



Master's Thesis

# Computer-Aided Design in Virtual Reality

Author: Benjamin R uth  
Field of Studies: Computer Science, Computational Science and Engineering  
1<sup>st</sup> Examiner: Univ.-Prof. Gudrun Klinker, Ph.D. (TUM)  
2<sup>nd</sup> Examiner: Univ.-Prof. Dr. Hans-Joachim Bungartz (TUM)  
Academic Supervisor: Dr. Frieder Pankratz (TUM)  
Assistant Supervisors: Dr. Philipp Emanuel Stelzig (Siemens, Corporate Technology)  
Dr. Dirk Hartmann (Siemens, Corporate Technology)  
Date: 28.2.2017

Technische Universit t M nchen  
Fakult t f r Informatik  
Lehrstuhl f r Wissenschaftliches Rechnen  
Boltzmannstra e 3  
85748 Garching bei M nchen

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

---

Benjamin R uth, M unchen, February 28, 2017

# Acknowledgements

I express my gratitude to everybody who supported me during the past six months creating this work. I greatly enjoyed the research, the implementation, and the experiments conducted during this thesis.

I thank my supervisors Dr. Frieder Pankratz, Dr. Philipp Emanuel Stelzig and Dr. Dirk Hartmann for their support, great ideas, guidance and motivation. I thank Prof. Gudrun Klinker, Ph.D. and Prof. Dr. Hans-Joachim Bungartz for the examination of the thesis.

I thank Manuel Biedermann, Dr. Andreas Dippon and Dr. Christoph Kiener for sharing their point of view in expert interviews. I also thank the participants of the user study for the time and energy they put into solving tasks and filling out questionnaires. I am grateful for their valuable feedback.

I thank my colleagues at Siemens, Corporate Technology for their company and support. I thank Stefan for introducing me to IDeAs, Christof for his help with VTK, Christian for the introduction to Unity, Utz for mathematical guidance, Theo for his expertise with NX and Christa for knowing the right solution in every situation.

I especially thank Erik, Michaela and Julian for their help with proofreading and refining this thesis.

Finally I thank my parents Johanna and Norbert, my sister Anna and my partner Catrin.



# Abstract

*Computer-aided design* (CAD) and *computer-aided engineering* (CAE) play an important role in the product design process. CAD is commonly used for the creation of geometry while CAE allows us to assure the functionality of a product and optimize it with respect to the requirements of the application. These tools are restricted to trained users. In this thesis the novel concept of using virtual reality technology to provide an intuitive and easy-to-learn user interaction is proposed. This enables the layman to work with CAD.

The prototype CADinVR, developed at Siemens, Corporate Technology is extended with the *Virtual Reality Surface Fitting Extension* (SurfFitX). Here the classical topology optimization design workflow is realized: The user starts with an initial design which is optimized to obtain a lightweight structure. A mesh representation of the optimized geometry is created that is consistent with boundary condition shapes. The mesh is partially reconstructed using SurfFitX and finally the output is given in a native CAD file format (.step). This reconstructed geometry is superior to the original mesh geometry, since it provides a flexible and parametrized B-spline representation of the geometry. This representation can be further processed using standard CAD tools.

The interaction concepts that are implemented in CADinVR and SurfFitX are evaluated by performing a user study: Here the user's performance in SurfFitX is compared to the performance in the traditional CAD system FreeCAD. Further potential fields of use for VR technology in CAD are described on the basis of the user study.



# Zusammenfassung

*Rechnergestützte Konstruktion* (CAD) und *rechnergestützte Entwicklung* (CAE) sind wesentliche Bestandteile des Entwurfprozesses von Produkten. CAD wird dabei allgemein für die Erstellung der Geometrie genutzt, während CAE dazu dient die korrekte Funktion des Bauteils sicherzustellen und das Bauteil für die gewünschte Anwendung zu optimieren. Der Einsatz dieser Werkzeuge ist auf geschulte Benutzer beschränkt. In dieser Arbeit wird gezeigt wie der Einsatz von Technologien aus dem Bereich der *Virtuellen Realität* (VR) in einem neuartigen Konzept dazu dient eine intuitive und leicht erlernbare Benutzerschnittstelle zu entwerfen mit der auch ein Laie im Bereich der CAD arbeiten kann.

Der Prototyp CADinVR, entwickelt in der Abteilung Corporate Technology der Firma Siemens, wird durch *Virtual Reality Surface Fitting Extension* (SurfFitX) erweitert. Darin wird der klassische Topologieoptimierungszyklus realisiert: Der Anwender stellt einen grundlegenden Entwurf bereit. Dieser wird optimiert um eine Leichtbaustruktur zu erhalten. Es wird ein Gitter erzeugt, das die optimierte Geometrie darstellt und konsistent mit geometrischen Randbedingung ist. Im folgenden wird ein Teil des Gitters mithilfe von SurfFitX rekonstruiert und im nativen CAD-Format (`.step`) bereitgestellt. Diese Geometrie ist der ursprünglichen Gitterdarstellung überlegen, da darin flexible und parametrisierte B-Splines verwendet werden, welche in herkömmlichen CAD Werkzeugen weiter bearbeitet werden können.

Die Benutzerschnittstelle von CADinVR und SurfFitX wird in einer Studie bewertet: Dabei wird der Erfolg des Benutzers bei der Bearbeitung einer Aufgabe in SurfFitX sowie in dem herkömmlichen CAD-System FreeCAD verglichen. Weitere potentielle Anwendungsgebiete von VR-Technologie im Bereich der CAD werden auf der Grundlage der Studienergebnisse beschrieben.





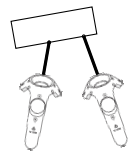
# Contents

<b>Abstract</b>	<b>iii</b>
<b>Zusammenfassung</b>	<b>v</b>
<b>1. Introduction and background</b>	<b>1</b>
1.1. Potential use of virtual reality technology in computer-aided design . . . .	1
1.2. Design assistance: Topology optimization and reverse engineering . . . . .	3
1.3. Siemens CADinVR prototype and surface fitting extension . . . . .	4
1.4. Structure of the thesis . . . . .	4
<b>2. Methods</b>	<b>7</b>
2.1. Description of geometry . . . . .	7
2.1.1. Exact representation of geometry . . . . .	7
2.1.2. Approximate representation of geometry . . . . .	10
2.1.3. Geometry conversion and comparison . . . . .	11
2.2. Computer-aided design . . . . .	12
2.2.1. Philosophy and requirements . . . . .	12
2.2.2. Open CASCADE Technology . . . . .	12
2.2.3. FreeCAD . . . . .	14
2.3. Virtual reality . . . . .	15
2.3.1. Hardware . . . . .	15
2.3.2. Design in virtual reality . . . . .	16
2.3.3. Unity3D . . . . .	17
2.3.4. Virtual reality software interface . . . . .	17
2.4. Topology optimization . . . . .	19
2.4.1. Theory . . . . .	19
2.4.2. Input and output . . . . .	19
2.5. Level set and voxel description . . . . .	20
2.5.1. Level sets . . . . .	20
2.5.2. Fast marching method . . . . .	21
2.5.3. Level set propagation . . . . .	22
2.5.4. Boolean operations on level sets . . . . .	23
2.5.5. Visualization Toolkit . . . . .	23
2.6. B-splines . . . . .	23
2.6.1. B-spline curves and surfaces . . . . .	23
2.6.2. Continuity . . . . .	24

<b>3. Design Workflow</b>	<b>27</b>
3.1. Topology optimization . . . . .	30
3.1.1. Definition of boundary conditions . . . . .	30
3.1.2. Optimized voxel geometry . . . . .	30
3.2. Consistent level set . . . . .	33
3.3. Parametric geometry reconstruction . . . . .	39
3.3.1. Mesh visualization . . . . .	40
3.3.2. Contour extraction from meshes . . . . .	40
3.3.3. Loft surface creation . . . . .	41
3.3.4. B-Spline sketching . . . . .	44
<b>4. User interaction</b>	<b>45</b>
4.1. Shape interaction . . . . .	45
4.2. Sketching . . . . .	47
4.3. Camera adjustment . . . . .	47
<b>5. Implementation</b>	<b>51</b>
5.1. Frontend . . . . .	51
5.1.1. General overview . . . . .	51
5.1.2. Large meshes and compound objects . . . . .	52
5.1.3. The menu . . . . .	52
5.1.4. Coordinate mapping . . . . .	53
5.2. Backend . . . . .	54
5.2.1. General overview . . . . .	54
5.2.2. Exchange data types . . . . .	54
<b>6. User study</b>	<b>57</b>
6.1. The task . . . . .	57
6.2. Experimental setup . . . . .	59
6.3. Evaluation procedure . . . . .	60
<b>7. Results</b>	<b>61</b>
7.1. Topology optimization workflow using Virtual Reality Surface Fitting Extension . . . . .	61
7.1.1. GE Bracket . . . . .	61
7.1.2. Generate Quadcopter . . . . .	62
7.2. User study . . . . .	65
7.2.1. The Task . . . . .	66
7.2.2. System Usability Scale . . . . .	69
7.2.3. Interview . . . . .	71
7.3. Expert Interview . . . . .	75
7.3.1. Today's engineering workflow . . . . .	75
7.3.2. Sketches and drawings in virtual reality . . . . .	76

<b>8. Discussion, conclusion and future directions</b>	<b>77</b>
8.1. The design workflow . . . . .	77
8.2. User Study . . . . .	78
8.2.1. Simple and intuitive user interface . . . . .	78
8.2.2. Easier learning experience . . . . .	79
8.2.3. Better access to 3D design tools . . . . .	79
8.2.4. Concept design . . . . .	80
8.3. The role of virtual reality in computer-aided design . . . . .	80
8.4. Conclusion . . . . .	81
8.5. Future directions . . . . .	83
<b>A. User study: Material</b>	<b>85</b>
A.1. The task . . . . .	85
A.1.1. Userguide FreeCAD . . . . .	85
A.1.2. Userguide CADinVR . . . . .	90
A.2. Questionnaires and interview . . . . .	95
A.2.1. The Background Questionnaire . . . . .	95
A.2.2. The SUS Questionnaire . . . . .	96
A.2.3. Interview questions . . . . .	97
<b>B. Tutorials</b>	<b>99</b>
B.1. Open CASCADE Technology: Tutorial and Examples . . . . .	99
B.2. Visualization Toolkit: Tutorial and Examples . . . . .	109





# 1. Introduction and background

Today there exists a broad range of professional general purpose desktop *computer-aided design* (CAD) and *computer-aided engineering* (CAE) tools such as NX or AutoCAD [1–4]. But there also exist open source solutions such as FreeCAD [4] and online tools such as OnShape [3]. These tools provide advanced user interfaces and modeling functionality and allow the user to design 3D models of engineering parts with a high level of detail and accuracy. The use of the aforementioned tools is usually restricted to trained experts due to high cost and complex user interfaces<sup>1</sup>.

Today’s 3D modeling tools obviously restrict the user to a projection of the 3D model onto the two-dimensional computer screen. The computer screen and input devices such as the computer mouse force the user to understand abstract concepts for performing 3D operations. Therefore the user needs training in order to understand the abstract concepts of CAD which are needed to operate the tools. On top of that the user interaction strongly differs between different tools.

But 3D modelling and simulation software is not only demanded by trained engineers who know how to work with a broad range of tools but also by amateurs and untrained professionals: additive manufacturing has reached the consumer market and the layman wants to design appealing and functional geometry that can be fed into the hobbyist’s 3D printer. In order to support people designing custom parts, very simple design tools such as Tinkercad exist [6]. Also professional workflows can be improved by incorporating simulation methods into product design workflows; here also non experts have to be involved [7].

## 1.1. Potential use of virtual reality technology in computer-aided design

We observe a gap between the demand and the accessibility of design tools. Using *virtual reality* (VR) technology one can overcome these deficits and CAD tasks become considerably easier to solve – especially for the untrained user – because VR hardware gives access to a real 3D model. Instead looking at a 2D projection of the model on the screen the user receives a stereoscopic 3D image of the model in a virtual environment. A more intuitive and accessible *user experience* (UEx) is possible because the user does not have to understand abstract 3D concepts of geometry representation. With this also inexperienced users are able to use CAD software.

Today’s VR hardware is affordable and accessible for a wide range of people, software interfaces to popular hardware setups [8, 9] exist, but there are only few specialized VR

---

<sup>1</sup>As an example: a book on CAD technology can easily have more than 700 pages [5].

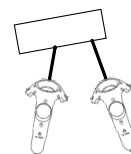
Tools	CAD tool	CAD visualization	manipulation	simple	VR/AR	professional	open source
NX [1]	✓	✓	✓	✗	✗	✓	✗
FreeCAD [4]	✓	✓	✓	✗	✗	✗	✓
FreeCAD VR [19]	✓	✓	?	✗	✓	✗	✓
Mindesk [15]	✗	✓	✗	?	✓	✓	✗
Tinkercad [6]	✗	✗	✗	✓	✗	✗	✗
Unreal VR [13]	✗	✗	✓	?	✓	✓	✗
Tiltbrush [11]	✗	✗	✓	✓	✓	✗	✗
VR VTK [14]	✗	✗	✗	✗	✓	✓/✗	✓
OCCT [21]	✓	✓	✓	✗	✗	✓/✗	✓
DualCAD [20]	✓	?	✓	?	✓	✗	✗

Table 1.1.: **Summary of different design tools** – We compare different design tools with respect to the following categories: Is the tool a full fledged CAD tool (*CAD tool*)? Is the tool able to load and visualize CAD geometry in *.step* format (*CAD visualization*)? Does it allow to manipulate geometry (*manipulation*)? Is it easy to use (*simple*)? Is it a VR or AR tool (*VR/AR*)? Is it considered a professional tool and used in professional applications (*professional*)? Is it an open source tool (*open source*)? We use the signs ✓ and ✗ for indicating true or false. A ? indicates that we could not evaluate the respective category.

software systems [10]. In the following we summarize VR tools that support components necessary for design in VR: There exist tools that concentrate on different aspects of design such as drawing and sketching [11] or games development [12, 13]. VR technology is also used for scientific visualization using the VR extension of *Visualization Toolkit* (VTK) [14]. Recently VR and *augmented reality* (AR) technology have also been used in the context of CAD for purposes of visualization without manipulation of the underlying geometry in professional tools such as Mindesk [15–17]. Part analysis and assembly are realized in a VR environment in IC.IDO [18]. An interface for the popular VR headset Oculus [9] exists in the open source CAD software FreeCAD version 0.15 [19]. There are AR systems that go beyond visualization and allow the user to even model the geometry such as DualCAD [20]; anyhow in this approach only modeling of primitive shapes is supported. For a comparison of the different tools see Table 1.1.

In this thesis we further develop the *Virtual Reality Surface Fitting Extension* (SurfFitX) on top of the prototype CADinVR, a CAD tool that heavily relies on VR technology. The development is mainly relying on the game engine Unity3D [22] and the geometry kernel *Open CASCADE Technology* (OCCT) [21].

We evaluate the usability of SurfFitX in a user study. By comparing SurfFitX with an existing desktop CAD tool, we want to analyze the benefits of VR in the context of CAD. Additionally we summarize the feedback of a CAD expert regarding the potential of VR technology in the context of CAD and product development.



## 1.2. Design assistance: Topology optimization and reverse engineering

In this work we want to concentrate on the use case of the reverse engineering of a topology optimized part. This task is especially difficult in traditional CAD since a topology optimized geometry is not built in a bottom-up approach, where the whole part is constructed from simple primitive shapes. The geometry is rather defined by the specifications of the part in a top-down manner: it is the output of an optimization algorithm which computes the optimal<sup>2</sup> geometry and topology for specified loads and fixtures.

The optimized geometry has to be analyzed and rebuilt by the CAD modeler in several hours of manual work. There are two main reasons for this additional step: Firstly, it cannot be guaranteed that the output geometry complies with specifications that are imposed on boundary condition shapes like, for example, screw threads; therefore, without special care at this point the optimized part might not fit in the application anymore. Secondly, the output geometry format of the optimization procedure is usually a mesh geometry or a voxel grid that is hard to process further. Automatized approaches for the reconstruction of CAD geometry from mesh geometry exist [23, 24], but they do not provide the flexibility of manual reconstruction, they are very complicated and the results are often not satisfactory with respect to the mesh quality. Additionally very general shapes originate from topology optimization and the structures have a complex topology which makes it even harder to automatically reconstruct the geometry. Therefore the reconstruction of a topology optimized structure in CAD can only be done by an expert with a large toolset in geometric modeling.

In this thesis we propose a workflow that eases the process of the creation of CAD geometry from topology optimization output: We decouple the simulation driven design phase from the constructive reconstruction phase. In the first phase the topology optimized geometry is created and design decisions are made; in the second phase a CAD expert creates the production ready parametrized geometry. We first automatically postprocess the optimized geometry in order to guarantee that the resulting mesh is consistent with boundary condition shapes. Then we manually reconstruct the consistent mesh geometry using the prototype SurfFitX in a beginner-friendly VR environment, where sketches and simple surfaces can be created in a fast and intuitive way. Working in 3D space helps the user to better visualize and understand complex structures. At the same time the user can reconstruct the structures in 3D without the need of falling back to 2D concepts of traditional CAD like planar sketches. Therefore, we can significantly speed up the process of surface reconstruction using VR technology.

SurfFitX outputs B-Spline curves and loft surfaces in the standardized .step file format that can be easily used in CAD. The expert CAD modeler may use these curves and surfaces for the reconstruction of watertight, parametrized and smooth CAD geometry in a professional CAD tool like NX or AutoCAD. The quality of the final model complies with established engineering standards.

---

<sup>2</sup>The geometry is optimal with respect to a certain goal like minimum weight etc.

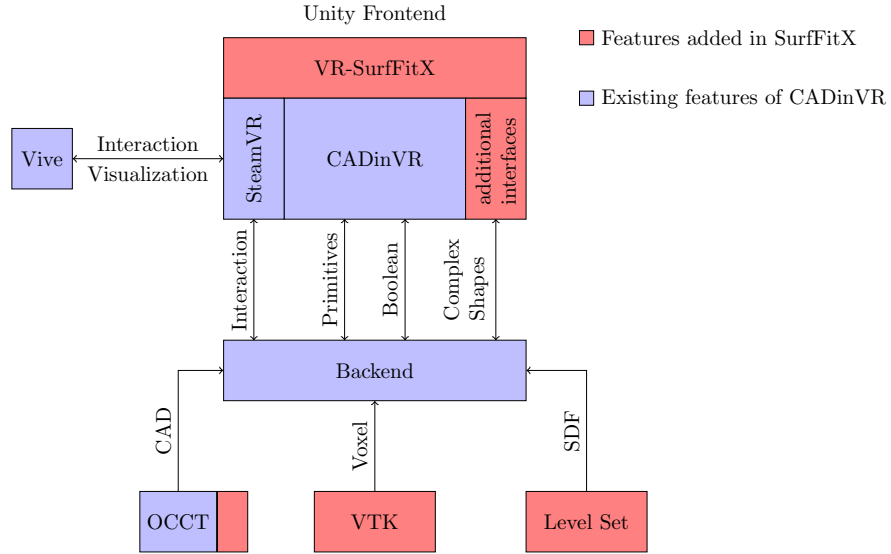


Figure 1.1.: **Overview over CADinVR with SurfFitX** – CADinVR provides basic CAD functionality for interaction with primitive shapes in a virtual environment (blue regions). We extended CADinVR with functionality for voxel data visualization and manipulation. We also added mechanisms for the creation of complex shapes such as loft surfaces or B-spline curves. The additional functionality is provided in SurfFitX (red regions).

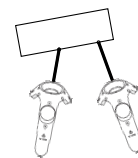
### 1.3. Siemens CADinVR prototype and surface fitting extension

SurfFitX is an extension for the prototype CADinVR developed at Siemens, Corporate Technology in 2016. CADinVR enables the user to create and manipulate primitive shapes that are modeled through CAD technology directly in a VR environment. A core functionality is the possibility to perform boolean operations on shapes for the creation of complex geometry. CADinVR consists of two layers: A frontend based on Unity3D providing interfaces to the VR hardware, and a backend written in C++, which provides an interface to the CAD-kernel OCCT.

The functionality of CADinVR is extended in the scope of this thesis by providing SurfFitX. Here we provide means of visualization for voxel or mesh geometry through the visualization library VTK [25]. We also provide a collection of level set methods for the creation and manipulation of a *signed distance function* (SDF) originating from topology optimization data.

