The Extended Isogeometric Segmentation Pipeline

Dissertation
zur Erlangung des akademischen Grades
Doktor der technischen Wissenschaften
im Doktoratsstudium
Technische Wissenschaften
Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 25. Februar 2019

Michael Haberleitner
First and foremost I want to thank my supervisor Prof. Bert Jüttler for giving me the chance to join his research group and for his guidance over the last years. Discussing scientific challenges with him is always very constructive and inspiring.

I also want to thank Prof. Bernd Simeon from TU Kaiserslautern for reviewing my thesis, and the Austrian Science Fund (FWF), as well as the European Research Council (ERC) for the financial support that made this work possible.

During my first few months at the institute I had the pleasure of working with Dr. Michael Pauley. He went out of his way to help me understand the subject, especially the existing implementation. Even after he had left the institute he was eager to answer my questions as quick and detailed as possible.

To become a member of the Institute of Applied Geometry (a.k.a. “The Cake Institute”) was a very special honor. Getting to know so many people from different countries and cultures broadened my horizon immensely. I will try my best to stay in contact.

A special thanks goes to Monika, who puts so much effort in to keep the atmosphere in the office as pleasant as it is. The fun little side-events and social gatherings she plans become the highlights of the year.

To be able to consider my family also my friends and my friends also my family is sincerely one of the greatest gifts I could have ever received. I am forever grateful for the people in my life and cannot thank them enough for everything. Especially, I want to thank Lisa for her constant support and all the joy she brings to my life.
Abstract

Many challenging problems (in mechanics, thermodynamics, electrodynamics etc.) are modelled by partial differential equations (PDEs). Since the middle of the 20th century Finite Element Analysis (FEA) is one of the most effective tools to solve those equations. More precisely, the solution of a PDE gets approximated by a linear combination of relatively simple basis functions.

However, it turns out to be beneficial to use more sophisticated basis functions, i.e. the same ones that are used to describe the geometric domain on which the PDEs are defined. This concept, which is known as Isogeometric Analysis (IgA) has the advantage that geometric models can be designed, analyzed and modified within the same dataset.

Unfortunately, the industrial standards in the computer aided design (CAD) technology do not allow for a direct application of the IgA framework to the designed models. The given boundary represented CAD data first need to be converted into parameterized volumetric objects. A common practice to achieve this goal is to first decompose the object into simpler sub-solids and to obtain their volumetric representation in a subsequent parameterization step.

The subdivision of a boundary represented solid object into a collection of topological cuboids has been denoted as the Isogeometric Segmentation Problem in [1]. A semi-automatic procedure to decompose a given input solid, which is considered to be contractible, free of double edges and to possess a 3-vertex-connected edge-graph into topological cuboids, has been provided in a series of articles [1, 2, 3]. The focus of this thesis is a number of modifications and ex-
tensions of the original algorithm that will improve the quality of the obtained output shapes.

The method consists of two main steps. First, we split the object recursively into smaller ones, until we obtain a decomposition into sufficiently simple sub-solids. By introducing a cutting loop on the solid’s boundary we indicate the most suitable possibility to decompose the solid. The actual split is performed by the construction of a cutting surface, which needs to fulfill several criteria. Most importantly, the boundary of the cutting surface must interpolate the cutting loop, while otherwise there are no intersections between the cutting surface and the boundary of the solid. In order to find a suitable cutting surface that stays within the solid’s boundaries, we first construct an implicit guiding surface, which is confined to the solid’s interior and approximate it in a subsequent least squares fitting step. Thus the constructed parameterized cutting surface is valid.

Additionally, we perform a face pre-segmentation on the input solid before starting the recursive decomposition. Inserting artificial edges in carefully chosen positions allows for more suitable cuts and eventually improves the shapes of the resulting sub-solids.

In the second step we use midpoint subdivision to further decompose the obtained sub-solids into topological cuboids. While this technique is straightforward for convex, polyhedral domains, its application to general shapes is very challenging. We present a method, based on the subdivision of a simple template domain and the construction of a regular mapping between the template and the solid, to obtain the desired collection of topological cuboids.

As a byproduct of constructing the template mapping one simultaneously obtains a volumetric parameterization of the sub-solid. Thus our method is able to semi-automatically convert given boundary represented CAD data into IgA-suitable volumetric models.
Zusammenfassung


Wie sich jedoch herausstellte, bieten anspruchsvollere Basisfunktionen durchaus Vorteile. Im Bereich der Isogeometric Analysis (IgA) verwendet man die selben Basisfunktionen, mit welchen das geometrische Gebiet beschrieben wird, auch für die Annäherung der Lösung der Differentialgleichung. Dieses Konzept ermöglicht es, geometrische Objekte mit nur einem Datensatz zu modellieren, zu analysieren und zu modifizieren.


Die Artikel [1, 2, 3] behandeln die Segmentierung eines Genus-0 Objektes, dessen Kantengraph keine Doppelkanten enthält und 3-zusammenhängend ist.
Wir behandeln diverse Änderungen und Erweiterungen des beschriebenen Algorithmus, welche zur Verbesserung der Qualität der Teilbereiche führen.


Um den Einsatz von stark gekrümmten Schnittflächen zu reduzieren, führen wir eine Vor-Segmentierung der Facetten des gegebenen Objektes durch. Strategisch platzierte Hilfskanten erlauben die Auswahl von besser geeigneten Schnitt-Positionen, was meistens zu einfacheren Schnittflächen führt.

Im zweiten Schritt der Segmentierung verwenden wir eine Mittelpunkt-Unterteilung (Midpoint Subdivision), um die eben konstruierten Teilbereiche weiter in topologische Quader zu zerlegen. Während die Mittelpunkt-Unterteilung für konvexe Polyeder keinerlei Probleme verursacht, ist die Anwendung auf allgemeine Objekte ziemlich schwierig. Wir machen uns die einfache Anwendung auf konvexe Polyeder zunutze und führen eine neue, auf Templates basierende Methode ein. Hierfür wird zunächst eine bijektive Abbildung vom zu zerschneidenden Objekt auf ein topologisch äquivalentes, konvexes Polyeder konstruiert. Anschließend wird sie dafür benutzt, um die Zerlegung vom Template auf das Objekt zu übertragen.

Darüber hinaus dient die soeben konstruierte bijektive Abbildung gleichzeitig als volumetrische Parametrisierung des Objektes. Unsere neue Methode kann also als halb-automatisches Konvertierungsverfahren eingesetzt werden, um gegebene CAD Daten in IgA-kompatible zu übersetzen.
# Contents

1 Introduction ................................................................. 1

2 The extended segmentation algorithm .................................. 5
   2.1 Preliminaries ......................................................... 5
   2.2 The algorithm ....................................................... 8

3 The construction of cutting surfaces .................................... 13
   3.1 Notation ............................................................... 13
   3.2 Implicit guiding surface ............................................ 18
   3.3 Parameterization of the guiding surface ......................... 24
      3.3.1 Parameterization of a guiding curve ....................... 24
      3.3.2 Parameterization of a guiding surface ..................... 26
   3.4 Results ............................................................... 31
      3.4.1 Cutting curves ................................................ 31
      3.4.2 Cutting surfaces .............................................. 35

4 Face pre-segmentation .................................................... 43
   4.1 Subdivide a domain using the medial axis transform ........... 44
   4.2 Subdivide a domain using a distance function .................. 46

5 Midpoint subdivision suitable solids (MS$^3$) ....................... 49
   5.1 Existence of a valency-reducing cutting loop ................. 50
   5.2 Template MS$^3$ ..................................................... 56
## CONTENTS

6 Midpoint subdivision 59
  6.1 Motivation ........................................ 59
  6.2 Midpoint subdivision of a convex polytope .......... 61
  6.3 Planar midpoint subdivision of a curved polygon .... 63
  6.4 Midpoint subdivision of general MS$^3$ ............ 66
    6.4.1 Multi-patch structure .......................... 68
    6.4.2 Template mapping problem ...................... 70

7 Examples 73
  7.1 Regular dodecahedron ............................... 73
  7.2 Bi-slotted cube ................................... 75
  7.3 Slotted quarter-pipe ............................... 76
  7.4 Hammer ............................................ 78

8 Conclusion and outlook 85
Chapter 1

Introduction

Since several decades, Finite Element Analysis (FEA) is a well-established field of research and of major importance in a variety of industrial applications. The accuracy of the approximated solution depends strongly on the number and quality of the used finite elements. Generally speaking, a finer subdivision leads to better results, but the complexity of the problem increases with the number of basis functions.

In 2005 Hughes et al. [4] introduced the concept of Isogeometric Analysis (IgA) as a modification of the classical FEA. The idea is to use more sophisticated but fewer basis functions to improve the quality of the solution. Numerous applications are based on domains constructed with the help of techniques from Computer Aided Design (CAD). Since NURBS curves and surfaces are a common standard within the CAD technology, using the same spline spaces also for approximating the solution seems like a natural choice. The survey article [5], written by two eminent experts in this field, gives an overview of the current state-of-the-art.

IgA has drawn the attention of many researchers due to its very promising perspective. Together with the many benefits that IgA yields, there are also new challenges arising. Besides the more involved integrals that occur due to the more complicated basis functions and the need for new methods for fast matrix assembly, there is also the issue of converting the given CAD data into IgA-suitable
ones. The geometric modelling community widely uses boundary representation for their objects but in order to solve PDEs using IgA one needs parameterized volumetric spline patches instead.

These parameterizations may be classified into single- and multi-patch representations. The first class is conceptually simpler but has limited flexibility. The construction of single patch spline models from boundary data has been addressed by numerous publications. We briefly mention some of them: Creating a regular single-patch domain parameterization from boundary data is discussed in [6]. A method for volumetric parameterization and trivariate B-spline fitting using harmonic mappings has been described in [7] for objects of cylindrical topology. Harmonic functions are particularly useful, since they can be employed to guarantee injectivity [8] and yield high-quality parameterizations [9]. Given a representation as a tetrahedral mesh, a B-spline volume representing a solid is constructed by first computing parameterizations via harmonic functions and then performing a spline approximation [10]. Single-patch spline representations for swept volumes and generalized cylinders were constructed in [11, 7].

The limited flexibility of single-patch NURBS parameterizations can be enlarged by employing more advanced spline functions. Several publications address the computation of parameterizations defined by trivariate T-splines. These combine optimization methods with refinement strategies and techniques from traditional mesh generation [12, 13]. The use of polycube domains allows to deal with objects of non-zero topological genus [14], such as solids defined by Boolean operations [15]. The preservation of boundary features is studied in [16]. A polycube-type construction for cubic polynomial splines over hierarchical T-meshes (PHT-splines) is presented in [17].

Although it has some benefits, the use of a single patch imposes severe constraints on the topology of the domain. Algorithms for creating multi-patch representations that provide increased flexibility, are therefore of vital interest. Typically, such algorithms consist of two steps: First the domain is subdivided into a collection of topological cuboids. Second, one constructs a trivariate spline
parameterization for each of these blocks. In order to benefit from the potential advantages of isogeometric analysis, one should construct segmentations into relatively few cuboidal patches. This is quite different from the usual approach to hexahedral mesh generation, which has been studied in the context of the classical finite element method, see e.g. [18] and the references cited therein.

A geometric modeling framework is established in [19]. Methods based on polycubes apply mesh deformation techniques to a uniform grid [20, 17]. While this gives satisfying results in many cases, it does not preserve the patch structure of a given CAD model. Alternatively, the segmentation into topological cuboids can be constructed by considering pants decomposition of the boundary surface [21], or by recursively applying splitting steps (isogeometric segmentation) that reduce the object’s topological complexity [1, 2, 22, 23]. Once the segmentation has been found, the multi-patch parameterization can be constructed by employing variational techniques and Coons interpolation [24, 25]. A combinatorial approach to planar multi-patch domains, which is based on a complete enumeration of the possible patch layouts, has been described recently in [26]. Suitable spline spaces for multi-patch domains have been analyzed in [27, 28].

Hybrid approaches, which combine an exact representation of the solid’s boundary with a tetrahedral mesh in the interior via intermediate patches defined by offsetting, maintain the compatibility with the initial representation [29].

This thesis contributes to the splitting-based approach to isogeometric segmentation [23]. Using cutting surfaces one recursively splits a solid into topologically smaller ones, until the resulting sub-domains belong to a predefined class of base solids. In each step the cutting surface is obtained from a cutting loop, which is a cycle of curves on the solid’s boundary surface. Although the proposed method already returns valid and promising results, there are still challenging problems and possible improvements to investigate.

While the selection of the cutting loop and the construction of its curve segments is now well understood, the actual construction of the cutting surface has not yet been discussed in detail. Furthermore, strongly non-convex solids are
not only very difficult to segment, the quality of the resulting base-solids is in
general far from ideal. Another handicap of the approach is the use of only one
cutting surface in each segmentation step. For some domains it is beneficial to
use multiple cuts at once in order to improve the quality of the output.

