Bavarian Graduate School of Computational Engineering

Technische Universität München

BGCE Honours project report

CAD-integrated Topology Optimization

Authors: S. Joshi,
J.C. Medina,
F. Menhorn,
S. Reiz,
B. Rüth,
E. Wannerberg,
A. Yurova
Advisors: Arash Bakhtiari (TUM),
Dirk Hartmann (Siemens),
Utz Wever (Siemens)

Preface

The Bavarian Graduate School of Computational Engineering (BGCE) honours project at the Computational Science and Engineering (CSE) Institute of Technische Universität München (TUM) is a 10-month project where students conduct research on cutting-edge topics in the field of Computational Engineering, in cooperation with a partner in industry or academia. The 2015-16 project is titled *CAD-Integrated Topology Optimization* and is initiated and supervised in a cooperation between TUM and Siemens AG in Munich.

Acknowledgments

This Honour's project is carried out under the supervision of Dr. Dirk Hartmann, Dr. Utz Wever from the side of the customer (Siemens AG) and Arash Bakhtiari (TUM). We would like to thank Bavarian Graduate School of Computational Engineering for providing us an opportunity to participate in a project closely related to the industry in a highly relevant and challenging topic.

Contents

Outline and Overview

Purpose of the document

The purpose of this document is to describe the *CAD-integrated Topology Optimization* software tool along with the theoretical background it relies on.

Document overview

The document is arranged in chapters. The current and first chapter provides an overview of the document structure, brief introduction to the topic, motivation and administrative details of the project. The second chapter provides a thorough review of the theory and techniques used within the project.

CHAPTER 1: INTRODUCTION
This chapter presents an overview of the motivation behind *CAD–integrated Topology Optimization*, including the current state of the art. It also provides general organizational information about the project execution, such as timeline and structure.

CHAPTER 2: BACKGROUND THEORY
This chapter introduces the theoretical background for the implementation of the *CAD–integrated Topology Optimization* tool. It consists of the five parts, describing the essential steps of the *Topology Optimization* pipeline. Furthermore, the detailed description of algorithms and libraries used in each step is given.

CHAPTER 3: IMPLEMENTATION
This chapter provides details on the implementation and structure of the *CAD–integrated Topology Optimization* tool itself. The different parts of the *Topology Optimization* pipeline are presented along with underlying implementation details.

Part I

Introduction

1 Introduction

In this chapter a motivation for the project and a short description of the problem task is stated. A brief introduction to topology optimisation and the project structure is also given.

1.1 Motivation

A common problem in product design is to create a functional structure using as few material as possible. Three decades ago engineering design versions were drawn, prototypes created and experimental test performed. Nowadays, the field of topology optimisation simplifies this process and has become a great help in all fields of engineering.

Topology optimisation tackles the problem of material distribution in a structure in order to fulfil certain target loads. Several topology optimisation open-source tools exist that are ready to use; however, it is still a challenge to incorporate these tools handily in the design process. The idea of this project is to allow these tools to work directly from CAD files and to transfer the obtained mesh based solution back to the CAD world. Unfortunately, at the moment, there is no open source solution for the conversion mesh based geometry to the spline-based CAD format (where in our case Non-Uniform B–Splines, a.k.a. NURBS, are used). The would-be straightforward approach – to convert each triangle of mesh geometry directly to CAD format – results in enormous file sizes. One of the biggest challenges of this project is thus to develop a conversion tool that feasibly provides a useful CAD-representation of the optimized surface.

1.2 Project structure

The aim of the project is to provide a tool that allows to *Topology Optimisation* without leaving CAD–framework. Therefore, main goals of the project are as follows:

- Implementation of *Topology Optimisation* by using and extending the available open source libraries
- Development of a flexible tool for the conversion of the optimized surface back to the CAD format.

The duration of the project has been set to 10 months. Hence, this project has been divided into 4 phases:

Phase 1: Getting familiar with the topic and agreement on the project specification.

Phase 2: Implementation of the first part of the pipeline (Topology Optimisation from CAD surface using existing tools); investigating the tools and algorithms available for the conversion of the geometry generated after topology optimisation back to CAD format (later referred as

NURBS fitting pipeline); prototyping (using MATLAB) and evaluating of found results.

Phase 3: Implementing the prototypes, developed on the previous stage, using a non-proprietary languages, such as Python or C++, extension of the NURBS fitting algorithm to more complex cases, finalising the first part of the pipeline.

Phase 4: Implementation of the extended NURBS fitting algorithm, integrating it with the *Topology Optimisation* part and delivering the final product to costumer.

Part II

Background Theory

2 Background Theory

In this chapter the theoretical background for the implementation of the *CAD–integrated Topology Optimization* tool is introduced. Also the detailed description of algorithms used in each step of the *Topology Optimization* pipeline is given.

2.1 CAD overview

Computer Aided Design (CAD) refers to the process of designing a product using a computer system. Before CAD applications were used, products were designed using a sketch board. It was a challenge to incorporate changes in the construction drafts as well as to keep documentations up to date; hence, it was no surprise that CAD systems spread rapidly across all design development branches. Computer aided design is now of irreplaceable use in architecture, mechanical, electrical and civil engineering.