Using SurfFitX the user can interactively extract contour curves from the mesh by defining cutting planes and draw guiding lines. Contour curves may be used for the creation of parametrized surfaces that approximate the mesh geometry. All created objects are modeled using OCCT and therefore comply with established CAD standards.





## 1.4. Structure of the thesis

The development of an interactive VR application for CAD demands knowledge and tools from a wide range of different fields. We provide the required theoretical framework for the thesis in chapter 2. This includes explanation of basic concepts of computational geometry, CAD and VR, introduction of the software frameworks that are used, the theory of topology optimization, and a short summary of level set methods and the fast marching method.

In chapter 3 we describe the workflow for the creation and reconstruction of a topology optimized shape using SurfFitX. We first start with modelling the topology optimization problem, then we optimize the part using the prototype *Interactive Design Assistant* (IDeAs) [26], postprocess the results with level set techniques, extract contours and guiding lines using SurfFitX and finally create a volumetric part on the basis of the extracted geometry using NX [1].

In chapter 4 we summarize the concepts for user interaction that exist in SurfFitX in order to provide an intuitive UEx for a CAD application in VR. This involves existing functionality from CADinVR, but also new concepts of user interaction developed in the scope of this thesis.

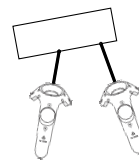
In chapter 5 we summarize the implementation aspects of the Unity frontend and the CAD backend. Here we also present a clear classification of parts of SurfFitX that originate from CADinVR and functionality that has been developed in the scope of this thesis.

In chapter 6 we summarize the setup of the user study that has been performed to evaluate the usability of SurfFitX, especially with respect to the interaction methods described in chapter 4. This includes the explanation of a surface fitting task that represents a part of the workflow described in chapter 3. Additionally we explain the evaluation procedure, where we relied on the *System Usability Scale* (SUS) questionnaire [27] and an interview.

In the first part of chapter 7 two example parts obtained from the workflow described in chapter 3 are presented: The parts result from a topology optimization process and they have been reconstructed using SurfFitX. We optimized and reconstructed a jet engine bracket [28] and the body of a quadcopter [29]. We also present the results of the user study on the basis of evaluation procedure explained in chapter 6. Finally we summarize feedback that we received from a CAD expert with respect to SurfFitX and potential use in professional workflow.

In the final chapter 8 we discuss and evaluate the results from the previous chapter, we describe the potential of VR technology in CAD, and provide future directions regarding SurfFitX.





## 2. Methods

The development of a *computer-aided design* (CAD) application with a *virtual reality* (VR) user interface an interdisciplinary task where materials from different fields are needed. In the following section we collect background material from the fields of CAD, VR, simulation and computational geometry that is important in the scope of this thesis.

Depending on the application – CAD, visualization or simulation – different representations of geometry are in use. We give an overview over different concepts for describing geometry in section 2.1.

We provide an overview over the different fields that we are covering in the scope of this thesis. CAD technology and related software that is used for the exact description of shapes is introduced in section 2.2. We give a short summary of VR technology and the game engine Unity3D that we are using for visualization and manipulation of geometry in section 2.3. Then we summarize the basic concepts of topology optimization, a simulation method for optimal shape generation, in section 2.4.

Finally we provide the mathematical framework that is needed in the scope of this thesis: We rely on level set methods for the manipulation of voxel data that emerge from topology optimization (section 2.5), and B-splines and *non-uniform rational B-spline* (NURBS) are introduced, as they are the standard for geometry description in CAD (section 2.6).

### 2.1. Description of geometry

There exist many different ways of describing geometry. Depending on the application certain representations are more commonly used than others.

In this section we will shortly review different descriptions of geometry and their respective fields of application. Major classifications for describing geometry are, firstly, describing solid bodies through their volume or describing them with their surrounding surface, and, secondly, describing geometry exactly or approximately (see Figure 2.1).

In subsection 2.1.1 we summarize methods for exact geometry representation that are, for example, used in CAD. In subsection 2.1.2 approximate representations are revised; these are frequently used in the field of simulation and visualization. Finally we give a short overview over methods of conversion between different representations in subsection 2.1.3.

#### 2.1.1. Exact representation of geometry

An exact representation of geometry is usually utilized through parametrized objects such as, a circle that is defined by origin, radius and normal vector. More general and

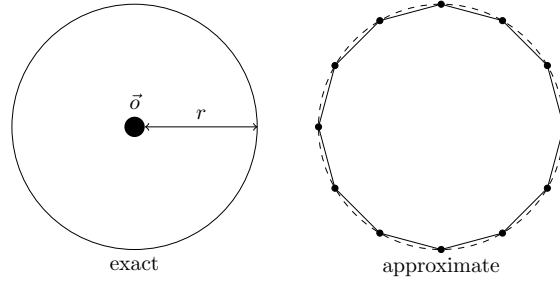


Figure 2.1.: **Two geometric descriptions of a circle** – A 2D circle can either be described exactly through the parameters radius and origin or approximately with a mesh. Both descriptions are used in different context, the exact representation is used in CAD for parametrized geometry, while an approximate representation is often used in computer graphics.

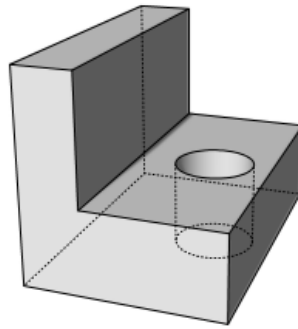
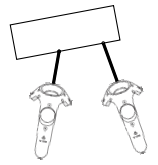
flexible concepts like NURBS exist (see section 2.6 for more information). We call this a parametric description of a curve or surface. Theoretically solid objects can also be described through parametrized objects [30].

The following two philosophies for the exact description of volumetric geometry exist: *constructive solid geometry* (CSG) and *boundary representation* (BREP) (see Figure 2.2):

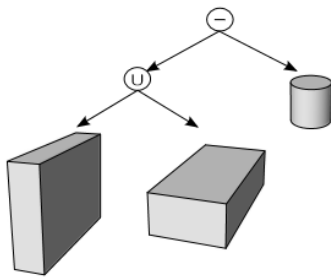
- CSG describes the shape as a set of primitive shapes that have been joined using Boolean operations. Primitive solid shapes – cube, cylinder or sphere – are fused, subtracted or the intersecting region is determined for the creation of new shapes (see Figure 2.2b).
- BREP describes the shape on the basis of its surface, which is built from several faces with a given topology. Solid bodies are described by a watertight surface, that is, a closed shell without holes that surrounds the region occupied by the solid (see Figure 2.2c).

Both of the two philosophies use parametric curves and surfaces in order to be able to exactly represent geometry at arbitrary resolution. In order to be able to represent arbitrary shapes and provide sufficient flexibility we either need a large range of parametrized shapes or very flexible basis functions (e.g. NURBS see section 2.6). With this additional degree of complexity comes additional structural information about the geometry: a circle, for example, is explicitly denoted as a circle.

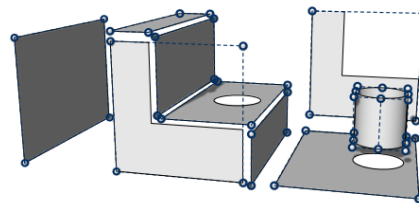
In CAD the geometry is mostly modelled using the aforementioned concepts. But also in the field of numerical simulation, branches exist that use exact representation of geometry (*isogeometric analysis* (IGA) [30]). In the scope of this thesis we use a standardized file format based on the *Standard for The Exchange of Product model data* (.step) [31, 32] for data storage and exchange of BREP data.



(a) A simple shape

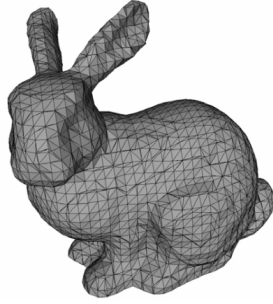


(b) Representation using CSG.

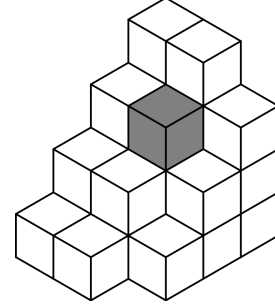


(c) Representation using BREP

Figure 2.2.: **Representation of a simple shape with CSG and BREP** – Using CSG (Figure 2.2b) we first fuse two blocks ( $\cup$ ) and create an L-shaped compound object, then we drill a hole, by subtracting a cylinder from the resulting shape ( $-$ ). BREP (Figure 2.2c) fully describes the shape using the boundary surfaces to create a watertight shell. The connection of the several faces is described through the topology. From [33]



(a) Triangular mesh of the Stanford bunny.  
From [34].



(b) A voxel grid with one colored voxel.  
From [35].

Figure 2.3.: **Approximative geometry representation** – The first dataset is a visualization of the Stanford Bunny dataset using a triangular mesh (Figure 2.3a), the second dataset is a voxel dataset with a single voxel colored differently (Figure 2.3b).

### 2.1.2. Approximate representation of geometry

Geometry is usually represented in an approximative sense if an exact representation is not feasible, not needed or too complicated (For example geometry see Figure 2.3). Using concepts of discretization a geometry can be approximated in one of the following ways:

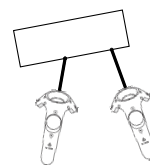
**Unstructured mesh** We represent the surface of a geometry or the interface between two regions using a mesh. In a two dimensional setting the mesh is a set of vertices and a set of lines connecting the vertices; in a three dimensional setting the mesh is a set of vertices that form polygonal shapes (very often triangles or quadrilaterals are used). Volumes are described by a watertight surface with face normals pointing into the inside or outside direction (depending on the convention).

In general a mesh geometry is very flexible and arbitrary shapes are described approximately using meshes. Anyhow, modification of meshes is difficult because there do not exist any control parameters and every vertex has to be modified on its own.

Mesh representations of geometry are widely used in computer graphics, because they can provide a visualization of the geometry in a cheap and efficient way. Exchange formats for mesh geometry are *stereolithography* (.stl) and .obj file format.

**Structured cells** A pixel (2D) or voxel (3D) grid is a structured, Cartesian grid of cells. Using this representation, additional volume information can be stored and the surface of the geometry is only represented implicitly by the volume information: In a 2D pixel image we do not describe a boundary but identify region of different color. Each cell is defined by its grid coordinates and its value.

Cell geometry is bound to the resolution of the underlying mesh and – without further improvements – not adaptive. Features that are finer than the resolution cannot be



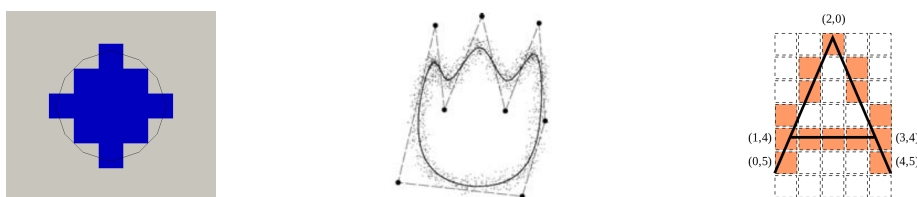
resolved. Modifying voxel geometry and performing computations on voxel geometry is particularly simple due to the simple Cartesian structure of voxel grids.

Voxel representations of geometry are often used in simulations due to their simple structure – many topology optimization frameworks use a voxel representation of the geometry, where each voxel represents a cell of variable density. A possible way of storing voxel data in 3D is the `.vtk` format, examples for 2D pixel data formats are the widely used image formats `.bmp`, `.jpg` and `.png`.

### 2.1.3. Geometry conversion and comparison

Depending on the intended application and the storage format of the data, geometry representations have to be converted. There exist a number of concepts for transforming one representation of geometry into another one (see Figure 2.4):

- **Contouring:** Contouring allows us to create a contour at a certain threshold value. This converts voxel or pixel data to mesh data and the boundary between regions of different value is described explicitly. Common algorithms are marching cubes [36] or dual contouring [37].
- **Fitting:** Fitting allows us to fit parametric curves or surfaces to point or mesh data. Different algorithms, like least squares fitting of NURBS exist [38].
- **Voxelization:** Voxelization is the process of converting mesh data or parametrized curves into their voxel or pixel<sup>1</sup> representation. We can either detect the voxels that are overlapped by a curve or surface (surface voxelization), or the voxels that are inside of a closed contour (solid voxelization)[39]. We either provide simple boolean information, whether a voxel is overlapped by the shape, respectively lies inside or outside, or we provide more detailed information like color or distance to the surface.



(a) Contour. Created with [40]. (b) NURBS fit. From [38]. (c) Rasterization. From [41].

Figure 2.4.: **Conversion of geometry representations** – We create the contour (black) of a pixel image to obtain a mesh representation (Figure 3.10b), fit a NURBS curve to a set of point data to obtained a parametric representation (Figure 2.4b) or rasterize a vector image of the letter "A" to obtain a pixel version (Figure 2.4c).

---

<sup>1</sup>Here the term rasterization is used

## 2.2. Computer-aided design

“*Computer-aided design* (CAD) can be defined as the use of computer systems to assist in the creation, modification, analysis or optimization of a design.” ([5], p.4) In the following we will first give a short overview over existing CAD software and the general architecture of a CAD program. Then we give an overview over the open source CAD kernel *Open CASCADE Technology* (OCCT), that is used for geometric modeling, and the open source CAD software FreeCAD, which relies on OCCT.

### 2.2.1. Philosophy and requirements

Due to the wide range of demanded functionality CAD systems are usually very complex pieces of software. There are many different software packages for different purposes: Powerful commercial packages are Siemens NX and AutoCAD [1, 2], but also open source tools like FreeCAD [4] and even online tools, only relying on a browser and a connection to the internet exist [3]. Also specialized niche products that only support a subset of CAD functionality exist: the design assistant SolidThinking Inspire [42] can be used for topology optimization; the modelling tool Blender [43] also supports techniques that are rather used by the computer graphics community than by engineers.

Usually the user is able to perform different operations through a *graphical user interface* (GUI). The toolset that the software offers usually depends on the field of application and type of task the user wants to perform: Software for architectural design offers tools for building doors and windows, while engineers need tools that help in the construction of screws and threads.

At their core CAD systems rely on the kernel that handles the geometric modelling and provides the algorithmic backbone. There exist commercial kernels (Parasolid used by Siemens NX, ACIS used by AutoCAD [44, 45]) and some open source kernels like OCCT [21].

Most CAD systems provide a mixture of BREP and CSG for the exact description of geometry. Approximative mesh geometry formats are often only supported as exchange formats, since they cannot be modified as easily as parametrized geometry (see section 2.1). OCCT uses at its core BREP for geometry representation, while boolean operations on primitive shapes are supported as well. The Online CAD system Tinkercad [6] solely relies on CSG.

### 2.2.2. Open CASCADE Technology

For our work we rely on the open source kernel *Open CASCADE Technology* (OCCT), that is written in C++. OCCT is designed in an object oriented manner and provides a mature, extensible and portable architecture. A wide range of functionality that is required in the context of CAD, but also advanced frameworks for visualization and application development exist (for technical overview see Figure 2.5). In the scope of this thesis we use version 7.0.0 of OCCT. For licence information and a thorough explanation please refer to [46].



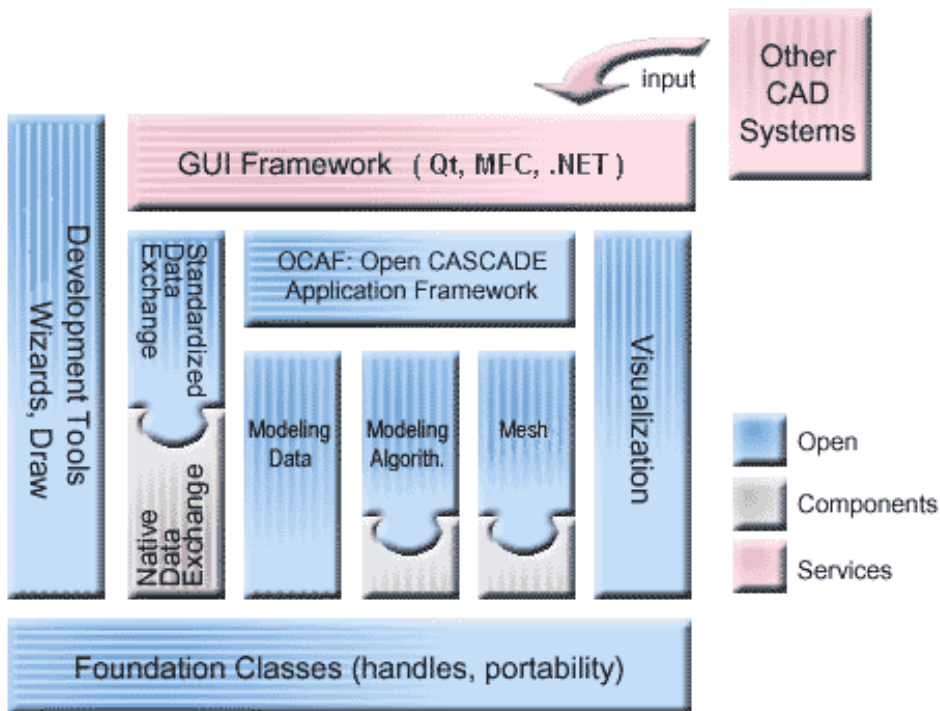
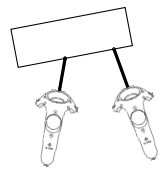


Figure 2.5.: **OCCT technical overview** – OCCT focuses on geometrical modelling (the modules Modelling Data, Modelling Algorithms and Mesh). But many additional frameworks exist. From [21].

OCCT models geometry through BREP, but also supports CSG concepts like boolean operations on geometry. Modelling capabilities of OCCT range from simple 2D geometry to complex freeform surfaces. Additionally approximation and interpolation algorithms exist for the generation of geometry. OCCT also supports the triangulation of surfaces and sampling of parametric curves. The dedicated voxel module has been removed in the update from version 6.9.0 to version 7.0.0. For data exchange OCCT provides a framework for reading and writing `.step` files. With those features OCCT provides a host of functionality: Exact geometric modelling of shapes, an algorithmic backbone for approximation and interpolation, meshing of arbitrary geometry and writing files that comply with the `.step` file format. For useful tutorials and explanation on OCCT see [47]. In section B.1 we give code examples and detailed explanation of OCCT components that have been used in the scope of this thesis.

### 2.2.3. FreeCAD

As an example CAD tool we use FreeCAD. FreeCAD realizes geometric modelling through a mixture of the BREP and CSG philosophy and provides a similar toolset as professional CAD tools. For detailed explanation of the software, please refer to [4].

#### Architecture

FreeCAD is a freely available open source tool written in the programming language Python [48]. Additionally Python is used in the FreeCAD Python interpreter as a scripting language to allow the user to customize FreeCAD or perform complex operations, which cannot be realized or are very cumbersome using the GUI.

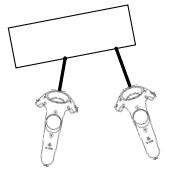
FreeCAD relies on the kernel OCCT. Additional technologies in use are Coin3D [49] for visualization of 3D geometry and Qt [50] for the creation of GUI elements. All the aforementioned tools provide a Python interface and can be accessed using the FreeCAD Python interpreter.

FreeCAD is designed in a modular and extensible way and provides an interface for homebrewn extensions through so-called *workbenches*. Adding new features to FreeCAD is explicitly supported by the developers: Collaborators can design new workbenches and easily access the libraries that are already in use by the main program for tasks of modelling, visualization and user interactions. This guarantees a coherent user interface and workflow.

#### Possible use cases for FreeCAD

With its open architecture, FreeCAD can be used in a variety of different contexts. The following use cases of FreeCAD are relevant in the scope of our work:

1. **CAD tool:** FreeCAD is a freely available CAD tool. FreeCAD supports the design and modification of parts and the visualization of `.step` geometry. Therefore, FreeCAD is a free alternative to the commercial packages.
2. **OCCT prototyping:** Using the FreeCAD Python interpreter we have an easy-to-use implementation of OCCT at hand. Therefore, we can prototype ideas and see the results in one single application. After validation, the prototype is finalized using the C++ implementation of OCCT.
3. **Extensible CAD system:** FreeCAD being open source allows us to customize the tool to our needs. The possibility of adding workbenches is very helpful in this context.



## 2.3. Virtual reality

In this section we shortly summarize available VR and *augmented reality* (AR) hardware (see subsection 2.3.1) and give a short overview over design tools for VR and AR (subsection 2.3.2). We describe the game engine Unity3D that we are going to use for development in the scope of this thesis (see subsection 2.3.3) and we introduce the asset SteamVR, that provides a high level interface between the the VR hardware and Unity3D (see subsection 2.3.4).