The remaining thesis is structured as follows. In the next chapter we introduce
the extended segmentation algorithm and some important notations. Chapter 3
focuses on the robust construction of a cutting surface, which is an essential in-
gredient of the extended segmentation algorithm. Given a three-dimensional solid
in boundary representation as a collection of trimmed NURBS surfaces, and a
cutting loop, we generate a representation of the cutting surface as a trimmed
NURBS surface patch. Its boundary interpolates the given cutting loop, but its
interior must not intersect the boundary of the solid. In Chapter 4 we investigate
two methods of pre-processing the given input solid, in order to obtain better
results. The main idea is to add artificial edges to the solid, such that the number
of possible cutting loops increases. This usually leads to an additional split into
two simpler sub-solids, which are easier to decompose further and thus returning
output domains with improved shapes. The article [2] examines the elimination
of non-convex edges, by choosing appropriate cutting loops. In a similar man-
ner, we remove high-valent vertices of the solid in Chapter 5. Consequently, we
obtain a catalog of base solids, with only tri-valent vertices. Midpoint subdivi-
sion is a method to decompose those base solids into a collection of topological
cuboids. This is studied in Chapter 6, where we first explain the basic concept for
decomposing a convex polyhedron, and then propose a novel, template mapping
approach to perform a subdivision on non-convex domains with non-planar faces.
The latter is simultaneously useful for obtaining a volumetric multi-patch NURBS
parameterization of the base solids. Before we finally summarize our results and
discuss open problems in Chapter 8 we present several numerical examples in
Chapter 7.
Chapter 2

The extended segmentation algorithm

The first step of converting boundary represented CAD data into volumetric NURBS patches is to subdivide the solid object into a collection of topological cuboids, which can then be parameterized by tensor product NURBS patches in the second step. The Isogeometric Segmentation Pipeline [1, 2, 23] provides a framework that generates such a subdivision via a recursive algorithm, with the focus on creating a small number of entities. This thesis discusses several extensions to the original algorithm, which improve the quality of the output shapes.

2.1 Preliminaries

In this section we discuss some important notations and concepts. The initial solid is given in boundary representation by its

- faces: trimmed NURBS surface patches,
- edges: intersection of two adjacent faces and
- vertices: intersection of neighboring edges.
Figure 2.1: A solid and its associated edge graph.

Removing the geometric information of a solid, and only considering the topological connectivity leads to the solid’s edge graph (cf. Figure 2.1). By definition any given solid has a unique edge graph, while on the contrary, to any given edge graph there are infinitely many different realizations as solids.

There are several assumptions a solid needs to fulfill in order to be suitable for the segmentation algorithm.

- The solid is contractible, i.e., homeomorphic to a point. More precisely, the solid has genus 0 and does not contain hollow regions (see [22]).
- The solid is free of double edges. Any two vertices are connected by at most one edge.
- The solid possesses a three-vertex-connected edge graph. Specifically that means removing any two vertices (and adjacent edges) of the graph results in a sub-graph, which is still connected.
- The angles formed by neighboring edges and faces lie between 0 and $2\pi$. Hence we exclude solids with “sharp” edges.
2.1. PRELIMINARIES

Figure 2.2: Instances of convex, non-convex and auxiliary edges. Note that the two normal vectors in the right picture actually coincide.

The contractibility of the solid implies that the solid’s edge graph is planar. More precisely the vertices can be arranged in the plane, such that the edges do not intersect each other, except in vertices where they are supposed to meet.

We recall the definition of convex and non-convex edges from [2]. Let $p$ be an interior point on an edge between the two faces $f_1$ and $f_2$. The normal plane of the edge in the point $p$ intersects the two faces in two planar curve segments and it contains the two outward facing normal vectors $n_1$ and $n_2$. We denote by $t_1$ and $t_2$ the two tangent vectors of the respective curve segments in $p$, oriented in such a way, that they point away from the face. We consider the two convex cones spanned by $t_1, n_1$ and $t_2, n_2$ respectively. The edge is said to be convex in the point $p$ if the intersection of the two cones is $p$. An edge is called convex if it is convex in every interior point. Otherwise it is called non-convex. Figure 2.2 shows three examples of edges. Note that in those examples, the non-convex edges are actually non-convex in every point. This is not necessarily the case in general.

While the convexity of an edge depends on the geometric realization of the solid, the following classifications are also relevant on the edge graph. An existing vertex is defined by the intersection of at least three boundary faces. Additionally
we consider *auxiliary vertices*, which can be introduced anywhere in the interior of an existing edge of the solid. Similarly, *existing edges* correspond to non-empty intersections of adjacent boundary faces. In addition, we introduce *auxiliary edges* connecting existing or auxiliary vertices. Its end points must not belong to the same boundary edge. These edges split faces into two smaller ones that meet each other at an angle of $\pi$. Hence their two normal vectors coincide on every point of the auxiliary edge. This constitutes the limit case between the edge being convex or non-convex in that point. Therefore an auxiliary edge is by definition a non-convex one (cf. Figure 2.2).

It should be noted that the geometric realization of auxiliary edges and vertices requires special care, in order to avoid intersections and singularities. This is discussed in more detail in Chapter 3.

### 2.2 The algorithm

The segmentation procedure of the extended pipeline is summarized in the Extended Segmentation Algorithm (ESA, see page 9). Given a three-dimensional solid $S$ that fulfills the assumptions stated above, it recursively performs splitting operations, which are described in the algorithm for Cutting Loop-based Splitting (CLS, see page 10), until the solid is decomposed into a collection of solids that are sufficiently simple and hence suitable for midpoint subdivision. We denote those solids, which are still given in boundary representation, as *midpoint subdivision suitable solids* ($MS^3$). A more precise definition of $MS^3$ will be provided in Chapter 5.

Compared to the original method (see [23]), the procedure incorporates the following additional features:

- We perform a face pre-segmentation. The faces of the initial solid are subdivided if their shape appears to be unsuitable for the segmentation algorithm. This is described in more detail in Chapter 4.
Algorithm ESA: Extended segmentation algorithm

Input : A contractible three-dimensional solid $S$.
Output: A decomposition into MS$^3$.

1 Initialize the stack with the face pre-segmented solid $S$.
2 while stack is not empty do
3    Take the next solid from the stack.
4    if non-convex edges are present then
5        call CLS to remove at least one of them and put the two resulting solids on the stack.
6    else if high-valent vertices are present then
7        call CLS to remove at least one of them and put the two resulting solids on the stack.
8    else if the solid is not an MS$^3$ then
9        call CLS to generate two sub-solids with fewer vertices and put them on the stack.
10   else
11      output the solid.

- The extended procedure makes systematic use of midpoint subdivision. This type of decomposition, which is available for any solid with only tri-valent vertices, generates one cuboidal sub-domain for each vertex. This contribution is based on two ingredients: First, we need a catalog of MS$^3$ and a way to obtain them via the ESA, which is described in Chapter 5. Second, we need a method for actually performing the subdivision into cuboidal sub-domains, which will be simultaneously useful for creating the individual trivariate NURBS parameterizations. In Chapter 6 this will be described in more detail.

- The segmentation algorithm of [2] decomposed a solid into a collection of sufficiently simple sub-solids. Those were called base solids and categorized in three different types: topological cuboids, tetrahedra and prisms (with three-sided base surfaces). All of those original base solids fall under the category of MS$^3$, which serve as base solids in the ESA.
CHAPTER 2. THE EXTENDED SEGMENTATION ALGORITHM

**Algorithm CLS:** Cutting Loop-based Splitting

**Input:** A contractible three-dimensional solid $S$.

**Output:** Two sub-solids $S_1$ and $S_2$.

1. Find all possible cutting loops in the edge graph.
2. Rate each loop by a cost function and select the best one.
3. Construct auxiliary vertices and edges on the solid.
4. Use the cutting loop to construct a cutting surface.
5. Split the solid along the cutting surface.

The algorithm for Cutting Loop-based Splitting (CLS) was first introduced in [1]. We briefly recall some important notations and concepts:

A *cutting loop* is a cyclical path in the solid’s edge graph that connects existing or auxiliary vertices by existing or auxiliary edges, such that two different edges do not belong to the same face. Its physical realization on the solid is called cutting loop as well.

A cutting loop is said to be valid if it separates the edge graph into two sub-graphs. According to Proposition 5 of [1] this is the case, if the cutting loop contains at least 3 vertices and if any two non-neighboring vertices do not share a face. Intuitively speaking, when walking along the cutting loop, one must never visit a face more than once.

In CLS we first use Yen’s algorithm [30] to find all available cutting loops in the edge graph, which is possibly extended by auxiliary edges and vertices. This is done by considering all pairs of neighboring vertices in the edge graph and temporarily deleting the edge in between them. Yen’s algorithm then finds all possible loopless paths between the two vertices. Finally we insert the previously deleted edge back into the graph to close the loops.

Second, the quality of these loops is measured by evaluating a cost function, which takes combinatorial and geometric aspects into account. Finally, we select the best cutting loop with respect to the cost function, construct the cutting surface and split the solid, see Chapter 3. The main difference between the three
2.2. **THE ALGORITHM**

cases (ESA lines 5, 7 and 9) is the choice of the cutting loop. Since each case fulfills a different purpose, the used cutting loops need to fulfill different properties.

The next chapters provide further details regarding the ESA and the CLS. The first splitting operation (line 5 of the ESA) is covered by [2] and will not be discussed further throughout this thesis. Similarly, since the cost function is already well understood (see [1, 2]) we will focus on different challenges in this thesis.
Chapter 3

The construction of cutting surfaces

As mentioned in the previous chapter, we obtain a decomposition of an input solid into a collection of MS\(^3\) by the ESA (see page 9). During this process, the solid gets recursively split into two sub-solids by the CLS (see page 10).

In this chapter we will examine two main ingredients of the CLS, i.e. the robust construction of auxiliary edges and cutting surfaces. Since these two problems are closely related, we first introduce a notation that will allow us to discuss them both simultaneously. The results of this chapter were published in [31].

3.1 Notation

We state the problem for general dimension \(d\). The cases \(d = 2\) and \(d = 3\) correspond to the construction of an auxiliary edge and a cutting surface, respectively.

We consider the problem of decomposing a \(d\)-dimensional simply connected domain, which is given in boundary representation into two smaller simply connected domains for \(d = 2, 3\). Throughout this chapter, we will refer to the domain as \textit{solid}, independent of the dimension \(d\).

More precisely we are given a list of \(n\) facets

\[
F_i : \Omega_i \subset [0, 1]^{d-1} \rightarrow \mathbb{R}^d \quad \text{for} \quad i = 1, \ldots, n,
\]
such that $\bigcup_i F_i(\Omega_i)$ is the boundary of a solid object $S$ in $\mathbb{R}^d$. We distinguish between a parameterized facet $F_i$ and its geometric locus $F_i = F_i(\Omega_i)$. For simplicity we use the notion facet for both of them.

In the two-dimensional case ($d = 2$), the parameter domains $\Omega_i$ are intervals and the associated facets $F_i$ are planar curve segments. The solid $S$ is the planar domain that is bounded by these segments. For $d = 3$ the notion of a solid coincides with one in the previous chapter. The facets correspond to the solid’s faces (see Figure 3.1). Note that the boundary faces of a three-dimensional solid are given as trimmed surface patches. The trimming loop in the parameter domain of any face can be considered as the boundary of a planar solid (see Figure 3.1, right).

The non-empty intersections $F_i \cap F_j \subset \mathbb{R}^d$ for $i \neq j$ will be called ridges. In the planar case ($d = 2$), the ridges are the start and end points of the boundary curves and therefore vertices. The ridges can be either edges or vertices for dimension $d = 3$, see Figure 3.1.

In order to decompose the given solid into two smaller ones, we need to construct two new lists of facets $F^\ell_i(\Omega^\ell_i)$, $\ell = 1, 2$, with the following properties:

---

**Figure 3.1**: Left: A 2D solid with five facets (curve segments) and one highlighted ridge (vertex). Right: A 3D solid with five facets (trimmed surface patches) and two highlighted ridges (vertex and edge).
3.1. NOTATION

- The facets are represented by parameterizations, i.e.,
  \[ F^\ell_i : \Omega^\ell_i \subset [0, 1]^{d-1} \to \mathbb{R}^d \] for \( i = 1, \ldots, n^\ell \).

- Each list of facets \( \bigcup_i F^\ell_i \) forms the boundary of a simply connected solid object \( S^\ell \) in \( \mathbb{R}^d \).

- The input solid \( S \) is the disjoint union of the two newly generated ones, i.e., it satisfies \( S = S^1 \cup S^2 \) and \( (S^1) \cap (S^2)^\circ = \emptyset \).

Intuitively speaking, we are cutting the solid \( S \) into two smaller solids \( S^1 \) and \( S^2 \).

In addition to the properties listed above, we assume that exactly one facet needs to be generated in order to decompose the given solid. More precisely, exactly one facet of each solid \( S^\ell \) is not contained in one of the facets of the given solid \( S \). The newly generated facet will be called the cutting facet \( C \). It is also called cutting curve (or auxiliary edge) and cutting surface for dimension \( d = 2 \) and \( d = 3 \), respectively.

Without loss of generality, the first facet of the two sub-solids is assumed to be the cutting one, i.e., it satisfies

\[ \Omega^1_1 = \Omega^2_1 \quad \text{and} \quad F^1_1 = F^2_1 \]

while the remaining facets fulfill

\[ \exists k \quad F^\ell_i(\Omega^\ell_k) \subset F_k(\Omega_k) \quad \text{for} \quad i > 1. \]

We will construct the cutting facet \( C = C(\Omega) \),

\[ C : \Omega \to \mathbb{R}^d, \quad \text{where} \quad \Omega = \Omega^1_1 = \Omega^2_1 \quad \text{and} \quad C = F^1_1 = F^2_1 \]

from given cutting data \( \mathcal{D} = (L, \vec{t}) \), which is a pair consisting of boundary ridges \( L \) and associated tangent vectors \( \vec{t} : L \to \mathbb{R}^d \), see Figures 3.2 and 3.3.

