Bavarian Graduate School of Computational Engineering

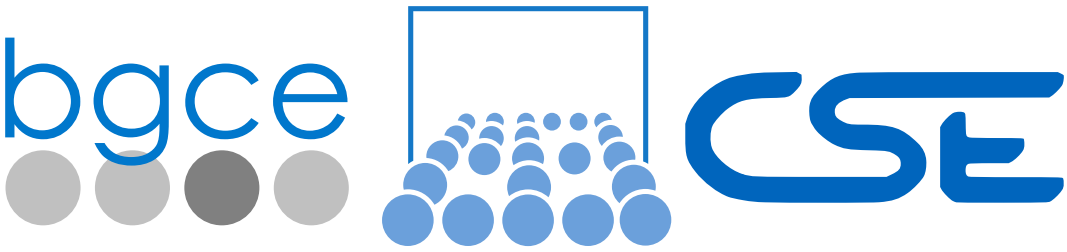
Technische Universität München

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, Dirk an Utz...



Preface

This report and the work described herein is the result from the 2015-2016 BGCE Honours Project within CSE at TUM, a 10-month project where students attending BGCE, the Bavarian Graduate School of Computational Engineering Elite Master's Programme, conduct a research project on cutting-edge topics in within the field of Computational Engineering, in cooperation with a partner in industry or academia. This project, *CAD-Integrated Topology Optimization*, was carried out by the BGCE students at the Computational Science and Engineering Master's Programme (CSE) at the Technische Universität München (TUM), initiated and supervised in a cooperation between TUM and Siemens in Munich.

Acknowledgments

This Honour's project is carried out under the supervision of Dr. Dirk Hartmann, (Topical Advisor), Dr. Utz Wever (Topical Advisor) from the side of the customer (Siemens AG) and Arash Bakhtiari (Team Advisor) from the side of the Technische Universität München. 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

This document describes an overall process of *Topology Optimization* with the following conversion of an obtained surface back to CAD. It includes the theoretical background, required for the successful completions of the project, a brief description of external libraries used within the project and the detailed information about algorithms developed for the implementation of the functionality not provided by open source libraries available.

Document overview

The document is arranged in chapters. The current and first chapter provides a 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.

Part I: Introduction

CHAPTER 1: INTRODUCTION

This chapter presents an overview of the general purpose of *CAD–integrated Topology optimization* and a current state of the art in this field. Also, in this chapter one can find a general administrative information about the project, such as timeline and structure.

Part II: Background theory

CHAPTER 2: BACKGROUND THEORY

This chapter provides the theoretical background for the implementation of *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 on each step is given.

Part I.

Introduction

1. Introduction

1.1. Motivation

1.2. Important concepts

1.2.1. Computer Aided Design - CAD

1.2.2. Topology Optimisation

1.3. Project structure

1.3.1. Aims and Goals

The aim of the project is to provide a tool, which allows to perform *Topology Optimisation* without leaving CAD–framework. This includes the reaching of the following goals:

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

1.3.2. Timeline and Structure

The project is to be completed within 10 months. This period is 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 solutions.

Phase 3: Reimplementing the prototypes, developed on a previous stage, using an open source language, 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 forming of final deliverables.

Part II.

Background Theory

2. Background Theory

2.1. CAD in computers

2.1.1. History of CAD

Computer aided design (short: CAD) refers to the process of designing a product using a computer system. Before CAD applications were used, products were constructed 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 irreplaceable used 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.

2.1.2. Geometry representations

In general, two different ways of describing a geometry are used in CAD systems: constructive solid geometry consisting of a set of primitive forms or storing the boundary of a part assuming that the interior is filled (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* (short: CSG). The basic idea is to start from a set of primitives, e.g. a sphere, cylinder and cube. Basic Boolean operations link these primitives towards a complex geometry. This procedure can be seen in figure 2.1.