Depending on the discipline, different requirements are set on the virtual model. One may imagine that in a civil engineering model of a building a 2D floor plan is often sufficient; however, in the design of a mechanical motor a 3D model is always necessary. Given these circumstances, various CAD software bundles evolved in the different disciplines with completely different modelling approaches. Besides the geometry representation additional parameters, such as material properties or manufacturing information, are stored. In order to move between different data structures standardized exchange interfaces are commonly used.

This section presents the relevant geometrical and computational aspects of CAD for the project; for a more thorough introduction we refer to [**?**].

2.1.1 Geometry representations

In general, two different ways of describing a geometry are used in CAD systems: a *constructive solid geometry* (CSG) or a *boundary representation* (BREP). Other approaches, such as a complete voxelised geometry are not common due to extensive memory consumption.

Constructive solid geometry

One way of representing a geometry in CAD is the approach of *constructive solid geometry*. The basic idea is to start from a set of primitives, e.g. spheres, cylinders and/or cubes. Basic Boolean operations link these primitives towards a complex geometry, as illustrated in **??**.

Key advantages of this format is the precise representation using very little storage memory. However, not all desired forms can be represented by CSG and hence, a second type of geometry description is needed.

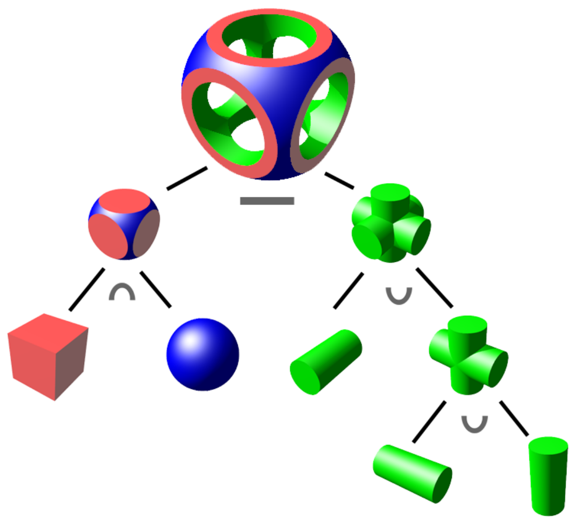

Figure 2.1: CSG object tree. The picture shows the construction of a complex object from a cube, a sphere, and a set of cylinders. Figure taken from [**?**].

Boundary representation

A different kind of modelling approach is the so-called *boundary representation*. Instead of storing the geometry information as geometrical objects, BREP formats only save the boundary surface of the body. The interior is assumed to be uniformly filled. Especially in complex geometries, this approach simplifies the model to such an extent, that the amount of data becomes much easier to handle. Surfaces can then be for example stored as a set of triangles (as in STL files, see **??** below) or in NURBS patches (see **??**). Furthermore, holes in the body are made possible by saving the surface normal of the respective boundary.

By the boundary representation arbitrary geometries can be created. While the data sizes are commonly larger than in CSG representation, BREP files are usually easier to work with. One also has to keep in mind, that non-physical geometries can result from BREP formats through a not closed surface.

2.1.2 Data exchange interfaces

CAD software programs usually use their own data formats; in order to exchange models standardized interface formats have been developed. Geometric models are compressed to certain geometry descriptions; transferring additional information, such as material properties or manufacturing information, is in general a difficult task and in some exchange file formats even prohibited. A few common exchange file types are described below, as also compared in [**?**].

STL file format

The STL (from *ST*ereo*L*ithography) file format describes the model only by its boundary and is thus a BREP format. The idea behind it files is simple. The geometric model is discretized into a cloud of points, where sets of three vertices form a triangle; hence, a connected surface

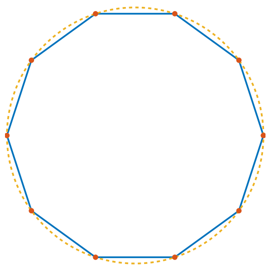

Figure 2.2: STL discretization for a circle, self-made in MATLAB. Note that the circle cannot be exactly represented by the vertices and edges.

of triangles emerges which describes the geometry. The procedure is shown in **??** for a two dimensional circle. The aforementioned triangles boil down to lines in two dimensions. The advantages and disadvantages of this approach become clear: It can be applied to an arbitrary geometry, but accuracy causes difficulties. In order to transfer high precision geometries many vertices are necessary, resulting in big files. Still, as can be seen in the figure, a perfect circle can never be represented.

ASCII STL files begin with a name and the data on the triangles is constructed as follows:

- a facet normal pointing outward
- a sequence of vertex coordinates

As this is the only information provided, no additional data such as material properties are transferred through STL files, reducing file size but also range of usage.

STEP file format

To overcome issues of insufficient precision also more elaborate exchange formats exist; these save e.g. a circle as a parameter where no discretization step is involved. Also, the possibility of passing additional parameter information (e.g. density, manufacturing information) is required by certain users. Popular file types that offer these two functionalities are STEP and IGES files.

The STEP file format is a newly developed CAD data exchange standard and documented in the ISO 10303 norm. On the contrary to STL files it uses a combination of CSG and BREP to store the geometry. Additional information (e.g. density) are passed through attribute sets that are stored besides geometry instances (e.g. a circle). A key disadvantage, however, is that STEP files carry much redundant information [**?**].