The boundary ridges specify the boundary of the cutting facet:
In the planar case we use two distinct vertices on the boundary. Those can be either existing or auxiliary vertices, which have been introduced in Chapter 2 (cf. Figure 3.2).

A closed loop $L$ of at least three curve segments on the boundary, meeting in vertices, is required for dimension $d = 3$, see Figure 3.3. The loop must not intersect itself. Again, we use either existing or auxiliary edges and vertices.

The tangent vectors $\vec{t}$ control the tangent vector respective the tangent plane of the cutting facet along the boundary. For each point $x \in L$, we specify a tangent vector $\vec{t}(x)$ that points to the interior of the solid $S$. For dimension $d = 3$, it is required that the vectors vary smoothly along the curve segments and define a
consistent tangent plane at the vertices.

It should be noted that the cutting data may be required to satisfy additional assumptions, which are needed by the ESA. Since a cutting loop (on a three-dimensional solid) is only valid, if two non-neighboring vertices do not belong to the same face (cf. Chapter 2), we need to restrict $L$ accordingly in the case of $d = 3$. For $d = 2$, a similar restriction occurs. Here the two points of $L$ must not lie on the same boundary segment.

The cutting facet $C$ has to satisfy the following conditions:

- **Boundary interpolation:** $C(\partial \Omega) = L$,
- **Boundary tangent interpolation:** $\vec{t}(C(u))$ is contained in the tangent plane of $C$ at $C(u)$ for $u \in \partial \Omega$,
- **No collision with the boundary:** $C(\Omega^o) \subset S^o$,
- **Regularity:** $C$ is injective and regular (in particular, it does not intersect itself).

In the case $d = 2$, the cutting facet $C$ is a curve segment that connects two given points on $\bigcup_i F_i$, while staying inside the domain bounded by the facets $F_i$, see Figure 3.2 (left). For dimension $d = 3$, the cutting facet is a surface patch whose boundary coincides with $L$. Again $C$ stays inside the domain bounded by the facets. In every point $x$ on $\partial C$, the tangent plane of $C$ contains the given vector $\vec{t}(x)$, see Figure 3.3 (left).

Given a $d$-dimensional solid given by its boundary facets $F_i(\Omega_i)$, and cutting data $D$, we present a novel method that generates a cutting facet $C$. For the planar case, this problem has already been dealt with in [3]. In this paper, the authors proposed a method based on the minimization of certain penalty functionals that led to satisfying results. In principle, it would be possible to adapt this method to the three-dimensional case. However, the computational effort is rather high. Alternatively, we introduce the Algorithm CFG (see page 18), which is simultaneously suitable for dimensions $d = 2$ and $d = 3$. 
Algorithm CFG: Cutting Facet Generation

| Input  | A solid $S$ and cutting data $D$. |
| Output | A parameterized cutting facet. |

1. Find an implicit guiding curve / surface
2. Parameterize the guiding curve / surface

Depending on the dimension $d$, we are looking either for a curve or for a surface. For simplicity, we will always refer to surfaces, independently of the dimension $d$. Similarly, we will always refer to $L$ as the cutting loop, even if this is not accurate for $d = 2$.

### 3.2 Implicit guiding surface

The cutting loop $L$ separates the boundary $\partial S$ into two simply connected, disjoint pieces $\overline{b}$ and $\underline{b}$, i.e., $\partial S \setminus L = \overline{b} \cup \underline{b}$. (We do not consider solids whose boundary is not split into two components by a cutting loop, such as the horned sphere. These solids are not likely to occur in CAD applications.) We obtain the guiding surface with the help of a level set function that takes positive values on $\overline{b}$, negative values on $\underline{b}$ and vanishes on the loop $L$.

We need to transform the given cutting data into suitable boundary data for the level set function. To do so, we compute a normal vector $\vec{n}(x)$ for each point $x \in L$.

In the planar case, we simply rotate the vector $\vec{t}(x)$ by $\pi/2$ towards $\overline{b}$, see Figure 3.4, left. In the case $d = 3$, we compute the tangent plane at all points $x \in L$ and pick the normal vector that points towards $\overline{b}$, see Figure 3.4, right. Note that we assumed a consistent distribution of the tangent vectors, thereby ensuring the existence of a tangent plane at all points of the cutting loop (including vertices).

The guiding surface is defined by a level set function that solves the cutting
3.2. IMPLICIT GUIDING SURFACE

Figure 3.4: Boundary data for the construction of the guiding surface for $d = 2$ (left) and $d = 3$ (right).

**Problem:** Find $f : \Omega \subset \mathbb{R}^d \to \mathbb{R}$ such that

$$
\begin{cases}
    f(x) = 0, & \forall x \in L, \\
    \nabla f(x) = \vec{n}(x), & \forall x \in L, \\
    f(x) > 0, & \forall x \in \bar{b}, \\
    f(x) < 0, & \forall x \in \bar{b}.
\end{cases}
$$

(CP)

In order to make this accessible to a numerical solution, we perform a discretization. We choose $d$-variate tensor-product B-splines $(\phi_i)_{i \in I}$, where $I$ is some index set, which are defined in the bounding box of the solid $S$. More precisely, we use cubic splines with open knot vectors and uniform inner knots, the number of which is specified by the user (typically either 3 or 7 knots per direction in our examples). The discretized level set function takes the form

$$f^*(x) = \sum_{i \in I} \phi_i(x)c_i$$

(3.1)
with unknown scalar coefficients $c_i$. They are found by solving the discretized cutting problem,

$$
\begin{aligned}
&f^*(x) \approx 0 \quad \forall x \in L^* \\
&\nabla f^*(x) \approx \vec{n}(x) \quad \forall x \in L^* \\
&f^*(x) \geq \delta \quad \forall x \in \overline{b}^* \\
&f^*(x) \leq -\delta \quad \forall x \in \overline{b}^*.
\end{aligned}
$$

(DCP)

The influence of the user-specified discretization parameter $\delta > 0$ will be studied later in Section 3.4.

The discretized sets $L^*$, $\overline{b}^*$ and $\overline{b}^*$ are created by sampling points on the curves and surfaces, respectively. We exclude a small strip around the loop $L$ when generating $\overline{b}^*$ and $\overline{b}^*$ in order to increase the numerical stability of the method.

Neither (CP) nor (DCP) have unique solutions. Therefore we introduce an optimization problem that allows us to identify the optimal solution with respect to a combination of certain quality measures,

$$
F(c) = \lambda_{\text{int}} F_{\text{int}}(c) + \lambda_{\text{tan}} F_{\text{tan}}(c) + \lambda_{\text{reg}} F_{\text{reg}}(c) \rightarrow \min
$$

where $c = (c_i)_{i \in I}$ is the vector of coefficients, see (3.1), and the parameters $\lambda_{\text{int}}$, $\lambda_{\text{tan}}$, $\lambda_{\text{reg}}$ are positive weights. They control the influence of the different contributions, cf. Section 3.4.

The first quality measure

$$
F_{\text{int}}(c) = \sum_{x \in L^*} f^*(x)^2
$$

represents the approximate interpolation constraint for the cutting loop, and the second one

$$
F_{\text{tan}}(c) = \sum_{x \in L^*} \| \nabla f^*(x) - \vec{n}(x) \|^2
$$

is used to constrain the normal vectors (and hence the tangent planes of the level set surface) along the cutting loop. Additionally we regularize the solution by
3.2. IMPLICIT GUIDING SURFACE

considering the functional

\[ F_{\text{reg}}(c) = \int_{[0,1]^d} \sum_{i,j=1}^{d} \left( \frac{\partial^2}{\partial x_i \partial x_j} f^*(x) \right)^2 \, dx, \tag{3.3} \]

that is related to the curvature of the level set surface.

We obtain the unknown coefficients by solving the \textit{regularized + discretized cutting problem}

\[
\begin{aligned}
F(c) \to \min \\
\text{subject to } f^*(x) \geq \delta & \quad \forall x \in \overline{D} \\
\text{and } f^*(x) \leq -\delta & \quad \forall x \in \overline{B}
\end{aligned} \tag{RDCP}
\]

\textbf{Lemma 1.} The regularized + discretized cutting problem RDCP has a unique solution if the set of feasible points (which is defined by the inequality constraints) is non-empty.

\textit{Proof.} The objective function is convex if the weights \( \lambda_{\text{int}}, \lambda_{\text{tan}}, \lambda_{\text{reg}} \) are all positive, see Proposition 1 of [32]. Thus we obtain a convex optimization problem, since the inequality constraints are linear. \hfill \Box

Minimization of the functional (3.2) requires the evaluation of the integral (3.3) multiple times. For a fast computation we exploit the tensor-product structure of the spline function \( f^* \) in order to precompute univariate integrals of B-spline basis functions. Let us consider the two-variate case:

\[
F_{\text{reg}}(c) = \int_{[0,1]^2} \sum_{i,j=1}^{2} \left( \frac{\partial^2}{\partial x_i \partial x_j} f^*(u,v) \right)^2 \, d(u,v)
\]

\[
= \sum_{i,j=1}^{2} \int_{[0,1]^2} \left( \sum_k c_k \phi_k^{(ij)}(u,v) \right)^2 \, d(u,v)
\]
CHAPTER 3. THE CONSTRUCTION OF CUTTING SURFACES

where we use the abbreviation $\phi_k^{(ij)}(u, v) = \frac{\partial^2}{\partial x_i \partial x_j} \phi_k(u, v)$.

With the vector representations $\mathbf{c} = (c_k)_i$ and $\phi^{(ij)}(u, v) = (\phi_k^{(ij)}(u, v))_k$ and the matrix representation $\Phi^{(ij)}(u, v) = \phi^{(ij)}(u, v) \cdot (\phi^{(ij)})^T(u, v)$ we can simplify further to

$$F_{\text{reg}}(\mathbf{c}) = \sum_{i,j=1}^{2} \int \int_{[0,1]^2} \left( \mathbf{c}^T \cdot \phi^{(ij)}(u, v) \right)^2 d(u, v)$$

$$= \sum_{i,j=1}^{2} \mathbf{c}^T \cdot \left( \int \int_{[0,1]^2} \Phi^{(ij)}(u, v) d(u, v) \right) \cdot \mathbf{c},$$

Furthermore any entry of the matrix $\tilde{\Phi}^{(ij)}(u, v)$ is of the form

$$\tilde{\Phi}^{(ij)}_{m,n}(u, v) = \int \int_{[0,1]^2} \phi^{(ij)}_m(u, v) \cdot \phi^{(ij)}_n(u, v)$$

$$= \left( \int_0^1 \phi^{(ij)}_m(u) \cdot \phi^{(ij)}_n(u) \ du \right) \cdot \left( \int_0^1 \phi^{(ij)}_m(v) \cdot \phi^{(ij)}_n(v) \ dv \right). \quad (3.4)$$

Finally we can write the regularity term as

$$F_{\text{reg}}(\mathbf{c}) = \mathbf{c}^T \cdot \left( \sum_{i,j=1}^{2} \tilde{\Phi}^{(ij)}(u, v) \right) \cdot \mathbf{c}, \quad (3.5)$$

where each matrix entry can be precomputed as the product of univariate integrals as in (3.4). Any further evaluation of (3.3) can then be performed as a fast matrix-vector multiplication. Analogously one can find a similar representation as (3.5) for the three-dimensional case, where the matrix entries consist of products of three univariate integrals.

In our examples, we solved RDCP using the fmincon function of Matlab\textsuperscript{TM}. Typical problems, where the number of spline coefficients varies between 100 (for curves) to 1000 (for surfaces), can be solved within a few seconds or minutes on
3.2. IMPLICIT GUIDING SURFACE

Figure 3.5: Guiding curves and surfaces, represented as zero level sets of cubic spline functions.

standard hardware.

An empty set of feasible points results if the number of knots, which is used to define the tensor-product splines \((\phi_i)_{i \in I}\), is too small. We increase the number of knots by dyadic refinement until RDCP possesses solutions. Additionally we increase the number of knots if the errors

\[ f^*(x) \text{ and } ||\nabla f^*(x) - \hat{n}(x)||, \quad x \in L^*, \]

exceed a user-defined tolerance. Recall that the values \(f^*(x)/||\nabla f^*(x)||\) estimate the distance of the point \(x\) to the level set surface \(f^* = 0\). We may omit the denominator, since the gradients of \(f^*\) approximate unit normal vectors along the loop.

Figure 3.5 shows several instances of guiding curves and surfaces. All curves and surfaces are represented as zero level sets of cubic spline functions.

The zero level set of the function \(f^*\) guides the construction of the cutting surface. The latter surface is represented by a trimmed NURBS patch, the construction of which will be addressed in the next section. It should be noted that the guiding surface may have extraneous components as in Figure 3.6, which are not considered when creating the cutting surface. It is also possible to place auxiliary obstacles that influence the shape of the guiding surface.
3.3 Parameterization of the guiding surface

The parameterization step uses different approaches for dimensions two and three. We consider both cases independently.