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

Boundary representation

A different kind of modelling approach is the so-called *boundary representation*. Instead of storing the geometry information at every single point, *BREP* formats only save the

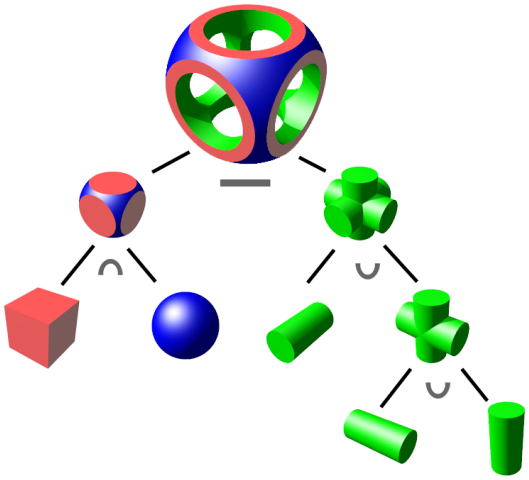


Figure 2.1.: CSG object tree

boundary surface of the body. The interior is assumed to be uniformly filled. Especially in big geometries this approach simplifies the model immensely to an extend that amounts of data are better to handle. Surfaces can be for example stored as a set of triangles (see later STL files) or in NURBS patches. Furthermore, holes in the body are possible by saving the specifying the surface normal of the respective boundary.

By the boundary representation arbitrary geometries can be created. Data amounts to fulfill a certain precision are larger than by the csg representation, but BREP files are usually easier to work with. Keep in mind that non-physical geometries can result from BREP formats through a not closed surface.

2.1.3. Data exchange file formats

CAD software programs usually use own data formats; in order to exchange models standardized interface formats are developed. Geometry information of the model is compressed to certain geometry descriptions and other programs can create a new model in its own file type by this information. Transferring additional information, such as material properties or manufacturing information is general a difficult task or prohibited by the exchange file format. A few common exchange file types are described in the following.

STL file format

The *ST*ereo*L*ithography file format describes the model only by its boundary (BREP); thus, only geometric information can be transferred.

The idea behind the STL files is simple; the geometric model is discretized into a cloud of points. In three dimensions three of these vertices form a triangle; this is done for all vertices and hence the connected surface of triangles describes the surface of the body (BREP). This procedure is shown in figure 2.2 for a two dimensional circle. In two dimensions the mentioned triangles are lines. The advantages and disadvantages of this

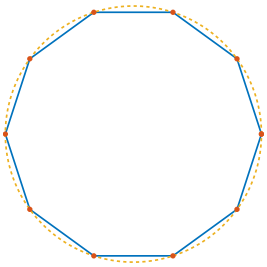


Figure 2.2.: STL discretization for a circle

approach becomes clear: It can be applied to an arbitrary geometry, but accuracy causes difficulties. In order to to transfer high precision geometries many vertices are necessary. This will result in big files; nevertheless, a precise circle can never be represented.

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

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

Note that no additional information such as material properties can be transferred through STL files.

IGES file format

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

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 additional information like the node color is saved. The parameter data section is used for storing the coordinate points and the terminate section signals the end of the file.

2.2. Topology Optimisation

2.2.1. Definition and motivation

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, we minimize the compliance. This is also analogous to minimizing the stress energy stored by the applied loads.