### 2.3.1. Hardware

We distinguish between *virtual reality* (VR) and *augmented reality* (AR) hardware: In AR the real world is augmented with artificial, virtual components. In VR the action takes place in a purely virtual environment. Many VR setups use a *head-mounted display* (HMD) worn by the user, that provides a stereoscopic 3D image and hides the real world. But there also exist CAVE setups where the image is projected onto the walls of a room. No HMD is needed and the user can see his own body [51]. In VR and AR the position of the user's head is tracked in space and the virtual environment is modified correspondingly. Therefore, the user gets the impression of acting in a real 3D environment.

In some setups the user is only an observer of the 3D environment [52]. But often the user is able to interact with the environment through a wide range of input devices: Traditional gamepads are just, but also special controllers with a spatial tracking in 3D space exist [8, 9, 53, 54]. There are devices that support a direct tracking of the user's hands [55].

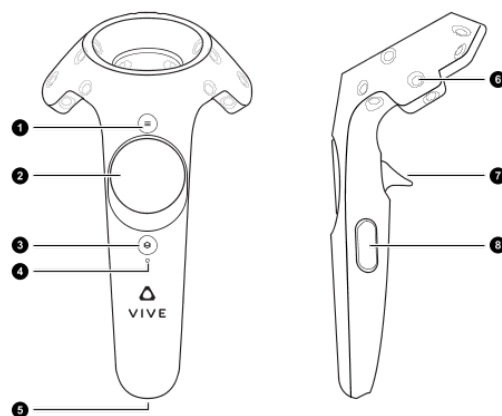


Figure 2.6.: **The Vive Controller** – Menu button (1), Trackpad (2), System button (3), Status Light (4), Micro-USB port (5), Tracking sensor (6), Trigger (7), Grip button (8). Picture and description from [8].

There exists a variety of VR and AR Hardware: An example for AR hardware is the HoloLens [56]. Pokemon GO [57], a recently popular AR game that is played on the smartphone, where the user does not wear a HMD. The Oculus [9] and the Playstation VR [54] can be controlled using a gamepad without spatial tracking or with optionally available controllers with spatial tracking. The Google Daydream [53] and the Google Cardboard [52] are HMD where a Smartphone has to be plugged in. In the following we concentrate on the Vive [8] as an example VR hardware package.

The Vive developed by HTC and Valve is a HMD, where the position of the head of the user is tracked. One can either use the HMD in a seating position or while moving around in a certain area. With a stereoscopic display the Vive comes with full 3D support. Additionally a pair of controllers can be used for user input (see Figure 2.6), the position of the controller is tracked and can be visualized in the VR environment.

In the last few years VR technology has reached the consumer market: In 2012 Oculus issued its Kickstarter campaign and gathered over US\$2.4 million. Facebook acquired Oculus in 2014 for an estimated US\$2 billion [58]. With Vive and Oculus, today's VR hardware is relatively affordable and accessible for a wide range of people. Good positional tracking and a high resolution stereoscopic HMD are provided. For both systems high level programming interfaces exist [59, 60] and there is a large community. With this, the development of advanced software applications for VR is possible and there is a large field of potential users.

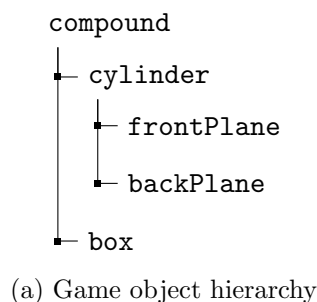
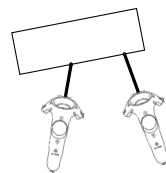
### **2.3.2. Design in virtual reality**

With stereoscopic visualization and input devices with 3D positional tracking VR and AR provide great possibility for design in 3D space. In the following we provide two examples of design tools in VR that inspired our work.

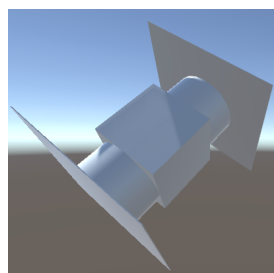
The VR interfaces for the Unreal Engine and for Unity3D allow the user to design complex scenes for 3D games in a virtual environment [12, 13]. Here the user has the possibility to inspect the whole scene in the role of an immersed observer. 3D positioning and scaling happens by "grabbing" objects using the controllers in an intuitive and easy to use way. A virtual tablet provides access to advanced functionality and a snapping mechanism for objects is implemented.

In the 3D sketching tool Tiltbrush [11] the user can create 3D sketches in a virtual environment. The user paints by moving the controller through the air, the controller is used like a brush and colorful lines are created. The user can choose from a wide range of different colors and painting devices with a menu that is attached to the second controller. Additional mechanisms for sketching of straight lines, redo/undo and export of data are provided.

We summarize that there exist design tools for constructive as well as creative work in VR. The tools heavily rely on the positional tracking of HMD and controller as well as the stereoscopic imaging for providing a more intuitive user experience. For more examples of design tools in VR and AR please refer to [20].



(a) Game object hierarchy



(b) A simple game object in the Unity editor

Figure 2.7.: **Unity game object** – The compound *game object* (GO) (Figure 2.7b) can be positioned in 3D space while the parts keep their relative local position due to the hierarchical structure of the GO (Figure 2.7a).

### 2.3.3. Unity3D

The game engine Unity3D [22] is widely used for the development of computer games and other interactive 3D applications. Unity already comes with fundamental functionality for game development and 3D visualization. For complex behavior Unity supports the scripting languages C# and JavaScript. Unity’s functionality can easily be extended by plugins<sup>2</sup> from Unity’s AssetStore.

The basic data structure of Unity is the *game object* (GO) with a transform<sup>3</sup> as its only mandatory property. Using a component<sup>4</sup> one can customize the GO. By hierarchically adding additional GOs as children, that inherit the global position of the parent GO, one can build compound objects (see Figure 2.7).

### 2.3.4. Virtual reality software interface

In the context of VR we need a software framework that provides an interface to the VR hardware, consistent handling of the position of the HMD, and stereoscopic 3D rendering.

The Unity asset SteamVR [59, 61] provides the above mentioned functionality and gives access to the basic functionality of the Vive from within the Unity scripting environment. It allows us to track the position of the HMD and the controllers as well as detecting buttons being pressed. But Unity’s asset store also provides interfaces to other VR hardware: We draw the reader’s attention to [60] for support of the *Oculus* [9] and [62] for Android devices like the *Daydream* [53]. Therefore, we can easily switch from the Vive to a different VR hardware package.

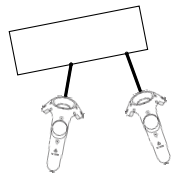
For user interaction in VR there exist several third party assets [63, 64]. They link actions performed with the controller to events that happen in the virtual environment.

<sup>2</sup>In the context of Unity a plugin is often also called an Asset.

<sup>3</sup>The transform is an object defining the GO’s position, rotation and scale in 3D space.

<sup>4</sup>A component can be anything from a mesh for geometry, a renderer for visualization of geometry or a script for complex behavior.

This supports the development of interactive applications, where the user can modify the environment by, for example, grabbing objects or interacting with widgets.



## 2.4. Topology optimization

Topology optimization allows us to optimize geometry of a continuum with respect to a given objective function. This is a very broad and general framework that is useful, for example, when designing specialized parts in the context of engineering problems.

Many professional and open source topology optimization frameworks exist [42, 65–67]. In this thesis we use the prototype *Interactive Design Assistant* (IDeAs) developed at Siemens, Corporate Technology [26]. IDeAs allows us to create a part with maximum stiffness.

### 2.4.1. Theory

We use topology optimization as a black-box: In the following we do not give a rigorous explanation of the underlying mathematical framework of topology optimization (see [68]), but only concentrate on the basics.

One popular approach to topology optimization is the *Solid Isotropic Microstructure with Penalization* (SIMP) approach, where the optimization domain is discretized with hexahedral elements with associated density values  $\rho$ . The density of each element is used as a design variable in an optimization problem where the compliance  $c(\rho)$ , a measurement for the work done by external loads, is minimized. The optimization procedure is constrained by a target volume fraction  $\alpha$  and the possible density values  $\rho_{\min} < \rho < \rho_{\max}$ .

As an additional constraint the linear elasticity boundary value problem has to be fulfilled. Using *finite element method* (FEM) the discretized form of the problem reads  $K_e(\rho)u = f$ , where  $K_e(\rho)$  denotes the stiffness matrix, that depends on the density distribution,  $u$  denotes the displacement and  $f$  the consistent force vector.

### 2.4.2. Input and output

The loads and fixtures, together with the optimization domain, define the design space of the topology optimization problem. For an example input setup see Figure 2.8. The optimization problem is solved using an iterative scheme (for more detail see [26]).

The final result of the topology optimization process is a mesh with hexahedral elements with an optimum a density value  $\rho$ . From the density values we can get information about the properties of the optimized structure:

- If  $\rho = \rho_{\max}$ , this means that material with the maximum density is needed at this place in order to obtain a structure that meets the requirements.
- If  $\rho = \rho_{\min}$ , this means that no material is needed at this place.

In order to manufacture an optimized part, the user usually defines a density threshold  $\rho_{\min} \leq \theta \leq \rho_{\max}$ . This threshold divides regions with material, where the density is above the threshold  $\rho > \theta$ , from regions without material, where the density is below the threshold  $\rho < \theta$ .

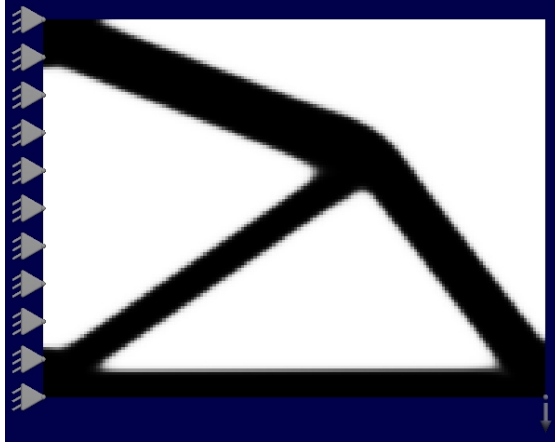


Figure 2.8.: **Screenshot from Interactive 2D TopOpt App** – Screenshot visualizes the necessary boundary conditions for a topology optimization problem: fixtures (triangles at the left border), forces (arrow at the right border) and the optimization domain (white box partially filled with black material). The resulting structure is optimal with respect to the boundary conditions. Interactive App from [69].

## 2.5. Level set and voxel description

For postprocessing the voxel output of the topology optimization process, we use level set methods. In this section we shortly describe level set methods and fast marching methods and their application in computational geometry and CAD. The descriptions are based on [70]. For a thorough introduction to the topic please refer to [70].

### 2.5.1. Level sets

Propagating interfaces are usually described in an explicit Lagrangian approach: parametrized functions or sets of vertices and edges define a curve or surface  $\Gamma$  that separates two regions. In the level set framework interfaces are described from an Eulerian perspective, where the interface is implicitly given as the isocontour of a multidimensional function.

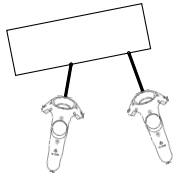
**The boundary value problem:** For an interface  $\Gamma$  moving with normal propagation speed  $F > 0$  we define the arrival time function  $T(\vec{x})$  that describes the arrival time of the moving interface  $\Gamma$  at the location  $\vec{x}$ . The arrival function  $T$  is defined through the following boundary value problem:

$$|\nabla T| F = 1, \quad T = 0 \text{ on } \Gamma, \quad F > 0 \quad (2.1)$$

The arrival function  $T(\vec{x})$  implicitly defines the propagating interface  $\Gamma(t)$  as

$$\Gamma(t) = \{\vec{x} \mid T(\vec{x}) = t\}.$$





Therefore,  $\Gamma(t = 0)$  describes the initial shape of the interface, while  $\Gamma(t = 1)$  describes the shape at  $t = 1$ . If we choose  $F = 1$ , the arrival time function  $T(\vec{x})$  represents the distance of any position  $\vec{x}$  to the interface  $\Gamma$ ; negative values denote regions inside of the interface, positive values denote regions outside of the interface. In this special case we call  $T(\vec{x})$  a *signed distance function* (SDF) and  $T$  fulfills  $|\nabla T| = 1$ .

**The initial value problem:** The arrival time function  $T(\vec{x})$  does not allow negative travelling speed  $F$ , since otherwise  $T(\vec{x})$  would not be single valued at arbitrary locations  $\vec{x}$ . Therefore, we define the time dependent level set function  $\phi(t, \vec{x})$  with the interface  $\Gamma(t)$  being the zero level set of  $\phi$ :

$$\Gamma(t) = \{\vec{x} \mid \phi(t, \vec{x}) = 0\}.$$

The level set function is defined by the following initial value problem:

$$\phi_t + F |\nabla \phi| = 0 \quad \text{given } \phi(t = 0, \vec{x}), \quad (2.2)$$

with arbitrary normal propagation speed  $F$ .

**Discretization:** By discretizing the functions  $T(\vec{x})$  and  $\phi(t, \vec{x})$  on a uniform grid

$$\Omega_h = \{\vec{x}_{ijk} = (x_i, y_j, z_k) \mid 0 < i < n_x, 0 < j < n_y, 0 < k < n_k\},$$

with

$$\begin{aligned} x_i &= x_0 + i\Delta x, \\ y_j &= y_0 + j\Delta x, \\ z_k &= z_0 + k\Delta x, \end{aligned}$$

we can numerically treat the associated partial differential equations (2.1) and (2.2). The interface can be visualized easily using common algorithms for the visualization of isocontours [36].

### 2.5.2. Fast marching method

In order to solve the boundary value problem (2.1), we have to first initialize the arrival function  $T$  such that  $T = 0$  on  $\Gamma$ . For this purpose we approximate the distance from the interface to all gridpoints that are close to the interface  $\Gamma$ . A gridpoint is close to the interface, if the interface lies between that gridpoint and one of its neighbors.

After having properly initialized  $T$ , we can use the *fast marching method* (FMM) for solving the boundary value problem (2.1). Finally we obtain the arrival function  $T$ . The complexity of the algorithm is  $\mathcal{O}(M \log M)$  on a grid with  $M$  points. Explanation and implementation instruction can be found in [70].

### 2.5.3. Level set propagation

We easily obtain a valid initialization of  $\phi$  for the initial value problem (2.2) by setting  $\phi(t = 0, \vec{x}) = T(\vec{x})$ .

Choosing  $F$  in an appropriate way, the level set framework can be used for different applications: Setting  $F = -\kappa_M$ , where  $\kappa_M$  denotes the mean curvature of the interface, smoothens the surface. An expression of the mean curvature in terms reads

$$\kappa_M = \frac{\left( (\phi_{yy} + \phi_{zz}) \phi_x^2 + (\phi_{xx} + \phi_{zz}) \phi_y^2 + (\phi_{xx} + \phi_{yy}) \phi_z^2 - 2\phi_x \phi_y \phi_{xy} - 2\phi_x \phi_z \phi_{xz} - 2\phi_y \phi_z \phi_{yz} \right)}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^{\frac{3}{2}}}, \quad (2.3)$$

where we approximate the derivatives using finite differences.

Finally the following upwind scheme can be used for solving (2.2):

$$\phi_{ijk}^{n+1} = \phi_{ijk}^n - \Delta t \left[ \max(F_{ijk}, 0) \nabla^+ + \min(F_{ijk}, 0) \nabla^- \right],$$

where

$$\begin{aligned} \phi_{ijk}^n &= \phi(t_n, x_i, y_j, z_k) = \phi(t_n, \vec{x}_{ijk}) \\ F_{ijk} &= F(x_i, y_j, z_k) = F(\vec{x}_{ijk}) \end{aligned}$$

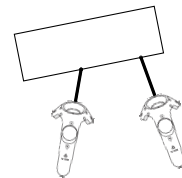
and

$$\begin{aligned} \nabla^+ &= \left[ \max(D_{ijk}^{-x}, 0)^2 + \min(D_{ijk}^{+x}, 0)^2 + \right. \\ &\quad \max(D_{ijk}^{-y}, 0)^2 + \min(D_{ijk}^{+y}, 0)^2 + \\ &\quad \left. \max(D_{ijk}^{-z}, 0)^2 + \min(D_{ijk}^{+z}, 0)^2 \right]^{\frac{1}{2}} \\ \nabla^- &= \left[ \max(D_{ijk}^{+x}, 0)^2 + \min(D_{ijk}^{-x}, 0)^2 + \right. \\ &\quad \max(D_{ijk}^{+y}, 0)^2 + \min(D_{ijk}^{-y}, 0)^2 + \\ &\quad \left. \max(D_{ijk}^{+z}, 0)^2 + \min(D_{ijk}^{-z}, 0)^2 \right]^{\frac{1}{2}}, \end{aligned}$$

with  $D_{ijk}^{\pm\alpha}$  denoting the forward, respectively backward, difference operator in  $\alpha$  direction at  $\vec{x} = (x_i, y_j, z_k)$ .

Additionally we require the following CFL condition:

$$\Omega \max F \Delta t \leq \Delta x.$$



#### 2.5.4. Boolean operations on level sets

The common Boolean operations on two shapes  $\Omega_A$  and  $\Omega_B$  can be applied to level sets:

$$\begin{aligned}\Omega_A \cup \Omega_B &= \min(\phi_A, \phi_B) \\ \Omega_A \cap \Omega_B &= \max(\phi_A, \phi_B) \\ \Omega_A - \Omega_B &= \max(\phi_A, -\phi_B),\end{aligned}$$

where  $\phi_{A,B}$  denote the respective level set representations of  $\Omega_{A,B}$ .

#### 2.5.5. Visualization Toolkit

In order to be able to work with level set and voxel datasets, we need a toolset that allows us to describe, store and visualize level set and voxel data: For this purpose VTK can be used. VTK [25] is a powerful open-source library for computer graphics, image processing and visualization. VTK is originally written in C++, but also provides an interface to the programming language python.

In this thesis we only need a small fraction of VTK's actual functionality, for a thorough explanation of VTK please refer to VTK's userguide [71], the VTK textbook [72] and the reference documentation [73]. For a detailed explanation of VTK functionality used in the scope of this thesis please refer to the material in section B.2.

### 2.6. B-splines

B-splines are a special type of parametric function that is, due to its flexibility, widely used in CAD for modelling of curves and surfaces. We do not explicitly introduce NURBS, regardless they are important in CAD, because they are not needed in the scope of this thesis.

In this section we first give a definition of B-spline curves and surfaces (see subsection 2.6.1) and explain basic concepts. After that we describe continuity requirements for joined B-spline segments or patches (subsection 2.6.2).

For detailed explanation of NURBS and B-spline theory we refer to [30] and [74].

#### 2.6.1. B-spline curves and surfaces

**B-spline curve** A B-spline curve is a parametric curve that is defined in the following way:

$$\vec{C}(\xi) = \sum_{i=1}^n N_i^p(\xi) \vec{P}_i,$$

where  $\xi$  describes the parametric coordinate, where the curve is evaluated,  $\vec{P}_i$  are the control points of the curve,  $n$  denotes the total number of control points and  $N_i^p(\xi)$  is the  $i$ -th B-spline basis function of polynomial degree  $p$  defined through the Cox-de Boor recursion formula:

- for  $p = 0$ :

$$N_i^0(\xi) = \begin{cases} 1 & \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

- for  $p > 0$ :

$$N_i^p(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_i^{p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1}^{p-1}(\xi).$$

The knot vector  $\vec{\xi}$  is vector of non-decreasing values that define the parametric space of our curve.

**B-spline surface** By using a tensor product, we can construct the B-spline surface

$$\vec{S}(\xi, \eta) = \sum_{i=1}^n \sum_{j=1}^m N_{i,j}^{p,q}(\xi, \eta) \vec{P}_{i,j},$$

with  $n \cdot m$  control points  $\vec{P}_{i,j}$ . The two-dimensional B-spline basis function

$$N_{i,j}^{p,q}(\xi, \eta) = N_i^p(\xi) N_j^q(\eta)$$

has an additional polynomial degree  $q$  and an additional knot vector  $H = [\eta_1, \eta_2, \eta_{m+q+1}]$  related to the second parametric direction  $\eta$ .