IGES file format

The Initial Graphics Exchange Specification (IGES) is an American National Standard since 1981 to exchange graphics information. Similar to the STEP format it uses a combination of CSG and BREP for the geometry representation. Instead of storing a set of manufacturing information as done in the STEP file, the IGES is build only to exchange graphics information. For example, the step file transfers a physical density information; in the IGES format the only additional parameter store node coloring information. Consequentially, IGES file sizes are significantly smaller compared to the STEP file format [**?**].

The IGES file format contains five different sections: a *Start*, *Global*, *Directory Entry*, *Parameter Data* and *Terminate* section. The *Start* and *Global* section are used for naming and part information. In the *Directory Entry* section additional information like the node color is saved. The *Parameter Data* section is used for storing the coordinate points; the *Terminate* section signals the end of the file [**?**].

2.2 Topology Optimisation

Topology Optimization describes the process of finding the optimal distribution of a limited amount of material for a given area or volume based on a predefined constraint/minimization problem. Possible optimization goals are for example [**?**]:

- **Minimum compliance** which seeks to find the optimal distribution of material that returns the stiffest possible structure. The structure is thereby subjected to loads (forces) and supports (boundary conditions). By maximizing the stiffness, the compliance is minimized. This is also analogous to minimizing the stress energy stored by the applied loads.
- **Heat conduction** tries to optimize the domain of a conductive material with respect to conductivity for the purpose of heat transfer. This maximization problem is the same as minimizing the temperature gradient over the domain– a poor conductor will create a large gradient.
- **Mechanism synthesis**' objective is to obtain a device that can convert an input displacement in one location to an output displacement in another location. Topology Optimization hereby seeks the optimal design which maximizes the output force for a given input or, respectively, minimizes the input force for a given output.

As one can already imagine by this short list of optimization goals, Topology Optimization has a wide field of possible applications. Hence, it has become a well established technology used by engineers in the fields of aeronautics, civil, materials, mechanical and structural optimization. Furthermore, due to the rising significance of additive manufacturing techniques in industry, the realisation of complex optimized designs is now much easier.

2.2.1 Minimum compliance: Problem formulation

In order to constrain the resulting structure as little as possible, the formulation of the topology optimization problem is generally given as follows: for a given set of external fixture points, external loads and/or body forces, the distribution of material within the reference domain should be found such that the structure has maximum stiffness. This is obtained when the structure has the minimum energy stored by external work for the applied forces. The problem is also usually formed to allow for regions in the domain to be specified as filled or empty of material (see **??**).

The formulation allows the problem to be cast as finding a displacement field u and a stiffness tensor field E that is in equilibrium with the applied loads, and that minimizes the external work done by the forces.

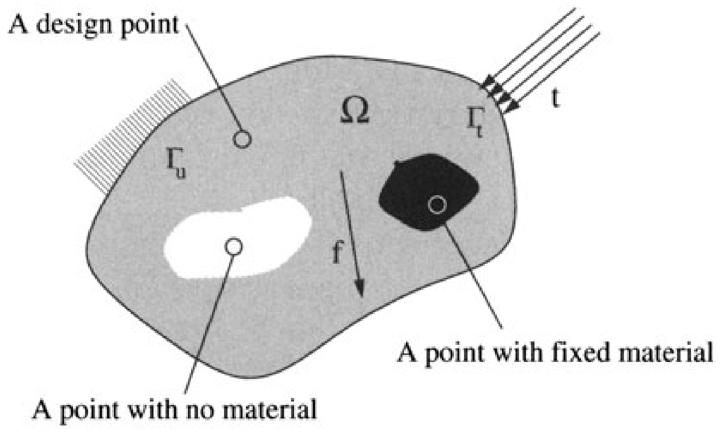

Figure 2.3: The reference domain Ω for the minimum compliance problem. The problem is formulated such that for a set of external loads t on boundaries λ_t, body forces f and a set of fixed support points λ_u, the material distribution within Ω is such that the stiffness with regards to these loads and forces is maximal and the energy stored by the application of those forces is minimal. The problem also allows defining areas which either cannot or must be filled with material. Figure taken from [**?**].

2.2.2 Physical and mathematical simplifications

To turn this into a more tractable mathematical problem, a few physical assumptions are also typically made: the material be isotropic and linearly elastic. From the assumptions of isotropy and linear elasticity of the material, the stiffness field becomes a constant of the material, defined where there is material in the domain.

The problem is also easy to cast into a weak form. First of all, we compute the integrated internal virtual work and external work. The former is the work of deforming the elastic material from equilibrium by an admissible displacement. The latter is done by the loads and forces to bring out this displacement. Having computed these, we set them equal to one another in order to conserve energy. As a result we obtain an equation that relates the equilibrium displacement, stiffness tensor, and the forces and loads. We then cast this into the weak form, which can be solved using Finite Element Methods (FEM). These can also incorporate the calculation of the external work done.