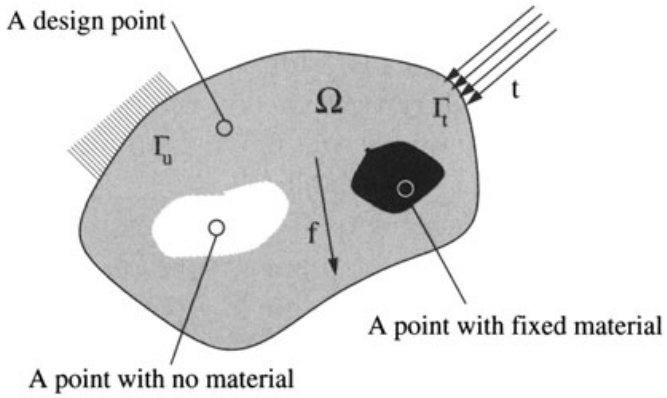


Figure 2.3.: The reference domain Ω for the minimum compliance problem. The problem is formulated such that for a set of external loads t on boundaries λ_t and body forces f, as well as a set of fixed support points λ_u, the material distribution within Ω is such that the stiffness with regards to these loads and forces is maximal, such that 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].

- **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 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, the rising significance of 3D-printers in industry, the realisation of computed optimized designs is now much easier.

2.2.2. Theory

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 Figure 2.3). 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.

To turn this into a more tractable mathematical problem, a few physical assumptions are also typically made: that the material is isotropic, and that it is 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 easily cast into a weak form: integrating the internal virtual work of deforming the elastic material by an admissable displacement from equilibrium, as well as the external work done by the loads and forces for the same displacement, and setting them equal in order to conserve energy, an equation is obtained relating the equilibrium displacement, the stiffness tensor, and the forces and loads, and the cast of this into weak form now allows for solving using Finite Element Analysis (FEM), which can also incorporate the calculation of the external work done.

SIMP: Solid Isotropic Material with Penalization

However, in trying to minimise this external work done by looking at different material distributions, 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 aren't infinite - but they still grow exponentially with the number of elements, so 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, where instead of either being present or not at a point, the material presence can take a continous set of values between one and zero – as some kind of density, fixing the total final volume by integrating this density over the domain, instead of constraining the allowed occupied space. In order to still obtain topologies where material is predominantly 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, such that an intermediate density value provied less elastic support, but still "costs" as much volume, and will thus be suboptimal.

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 [6]and ToPy described in subsection 2.2.3 below. For an extended explanation and discussion, as well as further alternative methods for topology optimisation, the interested reader is referred to [2].

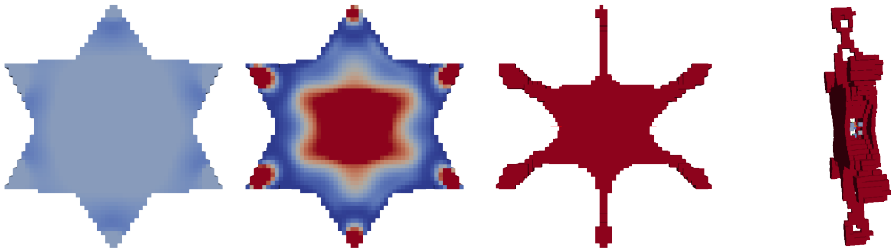


Figure 2.4.: Topology Optimization with minimum compliance of a star structure, given by an stl-file. The fixtures were applied in the corners of the star, while a load was set in the middle.

2.2.3. ToPy

ToPy [5] is a python library/program, written by William Hunter and documented in [4], implementing the SIMP model and method described above. It is based on the 99-line Matlab code by Sigmund's for minimum compliance [6]. The program can optimize the three above named problem types, minimum compliance, heat conduction and mechanism synthesis– in 2D as well as 3D. It uses available open source python software, as for example Pysparse and Numpy, leading to improved speed, porta- and scalability. The whole program is steered by an input file which– with the help of the documentation– is straightforward to use and easy to adapt.