3.3.1 Parameterization of a guiding curve

First we consider the curve case, i.e., dimension $d = 2$. A variety of parameterization techniques for implicitly defined curve segments exists. We chose a two-step procedure that first approximates the implicit curve by a polygon and uses a subsequent least squares fitting algorithm to obtain the final parameterized spline curve segment.

Step 1 - Marching Algorithm to construct a polygon First we approximate the implicit guiding curve by a polygon, whose start and end points are given by the cutting data. From a given point $p_k$ on the curve we predict the next point $\tilde{p}_{k+1}$, by walking a given step size $s$ along the curve’s tangent in $p$. The
3.3. PARAMETERIZATION OF THE GUIDING SURFACE

The next point \( p_{k+1} \) on the curve is the orthogonal projection of the prediction onto the curve (cf. Figure 3.7, center).

The start tangent vector, which is part of the cutting data, is used to predict the first new point (see Figure 3.7, left). We terminate this procedure, when a computed curve point is within the stepsize \( s \) to the end point (cf. Figure 3.7, right). Finally, we obtain a polygon \( (p_0, p_1, \ldots, p_{n-1}, p_n) \), with \( p_0 = v_S \) and \( p_n = v_E \), which approximates the guiding curve.

Step 2 - Spline curve fitting  
Since the computed polygon is an approximation of the curve, it can be used as data points for a least squares fitting algorithm. We are looking for a spline curve \( c(t) \), such that \( c(t_k) = p_k \), for all \( k = 0, \ldots, n \). The degree and knot vector are chosen in advance and the parameter values are computed by

\[
t_k = \frac{\sum_{i=1}^{k} \| p_i - p_{i-1} \|}{\sum_{j=1}^{n} \| p_j - p_{j-1} \|}.
\]

This leads to a linear system of equations, whose solution consists of the control points and thereby defines the spline curve.

In order to define the end points and boundary tangents exactly, we adjust the first two and the last two control points of the spline curve according to the given cutting data.

The accuracy of the final result depends on two main factors. First, on the sampling density, which is controlled by the stepsize \( s \) of the marching algorithm and second, on the number of control points, which is determined by the choice of the spline space. Figure 3.8 shows an example, where the marching algorithm fails, due to the unsuitable choice of the step size. The guiding curve is forced around the obstacles in a certain way, such that it almost touches itself at one point. The center picture of Figure 3.8 shows the predicted next point to be very close to the wrong part of the curve. Thus the polygon skips the middle part of the guiding curve and is therefore not a valid approximation. This problem can be resolved by choosing a sufficiently small step size.
3.3.2 Parameterization of a guiding surface

The second case ($d = 3$), where we need to find an approximate parameterization of the guiding surface, is far more challenging to deal with. In fact, it requires the solution of two non-trivial problems.

We construct an approximate parameterization of the guiding surface by a trimmed spline patch

$$p : \Omega \subset [0, 1]^2 \rightarrow \mathbb{R}^3,$$

$$(u, v) \mapsto \sum_i \psi_i(u, v)d_i.$$  \hspace{1cm} (3.6)

The parameterization is defined by tensor-product B-splines $\psi_i$, control points $d_i \in \mathbb{R}^3$ and a trimmed parameter domain $\Omega$. The spline patch has to satisfy two conditions:

1. Its boundary $p(\partial\Omega)$ interpolates the cutting loop $L$ and the tangent planes along the boundary contain the associated tangent vectors. In addition, the tangent vectors point to the interior of the surface patch.

2. The interior of the surface is an approximate parameterization of the guiding
Figure 3.8: Instance of the marching algorithm failing, due to a too large step size. The predicted point (green) is too close to the wrong branch of the curve. Thus the middle part of the curve is not approximated by the polygon.

surface. More precisely, the composition $f^* \circ p$ is close to zero in $\Omega$.

The patch is constructed in two steps.

**Step 1 - Construction of the trimmed parameter domain**  We choose the domain $\Omega$ as a planar (possibly curved) polygon, contained within the unit square, that mimics the shape of the cutting loop. This similarity will lead to better results in the second (parameterization) step.

We simply use the unit square $[0, 1]^2$ for four-sided cutting loops whenever this is appropriate, thereby avoiding trimming in this case. A more general polygon is used to define the domain if the square is not suitable.

More precisely, we choose this polygon such that the ratios of adjacent edge lengths are approximately equal to the ratios of the lengths of the corresponding curve segments, and the oriented angles between incoming and outgoing tangents at vertices are approximately the same and preserve the sign. Several techniques are used to find this polygon [33]:

- The first one simply projects the vertices of the cutting loop into a plane and connects them by straight lines. This simple method already leads to
good results in most cases, since the ESA generally uses relatively simple cutting loops (in particular near-planar ones).

- Some solids however do not allow for (near-) planar cutting loops, and therefore a projection into a plane would not result in a valid polygon. In this case, a second technique is used that relies on quadratic optimization:

\[
\arg\min_x (Ax - b)^T (Ax - b) \quad \text{subject to} \quad Cx = d \quad (3.7)
\]

where

\[
A = \text{diag}(\frac{1}{\ell_i}), \quad b = (1, \ldots, 1)^T, \quad d = (0, 0)^T, \quad \text{and}
\]

\[
C = \begin{pmatrix}
\cos(\alpha_1) & \cos(\alpha_1 + \alpha_2) & \cdots & \cos(\sum_{i=1}^n \alpha_i) \\
\sin(\alpha_1) & \sin(\alpha_1 + \alpha_2) & \cdots & \sin(\sum_{i=1}^n \alpha_i)
\end{pmatrix}.
\]

The values \(\ell_i\) are the lengths of the original cutting loop curves in 3d, while one obtains \(\alpha_i\) by scaling the tangent turning angles of the cutting loop, such that \(\sum_{i=1}^n \alpha_i = 2\pi\). As result of the optimization (3.7) one obtains a set of edge-lengths \(x_i\) that together with the angles \(\alpha_i\) form a closed planar polygon, which mimics the shape of the cutting loop.

- We explored another construction that is based on the minimization of the functional

\[
h = \sum_{i=1}^{n_v} \left( \omega_\kappa \int_0^1 r_{i,\kappa}^2 \, du + \omega_\alpha r_{i,\alpha}^2 + \omega_\ell \int_0^1 r_{i,\ell}^2 \, du \right), \quad (3.8)
\]

where \(\omega_\kappa\), \(\omega_\alpha\) and \(\omega_\ell\) are positive weights that regulate the influence of the individual terms. The values \(r_{i,\kappa}\) and \(r_{i,\ell}\) control the curvature and the length of the \(i\)-th boundary curve, respectively. With the term \(r_{i,\alpha}\) one can regulate the turning angle between the \(i\)-th and \((i+1)\)-th boundary curves. Minimizing the functional leads to a planar curved polygon whose
3.3. PARAMETERIZATION OF THE GUIDING SURFACE

curvature, turning angles and parametric speed resemble the ones of the three-dimensional cutting loop. More details can be found in [34].

- Additionally we also consider a fourth technique as fall-back strategy that generates a curved polygon with vertices placed on a circle. This technique is to be used if the optimization fails or returns a self-intersecting polygon.

Step 2 - Finding the control points The surface patch $p$ is a trimmed bicubic tensor-product spline patch with uniform knots. Initially we choose 3 inner knots per direction and refine if the accuracy of the result is not sufficient.

We generate the control points by solving a constrained optimization problem. The objective function is a combination of three terms:

- The first term

$$\int_{\Omega} \omega(t)[(f^* \circ p)(t)]^2 \, dt \quad (3.9)$$

measures the closeness to the implicit guiding surface. Note that the composition $(f^* \circ p)$ of the two functions $f^*$ and $p$ vanishes at the parameter $t$ if the implicit surface defined by $f^* = 0$ interpolates the point $p(t)$. The weighting function $\omega$ is procedurally defined such that it is zero at the domain boundary and increases towards the center of the domain, finally reaching the value 1 in the center of the domain. Using the weighting function improves the accuracy of the boundary interpolation, since the cutting surface approximates the exact cutting data but not the implicit guiding surface near the boundary.

- The second term ensures the approximation of the cutting data. It takes the form

$$\int_{\partial \Omega} \|p(t) - L(t)\|^2 + \lambda \|\nabla p(t) \cdot \vec{n}(t)\|^2 \, dt \quad (3.10)$$

where $L(.)$ and $\vec{n}(.)$ are parameterizations of the boundary data over the domain boundary $\partial \Omega$ and $\lambda$ is a positive weight. The first part ensures
positional accuracy, while the second one controls the tangent planes along the boundary. The positive weight $\lambda$ controls the relative influence of both terms.

- The third term is the approximate thin-plate energy, which is a standard fairness measure for surface fitting in geometry reconstruction [35]. It is used to regularize the fitting result.

The three terms are combined using non-negative weights, whose influence is discussed in the next section. In our experiments, we always scaled the geometry such that the bounding box fits into the unit cube and used standard weights $(10, 1, 1)$ for the three terms.

In addition to the objective function, the optimization also considers equality and inequality constraints. The first ones ensure the interpolation of the cutting loop at the vertices, while the second ones enforce the correct orientation of the boundary tangents, by making sure that the prescribed tangents along the cutting loop point towards the interior of the trimmed surface patch.

For the minimization of the non-linear objective function we need an initial solution, which we find by solving a simplified problem. When the first weight is set to zero, the resulting objective function becomes quadratic and can be solved rather easily. Note that the returned surface will be a regular trimmed surface patch, which interpolates the cutting loop on its boundary. In most situations (characterized by a relatively simple form of the solid and reasonable choice of the cutting loop), the interior of this surface patch will not penetrate the solid’s boundary faces, thus the initial solution can be used as a valid cutting surface.

For more complicated cases, however, the initial surface may intersect the boundary of the solid. We then obtain a valid result by minimizing the objective function with a positive first weight, using an iterative method that starts with the initial solution. More precisely, we then apply the fmincon function of Matlab™ to compute the control points by solving the resulting optimization problem. Examples are presented in the next section.
3.4 Results

This section consists of two parts. First we focus on the two-dimensional case, i.e., on the problem of finding a curve inside a given planar domain, connecting two points on the domain’s boundary. We consider the same test cases as used by Nguyen et al. [3] and show that the new method solves these problems as well while significantly accelerating the computation. In the second part we present results for three-dimensional solids, which are subdivided into topological cuboids using the algorithm of the isogeometric segmentation pipeline [23]. In the considered examples, the original construction of the cutting surface fails, since the generated surfaces intersect the solid’s boundary. We show that using implicit guiding surfaces, resolves these problems.

3.4.1 Cutting curves

The first example visualizes three different stages of the algorithm. In the top row of Figure 3.9 the given domain with two different sets of start and end points and the associated tangent directions are depicted. The computed implicit guiding curves, as well as the final parameterized ones are visualized in the second and third row, respectively.

The next examples are shown in Figure 3.10. Three relatively simple domains are given, along with end points and desired tangent vectors of the cutting curve. Clearly, our method produces results that are different from the curves obtained by the method in [3]. This is hardly surprising, as the methods rely on different approaches. Nevertheless, the resulting curves are also valid solutions as they fulfill all necessary criteria.

The next two test cases, shown in Figure 3.11, involve more complicated domains, which are again dealt with by both methods. Again we obtain valid results and a speed-up of the computing time. The new algorithm is about 27 times faster for the example of the maze and provides a tremendous improvement for the snake domain. While the original algorithm takes slightly over two hours, the
CHAPTER 3. THE CONSTRUCTION OF CUTTING SURFACES

Figure 3.9: Top: Two different start and end points with the associated tangent vectors. Center: The implicit guiding curves. Bottom: The parameterized cutting curves.

Figure 3.10: Guiding curves (blue) and cutting curves (purple) for the three character-shaped domains considered in [3]. The red curves have been generated by the method described in [3].
new method returns the result after just 27 seconds.

Figure 3.11: Guiding curves (blue) and cutting curves (purple) obtained for two more complicated domains: Maze (left) and snake (right). The red curves have been generated by the method described in [3].

<table>
<thead>
<tr>
<th>Example</th>
<th>J</th>
<th>K</th>
<th>U</th>
<th>Maze</th>
<th>Snake</th>
</tr>
</thead>
<tbody>
<tr>
<td>method from [3]</td>
<td>272s</td>
<td>3,309s</td>
<td>337s</td>
<td>410s</td>
<td>7,657s</td>
</tr>
<tr>
<td>new method</td>
<td>10s</td>
<td>11s</td>
<td>9s</td>
<td>15s</td>
<td>27s</td>
</tr>
<tr>
<td>speedup-factor</td>
<td>27.2</td>
<td>300.8</td>
<td>37.4</td>
<td>27.3</td>
<td>283.6</td>
</tr>
</tbody>
</table>

Table 3.1: Computing times for the examples shown in Figs. 3.10 and 3.11.

Examining the runtimes of both algorithms (see Table 3.1) shows a significant advantage of the new method: Depending on the domain’s complexity, we accelerate the method by about one or two orders of magnitude. There are several reasons for this speedup:

1. Computing the implicit guiding curve involves evaluating the objective function (3.2) multiple times. The bottleneck of this computation is the regularity term (3.3), which consists of a double integral over a bivariate function. As mentioned earlier, one can exploit the tensor-product structure of the spline function $f^*$ and rewrite the regularity term in the form $F_{\text{reg}}(c) = $
\( c^T \cdot M(u, v) \cdot c \), where each entry of the matrix \( M \) is the product of univariate integrals over products of derivatives of B-spline basis functions. The matrix \( M \) can be easily computed in advance and every further evaluation of the objective function can be done by a matrix-vector multiplication, which is usually very fast.