### 2.6.2. Continuity

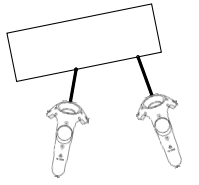
**$G^0$  continuity of B-splines** Since a B-spline curve with an *open knot vector* interpolates the first and last control point,

- a single curve is closed into a loop by setting  $\vec{P}_1 = \vec{P}_n$  and
- two curves are connected by setting  $\vec{P}_n = \vec{Q}_1$ .

For surfaces the same rules apply and we require

- $\vec{P}_{1,j} = \vec{P}_{n,j} \forall j = [1, m]$  for joining two opposite edges of a single patch to form a cylindrical surface and
- $\vec{P}_{n,j} = \vec{Q}_{1,j} \forall j = [1, m]$  for joining two different patches in  $\xi$  direction along the edges pointing in  $\eta$  direction. Two patches are connected at arbitrary edges applying similar rules.

Such a joint is called  $G^0$  continuous.



**$G^1$  continuity of B-splines** Anyhow, setting the first and last control point of two different curves (or of one and the same curve respectively) equal, only guarantees  $G_0$  continuity. In order to accomplish a  $G_1$  continuous joint<sup>5</sup>, additionally to  $G_0$  continuity<sup>6</sup> we require

$$\frac{\partial}{\partial \xi} \vec{C}^1(\xi_n) = \alpha \frac{\partial}{\partial \xi} \vec{C}^2(\xi_1).$$

The first derivative of the B-spline basis function reads

$$\frac{\partial N_{i,p}(\xi)}{\partial \xi} = \frac{p}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) - \frac{p}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi).$$

Inserting this into the definition of the B-spline curve with open knot vector yields after some simplifications (consult [30] for details) the condition for  $G^1$  continuity of a B-spline curve:

$$(\vec{Q}_2 - \vec{Q}_1) = \alpha (\vec{P}_n - \vec{P}_{n-1}).$$

Since  $\vec{Q}_1$  and  $\vec{P}_n$  have to be identical due to  $G^0$  continuity,  $G^1$  continuity requires the additional property, that the two control points  $\vec{Q}_2$  and  $\vec{P}_{n-1}$  are collinear with the control point  $\vec{Q}_1 = \vec{P}_n$  where both curves meet.

For B-spline patches we require this criterion across the whole connecting edge

$$(\vec{Q}_{2,j} - \vec{Q}_{1,j}) = \alpha (\vec{P}_{n,j} - \vec{P}_{n-1,j}) \quad \forall j.$$

**$G^2$  continuity of B-splines** We require  $C^0$ ,  $G^1$  and additionally

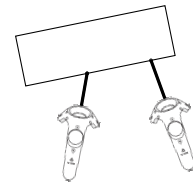
$$\vec{Q}_1 + \frac{1}{\alpha} (\vec{Q}_1 - \vec{Q}_2) = \vec{P}_{n-1} + \alpha (\vec{P}_{n-1} - \vec{P}_{n-2}).$$

---

<sup>5</sup>we only require *geometric* ( $G_1$ ) continuity, not *parametric* ( $C_1$ ) continuity; therefore we introduce the arbitrary scaling factor  $\alpha \neq 0$ .

<sup>6</sup>per definition  $G_n$  continuity always requires  $G_{n-1}$  continuity plus additional constraints.





### 3. Design Workflow

In this chapter we describe how the methods described in chapter 2 are combined to accomplish the task of designing a topology optimized part and then partially reconstructing it in a *virtual reality* (VR) environment.

Throughout this section we use the GE Bracket [28] as an example problem. It originates from a design challenge issued by GrabCAD and General Electric in 2013, where the design of a jet engine bracket has to be optimized in order to obtain a lightweight part. The optimized part has to comply with a set of requirements: Firstly the part has to withstand different load scenarios without breaking and, secondly, load interfaces, where loads are applied or the part is fixed, should remain unchanged. The load interfaces are given as geometry in *Standard for The Exchange of Product model data* (.step) file format. The winner of the competition [75] was finally able to reduce the weight of the initial design by nearly 84% (see Figure 3.1).

We propose topology optimization methods for the design of lightweight structures. We describe the necessary steps in modelling and simulation for the creation of a topology optimized geometry in section 3.1. In order to guarantee that the optimized design complies with boundary condition geometry, we introduce an intermediate step before the actual geometry reconstruction starts. Here we finally obtain a consistent description of boundary conditions and optimized geometry (section 3.2). The resulting level set is used in the following geometry reconstruction step (section 3.3), where we use VR technology for the extraction of B-spline curves and *non-uniform rational B-spline* (NURBS) surfaces (section 2.6). In a postprocessing step we use a professional *computer-aided design* (CAD) tool with a rich toolset for geometric modelling in order to create the final CAD geometry. The design workflow is visualized in Figure 3.2.

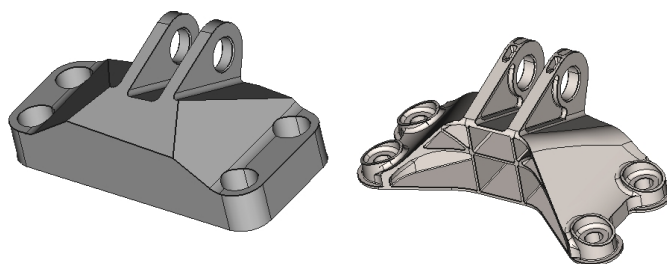


Figure 3.1.: **GE jet engine bracket designs** – The original part (left) and the winner of the challenge right (right) with a weight reduction of nearly 84% from 2033 grams to 327 grams. Original design from [28] and optimized design from [75]. Visualized using FreeCAD [4].

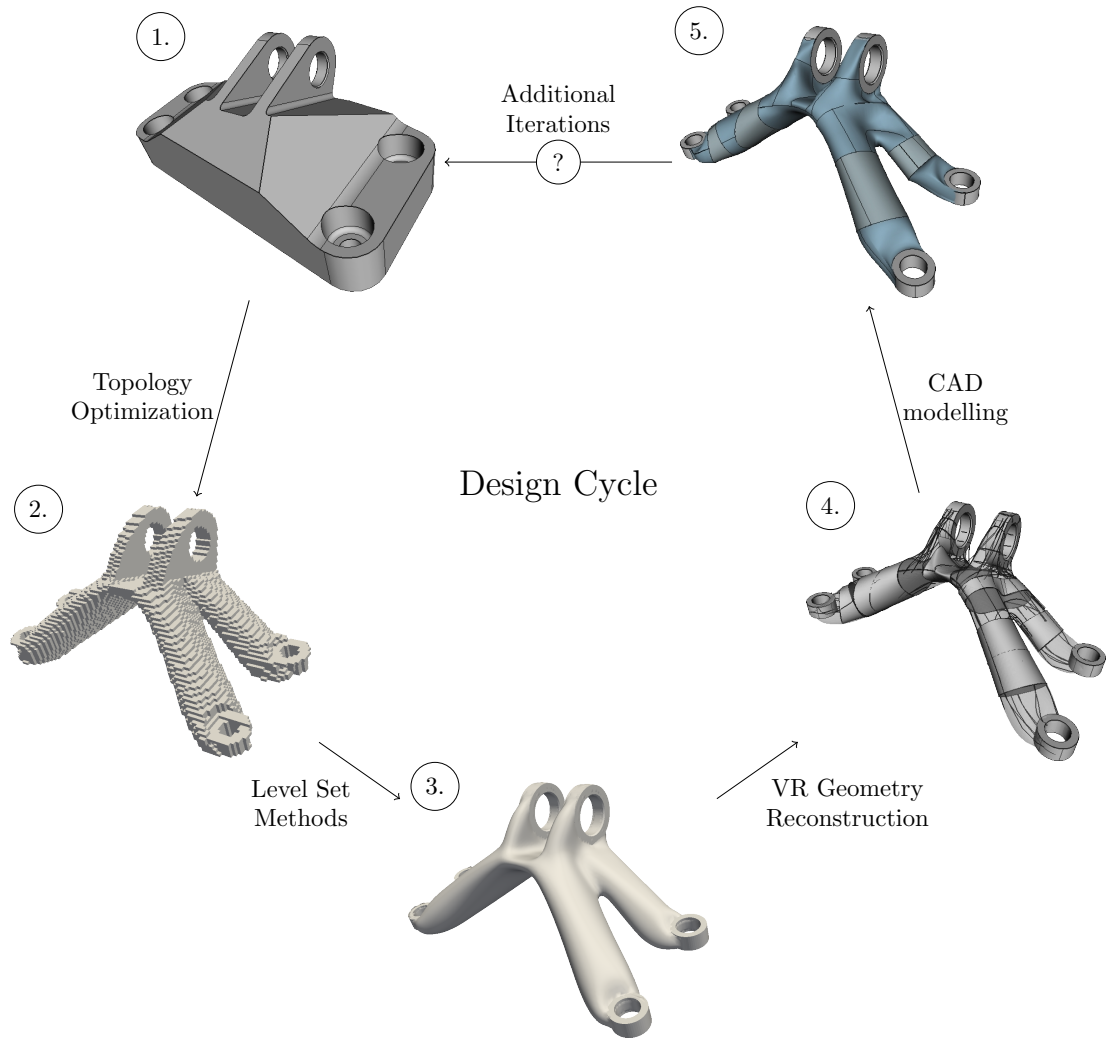
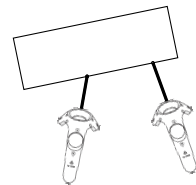


Figure 3.2.: **Design workflow realized in this thesis** – We start with an initial design (1), optimize the topology (2), create a consistent geometry representation using level set methods (3), extract cross sections, loft surfaces and guiding lines from the mesh using VR technology (4) and finally model the geometry using a full fledged CAD tool (5). The resulting part can be used for additional iterations.





## 3.1. Topology optimization

*Interactive Design Assistant* (IDeAs), a prototype for fast parallel topology optimization developed at Siemens, Corporate Technology [26], supplies the topology optimization framework described in section 2.4.

In subsection 3.1.1 we explain how to define the parameters that are necessary in order to start a topology optimization process using IDeAs. Secondly in subsection 3.1.2 we describe the resulting output.

### 3.1.1. Definition of boundary conditions

As already described in section 2.4 topology optimization is a procedure that tries to find an optimal geometry with respect to a certain goal and a set of boundary conditions. Using IDeAs we can easily create a lightweight structure using topology optimization techniques. In IDeAs the boundary conditions are provided through geometry shapes in `.step` file format (see Figure 3.3):

- **Initial geometry:** Geometry of the original, not optimized part. Defines the initial material density distribution.
- **Forces:** Regions where forces are applied to the part.
- **Fixtures:** Regions where the part is fixed and no displacement occurs.
- **Optimization domain:** Region, where the algorithm may modify the initial material density distribution. In regions outside of the optimization domain the initial geometry shall be conserved. This is for example useful if a region should remain empty (void regions)<sup>1</sup> or the exact shape has to remain untouched<sup>2</sup>.

Additional quantities – like material properties or discretization parameters – have to be implemented as well.

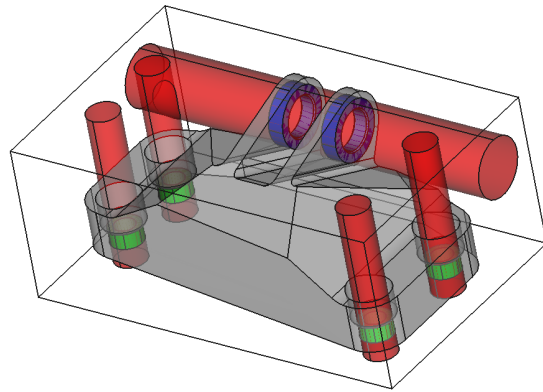
### 3.1.2. Optimized voxel geometry

The optimized geometry is represented with a density field on a regular voxel grid. By specifying a density threshold  $\theta$  we can visualize the resulting geometry as a set of voxels that have a density value  $\rho \geq \theta$ . The `.vtk` file format [25] is used for outputting and saving data (Figure 3.4).

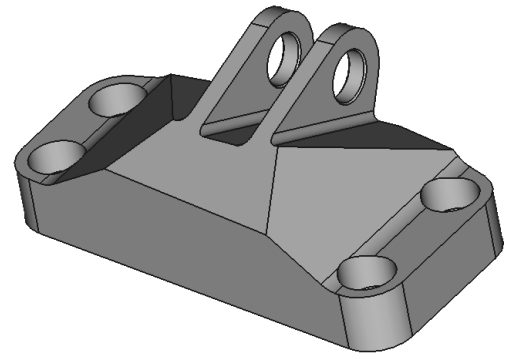
---

<sup>1</sup>One possible reason is if space for additional parts is needed in this region.

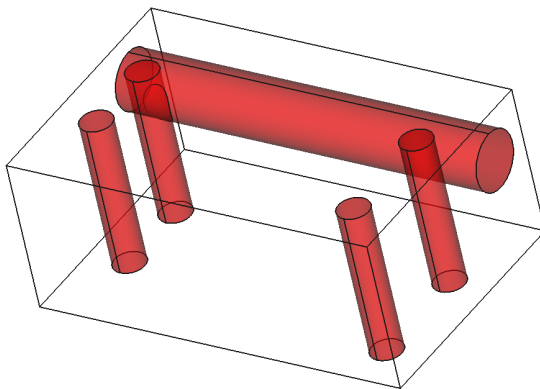
<sup>2</sup>The thread of a screw, for example, must not be changed.



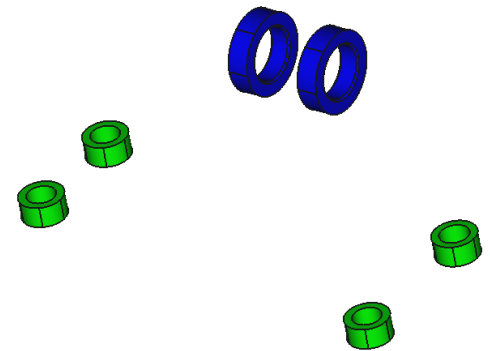
(a) Full input geometry



(b) Initial geometry to be optimized

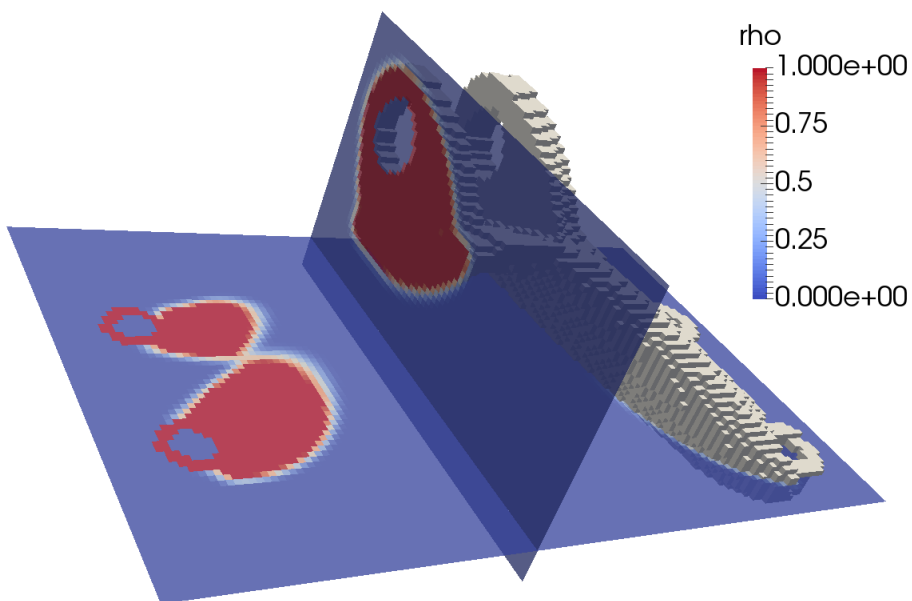
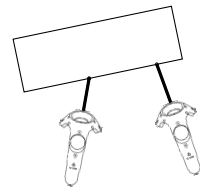


(c) Optimization domain with not optimized regions (red).

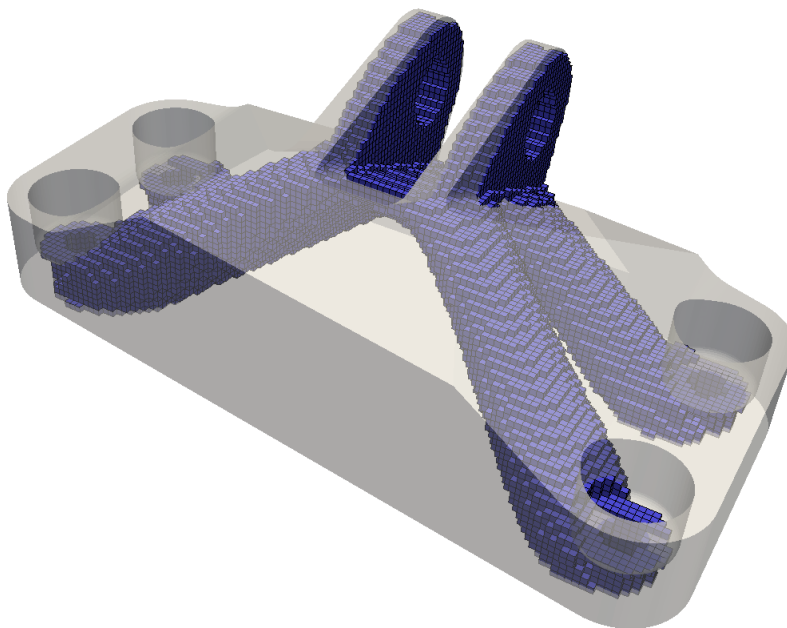


(d) Parts with forces (blue) and fixtures (green).

Figure 3.3.: **IDeAs input geometry** – IDeAs accepts input geometry defining the topology optimization problem (Figure 3.3a). The part (Figure 3.3b) is optimized in a given optimization domain, where certain regions can explicitly be excluded from optimization (Figure 3.3c). Force and fixture boundary conditions are applied in certain regions (Figure 3.3d). Note that the boundary condition regions are not lying inside the optimization domain and therefore remain untouched. Geometry visualized in FreeCAD [4]. The problem statement is based on [28].



(a) Voxel dataset



(b) Comparison of original (grey, transparent) and optimized (blue, opaque) shape.

Figure 3.4.: **IDeAs output** – The optimized geometry is output as a density field on a regular voxel grid. We visualize the optimized geometry, by only displaying voxels with a density value  $\rho$  above the threshold value  $\rho_t = 0.5$  (Figure 3.4a). Comparing the optimized to the original geometry we observe that the amount of material has been reduced (Figure 3.4b). Visualized using [40].

## 3.2. Consistent level set

In the following section we present the treatment of boundary condition geometry after having completed the topology optimization step described in section 3.1. The algorithm heavily relies on the level set methods introduced in section 2.5. This intermediate step between topology optimization (section 3.1) and geometry reconstruction (section 3.3) has been developed in the scope of this thesis and is necessary to create a representation of topology optimized voxel geometry, which is consistent with given boundary conditions.

IDeAs does not return a mesh representation of the optimized geometry, but a voxel representation with a density field  $\rho$  from which we can extract an optimal geometry by defining a density threshold  $\theta$  and creating a contour, where the density value  $\rho$  exceeds the threshold value  $\theta$ . This geometry is in general not satisfactory for the following reasons:

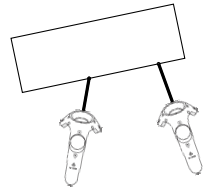
- Boundary condition shapes are not represented exactly.
- Regions outside of the optimization domain might have been modified during topology optimization.

In order to be able to create an optimized geometry that is consistent with the boundary conditions, we propose the following algorithm:

1. Transform all geometry to *signed distance functions* (SDFs) (Figure 3.5):
  - Interpolate the voxel density values onto the nodes, in order to obtain a spatially discrete representation of the density field  $\rho$ . This can be done easily using *Visualization Toolkit* (VTK).
  - Transform this dataset into a SDF  $\phi_0$  of the interface at the density threshold  $\theta$ . First we approximate the distance to the interface in a thin layer around the interface by detecting the roots in  $\rho(\vec{x}) - \theta = 0$ , where  $\theta$  denotes the density threshold where we want to create the interface, and then apply the *fast marching method* (FMM) for the initialization of the remaining domain.
  - Transform the optimization domain  $\Omega$  and the shapes outside of the optimization domain  $\bar{\Omega}$  – both defined as `.step` geometry – into their respective SDFs  $\phi_\Omega$  and  $\phi_{\bar{\Omega}}$  in a thin layer around the interface<sup>3</sup>. Then – again – use the FMM for the initialization of the remaining domain.
  - For all SDF use the same mesh that is already used for the voxel dataset.
2. Perform boolean operations on level sets (Figure 3.6):
  - Perform  $\phi_0 \cap \phi_\Omega = \phi_1$  to remove any material that is outside of the optimization domain.