2.2.4. Implementation

In terms of our implementation, we use ToPy as a blackbox topology optimizer. This means, we launch the program with an input file based on our scenario, let ToPy run and proceed by working with the output of ToPy. The intention is to touch the solver itself as less as possible to be able to just plug in different solvers later on. Implementation-wise that means, that we wrote a program which takes as input a voxelized CAD design in, for example, stl-format and outputs a tpd-file which can be used by ToPy. Results of the process can be seen in figure 2.4. Here, a star was given as input from a stl-file. We fixed the voxels in the corners of the structure, while we set a load in the middle, pointing into the structure. As can be seen, the optimization process "cuts" away unnecessary material in-between the corners and even in the middle of the material and returns stiff structure for a minimal amount of material. (maybe a bit wishy washy here)

2.3. From CAD to Voxels

2.3.1. Motivation

The goal of good design is to find the right balance between a set of parameters, which usually include efficiency, weight and aesthetics. For a long time, this process had been an ardous loop of minute modifications to the product, oscillating between the engineer and the designer. However, with the advent of Topology optimization (see Section) and

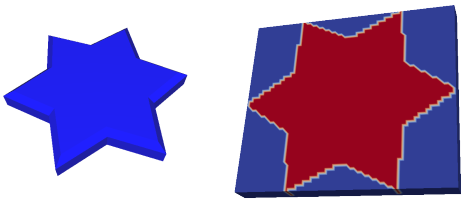


Figure 2.5.: The STL geometry of a star (left) and its voxelized form (right) obtained via the CVMLCPP voxelizer

additive manufacturing, it has been shrunk drastically to an efficient, compact task. In the previous section, we summarized the different ways of representing a design object digitally. On specifying the appropriate loads as boundary conditions on this object, one can choose their favourite topology optimization tool to compute the optimal dimensions and form.

The main hurdle with most state-of-the-art open source topology optimization tools is their input format - almost all of them 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. This section describes how we overcame this hurdle of converting CAD representations to voxelized input.

2.3.2. CVMLCPP

The Common Versatile Multi-purpose Library for C++ (”abbreviated” to CVMLCPP) 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.

2.4. From Voxels to a surface representation

2.4.1. Isosurface Contouring

Now when the optimized voxel data was obtained, the next step is to generate a *mesh based geometry*. It will be useful in the further NURBS implementation. In order to achieve it, the data will be represented by a contour of a smooth function, rendering an isosurface. The isosurface allows to visualize Scalar Volumetric Data in 3D. It furthermore permits a mesh representation of the volume data. The mesh can be composed of triangles or quads, according to the algorithm used. There are two main approaches to solve this problem, the most famous one is Marching Cubes.

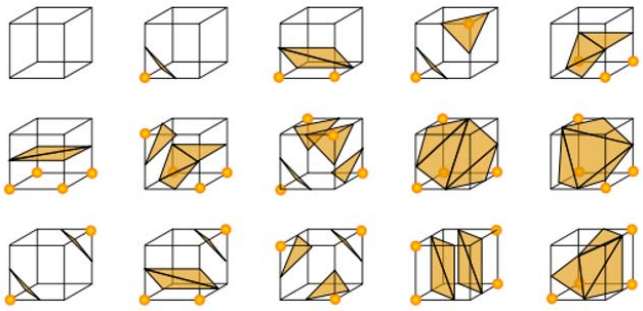


Figure 2.6.: Basis cases of Marching Cubes

Marching Cubes

This algorithm takes as an input a regular volumetric data set and extracts a polygonal 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. A cube that contains an edge is called active. Non active cubes are not further considered in the algorithm.

By connecting the vertices we obtain a polygon on each cube. There are 256 possible scenarios, but most of them are just reflections or rotationally symmetric cases of each other. Therefore there are 15 base cases which represent all the possibilities of the marching cubes (Figure 2.1). The original algorithm presents two main problems. Firstly it does not guarantee neither correctness nor topological consistency, which means that holes may appear on the surface due to inaccurate base case selection. The second problem is ambiguity, which appears when two base cases are possible and the algorithm chooses the incorrect one. These cases can be grouped into face ambiguities and internal ambiguities. 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 this algorithm is similar to Marching Cubes, but instead of generating vertices on the edges of the cubes, it locates them inside the cube. Figure 2.5 shows the basic differences in both approaches. The vertices associated with the four contigous 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 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 to work with.

