the shape, so as to keep the element size even and the resolution lower (b).

We detect the elements of the mesh where a change of resolution is needed (we call them *cones*) directly from the tubular structure derived from the curve-skeleton. As explained in Section D.4, the size of each hexahedron in such structure is proportional to the local thickness of the shape, therefore the morphology of the input shape is correctly encoded in this coarse, yet easy to process, hex-mesh.



We split each cone with a template subdivision that puts more sub-elements at the base and less at the tip and we propagate the resulting subdivisions throughout the whole model so as to generate a conforming hex-mesh. The effect of this approach can be seen in the inset above, where the foot of the WARRIOR is shown before (left) and after (right) the application of our cone-based resolution scaling technique. In the following subsections we illustrate how to detect cones (Section D.5.1) and how to propagate the subdivision they introduce (Section D.5.2).

D.5.1 Cone detection

Let us consider the hexahedron h depicted in Figure D.5a. In order to decide whether h is a cone or not we consider the ratio between the size of facets belonging to h_{prev} and h_{next} that are opposite to the facets of h; if this ratio is > 4 we mark h as a cone and we apply the volumetric subdivision schemes depicted in the right part of Figure D.6, each of which is capable of scaling the resolution of the mesh by a factor of 4. Note that these subdivision patterns are just a tiny subset of all the possible ways to scale the resolution; as pointed out in [Takayama et al., 2014] the problem of enumerating an exhaustive list of subdivision schemes is wide open.

In our experiments we observed that using too many cones may result in a very high resolution mesh; we therefore restrict our cone detection strategy only to the terminal branches of the curve-skeleton and we limit the presence

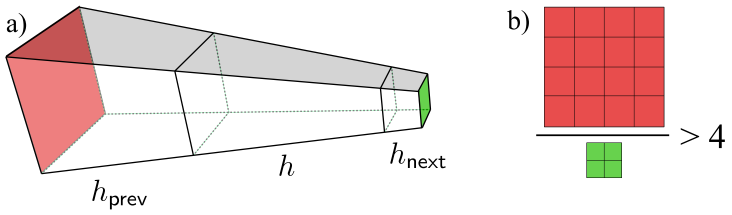


Figure D.5: A hexahedron h (a) is a "cone" if the ratio between the areas of the opposite facets of its two adjacent hexahedra (here in red and green) is larger than 4 (b).

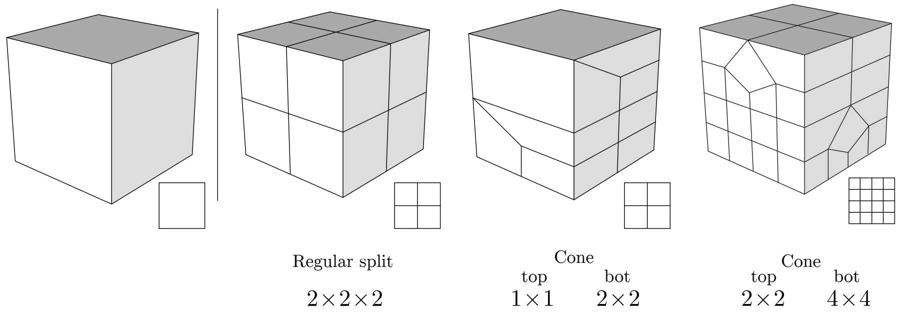


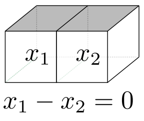
Figure D.6: The volumetric splitting schemes we use in our meshing algorithm. From left to right: a single hexahedron; a regularly split hexahedron; a cone that scales from 1×1 to 2×2; a cone that scales from 2×2 to 4×4.

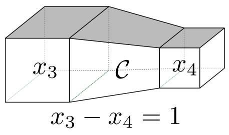
of cones at up to two for each skeleton branch. If more than two candidate cones are found along a skeleton branch we rank them according to the ratio described above and we select the two top ranked elements. Specifically, we enable the presence of one 1×1 to 2×2 and one 2×2 to 4×4 cones, thus achieving a maximum scaling factor of 4^2 (see Figure D.8). Our choice to restrict to at most two cones for each terminal branch is justified from the fact that the class of objects we are interested in (i.e., biological structures like humanoids, plants and animals) tend to have a thicker core and thinner terminal limbs, thus requiring a resolution scale only at the peripheries of the shape.

D.5.2 Subdivisions propagation

We propagate the subdivisions induced by the cones throughout the mesh by solving an Integer Linear Programming (ILP) problem. We associate to each regular element of the hex-mesh (i.e., not a cone) a variable x that represents the number of splits necessary to achieve mesh conformity. Regular elements are always split with the 2×2×2 pattern depicted in Figure D.6, iteratively applied as many times as indicated by the associated integer variable.

Specifically, for each pair of adjacent elements we require the number of splits to be equal (inset aside, top), whereas for each pair of hexahedra adjacent to the same cone $\mathcal{C}$ we require the element at the base to be split once more than the element at the tip (inset aside, bottom). Furthermore, since we are solving for the number of splits of each element, we ask each variable x to be positive. This results in the following integer linear programming problem





$$\begin{aligned} \min\ & AX = b \\ & X \geq 0 \\ & X \in \mathbb{I}^n \end{aligned}$$

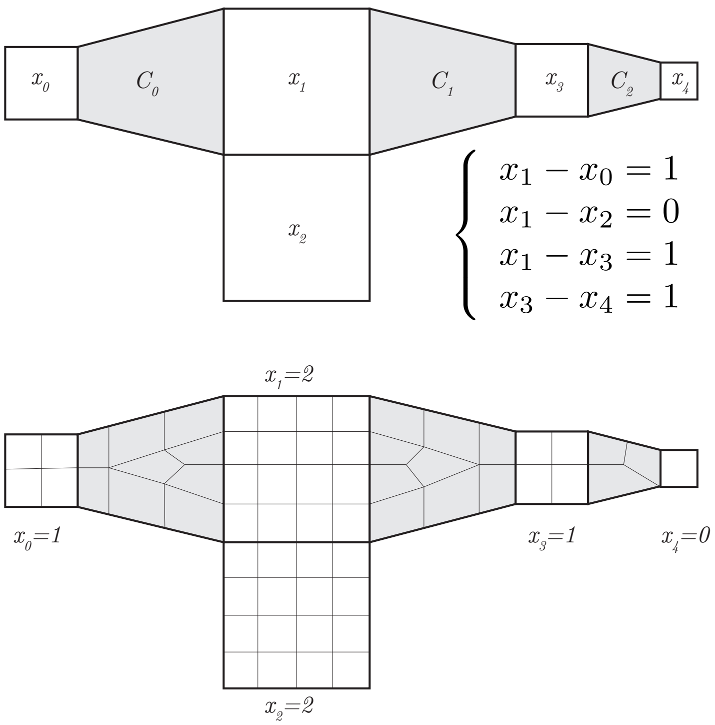


Figure D.7: A 2D example of our subdivision propagation system. We ask adjacent elements (e.g., x_1, x_2) to split the same number of times, and elements at the base of a cone (e.g., x_1, x_3) to split once more than the elements at the tip of the same cone (x_0, x_3, x_4). By solving the resulting ILP problem we know how many times we need to split each element in the mesh to make it conforming.