---

<sup>3</sup>Here we use *Open CASCADE Technology* (OCCT) for a – costly – thin layer approximation based on existing `.step` geometry



- Perform  $\phi_1 \cup \phi_{\bar{\Omega}} = \phi_2$  to fuse the exact representation of untouched geometry and the optimized geometry.
- After each boolean operation perform a reinitialization step<sup>4</sup> in order to prevent a deterioration of the SDF.

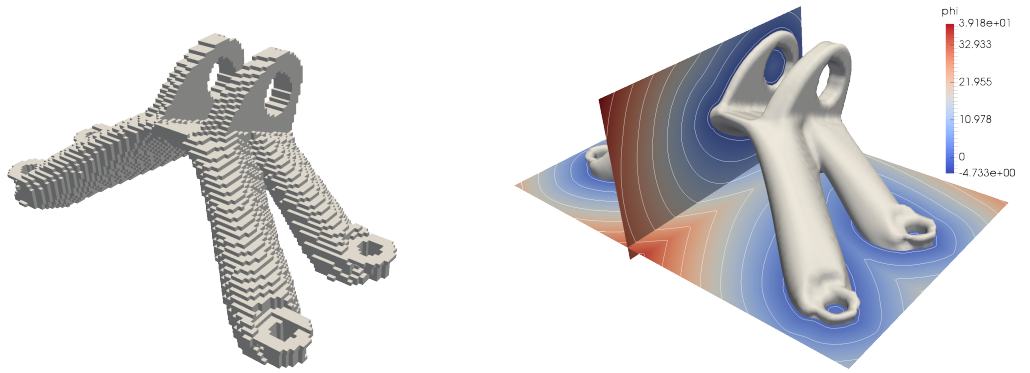
3. Smoothing of the result:

- Do some iterations of mean curvature smoothing on  $\phi_2$  to obtain a smooth geometry.
- Only perform smoothing inside the optimization domain  $\Omega$ . On  $\bar{\Omega}$  set the velocity equal to zero.

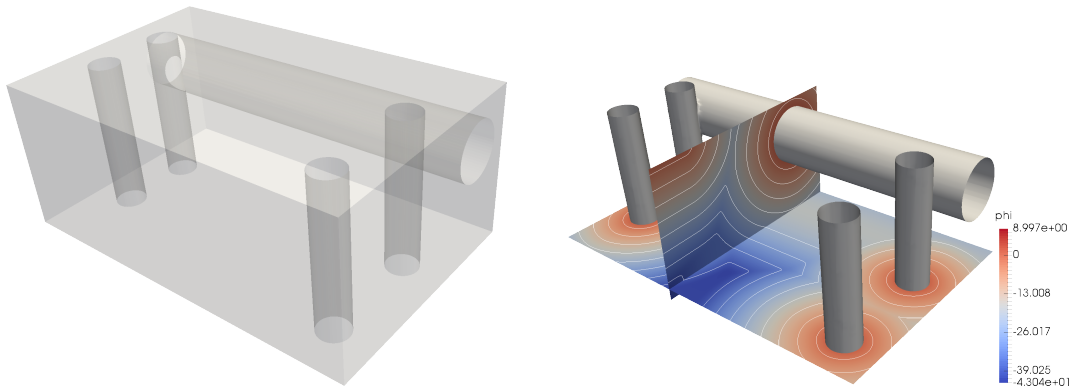
The resulting level set representation is consistent with the boundary conditions of the original optimization problem (See Figure 3.8).

---

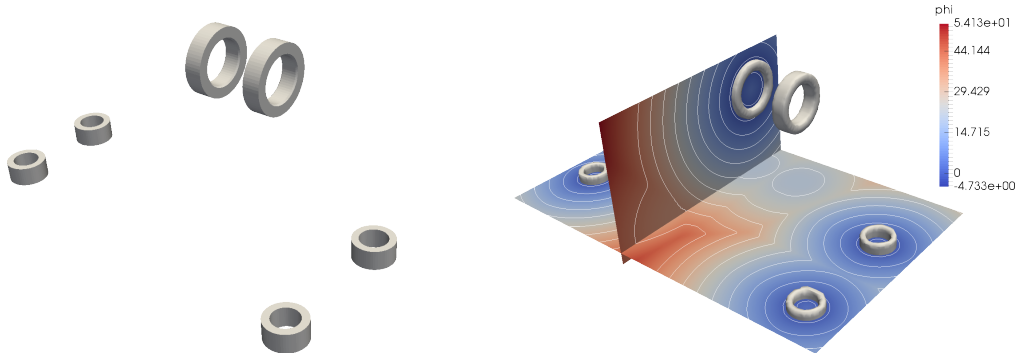
<sup>4</sup>This involves first performing a thin layer initialization at the interface, where  $\phi_i = 0$ , then applying FMM for the initialization of the remaining domain.



(a) Voxel representation of optimized geometry (b) SDF of geometry  $\phi_0$ . Isocontour at  $\phi_0 = 0$ . with threshold  $\theta = 0.5$ .

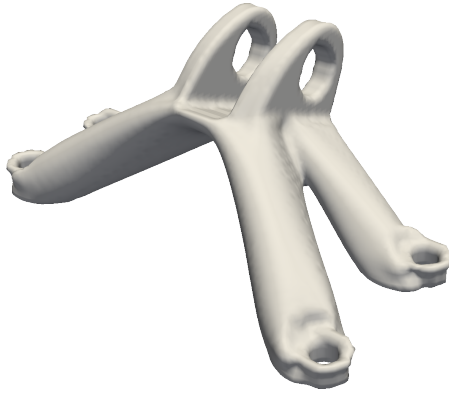
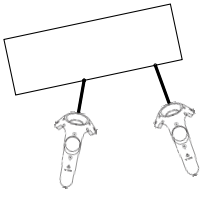


(c) .step representation of the optimization domain  $\Omega$ . (d) SDF of optimization domain  $\phi_\Omega$ . Isocontour at  $\phi_\Omega = 0$ .

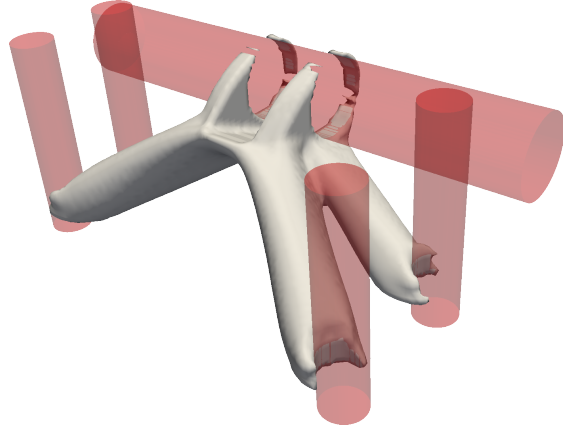


(e) .step representation of the unchanged domain  $\bar{\Omega}$ . (f) SDF of unchanged domain  $\phi_{\bar{\Omega}}$ . Isocontour at  $\phi_{\bar{\Omega}} = 0$ .

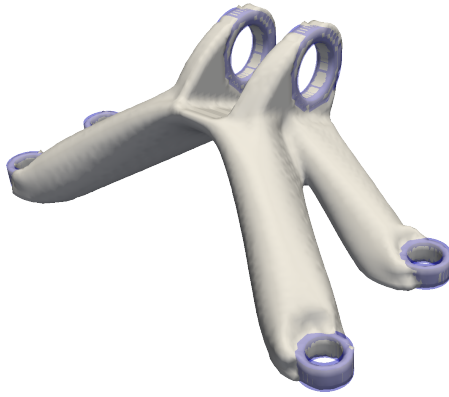
Figure 3.5.: **Transformation to level sets** – Transformation of the optimized voxel geometry (Figure 3.5a) to the SDF (Figure 3.5b) is done by first interpolating the voxel values onto the nodes and defining a surface at the density threshold  $\rho_t$ . We also transform .step geometry that defines the simulation domain (Figure 3.5c) and unchanged geometry (Figure 3.5e) to their respective SDFs (Figure 3.5d and Figure 3.5f). Visualized using Paraview [40]. .step files have been converted to .stl using FreeCAD [4].



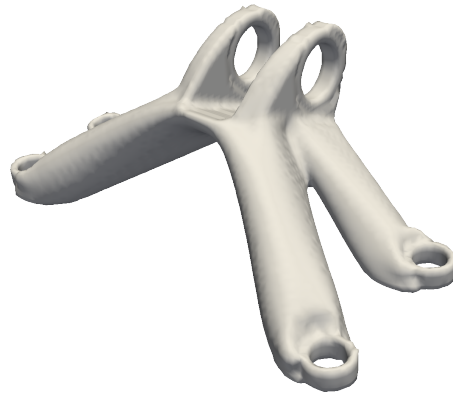
(a) Initial shape  $\phi_0$ .



(b) Shape  $\phi_1$  after removing unchanged regions (red).

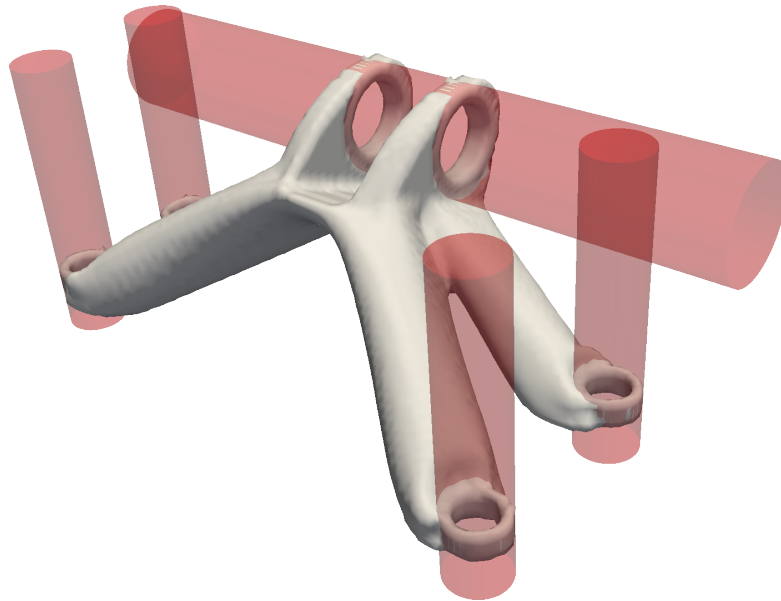


(c) Shape  $\phi_2$  after adding unchanged geometry (blue).

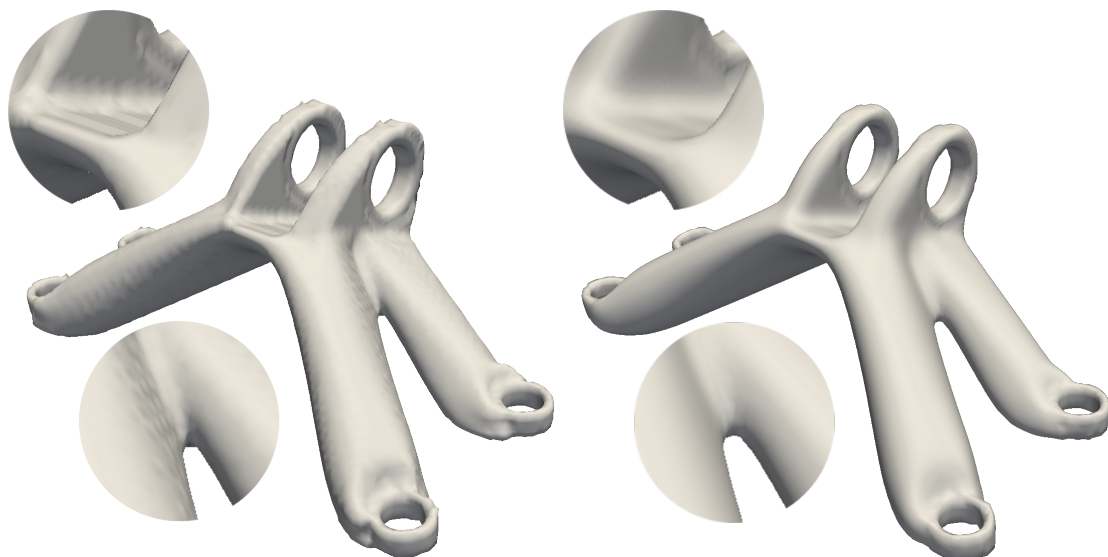


(d) Final shape  $\phi_2$ .

Figure 3.6.: **Boolean operations on level sets** – From the initial level set  $\phi_0$  (Figure 3.6a) we remove unchanged regions by building the intersection with the optimization domain  $\phi_0 \cap \phi_\Omega = \phi_1$  (Figure 3.6b). Then we add the unchanged, given geometry outside of the optimization domain  $\phi_1 \cup \phi_{\bar{\Omega}} = \phi_2$  (Figure 3.6c). The resulting shape is consistent with the optimization result and the geometry outside of the optimization domain remains unchanged (Figure 3.6d). Visualized using Paraview [40].



(a) Initial shape  $\phi_2$  with unchanged regions (red).



(b) At  $t = 0.0$ .

(c) At  $t = 2.0$ .

Figure 3.7.: **Level set smoothing** – The shape is smoothed using the mean curvature flow, some regions are excluded from smoothening (Figure 3.7a) . Smoothing under mean curvature removes noise, but also reduces the volume of the shape (Figure 3.7b and Figure 3.7c). Visualized using Paraview [40].



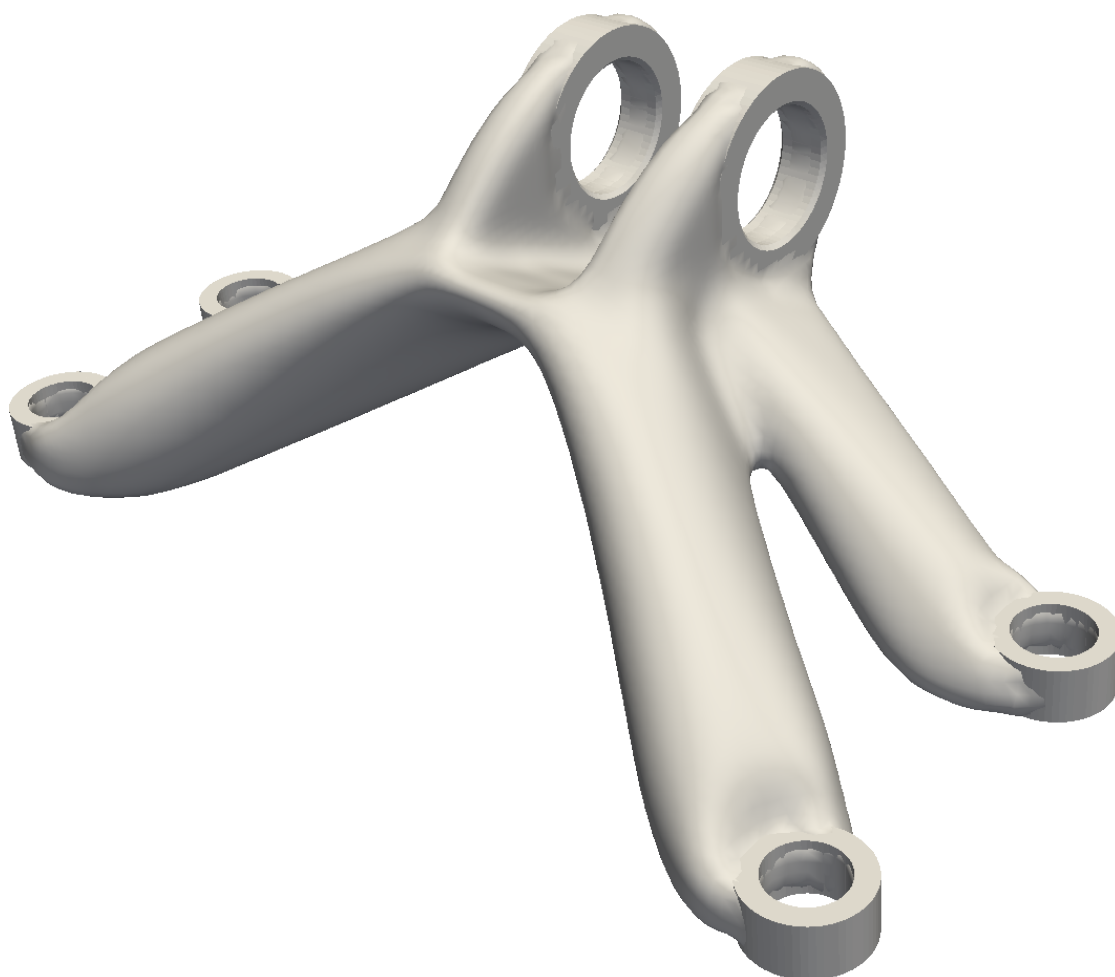
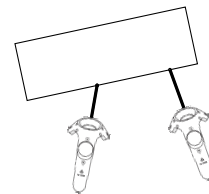


Figure 3.8.: **Consistent level set** – Isocontour at  $\phi = 0$  of the smoothed level set with smoothing time  $t = 2.0$ . The level set geometry is consistent with the boundary condition shapes that are given in `.step` format. The boundary condition shapes are also visualized. Created using the algorithms proposed in this work. Visualized using Paraview [40].

### 3.3. Parametric geometry reconstruction

In the previous sections we explained how to obtain a topology optimized geometry from boundary conditions (section 3.1) and how we postprocess it in order to comply with boundary condition shapes (section 3.2). Here we finally want to reconstruct the topology optimized structure as a CAD geometry in *boundary representation* (BREP) format (subsection 2.1.1).

We first visualize the topology optimized structure with a triangular mesh (see subsection 3.3.1). Now we want to reconstruct this mesh geometry using B-spline curves and NURBS surfaces.

Topology optimized structures created on the basis of the *Solid Isotropic Microstructure with Penalization* (SIMP) approach are usually truss structures. Therefore, for the reconstruction of the mesh geometry we propose the following specialized toolset (see also Figure 3.9):

In order to be able to create struts of variable diameter we provide a tool for the extraction of cross section contours from the mesh (subsection 3.3.2). These contours can then be joined into loft surfaces that approximate a series of contours in order to model struts (subsection 3.3.3). For more complicated parts like junctions of several struts or regions where a smooth connection to boundary condition shapes has to be created, we provide a sketching tool for the creation of guiding lines that lie on the mesh (subsection 3.3.4).

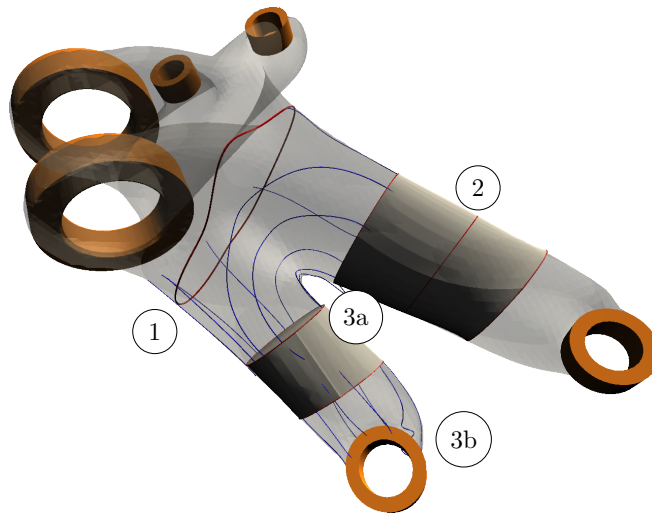
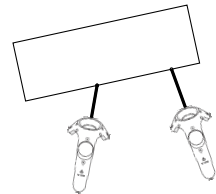


Figure 3.9.: **Toolset for geometry reconstruction implemented in SurfFitX** – The contour tool allows the extraction of contour shapes (orange) from the mesh (1). Several contours can be joined to form a loft surface that approximates a strut structure (2). For complicated regions like junctions of several struts (3a) or connections to boundary condition shapes (3b) we provide a sketching tool for the creation of guiding lines (blue).



For the creation of a smooth, watertight and parametrized surface we finally rely on a full fledged CAD tool like NX [1], which provides the wide range of tools that are necessary to impose continuity conditions and create complicated surfaces. We can import the contours, guiding lines and loft surfaces into NX and use them for the construction process.