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

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 nummerical treate-

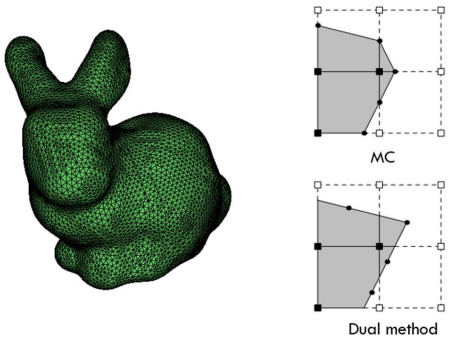


Figure 2.7.: *Right:* The famous Standford Bunny. *Left:* Main difference between MC and Dual methods

ment is needed. As proposed in [1] the best approach is to compute the SVD decomposition of A and form the pseudo-inverse by truncating its small singular values.

The main advantage of this method over MC is the adquisition 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.

2.4.2. The VTK Toolbox

Installing VTK

VTK was installed using the Linux platform, for it to be successfully implemented a gcc compiler must be already on the machine. VTK offers the possibility to use Python, TLC or C++ for development. VTK toolbox is actually a C++ library, which is implemented in other languages. We decided to continue the project with C++ since it gives the possibility to explore the original code. A few dependency problems were encountered, nevertheless they were easy to track back. If any problems were to be found at installation time, please refer to the VTK Wiki where the procedure is explained step by step.

Implementing the VTK Classes

The VTK toolbox was used in order to implement the algorithms on our optimized data. It is a heavily object oriented toolbox. Our first approach was to use the built in Marching Cubes algorithm, nevertheless it did not work with our unstructured grid data. It just works for ImageData and PolyData . For structured and unstructured grids the tool to render 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 cube algorithm.

The Contour Filtering seemed to work 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

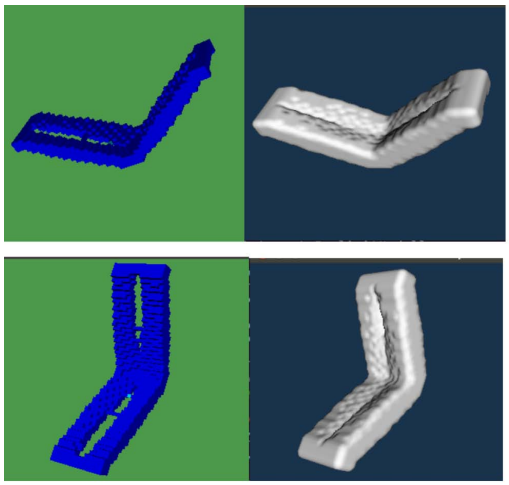


Figure 2.8.: Contour Filtering tool after Implicit Modelling

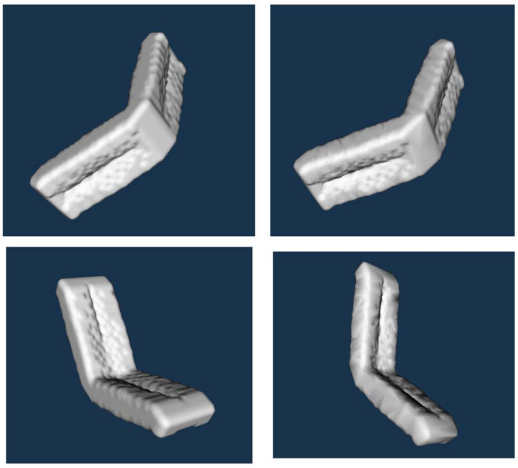


Figure 2.9.: Decimation of triangles. *Top:* 50% *Lower:*90%

new, offset surfaces from the original geometry. It finally allowed visualization but it created one problem. Holes are lost in the process.