with n being the number of regular elements in the mesh (here the cones do not count), A being a sparse $n \times n$ coefficient matrix and b being a sparse vector. We solve such problem with Gurobi [Gurobi,]. Note that since we admit cones only in the terminal limbs of a shape no cone will appear in a loop; consequently such system will always admit a solution regardless the topology of the shape, as discussed in [Usai et al., 2015]. In Figure D.7 we show a 2D example of our propagation system; real examples can be seen in Figure D.8, where both DINO and DINOPET are depicted.

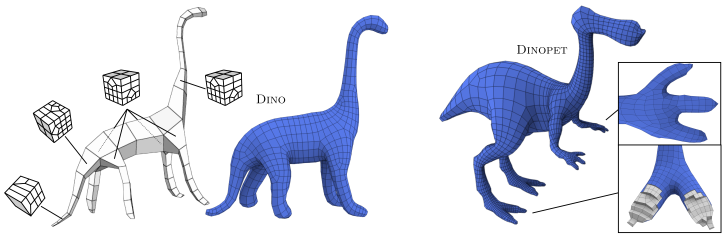


Figure D.8: Left: a hexahedral mesh of the Dino model obtained using our resolution scaling scheme; the volumetric tubular structure used to derive the meshing process. If a limb contains only one cone (legs, neck) we apply a 2×2 to 4×4 subdivision scheme; if there are two cones along the same limb (tail) we place one 1×1 to 2×2 and one 2×2 to 4×4 subdivision schemes, thus achieving a maximum scaling factor of 4^2. Right: another example of our resolution scaling technique applied to the Dinopet.

D.6 Projection and Finalization

As already mentioned in Section D.3 the projection phase is the same used in [Usai et al., 2015]. We briefly recap it here just for completeness. We first use ray-casting to inflate the boundary of the hexa mesh, so as to map it to the input triangle-mesh; we then optimize the resulting mapping using the *abstract domains* technique introduced in [Tarini et al., 2011]. Please refer to [Usai et al., 2015, Tarini et al., 2011] for further details.

Once the surface of $\mathcal{H}$ has been mapped onto the target surface $\mathcal{M}$, we optimize the position of internal vertices, by minimizing the following quadratic energy

$$\sum_{i}^{n} \sum_{j \in N(i)} \|v_i - v_j\|^2$$

Here $N(i)$ is the *volumetric* one ring of the vertex v_i. Minimizing the energy above requires the resolution of a sparse linear system $Ax = b$ in the least square sense, according to the normal equation $A^TAx = A^Tb$.

The hexahedral meshes so generated have good average quality but can, and in general do, contain *inverted* elements, that is, non-convex hexahedra [Knupp, 1999]. Even a single inverted element makes a mesh unusable for applications [Pébay et al., 2008], therefore a further optimization step aimed at removing inverted elements is needed. Since the focus of this work is the generation of a high quality topology, we rely on standard optimization tools for the improvement of per element quality. Notice that separating the generation of the connectivity from the optimization of the embedding is a classical approach in hex-mesh generation [Livesu et al., 2015].

In order to improve mesh quality we first add a *pillowing* (or padding) layer, extruding the surface quads to form a shell of hexahedra surrounding the outer

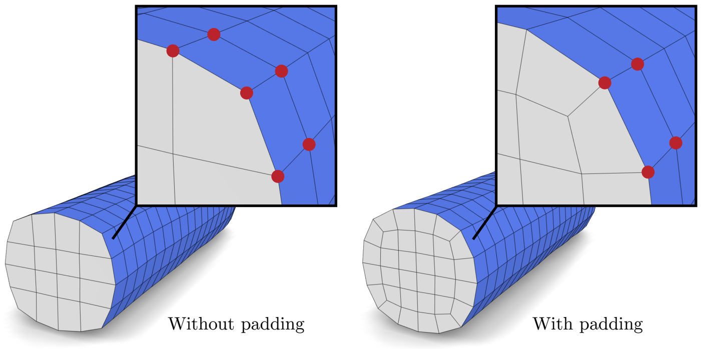


Figure D.9: Prior to padding boundary elements may have more than one facet exposed on the surface. If these facets are coplanar the element will become degenerate, with no chances of optimization as at least six vertices will be constrained on the surface. We therefore add a padding layer. This ensures that each element will have at most one facet exposed (four vertices), thus leaving enough degrees of freedom for the subsequent mesh optimization.

surface of the hexahedral mesh [Gregson et al., 2011]. After pillowing, each element of the mesh will have at most four vertices on the surface. Consequently, at least four vertices per element will be free to move in the interior, guaranteeing enough degrees of freedom for the subsequent quality optimization (Figure D.9). We then apply the edge-cone rectification algorithm [Livesu et al., 2015] to remove all the inverted elements from the mesh and improve on both minimum and average quality (see Table D.1). For the edge-cone rectification algorithm we always used the same parameters, that is: automatic estimation of the target edge-length, attraction to the surface weight (α) equal to 20 and attraction to the sharp features (β) equal to 0.

D.7 Results

We have implemented our algorithm as a single threaded C++ application and we run our tests on a MacBook Air equipped with a 1.7GHz Intel Core i5 and 4GB of RAM. The running times of our method vary from 1.5 seconds for a simple shape like CACTUS to approximately 1 minute for a complex shape like WARRIOR. Alternative meshing strategies such as [Huang et al., 2014, Livesu et al., 2013] are one order of magnitude slower.