### 3.3.1. Mesh visualization

We visualize the topology optimized structure using a mesh that can be created using the marching cubes implementation from VTK and a given threshold  $\theta$ . In the previous section we defined the topology optimized structure over its SDF  $\phi$ . Therefore, we pick the threshold  $\theta = 0$  in order to obtain the zero level set of the SDF  $\phi = 0$  as a triangular mesh. The visualization of the mesh is handled by Unity.

### 3.3.2. Contour extraction from meshes

We approximate the mesh geometry by creating closed B-spline curves lying in cross sections cutting the mesh. We call these B-spline curves contours. We implemented the following algorithm for the extraction of contours from the SDF  $\phi$ :

1. The user provides origin  $\vec{o}$ , normal vector  $\vec{n}$ , start direction  $\vec{d}_0$ <sup>5</sup> and radius  $r$  of a disk  $D$ . The origin of the disk lies inside the mesh geometry and the disk defines the cross section where the contour is going to be constructed.
2. We probe the SDF  $\phi(\vec{x})$  along  $n + 1$  straight lines  $\gamma(t)_{0\dots n}$  pointing from the origin  $\vec{o}$  into the search direction  $\vec{d}_i$ :

$$\vec{\gamma}_i(t) = \vec{o} + \frac{\vec{d}_i}{\|\vec{d}_i\|}t, \quad t \in [0, r],$$

with

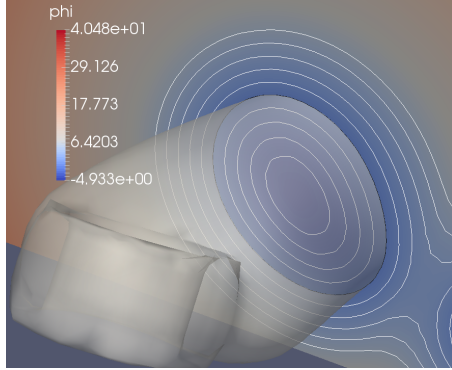
$$\vec{d}_i = R(\alpha_i, \vec{n})\vec{d}_0,$$

where  $R(\alpha_i, \vec{n})\vec{x}$  describes the rotation matrix rotating the vector  $\vec{x}$  around the axis  $\vec{n}$  by  $\alpha_i$  degrees. We create sample lines for  $\theta_i = 2\pi i/n, i = 0 \dots n$ .

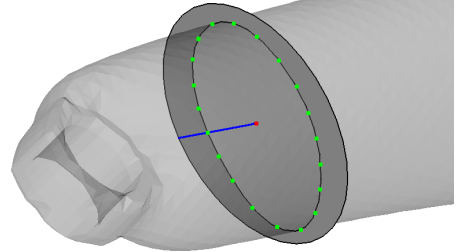
3. Along each line  $\gamma_i$  we determine the position where the line crosses the mesh by calculating the root of  $\phi(\vec{x}_i) = 0$  using bisection. We define a tolerance  $\epsilon$  and iterate until  $|\phi(\vec{x}_i)| < \epsilon$ .
4. We create a B-Spline curve that approximates the ordered array of  $n + 1$  sample points  $[x_0, x_1, \dots, x_{n-1}, x_n]$ . We set the resulting B-Spline to be closed ( $C_0$  continuity) and obtain the contour

---

<sup>5</sup> $\vec{d}_0 \cdot \vec{n} = 0$ , therefore  $\vec{d}_0$  lies inside of the disk  $D$ .



(a) Cutting plane with interpolated data and isocontours. Visualized using Paraview [40].



(b) Extracted contour in the disk. Origin  $\vec{o}$  (red), sample points  $\vec{x}_i$  (green) and root search line  $\vec{\gamma}_0(t)$  (blue) are highlighted. Visualized using FreeCAD [4]. [4].

Figure 3.10.: **Contour extraction** – The contour extraction algorithm samples the `.vtk` dataset. Interpolated values are obtained using VTK (Figure 3.10a). On a disk regularly spaced sample points are created by performing a root search on the `.vtk` dataset along straight search lines. Finally an approximating B-Spline curve is computed (Figure 3.10b).

To obtain a contour that approximates the mesh well and does not ignore important features, the origin  $\vec{o}$  of the disk should approximately lie in the center of the mesh region that is contoured. Additionally the number of  $n + 1$  sampling points should be neither too low, since features might be lost, nor too high, since effects of overfitting might occur.

For a visualization of the algorithm please refer to Figure 3.10. Dataset probing and necessary interpolations<sup>6</sup> are realized using VTK [25], the geometry operations as well as B-Spline approximation and modelling are realized using OCCT [21].

### 3.3.3. Loft surface creation

For the creation of surfaces from a set of closed or open wires, OCCT provides a mechanism for lofting. We use this mechanism for either creating a loft from contours defined using the algorithm mentioned above or creating a loft contours and already existing wires.

Creating a loft surface from contours is trivial: The contours have to be selected in the demanded order and a loft surface is created that tries to approximate the contours as good as it is required by the tolerance that has been defined by the user (Figure 3.11).

If we want to create a loft using already existing contours (e.g. contours from existing faces) we first have to extract the edges belonging to the faces, remove seam edges and create a closed wire with consistent orientation (Figure 3.12).

The necessary algorithms for loft creation, shape analysis and shape healing are provided by OCCT [21].

<sup>6</sup>The SDF  $\phi$  is provided in the discretized form of values lying on a regular grid stored as a `.vtk` dataset.

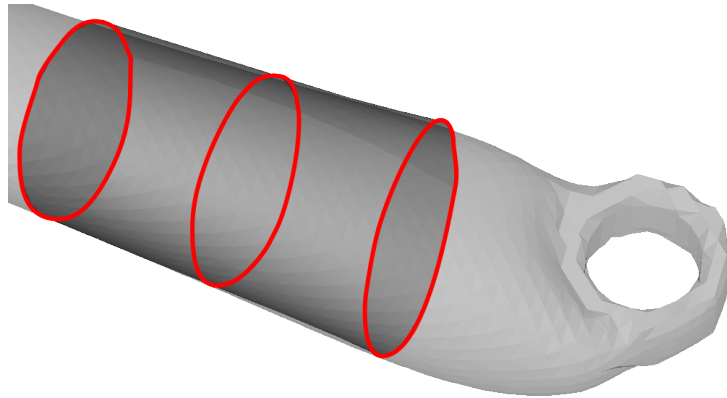
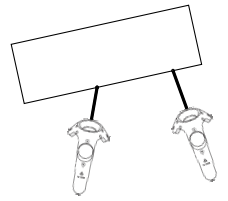


Figure 3.11.: **Loft from three contours** – Three extracted contours (orange) define a loft surface. Visualized using FreeCAD [4].

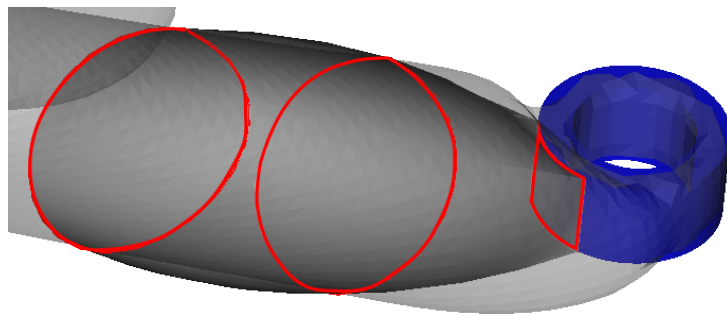


Figure 3.12.: **Loft connecting extracted contours and existing geometry** – Three extracted contours (orange) define a loft surface. One contour has been created from an existing boundary condition shape (blue). Visualized using FreeCAD[4].

### 3.3.4. B-Spline sketching

For shapes that cannot be described using contour curves, like junctions or connections with boundary condition shapes, we provide a tool for sketching of guiding lines that are aligned with the mesh. The lines are open B-spline curves.

An array of  $M$  sample points  $y_i$  provided by the user describes a line. In order to remove noise, originating from hand tremor and inaccuracy of measurement, we fit a B-spline curve to the sample points.

The resulting B-spline is smooth, but does not lie on the mesh. Therefore we extract  $N$  uniformly spaced sample points  $\vec{x}_i^0$  and shift them onto the mesh. This is equivalent to determining the root of the SDF associated with the mesh by using the following iterative gradient method:

$$\vec{x}_i^{n+1} = \vec{x}_i^n - \frac{1}{2} \phi(\vec{x}_i^n) \nabla \phi(\vec{x}_i^n),$$

where  $\phi$  denotes the SDF of the mesh and  $\nabla \phi$  the gradient of the SDF, which always points from the inside direction of the mesh to the outside. With  $\phi$  being the SDF it already provides an easily accessible estimation for both the error and the stepwidth of the gradient method. We iterate until  $|\phi(\vec{x}_i^n)| < \epsilon$ , where  $\epsilon$  denotes a tolerance.

After performing the gradient method, we obtain  $N$  shifted sample points that lie on the mesh. We again fit a B-spline to the points and obtain an open B-spline lying on the mesh.

For a visualization of the algorithm please refer to Figure 3.13. For B-spline fitting and evaluation we used OCCT [21] functionality. The computation of gradients and probing of the SDF can be easily done using VTK [25].

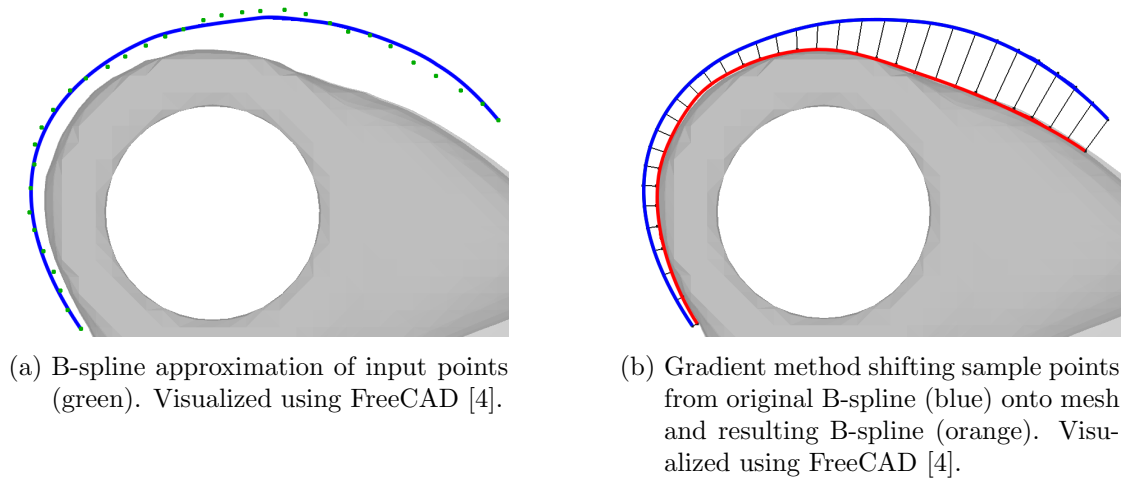
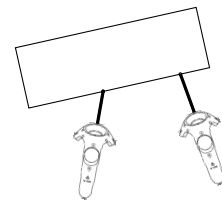


Figure 3.13.: **B-spline sketching on mesh** – We first remove noise from the sample points by creating an approximative B-spline (Figure 3.13a). Then we apply a gradient method for shifting sample points from the B-spline onto the mesh (Figure 3.13b).



## 4. User interaction

In this chapter we present the concepts of user interaction that allow the user of *Virtual Reality Surface Fitting Extension* (SurfFitX) to access the core functionality of CADinVR as well as the additional functionality that has been introduced in section 3.3.

For user interaction we use the HTC Vive [8], the Unity3D [22] games engine, and the SteamVR [59] interface (see also section 2.3): The user interacts with the *virtual reality* (VR) environment using the two controllers of the Vive (see Figure 2.6). The pose of the controllers is tracked and visualized in the virtual environment. The visualization of the scene happens in the stereoscopic display of the Vive *head-mounted display* (HMD).

In section 4.1 we explain concepts for the manipulation and selection of shapes in the virtual environment, this involves primitive shapes as well as complex geometry such as loft surfaces. In section 4.2 we explain the mechanism for sketching of lines in 3D space. Finally in section 4.3 we explain how the user controls the camera view in the virtual environment and how the user is able to inspect complex scenes.

### 4.1. Shape interaction

For the manipulation of 3D shapes in a VR environment we need to be able to interact with geometry that is represented through a triangular mesh. This first involves selection of shapes and secondly being able to transform (translate, rotate and scale) shapes. For more complex interaction we also provide a basic menu.

A majority of the user interaction metaphors that are presented in the following did already exist in CADinVR. These metaphors have been implemented from scratch and we decided to continue using them. However, there exist Unity Assets of high quality that provide VR interaction functionality [63, 64], which we decided not to use due to the already existing implementation in CADinVR.

We use the interaction methods described below for manipulation of the position, rotation and scale of the mesh representing the topology optimized geometry (subsection 3.3.1), the positioning of cross section planes that define the input variables for contour creation (see subsection 3.3.2), and the selection of contour shapes for the creation of loft surfaces (see subsection 3.3.3).

**Hover and selection:** We want to be able to detect if the user is pointing on an object with the controller, we call this mechanism hovering. Active hovering is visualized by a laser beam pointing from the controller to the object. Additionally we want to allow the user to persistently select and deselect objects. When an object is hovered, selection can be activated and deactivated by pressing the trackpad on the Vive controller. We

support selection and hovering for the whole object as well as for single faces of the object (see Figure 4.1). This mechanism has been developed and implemented in the legacy system.

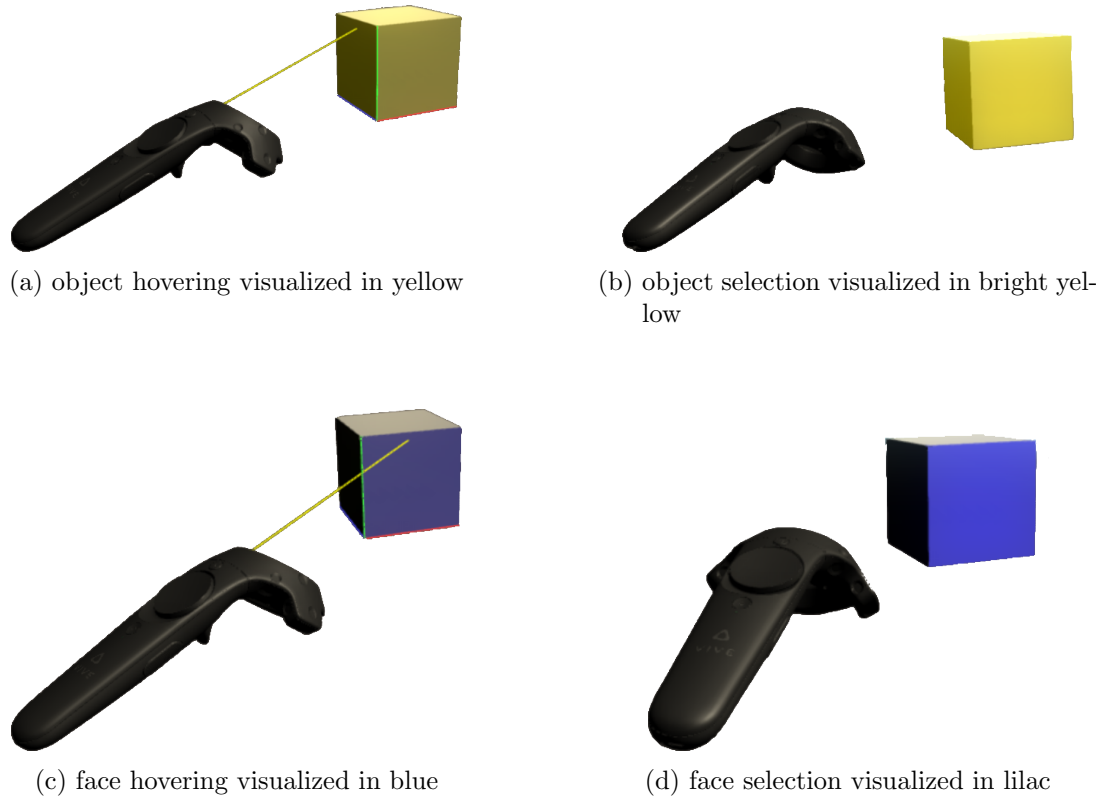
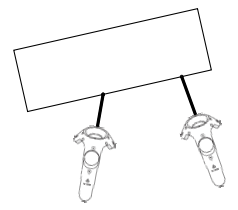


Figure 4.1.: **Hovering and selection** – A ray pointing from the controller to the object indicates, that the controller is hovering the object.

**Transformation:** The user is able to transform objects with respect to position, rotation and scale by using the controllers. The user grabs an object that is currently hovered by pressing the trigger button. If the user grabs the object with only one controller, this action binds the objects position and rotation to the movement of the controller (see Figure 4.2) and the object exactly follows the movement of the controller. This basic mechanism has been developed and implemented in CADinVR.

For objects that are far away we introduced a modified mechanism as the result of an iterative development process: If the object is grabbed while being far away from the user (the distance from the user to the object is larger than one armlength), translation into the direction normal to the camera plane is performed relatively to the distance of the user (pulling the object into the user’s direction by one armlength reduces the distance from the user to the object by 100%, regardless of the actual distance.). For a visualization see Figure 4.3.





Additionally grabbing the object with a second controller allows the user to scale the object (see Figure 4.4). This mechanism has been developed and implemented in CADinVR.

## 4.2. Sketching

Using the Vive Controller the user is able to draw lines in 3D space by pressing the grip button. For this we attached a virtual drawing pen to the controller and sample the position of the pen tip on each frame. While drawing we create a polygon passing through the sample points. The resulting polygon provides the input for the creation of guiding lines described in subsection 3.3.4.

## 4.3. Camera adjustment

An important and often tedious part of user interaction in 3D applications is the control of the camera. This is needed for being able to look at the scene from different perspectives or at different scale. In the following we explain the camera controls that are realized in SurfFitX.

**Head-mounted display adjustment:** Basic movement of the camera is intuitively realized by moving the head. The position of the HMD is tracked and the position of the camera in the virtual environment is adjusted correspondingly. This firstly forces the user to move the head in order to change the perspective, and secondly restricts the available perspectives to the physical space that is occupied by the user. A change in scale is not possible. This basic mechanism is implemented in SteamVR [59].

**Global coordinate system:** In order to allow the user to scale the scene or look at the scene from positions that exceed the physical abilities of the user we give the user the possibility to modify the position and scale of the whole scene while the camera position remains untouched. This concept is similar to the concept of scaling and positioning the world that is presented in [13]. The user can modify the global coordinate system by grabbing and transforming a reference object. The whole scene inherits the position, rotation and scale of the reference object and is transformed and scaled correspondingly. With this functionality we allow the user to inspect the scene at different scales and look at objects from different perspectives. We implemented this mechanism in SurfFitX.

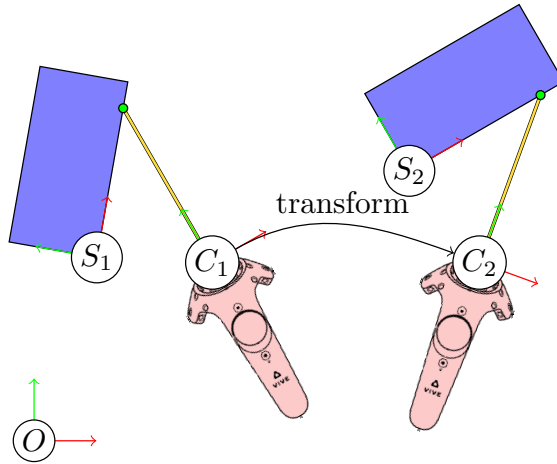


Figure 4.2.: **Schematic drawing of translation and rotation** – Translation and rotation of a shape  $S$  in the global coordinate system  $O$  happens by hovering and grabbing  $S$  at position  $S_1$ . This attaches  $S$  to the controller's coordinate system  $C_1$  until the controller releases  $S$  at the new position  $C_2$ . The new position of  $S$  is  $S_2$ .