2.2.3 SIMP: Solid Isotropic Material with Penalization

As described in the previous section, we try to minimise the external work done by looking at different material distributions. However, the usual problem of finding an optimum arises: where to look? After discretising the domain with FEM, the possibilities of where to put material at least are not infinite– but they still grow exponentially with the number of elements; hence, trying out one-by-one is not going to prove efficient. One popular way of recasting the problem to allow for easier solving is the SIMP model. Here, instead of either being present or not at a point, the material presence can take a continuous set of values between one and zero. The total final volume is then obtained and fixed by integrating this presence variable over the domain, instead of constraining the allowed occupied space. This allows for the interpretation as some

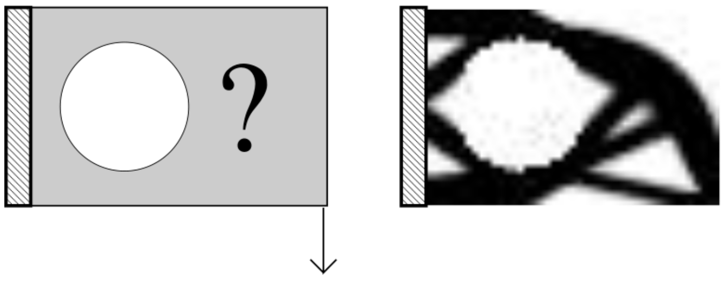

Figure 2.4: Topology optimisation of end-loaded cantilever with fixed hole. The optimisation of a loaded cantilever is one of the model problems in topology optimisation, due to its simplicity and its multitude of used solutions throughout the history of engineering. The picture is taken from a known paper by Sigmund [**?**] where a 99-line Matlab code for topology optimisation is introduced.

kind of density.

In order to still obtain topologies where material is predominant in certain areas– of densities one, with the rest being empty at densities close to zero– a ”penalty” is applied to the intermediate values. This is effected by raising the density to a power > 1 in the elastic energy calculation, but not in the volume calculation. That way, an intermediate density value provides less elastic support, but still ”costs” as much volume, and will thus be suboptimal.

2.2.4 Solution and implementation

In typical implementations, a heuristic iterative scheme is then used for finding a solution. The optimal solution is assumed to have all present parts stressed (as they would otherwise be unnecessary, not providing any support). Thus, at places where the elastic energy is high, material is added if possible, and where it is low, material is likewise removed, with the values ”high” and ”low” being determined dynamically to keep the total volume constraint.

This whole scheme is one of the simpler topology optimisation schemes to implement, and has been done so in several pieces of open-source software, including a known 99-line Matlab code by Sigmund [**?**] and ToPy described in **??**. An example optimised topology is shown in **??**. For an extended explanation and discussion, as well as further alternative methods for topology optimisation, the interested reader is referred to [**?**].

2.3 From Voxels to a surface representation

After obtaining the optimized voxel data, the next step is to generate a *mesh based geometry*, which is a mesh representation of the volume data. This will be used in conversion of our surface back to CAD format, in particular, to NURBS representation. In order to achieve it, the data will be represented by a contour of a smooth function, rendering an isosurface. Based on the algorithm used, the mesh can be composed of triangles or quadrilaterals, also known as quads. A well-know approach to tackle this problem is the *Marching Cubes* technique.

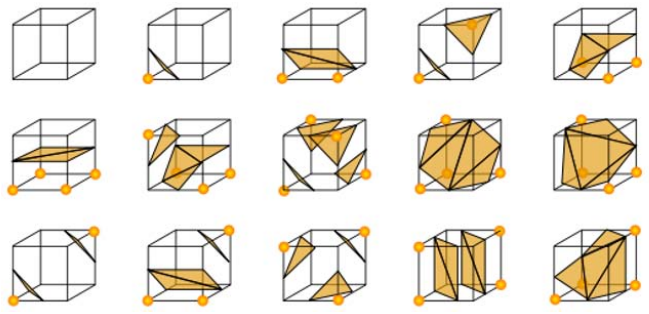

Figure 2.5: Base cases of Marching Cubes. Figure taken from [**?**]

Marching Cubes

The *Marching Cubes* algorithm [**?**] takes as an input a regular volumetric data set and extracts a mesh. It divides the space into cubes, which are defined by the volume information. Each cube has scalar information on its vertices, the value is equal or above a marked isovalue. Therefore each of the eight vertices of a cube can be marked or unmarked. According to these values vertices are drawn on the edges of the cube at calculated points with the use of interpolation.

By connecting the vertices we obtain a polygon on each cube. There are 256 possible cases. Many of these can be constructed by rotating or reflecting of previous cases. Therefore there are 15 base cases which represent all the possibilities of the marching cubes (see **??**). The original algorithm presents two main problems. First, it does not guarantee neither correctness nor topological consistency, which means that holes may appear on the surface due to inaccurate base case selection. Second, another problem is ambiguity, which appears when two base cases are possible and the algorithm chooses the incorrect one. There are many extended Marching Cubes algorithms that tackle the problems of the original one, getting rid of the ambiguities and providing correctness.