For the computation of the curve-skeletons we used different algorithms, specifically: ARMADILLO, BIG BUDDY, BLOCK, CACTUS, CLEF, DINOPET, DINOSAUR, OCTOPUS, SANTA, BLOOD, VESSEL and WARRIOR were skeletonized using the 3D-from-2D approach described in [Biasotti et al., 2015, Livesu and Scateni, 2013, Livesu et al., 2012]; FERTILITY and ROCKER ARM were skeletonized using the mean curvature approach described in [Tagliasacchi et al.,

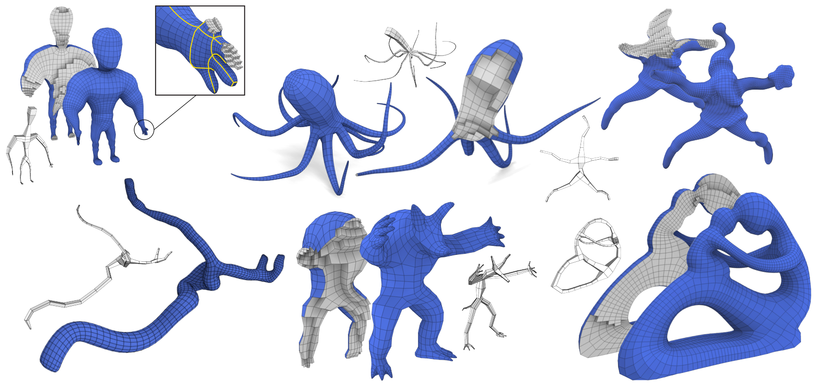


Figure D.10: A gallery of hexahedral meshes generated with our method without resolution scaling; the tubular structures used to derive the hexahedral meshes are shown in white. From top left to bottom right: BigBuddy, Octopus, Santa, BloodVessel, Armadillo and Fertility.

2012]. In the latter case, we manually simplified the skeleton, merging nearby branching nodes to better reflect the logical structure of the shape (we used Skeleton Lab [Barbieri et al., 2016] for each such editing). This operation was not necessary for the skeletons computed with [Biasotti et al., 2015, Livesu and Scateni, 2013, Livesu et al., 2012] as they already embed heuristics for the automatic collapse of spurious branches.

We tested our method on a wide range of objects (Figures D.10, D.1, D.8, D.12); next to each model we show the tubular structure we used to drive the meshing process. The meshes produced by our method embed a volume decomposition that reflects the logical structure of the input shape and naturally align with its main features. As acknowledged in [Blacker, 2001] a good alignment with the features of a shape leads to superior results in the simulations. Furthermore, recent studies have shown that simplifying the singularity graph of a hexahedral mesh by aligning its singular vertices helps to keep the resolution lower and at the same time improves the mesh quality [Gao et al., 2015b]. Our method builds upon the quad layout generation algorithm proposed in [Usai et al., 2015] so it naturally aligns singular vertices in a meaningful way, providing as-coarse-as-possible singularity layouts that turn into high quality, boundary conforming, hexahedral meshes (see yellow lines in the top closeup in Figure D.10).

We summarize our results in Table D.1, where we also compare against the skeleton-based method proposed in [Lin et al., 2012], two PolyCube-based methods [Huang et al., 2014, Livesu et al., 2013], one frame field-based method [Li et al., 2012], an Octree-based method [Maréchal, 2009] and the recently published Generalized Sweeping [Gao et al., 2016] and CVIF [Lin et al., 2015] methods. For each model we report: the mesh resolution (for both input and output), and the minimum and average quality of the output hex-mesh and the average

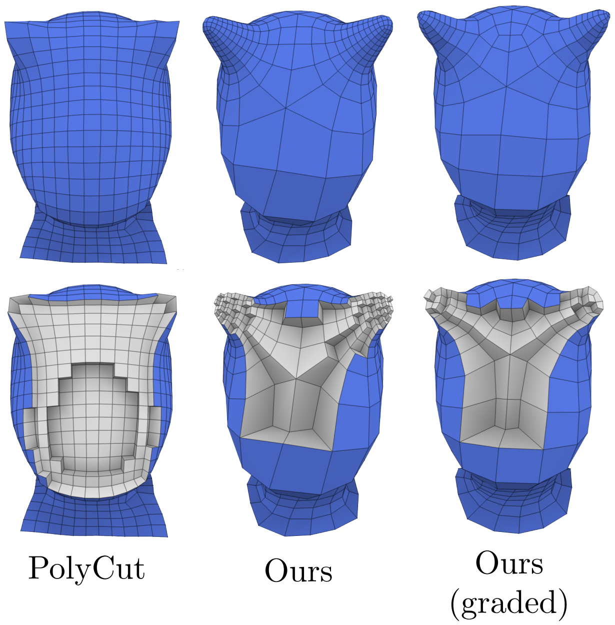


Figure D.11: The head of the WARRIOR meshed with PolyCut (left), our method without applying the resolution scaling step (center) and applying it (right).

distance from the input surface. We evaluate quality using the Scaled Jacobian (SJ), a popular metric that measures the deviation of each element from a perfect cube [Pébay et al., 2008]; the SJ is defined within the range $[-1, 1]$ with one being optimal and negative values denoting inverted elements. None of the models shown throughout the paper contains inverted elements (i.e., min SJ $>$ 0), this is a fundamental minimum requirement for many applications involving hexahedral meshes [Blacker, 2001].

Since we use [Livesu et al., 2015] in the final step of our method (Section D.6), for the sake of a fair comparison we optimized the meshes of our competitors with the same technique, whenever possible. Exceptions to this rule are: the Generalized Sweeping [Gao et al., 2015b] and Frame Field [Li et al., 2012] approaches, for which we report the quality from the orginal papers as with [Livesu et al., 2015] it was impossible to improve any further; and CVIF [Lin et al., 2015] and the skeleton-based approaches [Lin et al., 2012], for which we did not have the geometry available. Notice that both [Lin et al., 2015] and [Lin et al., 2012] already employ some optimization strategy to finalize their meshes, therefore we believe that the values reported in Table D.1 truly reflect the potential of the connectivity generated by their meshing strategies.