2. In addition to the efficient computation of the objective function of (RDCP), there are only linear inequality constraints, in contrast to the more expensive computation of the penalty functions in [3].

3. For curves, the parameterization step is less complicated than for surfaces. There is no need to solve a constrained optimization problem as there are faster methods available. We use a marching algorithm to create an approximating polygon, which is subsequently used to fit a parameterized spline curve.

4. We use standard settings for the degrees and knot vectors of the implicit guiding curve, as well as for the final parameterized curve and received valid results for all examples. Only for the snake example (cf. Figure 3.11) we got an improvement by manually choosing a different knot vector for the implicit guiding curve. However the vast speedup would allow to run several different settings and simply using the best result, while still reducing the runtime.

We conclude this section by analyzing the influence of the optimization weights, which are used when computing the guiding curve. We consider an L-shaped domain and compute several implicit curves by considering various settings of the parameter values, see Figure 3.12. More precisely, we minimize the objective function of RDCP with fixed values of \( \lambda_{int} \) and \( \lambda_{tan} \) and investigate the influence of the values of \( \lambda_{reg} \) and \( \delta \). For every curve, we calculate the minimum distance to the boundary of the domain and the maximum curvature of the resulting curve. It is clearly visible that an increase of \( \lambda_{reg} \) results in a decrease of the curvature,
3.4. RESULTS

while a higher value of $\delta$ increases the distance between the curve and the domain boundary.

![Diagram showing experimental results](image)

Figure 3.12: Experimental results showing the influence of the weights used for computing the guiding curve. The value $d$ is the length of the purple line (distance between corner and curve) and $\kappa_{\text{max}}$ is the maximum curvature of the curve segment. The red dot identifies the location of the curvature maximum.

3.4.2 Cutting surfaces

The first example (visualized in Figure 3.13) shows a portion of an “iron maiden”: two parallel walls with spikes pointing towards each other. The height of the spikes exceeds half the distance between the two walls, so a planar surface would be penetrated by those spikes. The bottom picture shows the resulting wave-shaped guiding surface, which is fully contained in the empty space left between these spikes.
The second example shows the influence of the regularity weight in the parameterization step. We consider a cube-shaped vase with a planar cutting loop (cf. Figure 3.5), visualized in side view in Figure 3.14. The computed implicit guiding surface is shown in red, and we obtain three different parameterized patches, depending on the weight for the fairness measure. As the weight $\lambda_{\text{reg}}$ (weight of the third term) increases, the resulting surface becomes less curved and does not approximate the guiding surface anymore: Note that this may lead to an invalid cutting surface. In all our examples, we reached satisfying results when setting
3.4. RESULTS

the weight to 1.

\[ \lambda_{\text{reg}} = 1 \]
\[ \lambda_{\text{reg}} = 10 \]
\[ \lambda_{\text{reg}} = 100 \]

Figure 3.14: Influence of the regularity weight in the parameterization step.

A more complicated version of the vase-shaped object is shown in Figure 3.15. The object has a five-sided planar base surface and five non-planar side-faces. All the faces on the top side of the object as well as the faces of the indentation are planar. The curves of the cutting loop correspond to straight line segments in the parameter domains of the respective faces. However, since they are defined on non-planar faces, they define non-planar spatial curves.

Constructing a valid cutting surface requires to find a trimmed NURBS surface patch, whose five-sided boundary curves interpolate the five curvilinear cutting loop curves, while its interior stays inside the domain. The constructed implicit guiding surface is depicted in the left picture in the second row, next to the finally obtained parameterized trimmed surface patch. Note that the parameter domain (bottom-right) is a five-sided planar polygon, which has been automatically generated.

The cutting loops considered in the previous examples were manually generated and did not introduce a meaningful segmentation. The intention of these loops was to show the properties of the cutting surface construction. In contrast to this, the remaining two examples use real cutting loops generated by the ESA and the CLS. However, instead of MS\(^3\) we use the original base solids of [2] (topo-
CHAPTER 3. THE CONSTRUCTION OF CUTTING SURFACES

Figure 3.15: Top: Vase-shaped object with non-planar faces seen from two different viewing directions. Bottom: Implicit guiding surface, parameterized surface patch, and its automatically created parameter domain.

logical cuboids, tetrahedra and prisms) and omit the face pre-segmentation step in the beginning of the ESA.

The first example is a cube with two slots, which requires a non-trivial segmentation, shown in Figure 3.16. We observed that this shape occurs frequently when applying the ESA and the CLS to engineering objects with holes, since they need to be pre-segmented in order to be contractible. Therefore the solids will be split along the holes (see [22]) resulting in similar shapes as the one in Figure 3.16.

The original construction of the cutting surface fails, since two opposite faces of one of the solids in the left part of Figure 3.16 intersect each other. This problem can now be resolved by using the novel cutting surface construction. The pictures on the right hand side show the same example, but with cutting surfaces...
3.4. RESULTS

Figure 3.16: Segmentation of the bi-slotted cube with invalid (left) and valid (right) cutting surfaces.

obtained by using implicit guiding surfaces. We obtain a valid segmentation into three topological cuboids.

The final example is a slightly simplified coupling piece of a garden hose, see Figure 3.17. Due to the symmetry, it suffices to consider only a quarter of this object.

Figure 3.17: Solid object (center and right from two different perspectives) representing a slightly simplified half of a coupling of a garden hose (left).
Figure 3.18: Automatic segmentation of a quarter of the hose coupling, using the ESA. The three solids in the red box lead to a problem with the construction of the cutting surface.
3.4. RESULTS

The result of the segmentation algorithm is presented in Figure 3.18. Most segmentation steps are rather simple, as they simply cut slices off the solid, using planar cutting surfaces. There is, however, a more challenging step, which is marked in red. Here, the original construction of the cutting surface fails, due to the particular shape of the solid. The cutting surface penetrates the solid’s boundary. This fact is visualized in Figure 3.19, left. Using the implicit guiding surface leads to a different result that resolves this problem, see Figure 3.19, right. Summing up, we obtain a valid segmentation into eight topological cuboids and one prism with a triangular base surface.

Figure 3.19: Failing (left) and valid (right) segmentation of a piece of the hose coupling example. Bottom row: The side view (with a viewing direction parallel to the chords of the circular arcs) shows the intersection (marked by the red arrows) between the opposite boundary faces obtained when using the initial version of the cutting surface (left), which is no longer there when applying the new algorithm (right).
Chapter 4

Face pre-segmentation

The splitting algorithm CLS (see Chapter 2) searches the edge graph of the solid for the most suitable cutting loop, i.e. the one that induces the most promising cutting surface. Clearly, the quality of the two new sub-solids depends not only on the combinatorial structure of the solid’s edge graph, but is also strongly influenced by its geometry. More complex shapes are more challenging than simpler ones and require more sophisticated techniques in order to obtain a satisfactory result. The proposed face pre-segmentation of the initial solid takes this fact into account.

In many cases, the more complicated parts of a solid either correspond to a large number of faces, or to fewer faces with more complex shapes. The latter case could not be dealt with successfully by the original algorithm [23], since it was unable to identify suitable segmentations of faces possessing a complex shape. We address this problem by splitting those faces into simpler ones, joined across newly introduced auxiliary edges.

Recall from Chapter 2, that before the ESA starts reducing the complexity of the solid, it first chooses cutting loops, which eliminate non-convex edges. Furthermore we know that auxiliary edges are always non-convex ones. Thus, the newly introduced edges trigger splitting operations that simplify the shape of the solid.

We assume that the shape of the face roughly matches the shape of the asso-
ciated parameter domain. This is the case if the parameterization has bounded distortion. Consequently, it suffices to analyze the shape of the parameter domain, which is a much simpler task than analyzing the shape of the face itself.

The face is split along one or several straight lines in the parameter domain. In this chapter we will introduce two different methods to determine the location of these lines.

4.1 Subdivide a domain using the medial axis transform

We consider a simply connected planar domain $D$. Its boundary curve is supposed to consist of a finite number of piecewise $C^2$-continuous curve segments (no fractals). An inscribed disk $d \subset D$ is called a maximal inscribed disk (MID) if it is not contained in another inscribed disk $d \neq \bar{d} \subset D$ (see Figure 4.1). The medial axis of the domain $D$ is formed by the center points of all maximal inscribed disks. Together with the associated radius information we obtain the Medial Axis Transform (MAT), see [36].

Some general observations about maximal inscribed disks and the medial axis:

- An MID touches the boundary of $D$ in more than one point (cf. Figure 4.1), or in a point, where the curvature of the boundary reaches a local extremum.

- The medial axis has a tree-like structure. At the meeting points of $n \geq 2$ branches, the associated MID touches the boundary in $n$ different points, provided that no part of the boundary coincides with an MID. There are two touching points of an MID with the boundary, if the center of the MID is located in the interior of a branch of the medial axis (cf. Figure 4.2).

- The radius of a MID is the shortest distance of its center to the boundary of $D$. 

4.1. **SUBDIVIDE A DOMAIN USING THE MEDIAL AXIS TRANSFORM**

Figure 4.1: The blue circles are MIDs, while the purple ones are contained within another circle.

Figure 4.2: Shrinking a rectangle to a square lets the middle branch of the medial axis disappear. The two crossings and the associated MID finally coincide. The number of touching points coincides with the number of branches meeting in the crossing of the medial axis.
Since the radius of a MID is the shortest distance to the domain’s boundary, a local minimum of the radius function along the MAT indicates a bottleneck-like shape of the domain. Splitting the domain across this narrow region will lead to two simpler sub-domains. Therefore, the local minima of the radius function along the MAT indicate where to split the face. Each of the associated maximal disks touches the boundary of the face in two points, cf. Figure 4.3. The disk’s diameter connecting them defines the splitting curve.

Figure 4.3: Left: Local minimum of the radius function along the MAT with corresponding MID and touching points. Right: The obtained splitting across the connection of the two touching points.

4.2 Subdivide a domain using a distance function

Although using the MAT to determine where to split the domain leads to the desired cuts in many cases, it fails in some situations. Figure 4.4 shows that the corresponding circle to the intuitive cut is not always a maximal inscribed disk. Therefore, its center point is not placed on the medial axis. In fact, the radius function does not possess a local minimum along the MAT in the example of Figure 4.4.

However, we can use a different method to reach our goal of pre-segmenting this planar domain. Let the boundary of the domain $\mathcal{D}$ be parameterized
4.2. SUBDIVIDE A DOMAIN USING A DISTANCE FUNCTION

Figure 4.4: Left: the necessary circle is obviously not a maximal inscribed circle. The radius function has no local minimum along the MAT. Right: the desired cut is found by the distance function approach.

clockwise by \( b(t) \), with \( t \in [0, 1] \) and \( b(0) = b(1) \). We consider the distance function

\[
d(u,v) = \|b(u) - b(v)\|, \quad \text{for } (u,v) \in [0,1]^2,
\]

(4.1)

to find the locations of the splitting lines. More precisely, potential splitting lines correspond to prominent local minima of this function, with well-separated end points. Typically this results in several candidates for splitting lines, and we select the shortest one.

Since usually the boundary contains non-smooth corners, the distance function \( d \) is not differentiable on the whole unit square. Thus, finding the local minima requires special care. One needs to treat values that correspond to corners of the domain accordingly to find the correct extrema.

We find an approximate solution by discretizing the problem. First, we sample points along the boundary of the domain and store the distance between any two points in a matrix. Second, we compare any value in the matrix with its eight direct neighbors. If a value is smaller (within some tolerance) than all the surrounding values, we consider it to be a local minimum. Based on the sample density and the geometry of the domain we might get clusters of potential minimal
locations, as is depicted in Figure 4.5.

In order to obtain only reasonable cuts we need to filter out unsuitable ones. We notice that long clusters in the parameter domain, with a descent of approximately 45 degrees correspond to parallel edges of the face’s boundary. Additionally, the splitting line must not intersect the boundary curve, cut through a convex vertex or result in non-convex sub-domains. By filtering out undesired cuts, we are left with only suitable ones (cf. Figure 4.6).

Figure 4.5: Left: Local Minima in the lower half of the parameter domain of the distance function. Right: Corresponding cuts in the physical domain.

Figure 4.6: The same example as in Figure 4.5, but unsuitable cuts were filtered out. Two equally valid cuts remain.
Chapter 5

Midpoint subdivision suitable solids (MS$^3$)

The ESA of Chapter 2 decomposes a given solid into a collection of sub-solids, which are suitable for midpoint subdivision. We denote those sub-solids as Midpoint Subdivision Suitable Solids (MS$^3$) and refer to Chapter 6 for details about midpoint subdivision. However, in order to specify, which solids classify as MS$^3$, we briefly explain the main idea of midpoint subdivision.