A further idea to solve this problem is to convert at the first step the volume data into point data and only then present it to the Contour Filtering Tool. This will be implemented in the next milestone.

The next step was to create a coarser mesh from the fine one. The 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 [2] 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.

2.4.3. The Long Road to NURBS

There are two possible roads to go from the voxel data to the NURBS representation.

Quad Contouring

This approach uses the dual contouring algorithm as first step in order to obtain a quad mesh representation from the voxel data. The first challenge is to implement correctly the algorithm with the ideas presented in [1]. The original marching cubes algorithm is implemented in VTK but the source code is not public, therefore not only an extension of it is needed, but a full implementation. Once this first step is done, the quads will be chosen for the NURBS parametrization. A second step considers multiple smaller quads which have to be combined into one larger patch. This is another challenge, since the remeshing of quad meshes is not as straight forward as with the triangles. Different approaches have been taken in order to achieve this coarsening. In [2] an incremental and greedy approach, which is based on local operations only, is presented. It depicts an iterative process which performs local optimizing, coarsening and smoothing operations. Another approaches, like the one presented in [3] uses smooth harmonic scalar fields to simplify the mesh.

Multiresolution Analysis of Arbitrary Meshes

With this approach there is no need to apply a Dual Contouring algorithm, since it takes as beginning data the triangles from the Marching Cubes. The main concepts are shown in the paper of the same name [4]. It mainly takes a series of intermediate steps which permits a parametrization of data. It includes a partitioning scheme based on the ideas of the Voronoi Diagrams and Delaunay triangulations. Large patches or quads are obtained with this method. Further discussion on this approach is explained in the following section.

Both approaches have not been implemented in open source documentation, therefore it will be a long road to achieve what is required. For the first part the second road was chosen and in case it leads to a dead end, the first way will be taken into consideration.

2.5. From a surface representation to NURBS

Having said about the importance of the NURBS representation, we define NURBS from a mathematical standpoint. For that, we first define so-called *Bezier curves* which we will use later for the definition of NURBS.

2.5.1. Bezier Curves

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

Analytical expression

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

$$b_i^n(t) = \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{(n-i)} t^i$$

is the i–th Bernstein polynomial 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.

Surfaces

Analogically to Bezier curves, but with $n \cdot m$ Points $\vec{P}_{i,j}$ and the analytical expression

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

one can define a *Bezier surface*
Please, note, that Bezier curves and surfaces may be unstable! Minor changes in control points might lead to major global changes!

2.5.2. NURBS basis functions

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

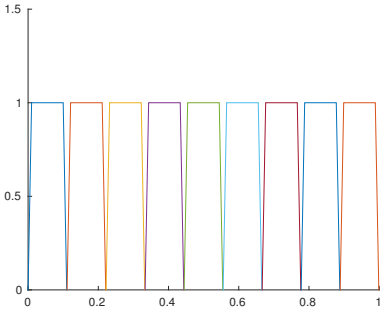
Unlike Bezier curves, for the B–spline basis a parameter domain is subdivided with, so-called, *knots*. In particular, given the parameter domain $[u_0, u_m]$ (in 1D), *knot vector* is given by $u_0 \leq u_1 \leq ... \leq u_m$. In most cases $u_0 = 0, u_m = 1$, so we get unit interval for our parameter 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:

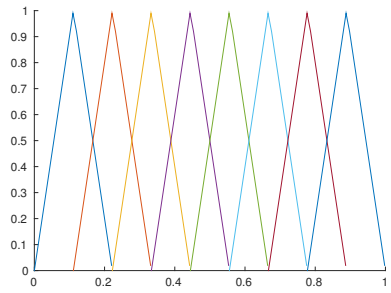
$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

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

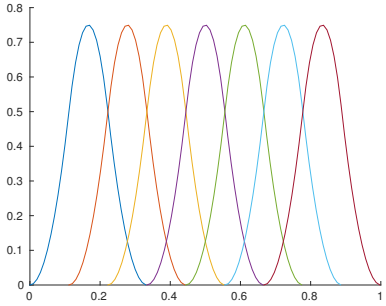
For $p = 0$ we get just step functions (see fig. 2.5.2), for $p = 1$ we get familiar hat functions (see fig. 2.5.2). Quadratic basis looks a bit more complicated, but also quite intuitive (fig. 2.5.2).