From the qualitative point of view the performance of our algorithm matches the parameterization-based methods [Huang et al., 2014, Livesu et al., 2013, Li et al., 2012] and outperforms Generalized Sweeping [Gao et al., 2015b], grid-

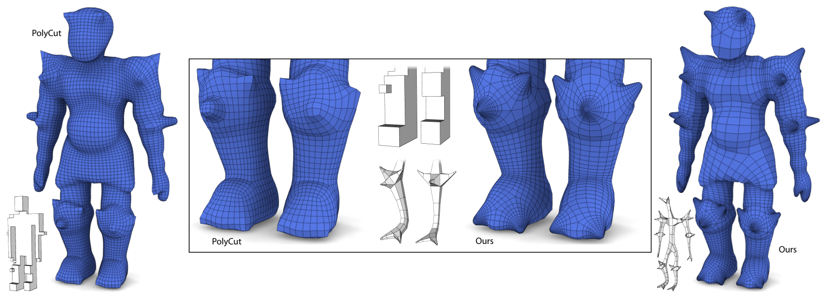


Figure D.12: Mapping to an axis aligned domain strongly limits the ability to represent off axis features; most of the spikes of the WARRIOR are not caught by the polycube, resulting in a poor meshing that hardly matches the original geometry (left). Our tubular structure nicely follows each and every spike, generating a better meshing (right).

based [Lin et al., 2015, Maréchal, 2009] and skeleton-based [Lin et al., 2012] algorithms. Our method achieved average SJ above 0.9 for the majority of the models we produced, outperforming previous skeleton-based methods like [Lin et al., 2012], whose hexahedral meshes hardly exceed 0.8 average SJ.

Another important feature that emerged from our tests is the ability to keep the resolution low, an important criterion for fast simulations. As can be noticed in Table D.1 we produced the coarsest hex-meshes in the majority of the comparisons. From this point of view the worst performances come from the Octree [Maréchal, 2009] and voxel-based CVIF [Lin et al., 2015] methods.

We also propose some visual comparison against a polycube-based method (i.e., PolyCut [Livesu et al., 2013]). As one can notice, in Figure D.1, polycube-based methods may not align with the surface of the model, placing unnecessary singularities that prevent the edges to align with the limbs of the CACTUS. This property, called *surface conformity*, serves to promote the placement of high quality elements close to the boundary of the model and it is an important factor to ensure accurate simulations [Ruiz-Girones, 2011]. The connectivity generated by our method nicely aligns with both the limbs and the core of the shape enabling the placement of high quality elements nearby the boundary of the shape. Our method is also able to generate a connectivity that nicely fits the assembly of spikes in the knees, elbows and shoulders of the WARRIOR (Figure D.12). We note that, because of the rigid structure of the parametric domain, such a meshing is impossible to achieve with a polycube-based method. As it can be noticed in the closeup in Figure D.12, some of the spikes are not caught by the polycube, resulting in a hexahedral mesh that hardly fits the target geometry.

In Figure D.11 we make a sample visual comparison also with our method once the resolution scaling is applied. As one can see from the picture on the right, our complete method obtains a reasonable compromise between element

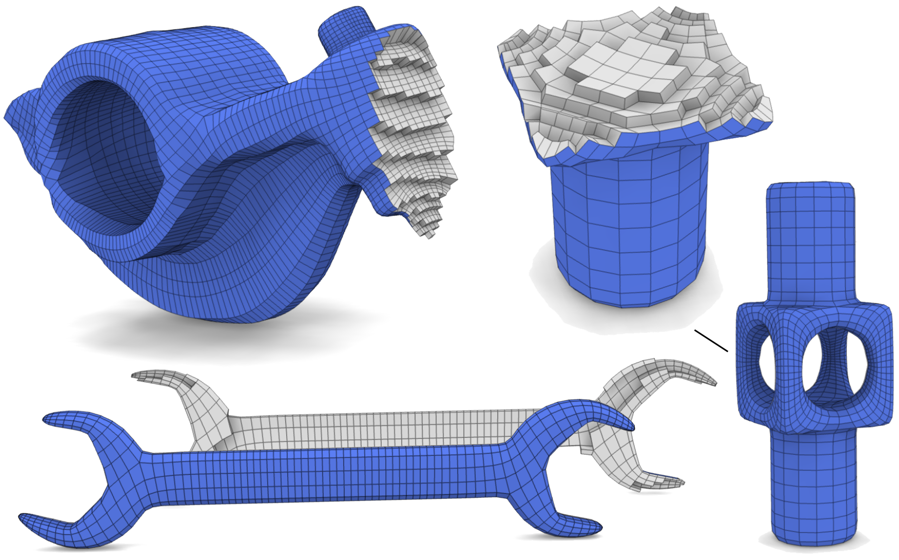


Figure D.13: Some preliminary test on three mechanical parts: RockerArm (top left), Block (right) and Clef (bottom). Although out of the scope of this work our method could produce full hexahedral meshes for each model, but it still fails at aligning the meshing to sharp features; something that we plan to work on in the future.

regularity (for which PolyCut is optimal) and alignment with the features (for gaining the optimum on this we should not apply the reduction scheme).

Finally, although out of the scope of our method, we run some preliminary tests on mechanical parts. As one can notice in Figure D.13, we have been able to produce full hexahedral meshes for our test models, but we still fail at aligning the edge flow with the sharp edges and features of the shapes. In the future, we plan to improve our meshing strategy by taking into account the alignment to sharp features, so as to be able to embrace a broader range of shapes.

D.8 Conclusions

We have introduced a skeleton-based algorithm for the automatic generation of structured hexahedral meshes of tubular shapes, and we have also presented novel techniques for the control of the resolution of the hexahedral mesh, both across and along the skeleton curves. To reach this goal we exploit the properties of the curve-skeleton, using it as a proxy to derive structural information about 3D shapes, as in [Usai et al., 2015]. We then use such information to construct volumetric meshes that nicely align with the branching structure of the target