Let $S$ be a solid in boundary representation with vertices $v_1,\ldots,v_{n_v}$. Midpoint subdivision decomposes $S$ into $n_v$ sub-solids $S_1,\ldots,S_{n_v}$, such that

$$v_i \subset S_i \quad \text{for all } i = 1,\ldots,n_v \text{ and}$$

$$v_i \not\subset S_j \quad \text{for all } i \neq j.$$ 

Since the goal of the isogeometric segmentation problem is to eventually decompose a given solid into a collection of topological cuboids, we want the sub-solids $S_1,\ldots,S_{n_v}$ to be topologically equivalent to a cube. The valency of any vertex of a cuboid is three and every resulting sub-solid $S_i$ contains the vertex $v_i$ of the original input solid $S$, for $i = 1,\ldots,n_v$. Thus the solid $S$ needs to possess only tri-valent vertices in order to be considered suitable for midpoint subdivision.
In Section 5.1 we show that the ESA can be used to decompose a given solid into a collection of sub-solids, which only possess tri-valent vertices. So far we took solely combinatorial properties of MS$^3$ into account. The quality of the output shapes depends strongly on the geometry of the input solid. Thus we also employ two simple geometric criteria: Since all generated sub-domains, that share one common face of the original solid, meet in a common edge, small interior angles will be present if the original solid has faces with many edges. Therefore we restrict ourselves to solids with at most 6-sided faces. Similarly, in order to prevent the occurrence of small angles at the central vertex, we restrict ourselves to solids with at most 12 faces. Combining all the considered criteria we can now give a definition of an MS$^3$.

**Definition 1.** A solid $S$ is considered a *Midpoint Subdivision Suitable Solid* (MS$^3$), if

- every vertex has a valency of 3,
- the number of faces is at most 12 and
- any face has at most 6 vertices.

In Section 5.2 we will see that this definition leads to 185 topologically different MS$^3$ (see Table 5.1, page 56).

### 5.1 Existence of a valency-reducing cutting loop

In [2] the authors showed that a given solid, which contains a non-convex edge $\varepsilon$ can be split by an appropriate choice of the cutting loop into two sub-solids, that both still contain $\varepsilon$, but as a convex edge. In this chapter we will prove the following similar statement about high-valent (valency greater than 3) vertices.

**Theorem 1.** Any solid $S$, which fulfills the assumptions of the ESA and contains a high-valent vertex $v$, can be split by the CLS into two sub-solids $S_1$ and $S_2$, ...
both containing a copy of \( v \) with reduced valency. The splitting does not increase the valency of any other vertex and creates only tri-valent ones.

Before we prove this theorem, we need to recall some simple observations about cutting loops. Let \( v \) be a vertex with valency \( n \). A valid cutting loop that passes through \( v \) can be used to construct a cutting surface, which splits the solid \( S \) into two sub-solids \( S_1 \) and \( S_2 \). Both sub-solids contain the vertex \( v \) with valencies \( n_1 \) and \( n_2 \) respectively. Since the original solid, and therefore also the two sub-solids, are 3-vertex-connected, we know that

\[
n, n_1, n_2 \geq 3.
\] (5.1)

The two edges of the cutting loop that contain \( v \) can be existing or auxiliary ones, cf. Figure 5.1. Let \( n_{\text{aux}} \in \{0, 1, 2\} \) be the number of auxiliary edges in the cutting loop that contain \( v \). Since the edges of the cutting loop are contained in both sub-solids \( S_1 \) and \( S_2 \) and an auxiliary edge is not contained in the original solid \( S \) we obtain the equation

\[
n_1 + n_2 = n + 2 + n_{\text{aux}}.
\] (5.2)

A careful analysis of the possible configurations gives the following result.

**Lemma 2.** The valencies \( n_1 \) and \( n_2 \) do not exceed the original valency \( n \) if the cutting loop contains at most one auxiliary edge through \( v \).

**Proof.** We show this statement by contradiction. Without loss of generality let \( n_1 > n \). Then by (5.2)

\[
n + n_2 < n_1 + n_2 = n + 2 + n_{\text{aux}}
\]

holds. Since \( n_{\text{aux}} \leq 1 \) it follows that \( n_2 < 3 \), which contradicts (5.1). \( \square \)

The following lemma is another important auxiliary result, we will need to prove Theorem 1.
Lemma 3. If two faces $F_1$ and $F_2$ of an edge graph share two vertices $v$ and $w$, the edge $(v, w)$ exists and is contained in both faces $F_1$ and $F_2$.

Proof. We prove this result by contradiction. Let $F_1$ and $F_2$ be two faces that share the vertices $v$ and $w$, but no edge, see Figure 5.2. We assume that the vertices on both faces are ordered counterclockwise and denote the paths from $v$ to $w$ and from $w$ to $v$ in face $F_1$ by $P_1$ and $Q_1$, respectively. Similarly, we denote the paths from $w$ to $v$ and form $v$ to $w$ in face $F_2$ by $P_2$ and $Q_2$, respectively (cf. Figure 5.2).

The loop $Q = Q_1 \cup Q_2$ contains at least one vertex besides $v$ and $w$. Otherwise, the two faces would share the edge $(v, w)$, since the edge graph is free of double edges. Analogously we conclude that the loop $P = P_1 \cup P_2$ contains at least one vertex besides $v$ and $w$ as well. Consequently, removing the vertices $v$ and $w$ disconnects the edge graph. This contradicts the assumption that the edge graph is 3-vertex-connected.

With the Lemmas 2 and 3 we are now able to prove Theorem 1. The main idea of the proof is based on the proof of Theorem 1 of [2], which showed that there always exists a valid cutting loop that passes through a given non-convex edge.

Proof of Theorem 1. We enumerate the faces that surround $v$ in counter-clockwise order by $F_1, \ldots, F_n$, and the adjacent vertices by $v_1, \ldots, v_n$, such that $(v, v_k) =$
5.1. EXISTENCE OF A VALENCY-REDUCING CUTTING LOOP

Figure 5.2: The loops considered in the proof of Lemma 3.

$F_k \cap F_{k+1}$ (see Figure 5.3). Note that indices are considered modulo $n$. From the $n$ adjacent vertices, we pick two vertices $v_i$ and $v_j$, which are neither connected by an edge, nor share one of the faces $F_1, \ldots, F_n$. This is always possible, since $n \geq 4$ and the edge-graph is planar.

We show that the polygon $(v_i, v, v_j)$ can be extended to a valid cutting loop, i.e., any face contains at most one edge of that loop. This loop reduces the valency of $v$, due to the choice of $v_i$ and $v_j$. Moreover, Lemma 2 ensures that the valencies of $v_i$ and $v_j$ do not increase. All other vertices are chosen as auxiliary (and hence tri-valent) ones. The construction of this cutting loop requires the analysis of two different cases:

Case 1: $v_i$ and $v_j$ share a face $F^*$ (cf. Figure 5.3). We connect $v_i$ and $v_j$ by an auxiliary curve in that face, thereby obtaining the desired cutting loop. It will not be valid, only if the face $F^*$ coincides with any of the faces $F_i, F_{i+1}, F_j$ or $F_{j+1}$. We show that this is not the case by contradiction. W.l.o.g. assume that $F^* = F_j$. It follows, that the faces $F_i$ and $F_j$ both contain the vertices $v$ and $v_i$. By Lemma 3 we can conclude that $(v, v_i) = F_i \cap F_j$. However, since $(v, v_i) = F_i \cap F_{i+1}$ holds, it follows that $F_j = F_{i+1}$, which contradicts the choice of the vertices $v_i$ and $v_j$. By similar reasoning one can rule out the possibilities that $F^*$ coincides with any other of the proposed faces. Consequently, the cutting loop is valid.

Case 2: $v_i$ and $v_j$ do not share a face (cf. Figure 5.4). Let $F^*_i$ be the face that shares an edge with $F_{i+1}$ and contains $v_i$ but not $v$. Similarly, let $F^*_j$ be the face
Figure 5.3: Theorem 1, case 1: \(v_i\) and \(v_j\) are contained in the face \(F^*\). We connect them by an auxiliary curve to close the cutting loop.

that shares an edge with \(F_j\) and contains \(v_j\) but not \(v\).

We consider the union of the faces \(F_{i+1}, \ldots, F_j\). From its boundary we remove the vertex \(v\) to obtain the path \(q_1 = v_i, \ldots, q_{\ell} = v_j\). The faces, which contain at least one of the vertices \(q_1, \ldots, q_{\ell}\) but not \(v\) form a 2-vertex connected sub-graph, whose dual graph is connected. The shortest path (with respect to the number of edges) from \(F_i^*\) to \(F_j^*\) in the dual graph indicates how to obtain the cutting loop.

The shortest path in the dual graph corresponds to a chain of faces \(H_1 = F_i^*, \ldots, H_m = F_j^*\) of the original graph, such that adjacent faces \(H_k\) and \(H_{k+1}\) share an edge. On each edge \(H_k \cap H_{k+1}\) we create an auxiliary vertex \(w_k\) for \(k = 1, \ldots, m-1\) and connect consecutive ones by auxiliary curves in the corresponding faces. Finally we close the cutting loop by connecting \(v_i\) to \(w_1\) and \(v_j\) to \(w_{m-1}\) by auxiliary curves in \(H_1\) and \(H_m\) respectively.

Since, by construction, none of the faces \(H_1, \ldots, H_m\) contains \(v\), they are different from \(F_i, F_{i+1}, F_j\) and \(F_{j+1}\). Thus, the cutting loop is valid.

\[\square\]

**Corollary 1.** The extended segmentation algorithm (ESA) terminates.

**Proof.** Recall from [1] that a split along a valid cutting loop reduces the combinatorial size of a solid. The proof of Theorem 1 indicates the existence of a valid
5.1. EXISTENCE OF A VALENCY-REduCING CUTOuNG LOOP

Figure 5.4: Theorem 1, case 2: \( v_i \) and \( v_j \) do not share a face. We construct a path in the chain of adjacent faces (green and red) connecting \( v_i \) and \( v_j \). The green faces contain the shortest path, with respect to the number of edges.

cutting loop, which reduces the valency of a high-valent vertex. Together with the observations made in [1, 2] we know that any call of CLS returns two sub-solids with smaller combinatorial size. Thus the sub-solids obtained by the ESA will eventually fulfill the criteria of Definition 1.

\[ \square \]

Note that while Theorem 1 guarantees the existence of a valency reducing cutting loop, it is not used directly to construct it during the execution of CLS. Instead we first assess the valencies of every vertex of the two new sub-solids corresponding to a cutting loop. We then order the cutting loops with respect to the resulting valencies according to a lexicographic ordering. Finally we use a suitable cost function, which leads to the selection of a cutting loop that splits the solid into two sub-solids, with fewer high-valent vertices.
5.2 Template MS$^3$

In Chapter 6 we need a way to construct a sufficiently simple topologically equivalent template to any given MS$^3$. According to Steinitz’s theorem the edge graph of any MS$^3$ possesses a geometric realization as a convex polyhedron. In [37] the authors provide an algorithm to automatically construct a convex polyhedron from a given edge-graph. Unfortunately this method cannot be used for our purpose, since the resulting polyhedron will generally possess a rather flat shape (cf. Figure 5.5).

Alternatively, we use the fact that due to the criteria of Definition 1 there is a finite number of different edge graphs that correspond to MS$^3$. We list the possible edge graphs by exploiting the duality between planar triangulations and planar graphs with only trivalent vertices. With the help of plantri [38] we generated the edge graphs of all possible MS$^3$. Table 5.1 reports the number of topologically different edge graphs for all solids that fulfill Definition 1.

<table>
<thead>
<tr>
<th># faces</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td># MS$^3$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>30</td>
<td>44</td>
<td>77</td>
<td>185</td>
</tr>
</tbody>
</table>

Table 5.1: Number of different edge-graphs with respect to the number of faces.
To every edge graph we pre-constructed a suitable convex polyhedron that can be used as a template. Figure 5.6 shows all the edge graphs and corresponding convex template polyhedra with at most 9 boundary surfaces.

(a) Planar edge graphs.  
(b) Associated template polyhedra.

Figure 5.6: The precomputed edge graphs and template polyhedra up to 9 faces.
Chapter 6

Midpoint subdivision

Recall from Chapter 2 that we eventually want a volumetric NURBS representation of our input solid \( S \). For that purpose, we first subdivide the solid into a collection of topological cuboids, which can then be parameterized, since they are topologically equivalent to the unit cube.

However, the ESA segments a given object into a collection of \( \text{MS}^3 \) and not into topological cuboids. In this chapter we will explain how to further decompose an \( \text{MS}^3 \) using midpoint subdivision. We start by explaining the benefits of using midpoint subdivision, instead of simply continuing to split the \( \text{MS}^3 \) by the segmentation algorithm.

6.1 Motivation

One of the biggest advantages of the ESA is the intuitive construction, which usually leads to a small number of computed \( \text{MS}^3 \). However, the choice of cutting loops in each step determines if and after how many steps the algorithm terminates. Theorem 6 and Corollary 7 of [1] imply the existence of a valid and complexity reducing cutting loop for any (sub-) solid of the segmentation process. In general it is necessary to cut through existing edges in order to split a solid into two topologically smaller ones. Every existing edge that has been cut
CHAPTER 6. MIDPOINT SUBDIVISION

Figure 6.1: Planar segmentation of a pentagon. The angles are measured in degrees. Left: Split through one existing vertex causes small angles. Middle: Split through two auxiliary vertices does not reduce the combinatorial size of the object. Right: midpoint subdivision creates 5 quadrilaterals with improved angles.

through will appear in both new sub-solids and the angle between its adjacent faces gets approximately split in half. Multiple segmentation steps might lead to very small angles in the final cuboids, which may cause problems in the subsequent parameterization step.