(a) B-spline basis for $p = 0$



(b) B-spline basis for $p = 1$



(c) B-spline basis for $p = 2$

Figure 2.10.: B-spline basis functions

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.

2.5.3. Minimization problem

Now, once we defined all necessary tools, we proceed to the fitting problem.
The goal: Fit in a parametric curve to the set of given data points.
In our case our given set of points is a mesh, obtained on a previous step.

2.5.4. 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:

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

The function space $\mathbb{B}_n$ is composed of functions of the following form:

$$\vec{B}_n(t) = \sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{2.5}$$

Therefore a control point for the Bezier–curve has the following form:

$$\vec{P}_i = \begin{pmatrix} a_{2i} \\ a_{2i+1} \end{pmatrix}. \tag{2.6}$$

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

$$F(\vec{B}_n) = \int_{t=0}^{t=1} \left(\vec{B}_n - \vec{s}_m\right)^2 \mathrm{dt} \tag{2.7}$$

Varying B_n yields

$$\delta F(B_n) = 2 \int_{t=0}^{t=1} \left(\vec{B}_n - \vec{s}_m\right)^T \delta\vec{B}_n \mathrm{dt} \tag{2.8}$$

using the definition of $\vec{B}_n$ and $\delta\vec{B}_n$

$$\vec{B}_n(t) = \sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{2.9}$$

$$\delta\vec{B}_n(t) = \sum_{i=0}^{n} \delta a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + \delta a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{2.10}$$

leads to

$$\delta F(B_n) = 2 \int_{t=0}^{t=1} \left(\sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} - \vec{s}_m \right)^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{dt} \stackrel{!}{=} 0 \tag{2.11}$$

rewritten

$$\int_{t=0}^{t=1} \left(\sum_{i=0}^{n} a_{2i} \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} + a_{2i+1} \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \right)^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{dt} = \int_{t=0}^{t=1} \vec{s}_m^T \left(\sum_{j=0}^{n} \delta a_{2j} \begin{pmatrix} b_j^n(t) \\ 0 \end{pmatrix} + \delta a_{2j+1} \begin{pmatrix} 0 \\ b_j^n(t) \end{pmatrix} \right) \mathrm{dt} \tag{2.12}$$

or using the notation

$$\phi_{2i} = \begin{pmatrix} b_i^n(t) \\ 0 \end{pmatrix} \tag{2.13}$$

$$\phi_{2i+1} = \begin{pmatrix} 0 \\ b_i^n(t) \end{pmatrix} \tag{2.14}$$

we get

$$\int_{t=0}^{t=1} \left(\sum_{i=0}^{2n+1} a_i \phi_i \right)^T \left(\sum_{j=0}^{2n+1} \delta a_j \phi_j \right) \mathrm{dt} = \int_{t=0}^{t=1} \vec{s}_m^T \left(\sum_{j=0}^{2n+1} \delta a_j \phi_j \right) \mathrm{dt} \tag{2.15}$$

or

$$\sum_{i=0}^{2n+1} \sum_{j=0}^{2n+1} a_i \delta a_j \int_{t=0}^{t=1} \phi_i^T \phi_j \mathrm{dt} = \sum_{j=0}^{2n+1} \delta a_j \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \mathrm{dt} \tag{2.16}$$

this has to be true for any j and δa_j, therefore

$$\sum_{i=0}^{2n+1} a_i \int_{t=0}^{t=1} \phi_i^T \phi_j \mathrm{dt} = \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \mathrm{dt} \; \forall j = 0 \cdots 2n+1 \tag{2.17}$$

using

$$A_{ij} = \int_{t=0}^{t=1} \phi_i^T \phi_j \mathrm{dt} \text{ and } b_j = \int_{t=0}^{t=1} \vec{s}_m^T \phi_j \mathrm{dt} \tag{2.18}$$

we get the linear equation system

$$Aa = b \tag{2.19}$$

which has to be solved for a.