Model	**#Tris**	**#Hexa**	**avg/min SJ**	**Avg dist**
ARMADILLO				
PolyCut[†]		30K	**.90**/.14	
Ours	331K	**4K**	.88/**.21**	3.7×10^{-5}
BIG BUDDY				
Ours	27K	15K	.90/.32	4.5×10^{-5}
BLOCK				
PolyCut[†]		**3K**	.87/.25	
Octree-based[†]		20K	**.91**/.27	
Ours	5K	4K	.81/**.36**	8.7×10^{-6}
CACTUS				
PolyCut[†]		8K	**.94**/.42	
Ours	11K	**4K**	.92/**.52**	2.1×10^{-6}
CLEF				
Octree-based[†]		10K	**.90**/**.34**	
Ours	3K	**2K**	.86/.29	2.9×10^{-5}
DINOPET				
Ours	9K	18K	.92/.18	5.3×10^{-5}
DINOSAUR				
Ours	47K	9K	.91/.41	4.7×10^{-5}
FERTILITY				
ℓ_1 PolyCubes[†]		18K	**.94**/.42	
PolyCut[†]		54K	.86/.34	
Skel-based[§]		16K	.75/.08	
CVIF[§]		107K	.90/.04	
Frame-field[‡]		14K	.91/.35	
Gen.Sweep[‡]		20K	.82/.18	
Ours	33K	**8K**	.90/**.50**	2.6×10^{-5}
OCTOPUS				
Ours	66K	5K	.88/.11	4.4×10^{-4}
ROCKERARM				
ℓ_1 PolyCubes[†]		24K	**.96**/**.59**	
PolyCut[†]		57K	**.96**/.58	
Frame-field [†]		**11K**	.94/.57	
Gen.Sweep[‡]		**11K**	.83/.11	
CVIF[§]		63K	.90/.06	
Ours	20K	19K	.91/.16	6.2×10^{-5}
SANTA				
Skel-based[§]		**15K**	.72/.08	
CVIF[§]		73K	.88/.04	
Ours	13K	26K	**.94**/**.37**	2.2×10^{-4}
BLOOD VESSEL				
Ours	60K	5K	.88/.32	9.0×10^{-4}
WARRIOR				
PolyCut[†]		24K	**.94**/.14	
Ours	27K	**19K**	.90/**.29**	8.7×10^{-5}

[†] optimized with [Livesu et al., 2015]
[‡] data from the original paper, we could not improve on quality any further using [Livesu et al., 2015]
[§] data from the original paper, models not available

Table D.1: Summary of our results. From left to right: number of input triangles, number of output hexahedra, average and minimjm Scaled Jacobian (compute using the Verdict Library [Stimpson et al., 2007]), average deviation from the input surface (measure using Metro [Cignoni et al., 1998]). We compare against: ℓ_1 PolyCubes [Huang et al., 2014], PolyCut [Livesu et al., 2013], Frame-field based [Li et al., 2012], Skeleton-based [Lin et al., 2012], Generalized Sweeping [Gao et al., 2015b], CVIF [Lin et al., 2015] and Octree-based [Maréchal, 2009] all-hexahedral meshing techniques. For each model, we highlight in bold both the lowest resolution and the highest min/avg quality.

shape. The method is easy to code. It does not require any user parameter and it generates quality meshes for any 3D model that admits a skeletal representation.

D.8.1 Limitations and further works

This method is inherently limited in its scope by the class of shapes that admit a skeletal representation. Although this is not a real limitation for biological shapes like humanoids, animals, vessels and plants, we would like to be more general and embrace a wider class of shapes. We are therefore looking for other shape descriptors or shape understanding processes that can be exploited to derive structural information about general 3D shapes, to be used to accomplish tasks like surface and volume remeshing.

Similarly to polycubes, the hexahedral meshes generated by our method have a simple block structure that does not conform to a frame field [Fang et al., 2016, Li et al., 2012]. As a consequence, the twisting component along the skeleton curves can hardly be controlled, possibly leading to poor meshing results.

The method also inherits the limitations of [Usai et al., 2015] regarding the alignment to sharp features. As it is mainly intended for biologically-inspired shapes, at the moment the preservation of sharp features is not addressed.

Bibliography

[Alemanno et al., 2014] Alemanno, G., Cignoni, P., Pietroni, N., Ponchio, F., and Scopigno, R. (2014). Interlocking pieces for printing tangible cultural heritage replicas. In *GCH*, pages 145–154. 7, 8

[Alliez et al., 2005] Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M. (2005). Variational tetrahedral meshing. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 617–625. ACM. 67

[Attene, 2015] Attene, M. (2015). Shapes in a box: Disassembling 3d objects for efficient packing and fabrication. In *Computer Graphics Forum*, volume 34, pages 64–76. Wiley Online Library. 9

[Attene et al., 2008] Attene, M., Mortara, M., Spagnuolo, M., and Falcidieno, B. (2008). Hierarchical convex approximation of 3d shapes for fast region selection. In *Computer graphics forum*, volume 27, pages 1323–1332. Wiley Online Library. 8

[Barbieri et al., 2016] Barbieri, S., Meloni, P., Usai, F., Spano, L. D., and Scateni, R. (2016). An interactive editor for curve-skeletons: Skeletonlab. *Computers & Graphics*, 60:23–33. 80

[Biasotti et al., 2015] Biasotti, S., Tarini, M., and Giachetti, A. (2015). Practical medial axis filtering for occlusion-aware contours. 79, 80

[Blacker, 2001] Blacker, T. (2001). Automated conformal hexahedral meshing constraints, challenges and opportunities. *Engineering with Computers*, 17(3):201–210. 67, 68, 80, 81

[Chazelle, 1984] Chazelle, B. (1984). Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM Journal on Computing*, 13(3):488–507. 8

[Chen et al., 2013] Chen, D., Sitthi-amorn, P., Lan, J. T., and Matusik, W. (2013). Computing and fabricating multiplanar models. In *Computer graphics forum*, volume 32, pages 305–315. Wiley Online Library. 6, 12

[Chen and Sass, 2016] Chen, L. and Sass, L. (2016). Fresh press modeler: A generative system for physically based low fidelity prototyping. *Computers & Graphics*, 54:157–165. 6, 12

[Cherchi et al., 2016] Cherchi, G., Livesu, M., and Scateni, R. (2016). Polycube simplification for coarse layouts of surfaces and volumes. In *Computer Graphics Forum*, volume 35, pages 11–20. Wiley Online Library. 70

[Cignoni et al., 2014] Cignoni, P., Pietroni, N., Malomo, L., and Scopigno, R. (2014). Field-aligned mesh joinery. *ACM Transactions on Graphics (TOG)*, 33(1):11. 5, 9

[Cignoni et al., 1998] Cignoni, P., Rocchini, C., and Scopigno, R. (1998). Metro: Measuring error on simplified surfaces. In *Computer Graphics Forum*, volume 17, pages 167–174. Wiley Online Library. 43, 84

[Cohen-Steiner et al., 2004] Cohen-Steiner, D., Alliez, P., and Desbrun, M. (2004). Variational shape approximation. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 905–914. ACM. 5

[Cook, 1984] Cook, R. L. (1984). Shade trees. *ACM Siggraph Computer Graphics*, 18(3):223–231. 7

[Cormen, 2009] Cormen, T. H. (2009). *Introduction to algorithms*. MIT press. 36

[Cornea et al., 2007] Cornea, N. D., Silver, D., and Min, P. (2007). Curve-skeleton properties, applications, and algorithms. *IEEE transactions on visualization and computer graphics*, 13(3):0530–548. 68