Dual Contouring

The idea of the *Dual Contouring* algorithm is similar to Marching Cubes, but instead of generating vertices on the edges of the cubes, it locates them inside the cube. **??** shows the basic differences in both approaches. The vertices associated with the four contiguous cubes are joined and form a quad. The question now is which place inside the cube is the ideal one to insert each vertex. Different dual algorithms are classified according to the answer for this question. Dual Contouring in particular generates a vertex positioned at the minimizer of a quadratic function which depends on the intersection points and normals. Therefore the method needs Hermite data (exact intersection of points and normals) to work with. The quadratic function is defined in [**?**] as follows:

$$E(x) = x^T A^T A x - 2x^T A^T b + b^T b$$

Where A is a matrix whose rows are the normals and b is a vector whose entries are the product of normals and intersection points. To solve this system, a numerical treatment is needed. As proposed in [**?**] the best approach is to compute the singular value decomposition of A and form the pseudo-inverse by truncating its small singular values.

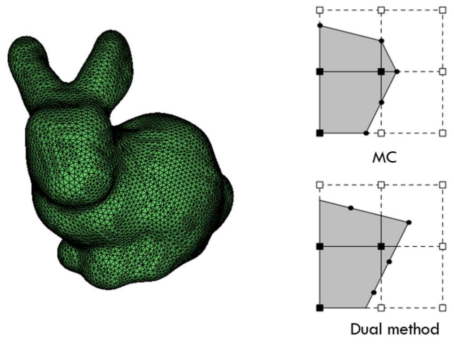

Figure 2.6: *Left:* The famous Stanford Bunny after application of Marching Cubes. *Right:* Main difference between MC and Dual methods. Figures taken from [?]

The main advantage of this method over MC is the acquisition of better aspect ratios. On the other hand the need of Hermite Data represents a disadvantage. Furthermore, there is no open source algorithm that implements the Dual Contouring scheme.

.

2.4 From a surface representation to NURBS

Parametrised geometries are often given in terms of Non-Uniform B–Spline (NURBS) descriptions (see, for example, documentation of FreeCAD software [?]). To define NURBS from a mathematical standpoint, we first define so-called *Bezier curves* [?] which we will use later for the definition of NURBS.

2.4.1 Bezier Curves

Bezier curve is a *parametric* curve, which is often used for producing a smooth approximation of a given set of data points.

An analytical expression for the Bezier curve is given by:

$$\vec{B}(t) = \sum_{i=0}^{n} b_i^n(t)\vec{P}_i$$

where $\vec{P}_i$ is the i–th control point (we have $n+1$ control points). And

$$b_i^n(t) = \left(\begin{array}{c} n \\ i \end{array} \right) (1-t)^{(n-i)} t^i$$

is the i–the Bernstein polynomial (see [?]) of degree n.

Additionally to the expression with the Bernstein polynomials, one can use a recursion formula (so-called **de Casteljau Algorithm** [?]) for the construction of the Bezier curve, which we will not cover here. For more information see [?].

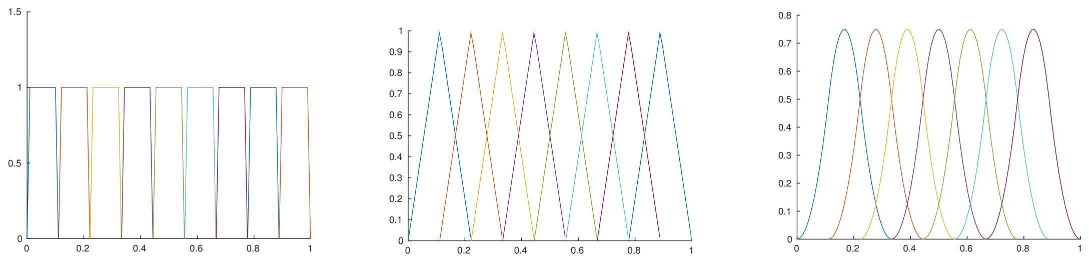
(a) B-spline basis for $p = 0$ (b) B-spline basis for $p = 1$ (c) B-spline basis for $p = 2$

Figure 2.7: B-spline basis functions

Analogically to Bezier curves, but with $n \cdot m$ Points $\vec{P}_{i,j}$, one can define a *Bezier surface*, given by the analytical expression

$$\vec{S}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_i^n(u) b_j^m(v) \vec{P}_{i,j}$$

Note, that Bezier curves and surfaces may be unstable – minor changes in control points might lead to major global changes.

2.4.2 NURBS basis functions

Extending the idea, described in previous section, one could use NURBS [?] basis functions instead of simple Bezier curves.

Unlike Bezier curves, for the B–spline basis a parameter domain is subdivided with so-called *knots*. In particular, given the parameter domain $[u_0, u_m]$ (in 1D), *knot vector* is given by $u_0 \le u_1 \le ... \le u_m$. In most cases $u_0 = 0, u_m = 1$, so we get unit interval for our parameter values. Recall, that, N in NURBS stands for *non-uniform*. This means, that our knots $u_0, ..., u_m$ are not equidistant.

Given *knot vector* $[u_0, u_m]$ and a degree of B-spline p one can find i-th B-spline basis function recursively as follows [?]:

$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \le u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{2.2}$$

For $p = 0$ we get just step functions (see fig. **??**), for $p = 1$ we get familiar hat functions (see fig. **??**). Quadratic basis looks more complicated (fig. **??**).