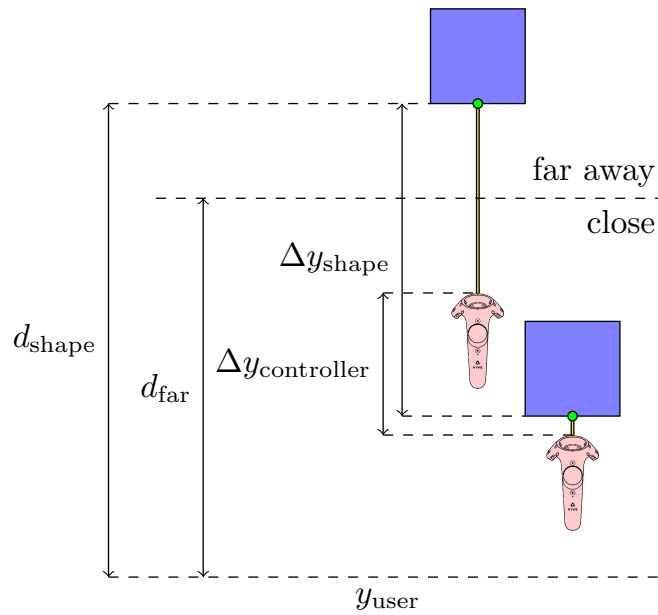


Figure 4.3.: **Transformation of far away objects** – If the object is far away ( $d_{\text{far}} < d_{\text{shape}}$ ) from the user, the distance  $\Delta y_{\text{shape}}$  that the object is translated towards the user is a multiple of the translation  $\Delta y_{\text{controller}}$  of the controller. The movement of the shape  $\Delta y_{\text{shape}}$  is calculated from the movement of the controller  $\Delta y_{\text{controller}}$  relative to the distance  $d_{\text{far}}$ .

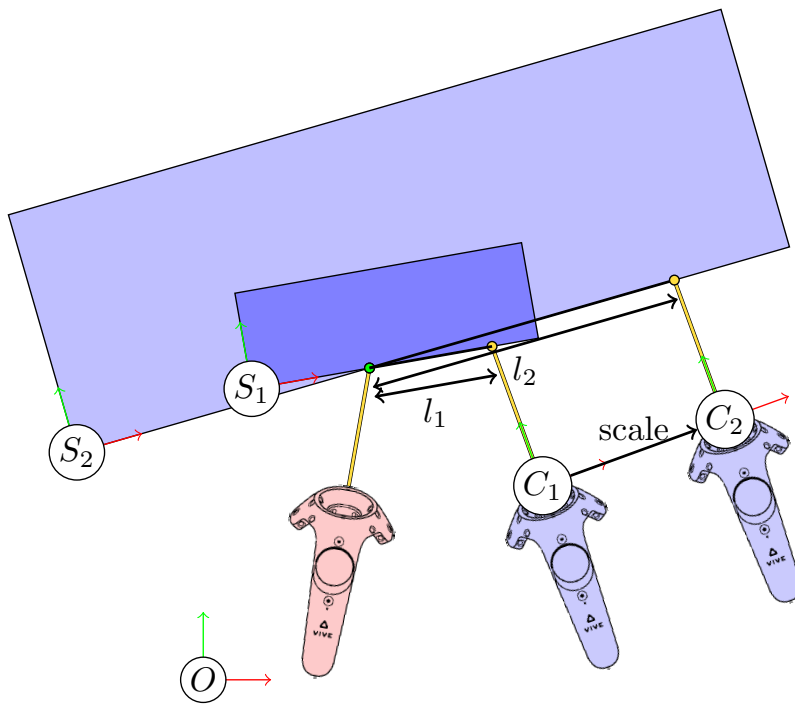
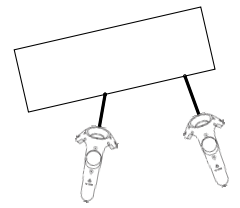
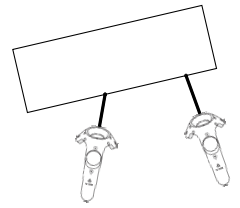


Figure 4.4.: **Schematic drawing of scaling** – Scaling of  $S$  is achieved by grabbing the object with a second controller and moving the controller from  $C_1$  to  $C_2$ .  $S$  is scaled by the fraction of the distances  $l_1$  and  $l_2$  between the collision points of the laser beams originating from the two controllers.





## 5. Implementation

In this chapter we describe the architecture of *Virtual Reality Surface Fitting Extension* (SurfFitX) in depth:

We rely on the core functionality of CADinVR described in section 1.3. This software system has been extended with respect to the visualization and reconstruction (see section 3.3) of as well as the interaction (see chapter 4) with topology optimized structures.

In section 5.1 and section 5.2 we describe the two main components – frontend and backend – of SurfFitX and their specific tasks. In general the frontend, that heavily relies on the game engine Unity, deals with user interaction and visualization, while the backend, wrapping the open source libraries *Open CASCADE Technology* (OCCT) and *Visualization Toolkit* (VTK), provides *computer-aided design* (CAD) core functionality and customized algorithms for the reconstruction of topology optimized geometry.

### 5.1. Frontend

For user interaction and visualization we use a frontend application that uses the game engine Unity. We first give a general overview over the tasks of the frontend application (subsection 5.1.1). In the remaining subsections we provide a detailed explanation of crucial implementational aspects.

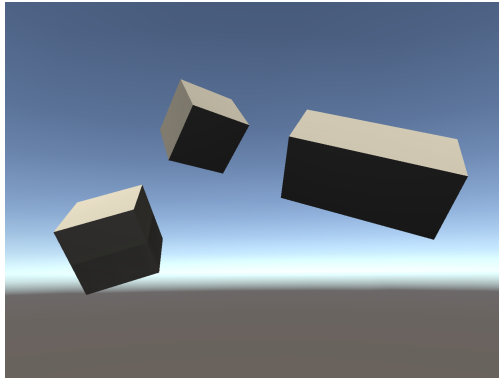
#### 5.1.1. General overview

For the visualization and manipulation of a CAD scene we rely on CADinVR functionality. The user has the possibility to create geometry and interact with it. In SurfFitX we implemented visualization methods for topology optimized geometry and we use the resulting optimized mesh geometry as the reference point of construction.

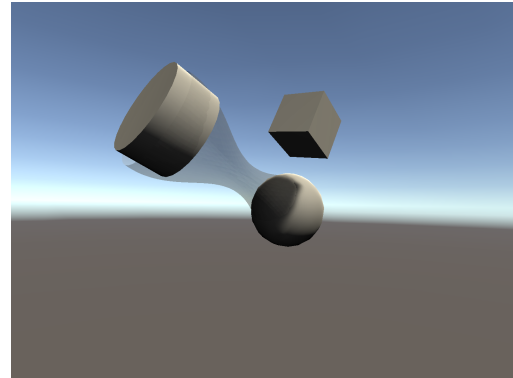
Unity already provides a large amount of functionality that is useful for the development of interactive applications (see subsection 2.3.3) and provides means of communication with the *virtual reality* (VR) hardware using the Unity asset SteamVR [59] (see subsection 2.3.4).

For user interaction we use the metaphors described in chapter 4. As a reference object for camera adjustment (see section 4.3) we defined the mesh that visualizes the topology optimized geometry (Figure 5.1).

We decided to prevent modification of geometry that has been created depending on the topology optimized geometry, like loft surfaces or extracted contours. However, this geometry can still be selected and therefore referenced.



(a) Transformation of a primitive.



(b) Scene with mesh (transparent).

Figure 5.1.: **Screenshots from the frontend application** – A primitive object is first transformed, then scaled (Figure 5.1a). The mesh defining the global coordinate system is part of a scene, all objects can be moved freely Figure 5.1b.

### 5.1.2. Large meshes and compound objects

Unity does not support more than 65535 vertices per mesh. The mesh that visualizes the topology optimized part is usually a large triangular mesh where this limit is exceeded; the mesh from the low resolution geometry introduced in chapter 3 has for example already about 15000 vertices and 20000 triangles. Thus, we have to pay special attention to large meshes:

1. The mesh has to be split up into multiple submeshes that do not exceed the vertex limit.
2. For each submesh an independent *game object* (GO) has to be created. This game object can be hovered, selected and transformed, if this behavior is desired.
3. Hovering, selection and transformations have to be applied to the whole mesh correspondingly, even though the user only interacts with a submesh.

We introduced the necessary logic for this scenario in order to be able to deal with large meshes that exceed Unity's vertex limit. With this logic we can also handle compound objects that are built from multiple primitive objects.

### 5.1.3. The menu

The legacy system already provided a menu that can be used for accessing additional functionality. The menu is directly attached to the controller. The user can select entries from the menu by hovering an entry with the other controller and pressing the selection button. By using the touchpad the user can scroll horizontally through several columns in the menu. Vertical scrolling can be used to scroll through a column, that has many entries (e.g. a list of input files).

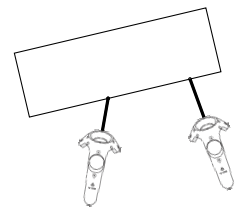


Figure 5.2.: **The menu** – The user can scroll sideways through different columns like *Import*, *Create* or *Boolean* using the trackpad. By selecting a menu entry with the other controller the corresponding command is executed.

Using the menu the user can access the following functionality (functionality of the legacy system is written in italics):

- *undo/redo*
- *.stp import/export*
- *.vtk import*
- *creation of primitives*
- *boolean operations on shapes*
- *selection and deletion of shapes*
- creation of planes for contour extraction
- creation of loft surfaces from selected contours

#### 5.1.4. Coordinate mapping

The frontend application maps the transform<sup>1</sup> of each shape that is transferred to the backend to the coordinate system of the backend. It converts the global transform that is used in Unity to the local transform in the coordinate system of the topology optimized geometry, since the topology optimized geometry is used as the reference object defining the global coordinate system on the backend(see section 4.3). This coordinate mapping can be easily done using Unity functionality. The coordinates that finally are transferred to the backend are always in the fixed coordinate system of the topology optimized geometry.

---

<sup>1</sup>position, rotation and scale

## 5.2. Backend

The backend is written in C++. CADinVR provides access to functionality of Open CASCADE Technology (described in subsection 2.2.2). In SurfFitX we added VTK functionality (subsection 2.5.5) and custom algorithms in order to be able to handle topology optimized geometry.

We first give a general overview over the backend functionality in subsection 5.2.1 and then concentrate on crucial parts of the implementation. In the following subsection we explain the exchange datatypes that are transferred to the frontend application for the visualization of the geometry.

### 5.2.1. General overview

CADinVR already provided bookkeeping for the CAD model on the backend. The model is stored, modified and consistently maintained on the backend. All shapes are modelled consequently in OCCT and are identified by a unique id. New shapes can be created and existing shapes can be modified. CADinVR provided functionality for the creation of primitive shapes, boolean operations on shapes, the modification of existing shapes, `.step` input and output and meshing of shapes.

In SurfFitX we added functionality for level set preprocessing (section 3.2), the generation of isosurfaces from voxel data using VTK, the extraction of contours from the topology optimized geometry, the creation of loft surfaces from contours and functionality for B-spline sketching (section 3.3). All functionality is requested by the frontend application, if the user provides the necessary input.

The topology optimized geometry cannot be modified on the backend and always maintains its initial position, rotation and scale in space. Reconstructed geometry is always created at a fixed position relatively to the topology optimized geometry.

### 5.2.2. Exchange data types

In CADinVR the shapes on the backend are uniquely identified by their respective id. If they are requested by the frontend in order to be visualized, the shapes that are modelled as OCCT objects have to be meshed using OCCT functionality. Each face of the shape is meshed and the triangulated face represented through an array of vertices and another array that contains the ids of vertices that form a triangle. In SurfFitX we extended this mechanism to support edge or wire in order to be able to handle contours (subsection 3.3.2) and guiding lines (subsection 3.3.4). Here the corresponding parametrized curve is sampled and the resulting array of sampling points is saved. After this the backend has converted complex OCCT objects to simple mesh or polygon datatypes that can be marshalled and transferred to the frontend for further processing (see Figure 5.3).



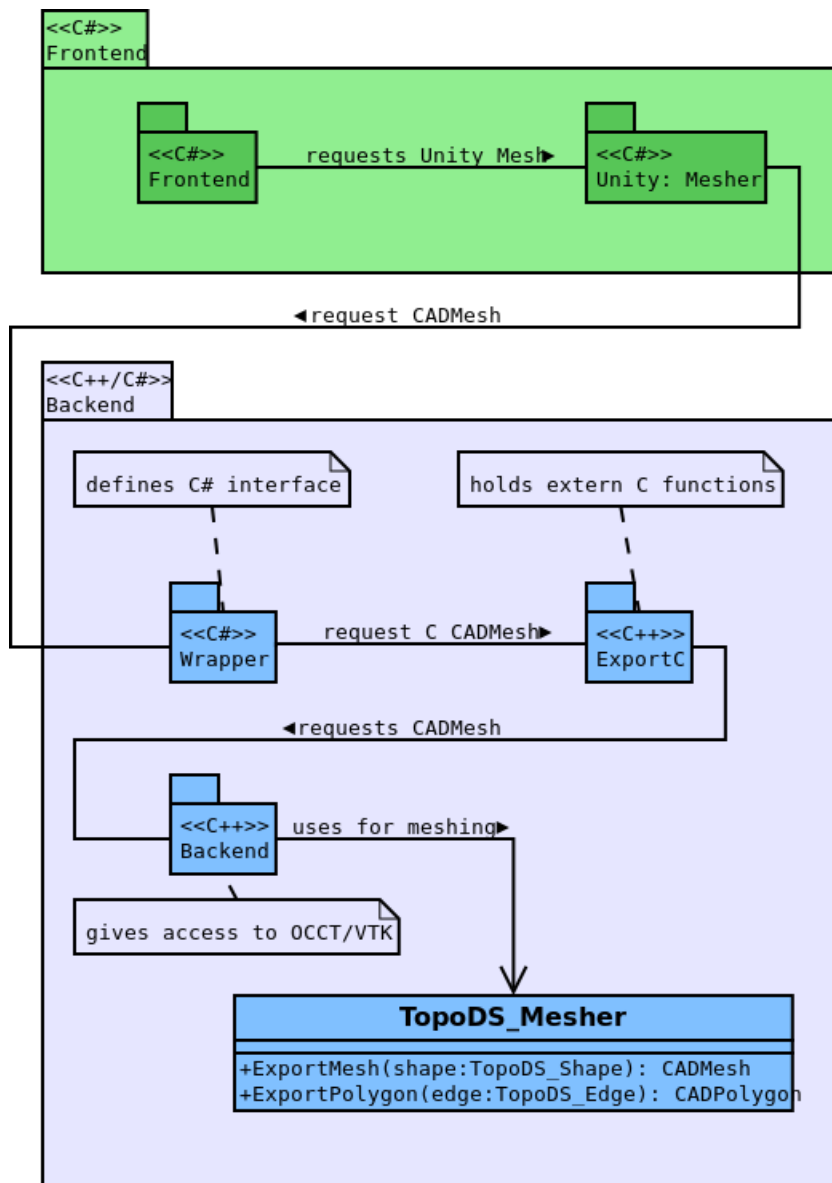
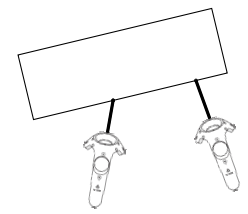
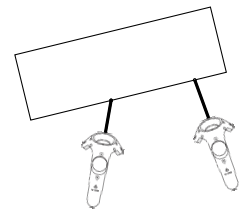


Figure 5.3.: **Frontend requesting a mesh from backend** – If the frontend application requests the mesh of a shape, the backend application first creates a mesh of the shape and then creates a simplified object that contains the necessary information. This object is then sent to the frontend and the mesh can be visualized.





## 6. User study

In chapter 1 we formulated the hypothesis that *computer-aided design* (CAD) is a considerably easier when performed in a *virtual reality* (VR) environment compared to established CAD systems. We want to evaluate this hypothesis by performing a user study that tries to answer the following two questions:

1. Is it easier and more efficient to do CAD in VR?
2. How does the participant experience working with CAD in a VR environment in general?

In the user study we evaluate of the performance of participants solving a task that mimics a part of the workflow described in chapter 3. We use *Virtual Reality Surface Fitting Extension* (SurfFitX) (described in chapter 5) as an example CAD system in VR.

In the following section we will give a detailed explanation of the task that the user has to solve, the experimental setup of the study and the procedure of evaluating the performance of the user. The whole user study was – depending on the preference of the participant – conducted in English or German language.

### 6.1. The task

The task is subdivided in two parts:

1. **Part one:** The participant has to position three planes in 3D space relatively to a given mesh. The Planes are used to extract three contours of a mesh. For solving this part of the task, the participant has a time constraint of 5 minutes.
2. **Part two:** After the creation of the contours the participant has to perform additional operations in order to produce a loft surface, that approximates the mesh geometry, from the cross sections.

The user guides corresponding to the tasks can be found in the appendix (see subsection A.1.1 and subsection A.1.2). The explanation of the task is done with the simple mesh shown in Figure 6.1, the participant has to solve the task with the bracket mesh shown in Figure 6.2.

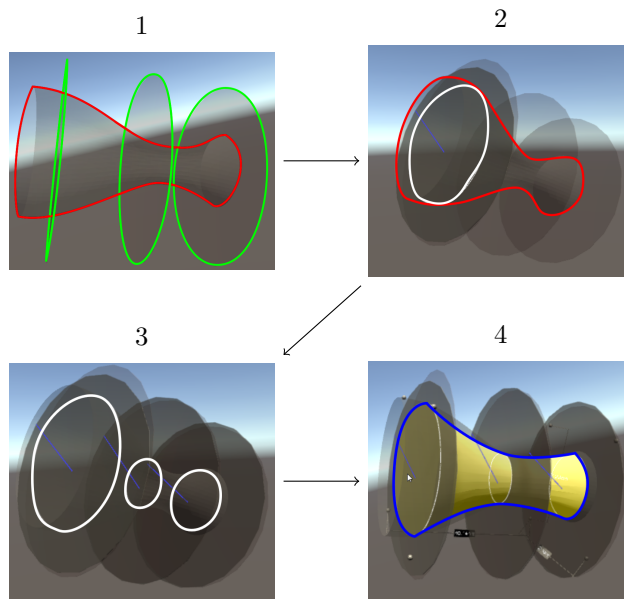


Figure 6.1.: **Schematic overview over task** – The participant has to reconstruct a mesh geometry (red) by positioning cross section planes at indicated positions (green). Having positioned the cross section planes a contour (white) is extracted. This procedure is performed three times in order to obtain three contours. On the basis of these contours a loft surface (blue) that passes through the contours is created.

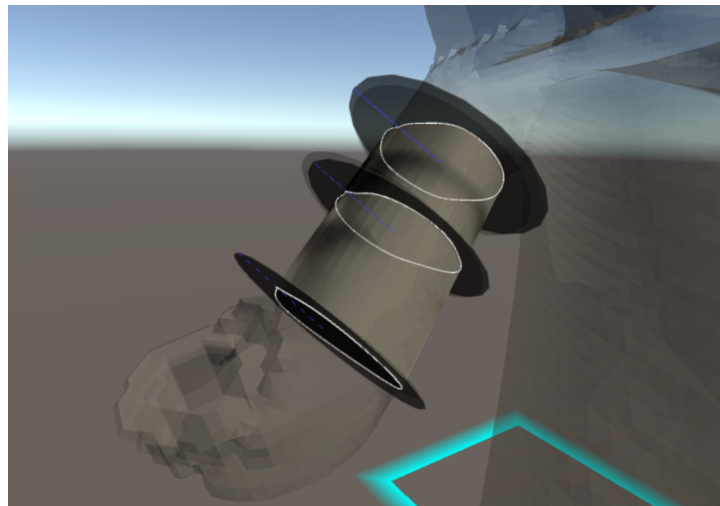
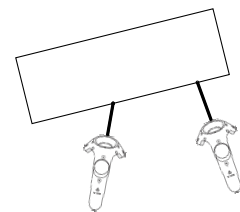


Figure 6.2.: **Reconstructed part of the GEBracket** – The participant has to reconstruct the topology optimized dataset of the GE Bracket [28].



## 6.2. Experimental setup

In an introductory part each participant completes a background questionnaire (see subsection A.2.1). Then the participant is allowed to play a short VR game (Postcards minigame in The Lab [76]). Most participants never used any VR hardware before and we decided to introduce the user to VR and the HTC Vive before performing the actual experiment.

In the main part of the experiment we compare the performance of the same participant solving the task described in section 6.1 in the two different environments SurfFitX and FreeCAD (within-subject design) in order to answer the aforementioned questions.