[Deuss et al., 2014] Deuss, M., Panozzo, D., Whiting, E., Liu, Y., Block, P., Sorkine-Hornung, O., and Pauly, M. (2014). Assembling self-supporting structures. *ACM Transactions on Graphics (TOG)*, 33(6):214. 9

[Dinh et al., 2015] Dinh, H. Q., Gelman, F., Lefebvre, S., and Claux, F. (2015). Modeling and toolpath generation for consumer-level 3d printing. In *ACM SIGGRAPH 2015 Courses*, page 17. ACM. 5

[Doggett and Hirche, 2000] Doggett, M. and Hirche, J. (2000). Adaptive view dependent tessellation of displacement maps. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 59–66. ACM. 7

[Fang et al., 2016] Fang, X., Xu, W., Bao, H., and Huang, J. (2016). All-hex meshing using closed-form induced polycube. *ACM Transactions on Graphics (TOG)*, 35(4):124. 70, 85

[Fekete and Mitchell, 2001] Fekete, S. P. and Mitchell, J. S. (2001). Terrain decomposition and layered manufacturing. *International Journal of Computational Geometry & Applications*, 11(06):647–668. 8, 27

[Gao et al., 2015a] Gao, W., Zhang, Y., Nazzetta, D. C., Ramani, K., and Cipra, R. J. (2015a). Revomaker: Enabling multi-directional and functionally-embedded 3d printing using a rotational cuboidal platform. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 437–446. ACM. 9, 32

[Gao et al., 2015b] Gao, X., Deng, Z., and Chen, G. (2015b). Hexahedral mesh re-parameterization from aligned base-complex. *ACM Transactions on Graphics (TOG)*, 34(4):142. 68, 71, 80, 81, 84

[Gao et al., 2016] Gao, X., Martin, T., Deng, S., Cohen, E., Deng, Z., and Chen, G. (2016). Structured volume decomposition via generalized sweeping. *IEEE transactions on visualization and computer graphics*, 22(7):1899–1911. 80

[Gregson et al., 2011] Gregson, J., Sheffer, A., and Zhang, E. (2011). All-hex mesh generation via volumetric polycube deformation. In *Computer graphics forum*, volume 30, pages 1407–1416. Wiley Online Library. 70, 79

[Guennebaud et al., 2010] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org. 46

[Gurobi,] Gurobi. Optimizer 6.5. `http://www.gurobi.com/`. 46, 77

[Hao et al., 2011] Hao, J., Fang, L., and Williams, R. E. (2011). An efficient curvature-based partitioning of large-scale stl models. *Rapid Prototyping Journal*, 17(2):116–127. 7

[Herholz et al., 2015] Herholz, P., Matusik, W., and Alexa, M. (2015). Approximating free-form geometry with height fields for manufacturing. In *Computer Graphics Forum*, volume 34, pages 239–251. Wiley Online Library. 7, 9

[Hildebrand et al., 2012] Hildebrand, K., Bickel, B., and Alexa, M. (2012). crdbrd: Shape fabrication by sliding planar slices. In *Computer Graphics Forum*, volume 31, pages 583–592. Wiley Online Library. 9

[Hildebrand et al., 2013] Hildebrand, K., Bickel, B., and Alexa, M. (2013). Orthogonal slicing for additive manufacturing. *Computers & Graphics*, 37(6):669–675. 7

[Hu et al., 2014] Hu, R., Li, H., Zhang, H., and Cohen-Or, D. (2014). Approximate pyramidal shape decomposition. *ACM Trans. Graph.*, 33(6):213–1. i, 8, 27, 28, 45, 49

[Huang et al., 2014] Huang, J., Jiang, T., Shi, Z., Tong, Y., Bao, H., and Desbrun, M. (2014). l-based construction of polycube maps from complex shapes. *ACM Transactions on Graphics (TOG)*, 33(3):25. 70, 79, 80, 81, 84

[Huang et al., 2011] Huang, J., Tong, Y., Wei, H., and Bao, H. (2011). Boundary aligned smooth 3d cross-frame field. In *ACM transactions on graphics (TOG)*, volume 30, page 143. ACM. 71

[Jacobson et al.,] Jacobson, A., Panozzo, D., et al. libigl: A simple c++ geometry processing library, 2016. 46

[Johnson, 1975] Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84. 40

[Jun et al., 2003] Jun, C.-S., Cha, K., and Lee, Y.-S. (2003). Optimizing tool orientations for 5-axis machining by configuration-space search method. *Computer-Aided Design*, 35(6):549–566. 2

[Kahn, 1962] Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562. 40

[Knupp, 1999] Knupp, P. M. (1999). Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities, part 1-a framework for surface mesh optimization. Technical report, Sandia National Laboratories, Albuquerque, NM, and Livermore, CA. 78

[Kowalski et al., 2014] Kowalski, N., Ledoux, F., and Frey, P. (2014). Block-structured hexahedral meshes for cad models using 3d frame fields. *Procedia Engineering*, 82:59–71. 71

[Kraevoy et al., 2007] Kraevoy, V., Julius, D., and Sheffer, A. (2007). Shuffler: Modeling with interchangeable parts. *The Visual Computer*. 8

[Kremer et al., 2014] Kremer, M., Bommes, D., Lim, I., and Kobbelt, L. (2014). Advanced automatic hexahedral mesh generation from surface quad meshes. In *Proceedings of the 22nd International Meshing Roundtable*, pages 147–164. Springer. 67, 70

[Li et al., 2012] Li, Y., Liu, Y., Xu, W., Wang, W., and Guo, B. (2012). All-hex meshing using singularity-restricted field. *ACM Transactions on Graphics (TOG)*, 31(6):177. 71, 80, 81, 84, 85

[Lien and Amato, 2007] Lien, J.-M. and Amato, N. M. (2007). Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 121–131. ACM. 8

[Lin et al., 2015] Lin, H., Jin, S., Liao, H., and Jian, Q. (2015). Quality guaranteed all-hex mesh generation by a constrained volume iterative fitting algorithm. *Computer-Aided Design*, 67:107–117. 69, 80, 81, 82, 84

[Lin et al., 2012] Lin, H., Liao, H., and Deng, C. (2012). Filling triangular mesh model with all-hex mesh by volume subdivision fitting. *State Key Lab of CAD & CG, Zhejiang University Report No: TR ZJUCAD*, 2:2012. 69, 80, 81, 82, 84