Given these basis functions, Non-Uniform Rational B-Spline (NURBS) curve is given by:

$$C(u) = \frac{\sum_{i=1}^{k} N_{i,n} \omega_i P_i}{\sum_{i=1}^{k} N_{i,n} \omega_i}, \tag{2.3}$$

where k is number of points, $\{P_i\}$ are given control points.

B–splines are have the following properties, which are useful for our problem:

- Degree n and number of control points $\vec{P}_{i\cdots m}$ are independent.
- B–Splines only change locally (depends on the degree n) when a control point is changed.

Analogically, one can define B–spline surfaces. For more information about NURBS see [**?**].

2.4.3 Minimization problem

Now, once we defined all necessary tools, we proceed to the fitting problem.
The goal is to fit in a parametric curve to the set of given data points. In our case our given set of points is a mesh, obtained from surface contouring.

2.4.4 Minimization Problem: Bezier curve

First we want to find a Bezier–curve $\vec{B}_n(t)$ of degree n which is approximating a given spline $\vec{s}_m(t)$ defined by m points in a optimal way. For this purpose we want to minimize the L2–error. This leads to the minimization problem:

$$\text{find} \min_{\vec{B}_n \in \mathbb{B}_n} \left\| \vec{B}_n - \vec{s}_m \right\|_{L_2} \tag{2.4}$$

Minimizing the L2–norm is equal to minimizing the functional:

$$F(\vec{B}_n) = \int_{t=0}^{t=1} \left(\vec{B}_n - \vec{s}_m \right)^2 \mathrm{dt} \tag{2.5}$$

Using the variational principle we get the the system of linear equations

$$Aa = b \tag{2.6}$$

where A is the matrix of pair-wise scalar products of basis functions (Bernstein polynomials), b is the vector of scalar products of the given spline s_m and basis functions and a is a required vector of coefficients for Bezier curve representation.

2.4.5 Minimization problem (least squares): NURBS

Although the approach used above showed good results, it appears to be very computationally intensive. Analogically, one could reduce the original problem to the *linear regression problem*, which allows us to reduce computational costs. Also, to improve the locality of our solution, from now on we are going to use NURBS basis functions with weights $\omega_j = 1$ instead of Bezier curves. For this purpose, we adopted an algorithm, provided in [**?**].

Let X^0 be the $n \times 2$ matrix of the given set of points, N^p - the basis functions of degree p ($n \times (n+p)$ matrix, where n - number of points), P^0 - the control points ($(n+p) \times 2$ matrix).

The original problem can be written as:

$$X_i^0 = \sum_{j=1}^{n+p} P_j^0 N_{i,j}^p, \quad i \in \{1, .., n\} \tag{2.7}$$

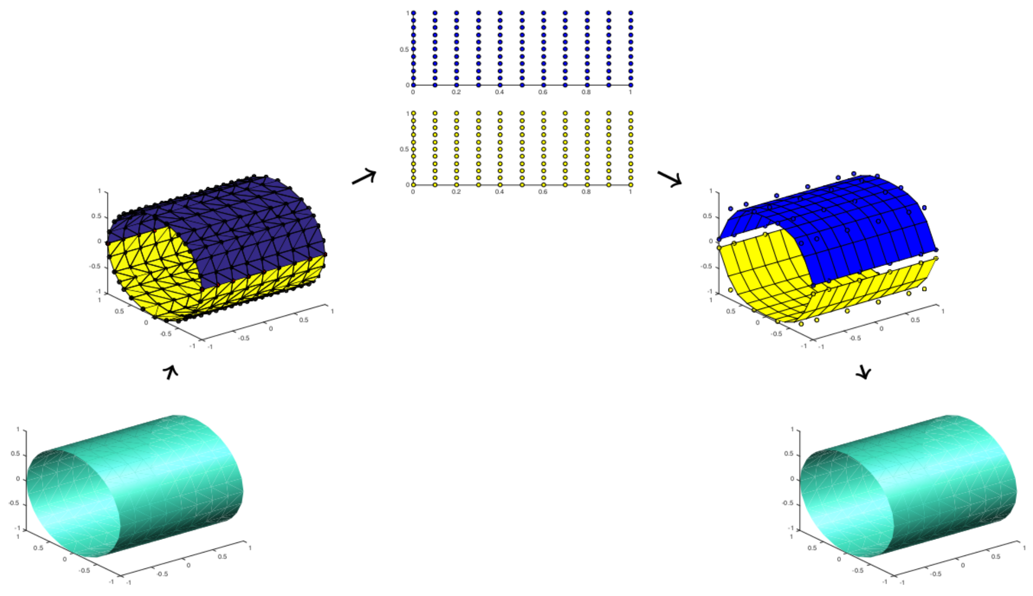

Figure 2.8: NURBS fitting pipeline

Or, in short:

$$X^0 = N^p P^0 \tag{2.8}$$

The above system needs to be solved for the unknown P^0. One of the ways to solve it is to use SVD decomposition.

2.4.6 Fitting pipeline

Since the geometry obtained after topology optimization can be arbitrary complex, we might not be able to find a good fit using only one patch. We seek a multi step algorithm, allowing us to break the overall big problem into smaller problems, which can be handled relatively easy. Based on the algorithm described in [**?**], our overall fitting pipeline looks as follows (see fig. **??**):

- Patch selection (breaking our problem in small pieces which can be solved using least squares)
- Parametrization of obtained patches
- B–spline fitting using least squares
- Smooth connection of patches
- Conversion back to CAD

The pipeline given above, once implemented, will provide us with a flexible algorithm for converting an arbitrary complex mesh based geometry into NURBS and, hence, CAD–representation.