Intuitively one might try to find a way to cut (nearly) orthogonally through the solid’s facets. In the three-dimensional case, this requires the construction of auxiliary edges in the solid’s faces, which has been discussed in Chapter 3. However, the termination of the algorithm must still be guaranteed. Unfortunately, this is not possible using a single cutting surface in each splitting step. We are now going to investigate the method of midpoint subdivision, which was already discussed in [39]. In that publication, the main emphasis was put on the compatibility of the generated hexahedral sub-meshes across the interfaces, which was achieved via integer programming. In contrast to this earlier work, we focus on the generation of a single NURBS volume parameterization for each of the generated cuboidal sub-domains. By midpoint subdivision, a solid $S$ with $n_v$ vertices gets decomposed into $n_v$ topological cuboids $S_1, \ldots, S_{n_v}$ in one step. Figure 6.1 visualizes the discussed issues in the planar case.
6.2. MIDPOINT SUBDIVISION OF A CONVEX POLYTOPE

Figure 6.2: Left: The center point (blue) gets connected to every edge-midpoint (purple) by a straight line segment. Right: The polygon gets split into quadrilaterals along the newly introduced edges.

The next section discusses the actual realization of the midpoint subdivision of a special class of domains for dimension $d = 2$ and $d = 3$.

6.2 Midpoint subdivision of a convex polytope

The concept of midpoint subdivision can be introduced for dimension $d = 2$ or $d = 3$ in a similar way. In this section, we will explain the method for convex polytopes with $n_v$ vertices, all of which have a valency of $d$.

The planar case. We consider a convex polygon with $n_v$ vertices $v_1, \ldots, v_{n_v}$. First, we compute the center of mass of every edge and of the polygon itself. We use those points as edge-midpoints and as the center of the polygon, respectively. Second, we create straight edges that connect each edge-midpoint to the center point. Finally we split the domain along those new edges into $n_v$ quadrilaterals. Note that every new sub-domain contains exactly one of the polygon’s vertices and the center of the polygon. This construction is visualized in Figure 6.2 for a particular convex polygon.
CHAPTER 6. MIDPOINT SUBDIVISION

The three-dimensional case  The concept of the planar case can be straightforwardly adapted to the three-dimensional one. Now the input domain is a convex polyhedron with $n_v$ vertices and $n_f$ planar faces. First, we choose a suitable center point for every edge and every face of the polyhedron. Second, we connect those center points by straight edges in the corresponding faces. Note that the first two steps can be viewed as a planar midpoint subdivision on every face of the polyhedron. Third, we choose the center point of the polyhedron as the center of mass and connect it to every face-midpoint by a straight edge. Fourth, we construct one bilinear cutting surface for each quadrilateral loop of newly created edges. Finally we split the solid along the cutting surfaces into $n_v$ topological cuboids. As in the planar case, every cuboid contains exactly one of the polyhedron’s vertices and the domain’s center point. Figure 6.3 depicts the midpoint subdivision of a particular instance of a convex polyhedron with only tri-valent vertices.

Note that in both cases ($d = 2$ and $d = 3$) the choice of the midpoints and the construction of cutting curves and surfaces are not restricted to the centers of mass, straight line segments and bilinear surfaces. One could use any point...
in the interior of the respective domain (edge, face or solid) as the corresponding midpoint. Similarly, any set of cutting curves or surfaces is feasible, as long as they do neither intersect each other nor the boundary of the domain.

This leads to a generalization of midpoint subdivision to more complex domains. However, it is particularly difficult to choose the midpoints and especially construct the cutting curves and surfaces robustly.

The next section studies the planar case and investigates the computational realization of the midpoint subdivision of a general, possibly curved polygon in the plane.

6.3 Planar midpoint subdivision of a curved polygon

As mentioned in the previous section, the subdivision of a general (curved) polygon requires special care. Due to the general shape of the domain, there are three new challenges to be considered.

Choosing the edge-midpoints A curved edge does generally not contain its center of mass. Hence one needs to find a different choice of edge-midpoints. However, that is not particularly difficult. We assume every edge-curve is given in arc-length parameterization and simply evaluate it at the midpoint of its parameter interval. Note that this will coincide with the center of mass if the edge is a straight line.

Choosing the center point Similar to the previous paragraph, the center of mass is no longer valid in the case of a general polygon, due to the fact that it is not necessarily contained inside the domain. Since the center should intuitively be far away from the boundary curves we can choose the center of the largest inscribed circle as domain-midpoint. By definition it will be located inside the
Constructing the cutting curves  The most difficult new challenge is the construction of non-intersecting cutting curves that also do not intersect the boundary. In Chapter 3 we constructed a single curve segment, connecting two points on the boundary of a given domain. The curve was limited to the interior of the domain and needed to be free of self-intersections. Now we need to find multiple curves with the same property. Additionally, we need to ensure that these cutting curves do not intersect each other.

In order to achieve this goal, we first decompose the polygon into non-intersecting sub-regions that all contain the center $q$ on the boundary. Afterwards we find a cutting curve in every sub-domain. The resulting cutting curves will not intersect the boundary due to the construction explained in Chapter 3. Neither will they intersect each other, since they are contained in disjoint regions. This procedure, which is visualized in Figure 6.4, consists of two main steps.

Step 1: We split the domain into two sub-domains by using a modified version of the implicit guiding curve-approach. We force the curve to pass through the center $q$ by adding the term $f^*(q)^2$ with an appropriate weight to the functional (3.2). Consequently, the splitting curve will pass through the midpoint $q$. Thus $q$ will be located on the boundary of the two new sub-domains.

If the domain has an even number of edges, we simply choose two edge midpoints opposite of each other as start and end points for the splitting curve. Otherwise we choose one edge-midpoint and the opposite vertex of the polygon. Either way we will save the curve segments of the constructed curve that connect the midpoint $q$ to an edge-midpoint $q_i$ as cutting curves.

Step 2: Now we recursively split the two sub-domains into smaller ones, by
constructing cutting curves from $q$ to edge midpoints $q_i$ or vertices of the sub-domains. More precisely, if the number of points $q_i$ in a sub-domain is odd and greater than one, we connect the middle one to the point $q$. If, on the other hand, the number of points $q_i$ in the sub-domain is even, we connect $q$ to the vertex in between the two middle edge-midpoints. Either way we split the domain into two sub-domains that contain fewer edge-midpoints than the larger domain.

Eventually we have decomposed the polygon into disjoint sub-domains, all of which contain the center point $q$ and only one of the edge-midpoints $q_i$. We finally compute the desired cutting curves by connecting $q_i$ to the edge-midpoint $q_i$ in every one of the sub-domains. Hence, these curves will neither intersect each other, nor the boundary of the domain.

The visualization of this concept in Figure 6.4 shows a regular pentagon that gets split into two sub-domains in the first step. The respective cutting curve connects one edge midpoint to the vertex on the opposite side and contains the center of the polygon. Thus, one half of the first cutting curve is already one of the final curves. The two sub-domains are split further by an auxiliary cutting curve, connecting the center $q$ to a vertex, such that both new sub-domains only contain one of the edge-midpoints $q_i$. Finally the remaining cutting curves are computed in the constructed sub-domains. Note that the domain was chosen to be a regular pentagon to simplify the illustration of the method.

In order to improve the results of this method we can use the fact that the implicit guiding curve approach allows for specifying the tangent vectors of the start and end points of the splitting curve. We choose the tangent vectors for the edge-midpoints $q_i$ as the inward-facing normal vectors of the boundary curves $F_i$ respectively. Thus we fulfill the goal of cutting (nearly) orthogonally through the domains boundary (cf. Section 6.1).

Since the $n_v$ constructed cutting curves will meet in the center point, the optimal angle between two neighboring curves will be $\frac{2\pi}{n_v}$. We choose the tangent vectors at $q$ in such a way that this condition is (approximately) fulfilled.

The Figures 6.5 and 6.6 show three different domains subdivided by the intro-
Figure 6.4: Recursive subdivision of a pentagon into non-overlapping sub-domains. Each sub-domain contains exactly one edge-midpoint. Dashed red lines are auxiliary splitting curves, while the solid red lines are the final cutting curves.

Although this construction returns valid results, it also introduces some issues. First, it is relatively slow, since it uses the implicit guiding curve technique repeatedly, and thus many times. Second, it returns different results, depending on the start and end points of the first split, as it is visualized in Figure 6.6. However, the biggest problem is that it does not admit a direct generalization to the three-dimensional case, in which we are most interested in. Therefore, we will present an alternative approach that generally leads to better results in the next section.

### 6.4 Midpoint subdivision of general MS$^3$

Instead of recursively using the methods of Chapter 3 we will exploit the fact that the midpoint subdivision of a convex polytope is straightforward. To a given poly-
6.4. MIDPOINT SUBDIVISION OF GENERAL MS$^3$

Figure 6.5: Planar Midpoint Subdivision on two (curved) polygons.

Figure 6.6: The same domain is subdivided in two different ways. The blue curves indicate the first split in both examples.

gon (for $d = 2$) or MS$^3$ (for $d = 3$) we consider a topologically equivalent template domain and construct a regular mapping from the template to the input domain. In the planar case, we simply construct a convex polygon, whose edge-length ratios are similar to the ones of the given polygon. In the three-dimensional case, we choose the pre-constructed template MS$^3$ (see Figure 5.6), which is topologically equivalent to the given MS$^3$. Afterwards we perform the midpoint subdivision on the template and use the mapping to transfer the subdivision to the input domain.

A well-established approach to construct the required mapping from a template to a target is the free-form deformation [40], where a box surrounding the template is mapped into a free-form volume around the target. This regular mapping transforms the boundary of the template into the boundary of the target. Its construction is relatively simple to implement and produces satisfying results if the shape of the target is similar to the shape of the template. Figure 6.7 shows
the result of this method for the three planar polygons of Figures 6.5 and 6.6.

For more complicated domains, especially in the three-dimensional case, the free-form deformation is sometimes not powerful enough to return a suitable mapping. We need a more sophisticated approach, which is based on a systematic use of a multi-patch structure that will be presented in the next section.

6.4.1 Multi-patch structure

Midpoint Subdivision decomposes a given domain into a collection of parameterizable sub-domains. We will now introduce a multi-patch structure that allows us to describe the entire domain in terms of the smaller sub-domains (see the part of Figure 6.8 that is within the dashed boundaries).

We consider a $d$-dimensional domain $S$ for $d = 2, 3$. It is assumed to be topologically equivalent (i.e., diffeomorphic) to a polygon for $d = 2$ and to a template $MS^3$ for $d = 3$. The number of vertices, all of which have the same
6.4. MIDPOINT SUBDIVISION OF GENERAL MS$^3$

valency $d$, is denoted by $n_v$.

The midpoint subdivision is realized by constructing a multi-patch parameterization

$$p_i : [0, 1]^d \to S, \quad i = 1, \ldots, n_v,$$

with $S = \bigcup_{i=1}^{n_v} S_i$ and $S_i^o \cap S_j^o = \emptyset$ if $i \neq j$, where $S_i = p_i([0, 1]^d)$.

More precisely, the boundary facets of $[0, 1]^d$ are split into two groups.

- The first $d$ facets represent the boundary of $S$

$$b_{i,j} : [0, 1]^{d-1} \to \partial S \cap F^{(i,j)}, \quad j = 1, \ldots, d,$$

with $b_{i,j} = p_i|_{[0,1]^{d-1} \times \{0\} \times [0,1]^{d-j}}$. 

Figure 6.8: Multi-patch structure (shown within the dashed boundaries) and template mapping problem for $d = 2$. 
where the $F^{(i,j)}$ denote the $d$ facets of $S$ that contain the $i$-th vertex $v_i$. 

- The remaining $d$ facets parameterize the interior interfaces

$$c_{i,j} : [0,1]^{d-1} \to S, \quad j = 1, \ldots, d,$$

with $c_{i,j} = p_i|_{[0,1]^{j-1} \times \{1\} \times [0,1]^{d-j}}$.

The interfaces are joined pairwise with $C^0$-smoothness. More precisely if the edge $(v_i, v'_i)$ exists in the solid $S$, there are indices $j$ and $j'$ and a permutation $\pi$ of the $d - 1$ parameters such that $c_{i,j} = c_{i',j'} \circ \pi$.

Note that

$$p_i(0, \ldots, 0) = v_i, \quad \text{and} \quad p_i(1, \ldots, 1) = q,$$

where $q$ is the center point of the domain $S$.

The multi-patch structure of a template MS$^3$ is readily available. We will transfer it to a topologically equivalent general MS$^3$ in the next section.

### 6.4.2 Template mapping problem

For subdividing a general MS$^3$, the simple techniques described in Section 6.2 will not suffice. Instead we consider a topologically equivalent template MS$^3$ and its associated multi-patch structure, and construct a regular mapping from the template to the solid, which is used to transfer the subdivision.

We denote the given domain, which is assumed to be a MS$^3$, by $S$ and the associated template by $\tilde{S}$. The multi-patch parameterizations of the solid and of the template will be denoted by $p_i$ and $\tilde{p}_i$, respectively. Similarly, we will use the tilde to distinguish between the MS$^3$ and its template.