[Liu et al., 2014] Liu, L., Shamir, A., Wang, C. C., and Whiting, E. (2014). 3d printing oriented design: geometry and optimization. In *SIGGRAPH ASIA Courses*, pages 1–1. 5

[Liu et al., 2015] Liu, L., Zhang, Y., Liu, Y., and Wang, W. (2015). Feature-preserving t-mesh construction using skeleton-based polycubes. *Computer-Aided Design*, 58:162–172. 69

[Livesu et al., 2017] Livesu, M., Ellero, S., Martínez, J., Lefebvre, S., and Attene, M. (2017). From 3d models to 3d prints: an overview of the processing pipeline. In *Computer Graphics Forum*, volume 36, pages 537–564. Wiley Online Library. 5

[Livesu et al., 2012] Livesu, M., Guggeri, F., and Scateni, R. (2012). Reconstructing the curve-skeletons of 3d shapes using the visual hull. *IEEE transactions on visualization and computer graphics*, 18(11):1891–1901. 73, 79, 80

[Livesu and Scateni, 2013] Livesu, M. and Scateni, R. (2013). Extracting curve-skeletons from digital shapes using occluding contours. *The Visual Computer*, 29(9):907–916. 73, 79, 80

[Livesu et al., 2015] Livesu, M., Sheffer, A., Vining, N., and Tarini, M. (2015). Practical hex-mesh optimization via edge-cone rectification. *ACM Transactions on Graphics (TOG)*, 34(4):141. 69, 78, 79, 81, 84

[Livesu et al., 2013] Livesu, M., Vining, N., Sheffer, A., Gregson, J., and Scateni, R. (2013). Polycut: monotone graph-cuts for polycube base-complex construction. *ACM Transactions on Graphics (TOG)*, 32(6):171. 6, 70, 79, 80, 81, 82, 84

[Lo et al., 2009] Lo, K.-Y., Fu, C.-W., and Li, H. (2009). 3d polyomino puzzle. In *ACM Transactions on Graphics (TOG)*, volume 28, page 157. ACM. 9

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM. 13

[Luo et al., 2012] Luo, L., Baran, I., Rusinkiewicz, S., and Matusik, W. (2012). Chopper: partitioning models into 3d-printable parts. 7

[Maréchal, 2009] Maréchal, L. (2009). Advances in octree-based all-hexahedral mesh generation: handling sharp features. *Proceedings of the 18th International Meshing Roundtable*, pages 65–84. 69, 80, 82, 84

[Medellin et al., 2007] Medellin, H., Lim, T., Corney, J., Ritchie, J., and Davies, J. (2007). Automatic subdivision and refinement of large components for rapid prototyping production. *Journal of Computing and Information Science in Engineering*, 7(3):249–258. 7

[Mitani and Suzuki, 2004] Mitani, J. and Suzuki, H. (2004). Making paper-craft toys from meshes using strip-based approximate unfolding. In *ACM transactions on graphics (TOG)*, volume 23, pages 259–263. ACM. 6

[Muntoni and Scateni, 2014] Muntoni, A. and Scateni, R. (2014). Simplifying the shape of triangle meshes for unfolding, milling and fabrication. 13, 14

[Nealen et al., 2006] Nealen, A., Müller, M., Keiser, R., Boxerman, E., and Carlson, M. (2006). Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library. 67

[Newman et al., 2015] Newman, S. T., Zhu, Z., Dhokia, V., and Shokrani, A. (2015). Process planning for additive and subtractive manufacturing technologies. *CIRP Annals-Manufacturing Technology*, 64(1):467–470. 1

[Nieser et al., 2011] Nieser, M., Reitebuch, U., and Polthier, K. (2011). Cubecover–parameterization of 3d volumes. In *Computer graphics forum*, volume 30, pages 1397–1406. Wiley Online Library. 71

[Nuvoli and Scateni, 2017] Nuvoli, S. and Scateni, R. (2017). Unfolding polygon meshes for the fabrication of simplified shapes. 22

[Pébay et al., 2008] Pébay, P. P., Thompson, D., Shepherd, J., Knupp, P., Lisle, C., Magnotta, V. A., and Grosland, N. M. (2008). New applications of the verdict library for standardized mesh verification pre, post, and end-to-end processing. In *Proceedings of the 16th International Meshing Roundtable*, pages 535–552. Springer. 78, 81

[Richter and Alexa, 2015] Richter, R. and Alexa, M. (2015). Beam meshes. *Computers & Graphics*, 53:28–36. 5, 12

[Ruiz-Girones, 2011] Ruiz-Girones, E. (2011). Automatic hexahedral meshing algorithms: from structured to unstructured meshes. 68, 82

[Ruiz-Gironés et al., 2015] Ruiz-Gironés, E., Roca, X., Sarrate, J., Montenegro, R., and Escobar, J. M. (2015). Simultaneous untangling and smoothing of quadrilateral and hexahedral meshes using an object-oriented framework. *Advances in Engineering Software*, 80:12–24. 70

[Scalas and Scateni, 2016] Scalas, A. and Scateni, R. (2016). Interactive editing of unfoldable reduced representations of 3d shapes. 20

[Schneiders, 1996] Schneiders, R. (1996). A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with computers*, 12(3-4):168–177. 69

[Schüller et al., 2013] Schüller, C., Kavan, L., Panozzo, D., and Sorkine-Hornung, O. (2013). Locally injective mappings. In *Computer Graphics Forum*, volume 32, pages 125–135. Wiley Online Library. 64

[Schwartzburg and Pauly, 2013] Schwartzburg, Y. and Pauly, M. (2013). Fabrication-aware design with intersecting planar pieces. In *Computer Graphics Forum*, volume 32, pages 317–326. Wiley Online Library. 5, 9

[Shamir, 2008] Shamir, A. (2008). A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library. 6

[Skouras et al., 2015] Skouras, M., Coros, S., Grinspun, E., and Thomaszewski, B. (2015). Interactive surface design with interlocking elements. *ACM Transactions on Graphics (TOG)*, 34(6):224. 9

[Sokolov et al., 2016] Sokolov, D., Ray, N., Untereiner, L., and Lévy, B. (2016). Hexahedral-dominant meshing. *ACM Transactions on Graphics (TOG)*, 35(5):157. 71

[Song et al., 2016] Song, P., Deng, B., Wang, Z., Dong, Z., Li, W., Fu, C.-W., and Liu, L. (2016). Cofifab: coarse-to-fine fabrication of large 3d objects. *ACM Transactions on Graphics (TOG)*, 35(4):45. 7, 12