Part III

Implementation

3 Implementation

3.1 From CAD to Voxels

One of the hurdles with most state-of-the-art open source topology optimization tools is their input format, where many of them (including ToPy, our topology optimizer of choice) require input to be specified as a 3-dimensional voxel grid. Presence (or absence) of material in these voxels is defined by a boolean variable, and boundary conditions are imposed on the appropriate locations. An example of voxelized data can be seen in **??**.

In the variety of toolboxes available for voxelization, we decided on the *Common Versatile Multi-purpose Library for C++* (CVMLCPP). This is a collection of mathematical algorithms whose objective is "to eliminate this redundancy by offering high-quality implementations of commonly needed functionality" [**?**]. The library offers an easy-to-use voxelizer, which we use for conversion of CAD input to a boolean voxel grid.

Another very popular open-source choice is the 3D modeling toolbox is OpenCascade [**?**], a versatile library with huge amounts of functionality. Although this also offers a voxelizer, we decided that its additional functionality did not justify its disadvantages in size and cumbersome installation requirements on for example a linux system.

In terms of implementation, the only thing required is the installation of the CVMLCPP library. The voxelizer is then included as a callable binary, that takes STL-file input (**??**) and converts it to a .dat binary file with dimension and voxel information, with specifiable voxel size. An example result of using the voxelizer is shown in **??**.

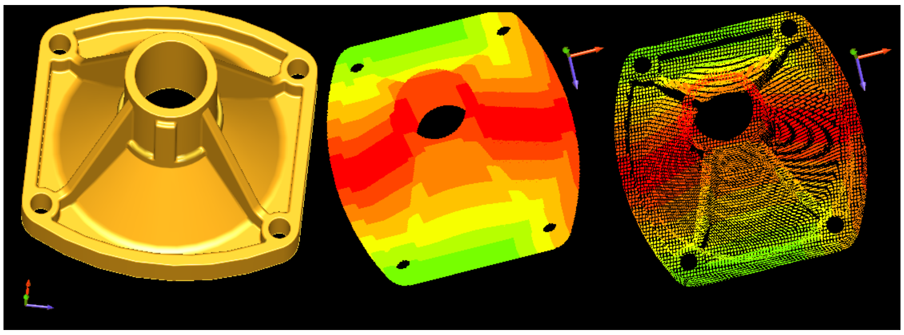

Figure 3.1: A shape and its voxel representation. *Leftmost picture*: The original parametrized shape. *Rightmost picture*: The voxel representation. Picture from OpenCascade [**?**].

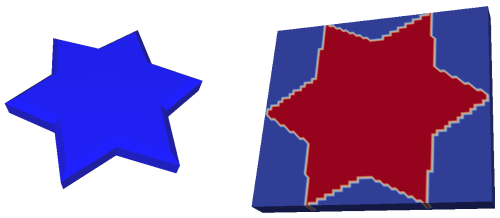

Figure 3.2: The STL geometry of a star (left) and its voxelized form (right) obtained via the CVMLCPP voxelizer, visualized by Paraview [?].

3.2 Topology Optimization

Due to the good range of topology optimization software available, we decided to adapt an open-source topology optimizer to our needs, namely *ToPy*.

ToPy [?] is a python library/program, written by William Hunter and documented in [?], implementing the SIMP model and method described in ??. It is based on the 99-line Matlab code by Sigmund's for minimum compliance [?]. The program can optimize the previously named problem types: minimum compliance, heat conduction and mechanism synthesis– in 2D as well as 3D. It uses highly optimized open source python libraries such as Pysparse [?] and Numpy [?], leading to improved speed, porta- and scalability.

We use ToPy as a black-box topology optimizer. This means, we launch the program with an input file based on our scenario and let ToPy run. Next, as soon as the output is available, we proceed with the next step. The intention is to create separate modules to be able to plug in different solvers later on. In order to specify our input, we wrote an auxiliary program taking a voxelized CAD design provided by CVMLCPP as input, outputting a .tpd file with geometry, load and fixture information in the format required by ToPy.

Results of the topology optimization process can be seen in figure ??. Here, a star was given as input from a stl-file. We set the voxels in the star's points as fixtures, while we set a load in the middle, in the direction normal to the plane of the star. As can be seen, the optimization process ”cuts” away unnecessary material in-between the corners and even in the middle of the material, returning an optimally stiff structure for the chosen remaining volume fraction.

3.3 From optimized voxels to surface representation

As was mentioned above, surface extraction is an intermediate step after Topology Optimization and NURBS representation in order to facilitate the conversion. In terms of implementing the surface extraction, we used VTK.

The VTK Toolbox is an open–source tool, providing algorithms for ”3D computer graphics, image processing, and visualization” [?]. Among the variety of tools, VTK offers algorithms that allow us to obtain a surface representation from voxel data. Among these algorithms, we could find Marching Cubes, Dual Contouring and also a Decimation method, which is useful for reducing the data size further for the NURBS-representation step.