In order to construct a mapping from the template to the target we consider the following **template mapping problem**, which is visualized in Figure 6.8:
We are given a template domain $\tilde{S}$ with the multi-patch parameterization $\tilde{p}_i$ and the boundary parameterizations $\tilde{b}_{i,j}$, for $i = 1, \ldots, n_v$ and $j = 1, \ldots, d$. In addition, we are given the target domain $S$ with associated boundary parameterizations $b_{i,j}$, where the mapping

$$\beta(\tilde{x}) = b_{i,j} \circ \tilde{b}_{i,j}^{-1}(\tilde{x}), \quad \text{if } \tilde{x} \in \text{domain}(\tilde{b}_{i,j})$$

is assumed to be continuous and bijective. For these given data, we construct a multi-patch parameterization

$$p_i, \quad i = 1, \ldots, n_v,$$

of the target domain $S$, which then defines the template mapping

$$u : \tilde{S} \to S $$

$$u(\tilde{x}) = p_i \circ \tilde{p}_i^{-1}(\tilde{x}), \quad \text{if } \tilde{x} \in \text{domain}(\tilde{p}_i).$$

(6.1)

In order to make this abstract problem accessible to a numerical solution, we first introduce a multi-patch spline space. It consists of tensor-product spline functions, defined on the union of $d$-dimensional unit cubes, which are $C^0$-smooth across the interfaces between these cubes (see [27]). Summing up, the multi-patch parameterization takes the form

$$p : [0, 1]^d \times \{1, \ldots, n_v\} \to S $$

$$p = \sum_{k \in \mathcal{K}} d_k N_k,$$

(6.2)

where $N_k$ are multi-patch B-splines and $\mathcal{K}$ their index set. The individual parameterizations are then obtained as restrictions to the patches $[0, 1]^d \times \{i\}$,

$$p_i(\xi) = p(\xi, i) = \sum_{k \in \mathcal{K}^i} d_k N_k^i(\xi), \quad \xi \in [0, 1]^d,$$

(6.3)
CHAPTER 6. MIDPOINT SUBDIVISION

with tensor-product B-splines $N_k^i(\xi) = N_k(\xi, i)$ and index sets $\mathcal{K}^i$.

We obtain the parameterization (6.2) by first choosing the degrees and knot vectors of the basis functions $N_k$ and then computing the unknown coefficients $d_k$. The latter are found by minimizing the functional

$$F(d) = P(d) + \lambda Q(d),$$

(6.4)

where $d = (d_k)_{k \in \mathcal{K}}$, and $\lambda \in \mathbb{R}^+$. This functional consists of two parts, whose relative influence is controlled by the positive weight $\lambda$. With the abbreviation $U_j = [0, 1]^{j-1} \times \{0\} \times [0, 1]^{d-j}$ we define the penalty term

$$P(d) = \sum_{i=1}^{n_v} \sum_{j=1}^{d} \int_{U_j} \| b_{i,j}(\tilde{x}) - p_i(\tilde{x}) \|^2 \, d\tilde{x},$$

which measures the accuracy of the boundary representation. Additionally we consider a quality measure $Q(d)$, which keeps the parameterization $p$ regular in the interior of $S$. Several possibilities are listed in [41] for the planar case, which admit extensions to the three-dimensional situation.

By minimizing the functional (6.4) we obtain the desired coefficients $d_k$ and thus the parameterization $p$. Together with equations (6.1) and (6.3) we finally obtain the mapping $u$.

Note that the construction of suitable boundary parameterizations $b_{i,j}$ for $d = 3$, which are needed as input for the template mapping problem, is a non-trivial problem itself. We obtain them by solving template mapping problems for $d = 2$ (one for each facet).

The next chapter provides several examples that use the template mapping approach to subdivide given MS$^3$ (output of the ESA) into topological cuboids.
Chapter 7

Examples

In this chapter we will decompose several different solids by the ESA (see page 9). In order to illustrate the power of the different tools introduced throughout this thesis, we will enable or disable certain features while running the algorithm and show the respective outputs with a focus on the differences.

7.1 Regular dodecahedron

The first example is a regular dodecahedron, see Figure 7.1 (left). Since it is a template $\text{MS}^3$ with many rotational symmetries it is perfectly suited for midpoint subdivision. Figure 7.2 shows the output domains obtained by the segmentation algorithm from [23] i.e., without face pre-segmentation and midpoint subdivision. The result is valid, but far from ideal for parameterization. In contrast, the use of midpoint subdivision leads to 20 symmetrical cuboids with planar boundary faces. One of those identical pieces is visualized in Figure 7.1 (right).
Figure 7.1: Left: a regular dodecahedron. Right: One of the 20 symmetrical topological cuboids, which are computed by midpoint subdivision of the dodecahedron. Note that the cutting surfaces are planar, due to the regular shape of the solid.

Figure 7.2: Base solids of the segmentation of the dodecahedron via the algorithm described in [23]. The computed cutting surfaces are all non-planar, some of them being highly curved, and therefore the resulting shapes are hard to parameterize.
7.2 Bi-slotted cube

The second example is a cube with two orthogonal slots on opposite sides. Although the shape of this object is not very complex, an automatic segmentation is not trivial, due to the non-convex shape of the object. Figure 7.3 shows the result of the original segmentation algorithm without face pre-segmentation and without the use of midpoint subdivision. The object is decomposed into three topological cuboids and two prisms with three-sided base surfaces. However, the shapes of those output solids is again not ideally suited for constructing a volumetric parameterization.

Figure 7.3: Segmentation of the bi-slotted cube via the method described in [23].

The result of the extended algorithm is shown in Figure 7.4. Due to the use of face pre-segmentation, the first few cuts are very intuitive, since they use planar cutting surfaces that split the object into 4 symmetrical convex polyhedra with only tri-valent vertices. In the last step, those MS3 are further decomposed into topological cuboids with bi-linear faces by midpoint subdivision. The final decomposition is visualized in Figure 7.5.
Figure 7.4: Segmentation of the bi-slotted cube with pre-segmented faces. Only one half/quarter is shown due to symmetry reasons.

Figure 7.5: Midpoint subdivision of one resulting MS$^3$ after the face pre-segmentation.

7.3 Slotted quarter-pipe

The next example is a non-convex solid, which does not contain a non-convex edge. The original segmentation algorithm [23] uses only one cut that involves
the use of an implicit guiding surface (cf. Section 3) due to the non-convex shape of the object. Since both resulting sub-domains, which are depicted in Figure 7.6, are also strongly non-convex, it is very challenging to represent them as trivariate NURBS patches.

Figure 7.6: Solid is split into two topological cuboids using the methods of [23] and Chapter 3. The resulting shapes are not ideal due to the highly curved cutting surface.

However, since this solid is already an MS$^3$, one can directly use midpoint subdivision, without any previous segmentation steps. This requires the use of a hexagonal prism as template MS$^3$ (see Figure 5.6, third row, third column). Although the 12 resulting cuboids (cf. Figure 7.7) are all valid, four of them contain edges with a fairly small interior angle, which may cause problems in the subsequent parameterization step.

The reason for the occurrence of these small angles is the highly non-convex shape of the object. The face pre-segmentation step resolves this issue, by fostering a split by a horizontal plane in the middle of the solid (cf. Figure 7.8). The subsequent use of midpoint subdivision leads to two symmetrical sets of 10 topological cuboids, which are visualized in Figure 7.9. The midpoint subdivision
requires again a suitable template MS$^3$ (see Figure 5.6, second row, first column). Clearly, the quality of this decomposition improves upon the previous results.

### 7.4 Hammer

The shape of the last example we present resembles the head of a hammer. We cut the hammer along a vertical plane into two symmetrical pieces, thereby eliminating the hole for the handle in order to make the object contractible. The object is depicted in Figure 7.10.

In [22] the authors use Reeb graphs to automatically place the necessary preprocessing cuts, to obtain a collection of contractible solids, which can be handled by the ESA. Due to symmetry reasons it suffices to only consider the left half of
7.4. HAMMER

Figure 7.8: Slotted quarter-pipe is split into two symmetrical halves after the face pre-segmentation step.

Figure 7.9: Midpoint subdivision after the face pre-segmentation.

the pre-processed hammer.

The ESA first splits off the tip of the hammer, see Figure 7.11. The resulting piece contains a vertex with valency 5. Therefore it is segmented further into four smaller pieces (topological prisms with three-sided base surfaces) by using cutting loops that pass through the high-valent vertex.

In the next step, the remainder of the hammer is split into two MS\(^3\). The bottom piece is a convex polyhedron with 10 tri-valent vertices. Therefore the
midpoint subdivision algorithm can be applied directly to decompose this piece into 10 cuboids with only planar and bi-linear faces, see Figure 7.12.

While decomposing the bottom piece is easy, the midpoint subdivision of the non-convex center piece is quite interesting. Based on the segmentation of a template polyhedron (see Figure 5.6, sixth row, second column) it is segmented into 14 topological cuboids, which are depicted in Figure 7.13. However, this results in four sub-domains that contain an edge with a very small interior angle.

A face pre-segmentation of the input solid resolves this issue. After the first two splits into the tip, center and bottom piece, the center piece is then subdivided by a plane into two smaller MS$^3$, which are shown in Figure 7.14. Finally those two sub-solids can be further decomposed by midpoint subdivision with the use of template MS$^3$ (see Figure 5.6, third row, second column). The results are shown in Figure 7.15. The subdivided top half of the center is shown in boundary representation, while the bottom half is depicted as its final multi-patch parameterization by NURBS volumes.

Figure 7.10: The head of an hammer gets pre-processed by splitting it into two symmetrical pieces along a vertical plane through the hole for the handle.
7.4. HAMMER

Figure 7.11: Segmentation of the hammer into $\text{MS}^3$. The tip contains a high-valent vertex and is segmented into four topological prisms. The remainder is split into two $\text{MS}^3$.

Figure 7.12: Midpoint subdivision of the bottom piece.
Figure 7.13: Midpoint subdivision of the center piece of the hammer.

Figure 7.14: Left: Center piece when the face pre-segmentation was enabled. Right: Splitting of the center piece along the artificial auxiliary edges.
Figure 7.15: Left: Final volumetric multi-patch parameterization of the bottom half of the center piece. Right: Final topological cuboids of the top half of the center piece.
Chapter 8

Conclusion and outlook

A representation of the computational domain as a bi- or trivariate spline model is a prerequisite for IgA. However, the standard within the CAD technology is to use boundary representations instead. A conversion into a format suitable for IgA is a challenging task, especially when dealing with volumetric domains. The semi-automatic subdivision steps described in [23] contribute to an isogeometric segmentation algorithm, where a given solid object in boundary representation is transformed into a collection of base solids and finally into topological cuboids. A subsequent parameterization step leads to the desired IgA-suitable domains. In this work we proposed several extensions to the isogeometric segmentation algorithm.

In every step of the segmentation algorithm, a cutting surface is constructed to match the previously computed cutting data. Constructing the surface involves fitting a trimmed NURBS surface patch to a given boundary curve. In order to obtain a valid split, it is crucial for the cutting surface not to intersect the solid’s boundary. Hence, the construction of the surface needs special care. We introduced a novel two-step method that first constructs an implicitly defined guiding surface, and subsequently fits a parameterized trimmed NURBS surface patch that approximates the guiding surface and thus stays inside the solid. The planar version of this technique is an alternative solution to the problem stated and
solved in [3]. This publication studies the construction of a spline curve connecting two points on the boundary of a known planar domain. The resulting curve needs to stay inside the given domain and to be free of self-intersections. A comparison of the runtimes of the two algorithms indicates a substantial improvement by the new method.

Besides the technical realization of the segmentation step, we also discussed possible improvements of the quality of the output shapes. While solids with a high number of faces seem to be more difficult to subdivide, they offer more flexibility in choosing the cutting loop appropriately. On the contrary, there are by far fewer possible cutting loops in solids of small combinatorial size. Hence, solids with small edge graphs and more complicated faces are generally very challenging to subdivide. By introducing artificial edges, one increases the flexibility of the cutting loop, thus allowing for different cuts. Placing these additional edges in carefully chosen positions may improve the quality of the sub-solids essentially. We offered two different constructions of these edges.

Another drawback of the original method is the restriction to only one cutting surface at a time. It might be favorable to perform multiple splits at once. This can be achieved by using midpoint subdivision. However, this method can only be applied to solids with certain properties. Midpoint subdivision returns a collection of topological cuboids, if and only if every vertex of the input solid possesses valency three. We showed that the segmentation algorithm is capable of subdividing an input solid into a collection of midpoint subdivision suitable solids. While performing midpoint subdivision for a convex polyhedron is straightforward, its application to more general (possibly non-convex) domains requires special techniques. We proposed a template mapping approach to resolve this issue. As a by-product, we directly obtain the parameterization of the base solids as volumetric multi-patch spline domains.

Despite these improvements, there are still limitations of the isogeometric segmentation algorithm. First, due to the exponential growth of the number of possible cutting loops with the combinatorial size of the input solid, our method
is restricted to solids of rather limited size. Second, although the constructed solids will be a valid decomposition of the input domain, their interfaces are not guaranteed to possess matching parameterizations. Thus new challenges arise when performing numerical simulations. These can be dealt with by employing discontinuous Galerkin methods [42] or similar approaches for weak coupling [43] of sub-domains in IgA. Alternatively, one may post-process the solids to generate matching interfaces. Finally, the segmentation and parameterization process relies on the choice of certain user-defined parameters, which influence the quality of the output. We are currently exploring how to use techniques of machine learning to automatize this process.
Bibliography