[Song et al., 2012] Song, P., Fu, C.-W., and Cohen-Or, D. (2012). Recursive interlocking puzzles. *ACM Transactions on Graphics (TOG)*, 31(6):128. 9

[Song et al., 2015] Song, P., Fu, Z., Liu, L., and Fu, C.-W. (2015). Printing 3d objects with interlocking parts. *Computer Aided Geometric Design*, 35:137–148. 7

[Sorkine, 2006] Sorkine, O. (2006). Differential representations for mesh processing. In *Computer Graphics Forum*, volume 25, pages 789–807. Wiley Online Library. 43

[Stimpson et al., 2007] Stimpson, C., Ernst, C., Knupp, P., Pébay, P., and Thompson, D. (2007). The verdict library reference manual. *Sandia National Laboratories Technical Report*, 9. 84

[Tagliasacchi et al., 2012] Tagliasacchi, A., Alhashim, I., Olson, M., and Zhang, H. (2012). Mean curvature skeletons. In *Computer Graphics Forum*, volume 31, pages 1735–1744. Wiley Online Library. 79

[Tagliasacchi et al., 2016] Tagliasacchi, A., Delame, T., Spagnuolo, M., Amenta, N., and Telea, A. (2016). 3d skeletons: A state-of-the-art report. In *Computer Graphics Forum*, volume 35, pages 573–597. Wiley Online Library. 68

[Takayama et al., 2014] Takayama, K., Panozzo, D., and Sorkine-Hornung, O. (2014). Pattern-based quadrangulation for n-sided patches. In *Computer Graphics Forum*, volume 33, pages 177–184. Wiley Online Library. 75

[Tarini et al., 2004] Tarini, M., Hormann, K., Cignoni, P., and Montani, C. (2004). Polycube-maps. In *ACM transactions on graphics (TOG)*, volume 23, pages 853–860. ACM. 6, 70

[Tarini et al., 2011] Tarini, M., Puppo, E., Panozzo, D., Pietroni, N., and Cignoni, P. (2011). Simple quad domains for field aligned mesh parametrization. *ACM Transactions on Graphics (TOG)*, 30(6):142. 73, 78

[Taubin, 1995] Taubin, G. (1995). Curve and surface smoothing without shrinkage. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 852–857. IEEE. 43

[Tautges, 2001] Tautges, T. J. (2001). The generation of hexahedral meshes for assembly geometry: survey and progress. *International Journal for Numerical Methods in Engineering*, 50(12):2617–2642. 68

[Tautges, 2004] Tautges, T. J. (2004). Moab-sd: integrated structured and unstructured mesh representation. *Engineering with Computers*, 20(3):286–293. 68, 72

[Tautges et al., 1996] Tautges, T. J., Blacker, T., and Mitchell, S. A. (1996). The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 39(19):3327–3349. 70

[Tor and Middleditch, 1984] Tor, S. B. and Middleditch, A. E. (1984). Convex decomposition of simple polygons. *ACM Transactions on Graphics (TOG)*, 3(4):244–265. 8

[Umetani et al., 2015] Umetani, N., Bickel, B., and Matusik, W. (2015). Computational tools for 3d printing. In *SIGGRAPH Courses*, pages 9–1. 5

[Usai et al., 2015] Usai, F., Livesu, M., Puppo, E., Tarini, M., and Scateni, R. (2015). Extraction of the quad layout of a triangle mesh guided by its curve skeleton. *ACM Transactions on Graphics (TOG)*, 35(1):6. 68, 69, 72, 73, 74, 77, 78, 80, 83, 85

[Vanek et al., 2014] Vanek, J., Galicia, J., Benes, B., Měch, R., Carr, N., Stava, O., and Miller, G. (2014). Packmerger: A 3d print volume optimizer. In *Computer Graphics Forum*, volume 33, pages 322–332. Wiley Online Library. 7

[Wang et al., 2016] Wang, W. M., Zanni, C., and Kobbelt, L. (2016). Improved surface quality in 3d printing by optimizing the printing direction. In *Computer Graphics Forum*, volume 35, pages 59–70. Wiley Online Library. 7

[Wu et al., 2016] Wu, R., Peng, H., Guimbretière, F., and Marschner, S. (2016). Printing arbitrary meshes with a 5dof wireframe printer. *ACM Transactions on Graphics (TOG)*, 35(4):101. 9

[Xin et al., 2011] Xin, S., Lai, C.-F., Fu, C.-W., Wong, T.-T., He, Y., and Cohen-Or, D. (2011). Making burr puzzles from 3d models. In *ACM Transactions on Graphics (TOG)*, volume 30, page 97. ACM. 9

[Yao et al., 2009] Yao, C.-Y., Chu, H.-K., Ju, T., and Lee, T.-Y. (2009). Compatible quadrangulation by sketching. *Computer Animation and Virtual Worlds*, 20(2-3):101–109. 69

[Yao et al., 2015] Yao, M., Chen, Z., Luo, L., Wang, R., and Wang, H. (2015). Level-set-based partitioning and packing optimization of a printable model. *ACM Transactions on Graphics (TOG)*, 34(6):214. 7

[Zhang and Chen, 2001] Zhang, C. and Chen, T. (2001). Efficient feature extraction for 2d/3d objects in mesh representation. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 935–938. IEEE. 41

[Zhang et al., 2015] Zhang, X., Le, X., Panotopoulou, A., Whiting, E., and Wang, C. C. (2015). Perceptual models of preference in 3d printing direction. *ACM Transactions on Graphics (TOG)*, 34(6):215. 45

[Zhang et al., 2007] Zhang, Y., Bazilevs, Y., Goswami, S., Bajaj, C. L., and Hughes, T. J. (2007). Patient-specific vascular nurbs modeling for isogeometric analysis of blood flow. *Computer methods in applied mechanics and engineering*, 196(29):2943–2959. 69

[Zhang et al., 2016] Zhang, Y., Gao, W., Paredes, L., and Ramani, K. (2016). Cardboardizer: creatively customize, articulate and fold 3d mesh models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 897–907. ACM. 9

[Zhou et al., 2016] Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. (2016). Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, 35(4):39. 46

[Zhou et al., 2013] Zhou, Q., Panetta, J., and Zorin, D. (2013). Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137–1. 41

[Zimmer et al., 2014] Zimmer, H., Lafarge, F., Alliez, P., and Kobbelt, L. (2014). Zometool shape approximation. *Graphical Models*, 76(5):390–401. 6