Since *Marching Cubes* algorithm only works with ImageData and PolyData, it is inapplicable to our case of unstructured grid data. For structured and unstructured grids the tool to ren-

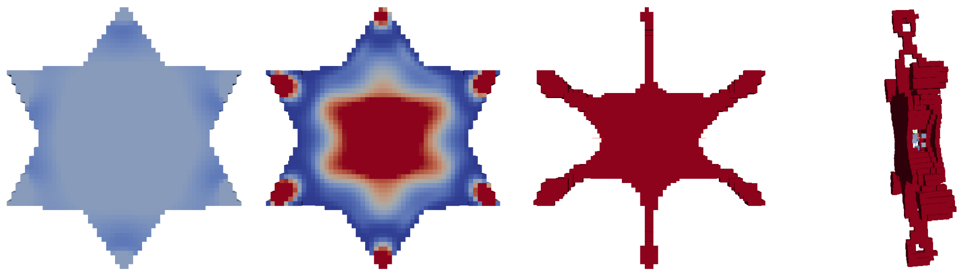
Figure 3.3: Topology Optimization by ToPy [**?**], with minimum compliance. *From left to right*: increasing number of SIMP iterations until convergence. The star-shaped structure was given by an STL-file which was processed into input readable by ToPy, with fixtures in the corners, and a load in the middle. Throughout the SIMP iterations, one can see how material from the less dense regions (blue) is concentrated into denser regions (red) that carry the load. The last picture gives a rotated view, to illustrate how material has been eliminated even from the inside of the star.

der the isosurface is the *Contour Filter* tool. Unfortunately, the documentation does not present which algorithm the tool uses. It can be inferred that it is an extended *Marching Cubes* algorithm. The *Contour Filtering* works fine but the visualization of our data was still not possible and an intermediate step was needed. We used the *Implicit Modelling* tool which is a filter that computes the distance from the input geometry to the points of an output structured point set. This distance function can then be "contoured" to generate new, offset surfaces from the original geometry. Although this approach allowed the visualisation, some crucial information was lost. In particular, holes are not represented in the final model.

A further idea to solve this problem is to first convert the volume data into point data and only then present it to the *Contour Filtering* tool (**??**).

In order to reduce computational costs of the following *NURBS fitting* process, presented in the next section, we need to create a coarser mesh from the fine one. The number of triangles that represent the isosurface can be reduced with the *Decimation* tool. A smoothing step is necessary in between to get the new connections right. The top part of figure **??** shows a 50 % reduction of the triangles, a noticeable difference can not be perceived. On the lower part a 90 % reduction is obtained, it is nevertheless still difficult to see a difference. Triangle meshes can be easily coarsened since there are many open source algorithms that simplify the triangles. VTK has the decimation tool which works for 3D triangle data.

3.4 From extracted surface to NURBS representation: an algorithmic journey

As of yet, there is no open source software which provides the conversion from a *mesh-based* geometry to NURBS representation. Hence, one of the main challenges of both the algorithmic and implementation part of this project is to develop one from scratch. Due to a variety of possible approaches to tackle this problem (e.g. [**?**],[**?**]), we have conducted a profound prototyping work. In order to avoid a cumbersome and time-consuming implementation overheads during

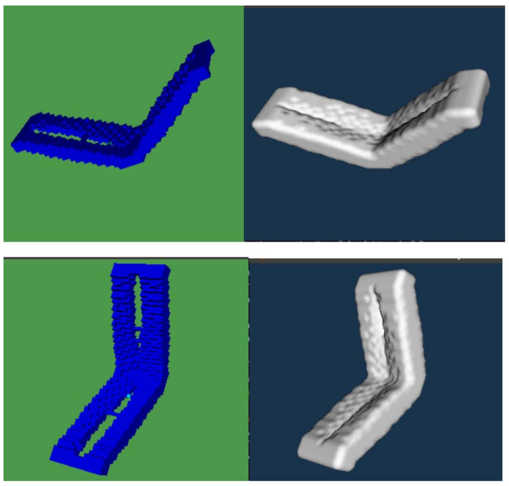

Figure 3.4: Contour Filtering tool after Implicit Modelling

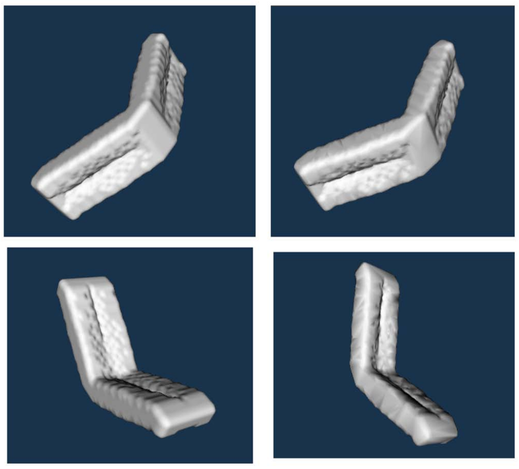

Figure 3.5: Decimation of triangles. *Top:* 50% *Lower:* 90%

the prototyping phase, we have used MATLAB [**?**]. Once the algorithms to be used are finalized, the prototypes are to be implemented using a non-proprietary language, such as Python or C++.