2.5.5. Minimization problem(least squares): NURBS

Although the approach used above showed good results, it appears to be very computationally extensive. 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 [1].

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:

$$X_i^0 = \sum_{j=1}^{n+p} P_j^0 N_{i,j}^p, \quad i \in \{1, .., n\} \tag{2.20}$$

Or, in short:

$$X^0 = N^p P^0 \tag{2.21}$$

The above system needs to be solved for the unknown P^0. One of the ways to solve it is to use SVD decomposition. So far, we used built-in MATLAB solver.

2.5.6. Fitting pipeline

Since the geometry obtained after topology optimization can be arbitrary complex, we might not be able to find a food 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 [3], our overall fitting pipeline looks as follows (see fig. 2.11):

- 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.

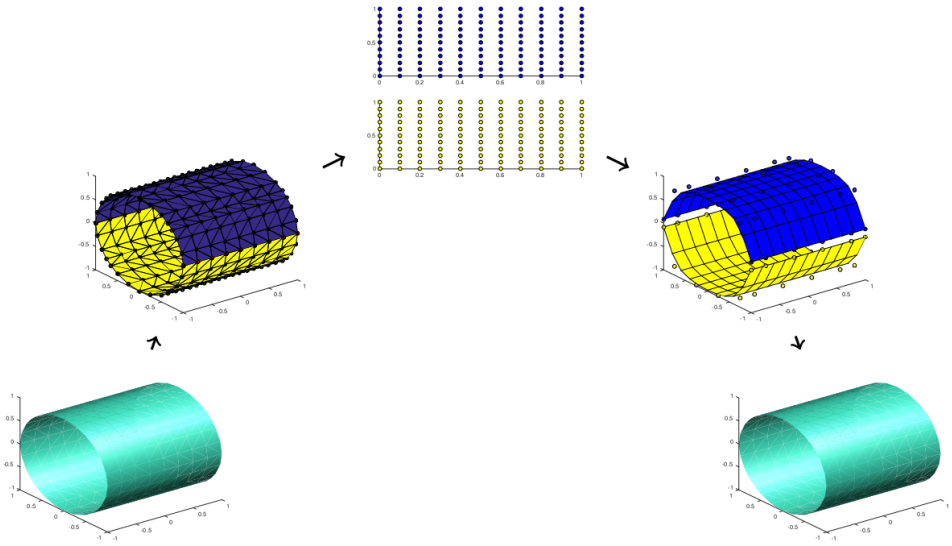


Figure 2.11.: Fitting pipeline

Appendix

A. Detailed Descriptions

Here come the details that are not supposed to be in the regular text.

Bibliography

[1] Gerrit Becker, Michael Schäfer, and Antony Jameson. *An advanced NURBS fitting procedure for post-processing of grid-based shape optimizations*. 2011.

[2] Martin Philip Bendsøe and Ole Sigmund. *Topology Optimization: Theory, Methods and Applications*. Springer Science & Business Media, 2003.

[3] Matthias Eck and Hugues Hoppe. *Automatic reconstruction of B-spline surfaces of arbitrary topological type*. 1996.

[4] William Hunter. Predominantly solid-void three-dimensional topology optimisation using open source software. Master's thesis, Stellenbosch University, 2009.

[5] William Hunter. Topy - 2d and 3d topology optimization using python. `https://code.google.com/p/topy/`, 2009. lastly accessed on 21/08/2015.

[6] Ole Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, 2001.