- Desktop environment **FreeCAD**: FreeCAD is a traditional CAD system with state of the art methods for modification of geometry. Keyboard and mouse are used as input devices, we decided not to use a 3D mouse, since many participants do not have experience with this input device. We customized FreeCAD by adding a new workbench with the functionality necessary for realizing the demanded workflow: On the one hand we had to add tools that allow the creation of cross sections, on the other hand we had to remove unnecessary features that might confuse the user (see Figure 6.3). We decided to use FreeCAD as a reference system, because it is open source, customizable and provides the same range of algorithms like the VR system<sup>1</sup>. Professional CAD systems usually provide a better usability than FreeCAD, but since they are not open source and use different geometry kernels, we decided to use FreeCAD as the reference point for our research.
- VR environment **SurfFitX**: SurfFitX is the CAD system in VR that has been described in chapter 5. Again, we removed existing functionality that is not necessary for solving the given task. This system is the subject of our study.

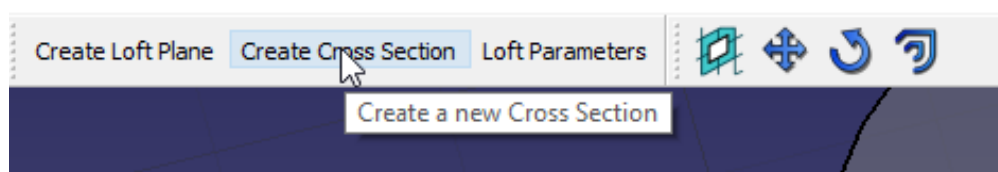


Figure 6.3.: ”Lofting” workbench in customized FreeCAD – We implemented a ”Lofting” workbench with a reduced toolset: ”Create Loft Plane” for the creation of a loft plane, ”Create Cross Section” for the extraction of a contour from the mesh at the position indicated by the loft plane, ”Loft Parameters” for printing position and rotation of the plane and shortcuts to standard FreeCAD functionality for transformation, rotation and scale and adjustment of the working plane.

<sup>1</sup>Both systems use the geometry kernel *Open CASCADE Technology* (OCCT).

The participant has to solve the task in both environments, we permute the order of solving the tasks in the two environments to account for effects of learning. We performed the study with 20 participants. Before solving the task, the participant first receives a demonstration of both tools: The basic workflow is shown and the necessary interactions are explained on the basis of the user guides (see subsection A.1.1 and subsection A.1.2). Then the user has a short period of time for experimenting with the software, revising the interactions and asking questions. While solving the task, the participant has access to the user guides (see above), and is allowed to ask for help.

### 6.3. Evaluation procedure

We quantify the performance of the participant in the different environments by measuring the time needed for positioning the three planes. In VR we additionally measure the amount of time needed for orientation. If the participant does not manage to position all the planes in 5 minutes time, we provide a substitute result.

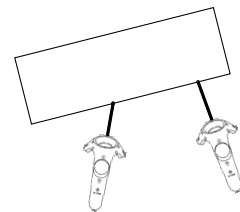
In order to evaluate the user experience, directly after the experiment the participant fills out the *System Usability Scale* (SUS) [27] questionnaire with 10 items. We decided to use the SUS questionnaire due to its brevity, ease of use and the acceptance in the community. The following formula is used for obtaining a single score from the 10 SUS items  $q_1, q_2, \dots, q_{10}$ :

$$(q_1 - 1) + (5 - q_2) + (q_3 - 1) + (5 - q_4) + \dots + (q_9 - 1) + (5 - q_{10}) = \text{SUS score},$$

where  $q_{1\dots 10}$  denotes the rating 1 . . . 5 of the participant for each item.

After the completion of both tasks, the user answers interview questions and gives feedback. Here we evaluate the experience of the participant with the particular workflow, the user interface and the general experience. We also ask for possible improvements regarding SurfFitX.

The SUS questionnaire and the questions asked in the interview can be found in the appendix (see subsection A.2.2 and subsection A.2.3).



## 7. Results

In this chapter we first present examples for topology optimized and reconstructed geometry created using the design workflow introduced in chapter 3. We used the prototype tool *Virtual Reality Surface Fitting Extension* (SurfFitX) in the reconstruction process.

We evaluated the usability of SurfFitX in a user study (chapter 6). The outcome of this study is summarized in section 7.2.

In the last section (section 7.3) we summarize an interview with a *computer-aided design* (CAD) expert. We discussed the potential of *virtual reality* (VR) technology in the context of professional CAD and demonstrated SurfFitX as an example application.

### 7.1. Topology optimization workflow using Virtual Reality Surface Fitting Extension

In the following chapter we present two examples that have been created using the workflow proposed in chapter 3. We used SurfFitX for the extraction of contour shapes, sketching of guiding curves and the creation of loft surfaces. Finally we used Siemens NX [1] for the creation of smooth CAD geometry. The resulting geometry is consistent with boundary condition shapes.

#### 7.1.1. GE Bracket

Based on the specification provided in [28] we create a topology optimized design for a jet engine bracket. The results of this process are presented throughout chapter 3, a summary is given in Figure 3.2.

First we model the boundary conditions, forces and simulation domain (see [28]) in NX[1]. Using *Interactive Design Assistant* (IDeAs) we perform the topology optimization simulation and obtain an optimized voxel geometry.

We use the algorithm described in section 3.2 to obtain a consistent mesh representation of the optimized part. We observe that the boundary reconstruction scheme greatly improves the mesh with respect to boundary condition shapes: The round parts of boundary condition shapes are conserved as they are intended. Additionally we observe that a few smoothing steps can remove noise from the geometry, but also significantly change the geometry in certain regions. Small errors on the scale of the mesh resolution are observed at the interface of boundary condition shapes and optimized domain.

Using SurfFitX we extract cross sections and guiding curves from the consistent mesh geometry (see section 3.3). They provide guidance for the final geometry reconstruction

step that is performed using NX. We experience that the creation of cross sections and guiding lines is simple and fast<sup>1</sup>.

In NX we create a smooth surface representation on the basis of the cross sections and guiding lines. The part of work that has to be done in NX in order to obtain a smooth geometry is high and requires experience with NX<sup>2</sup>. The resulting geometry is a smooth surface representation of the optimized part that is consistent with provided boundary condition shapes.

### 7.1.2. Generate Quadcopter

Based on the specifications provided at [29] we create a topology optimized quadcopter body. In the specifications, the quadcopter is only subject to forces. Since IDeAs requires at least one fixture, we have to slightly modify those specifications: We define the base plate of the Quadcopter, where the load is attached, as a fixture and we adjust the forces acting on the remaining boundary condition shapes correspondingly (see Figure 7.1).

The intermediate results after each step of the boundary condition reconstruction workflow are presented in Figure 7.2. We observe that the base plate is greatly damaged when the level set extraction from voxel data happens. We observe that the superposition of the boundary condition finally produces consistent geometry.

After a few iterations of smoothing we obtain a smooth geometry. We observe that the upper struts are damaged by the smoothing algorithm. The more smoothing steps we apply the larger amount of material is removed and finally the upper struts vanish completely (Figure 7.3).

<sup>1</sup>Only a few minutes are needed for this step

<sup>2</sup>This step requires about one week of training with NX and one day of work for the reconstruction of the Bracket geometry on the basis of the output of SurfFitX

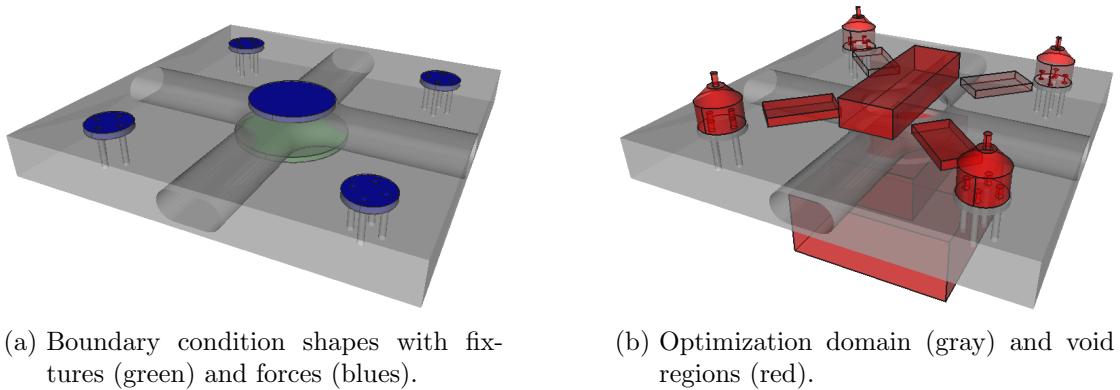
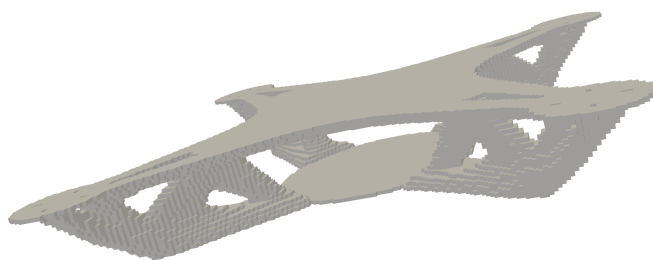
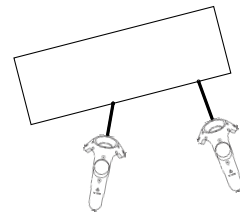
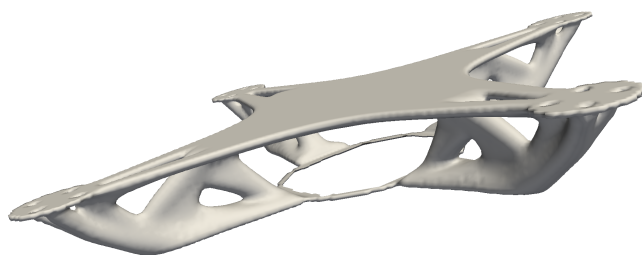


Figure 7.1.: **Model of the Quadcopter** – We had to modify the specification and set the base plate as a fixture (Figure 7.1a). The void regions and the simulation domain remain as specified (Figure 7.1b. Model specification from [29], visualized in FreeCAD [4].

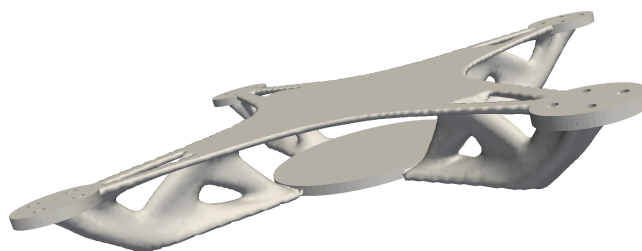




(a) Voxel image.

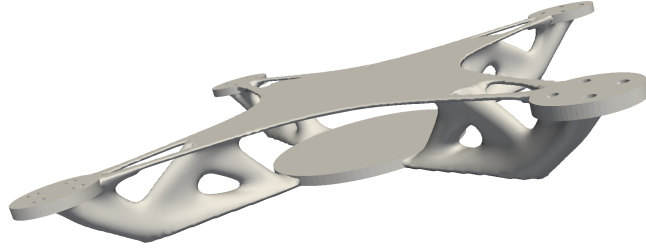


(b) Corresponding level set.

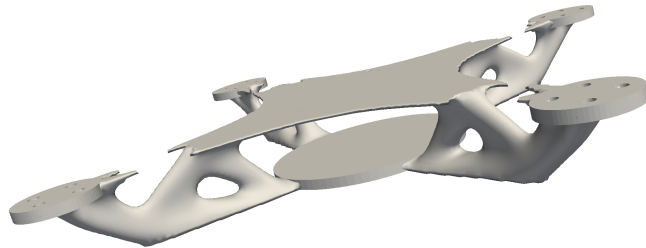


(c) Level set consistent with boundary condition shapes.

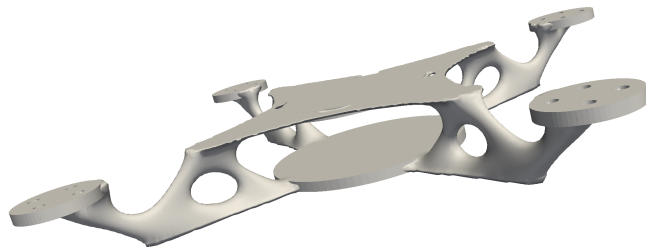
Figure 7.2.: **Topology optimized quadcopter** – From the voxel image of the quadcopter with a density threshold of  $\rho = 0.5$  (Figure 7.2a) we create the corresponding level set representation with an isosurface at  $\phi = 0$  (Figure 7.2b). We consistently superpose the boundary condition shapes (Figure 7.2c). Visualized using Paraview [40].



(a) Smooth level set at  $t = 0.3$ .

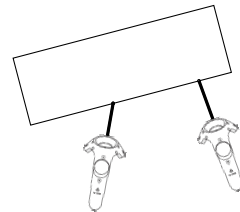


(b) Smooth level set at  $t = 0.5$ .



(c) Smooth level set at  $t = 2.0$ .

Figure 7.3.: **Level set smoothing** – The level set is shown at different smoothing times. Visualized using Paraview [40].



## 7.2. User study

In the following we present the results of the user study described in chapter 6. This involves the presentation of the quantitative results from the task being solved in the VR system SurfFitX and the desktop system FreeCAD (subsection 7.2.1), results from the *System Usability Scale* (SUS) questionnaire (subsection 7.2.2) and user feedback given in the interview session (subsection 7.2.3).

We decided to group the participants with respect to their success in the first part of the task. We distinguish the following groups (Figure 7.4)

- **Experts**, who successfully completed part one in both systems.
- **Successful beginners**, who were only able to complete the first part of the task in SurfFitX.
- **Failed beginners** who did neither succeed in SurfFitX nor in FreeCAD.
- One participant was not able to complete the experiment in SurfFitX environment due to software issues and also failed in FreeCAD. We do not regard his performance on the tasks and his SUS questionnaire. We are regarding his feedback in the interview session.
- There are no participants who failed to solve the task in SurfFitX and successfully solved the task using FreeCAD.

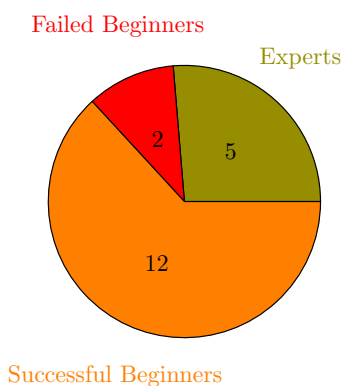


Figure 7.4.: **Classification of participants with respect to their performance in the both systems** – We distinguish the participants with respect to their performance in the two environments: Experts were successful both in SurfFitX as well as in FreeCAD. Successful beginners successfully finished the task in SurfFitX and failed in FreeCAD. Failed beginners failed in both environments. No participant failed in SurfFitX, while being able to solve the task in the Desktop environment.

### 7.2.1. The Task

In the process of solving the task we measured the time needed for teaching and learning the task, the time needed for solving part one and part two and finally whether the single parts have been solved successfully.

We observed that in SurfFitX the average time needed for teaching, as well as the average time the user requested for experimentation, was lower than in FreeCAD (see Figure 7.5). Comparing the time needed by the beginner’s group and by all participants we observe a statistic significance (p-value below 0.05). If we only compare the time needed by the experts we do not observe a statistic significance (p-values around 0.2).

A comparison between the groups in the same environment does not show clear statistic significance for explanation and experimentation in FreeCAD (p-values of 0.58 and 0.63) and significance for experimentation in SurfFitX (p-value of 0.01). No significance is observed for the explanation time in SurfFitX (p-value of 0.83).

For part one we observed that a higher portion of the participants successfully solved part one in SurfFitX than in FreeCAD. We observed that only a minority (5 of 19) was able to complete part one in FreeCAD in the given time of 5 minutes<sup>3</sup>. A majority (17 of 19) of the participants was able to complete part one in SurfFitX (Figure 7.6). Additionally we observed, that none of the participants was able to solve the task in the desktop environment while failing in VR.

In part two we observed that all participants performed well in both environments. On average they were slightly faster in SurfFitX (see Figure 7.7).

For successful completions of part one in FreeCAD and SurfFitX we measured the time needed for the positioning of the 3 planes and the construction of the cross section. In SurfFitX we additionally measured the time needed for orientation in the scene and positioning the dataset.

Using the grouping explained above we observe that the expert group was in general faster in SurfFitX than in FreeCAD. We detect a statistically significant difference (p-value of  $5.95^{-9}$ ) for the total time if we compare the performance of the expert’s group in both systems (Figure 7.8a and Figure 7.8b). Comparing the performance of the expert’s and the beginner’s group in SurfFitX we only observe a statistical significance for the time needed to for the positioning of the first plane. For the other times taken no statistically significant difference is detected (Figure 7.8b and Figure 7.8c). Among the failed beginners, we observe that they were not able to complete the last plane in time, but successfully positioned the first and second plane in SurfFitX while in FreeCAD they did not even finish the first plane. However we observe that the first plane was usually the most difficult of the three planes in FreeCAD, while all three planes were equally difficult in SurfFitX.

---

<sup>3</sup>Few participants received additional time to finish the step they were currently performing. One expert received additional time for positioning the last plane in FreeCAD.

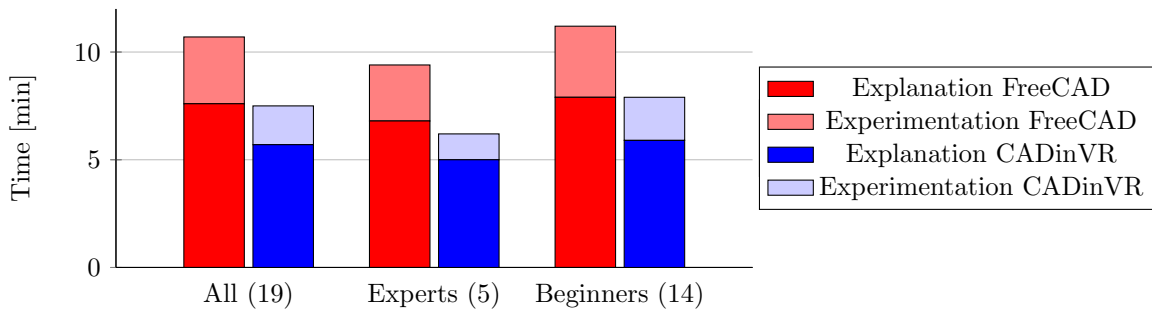
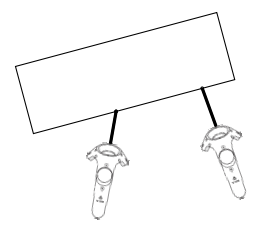


Figure 7.5.: **Comparison of time needed for teaching in SurfFitX and in FreeCAD** – Uncertainties are with  $\pm 1$  minute very high, but one can observe the following trend: both explanation of the task and the necessary steps to solve the task and experimentation time are smaller in SurfFitX.

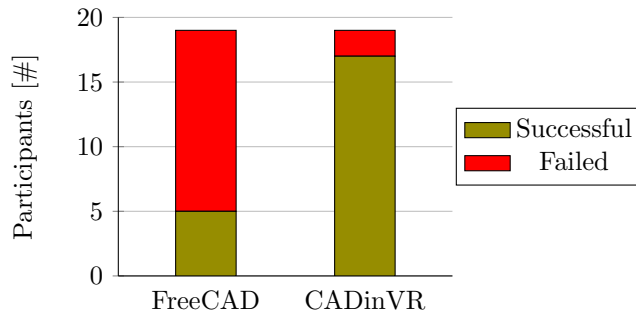


Figure 7.6.: **Success in part one** – Successful and unsuccessful participants when using FreeCAD and SurfFitX, respectively.

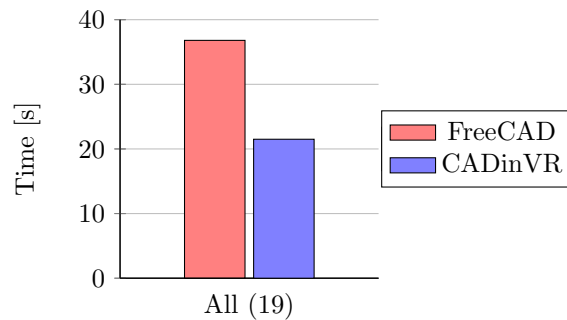


Figure 7.7.: **Comparison of the time needed for the second part of the task** – The time difference between the both systems is small and all participants performed well.

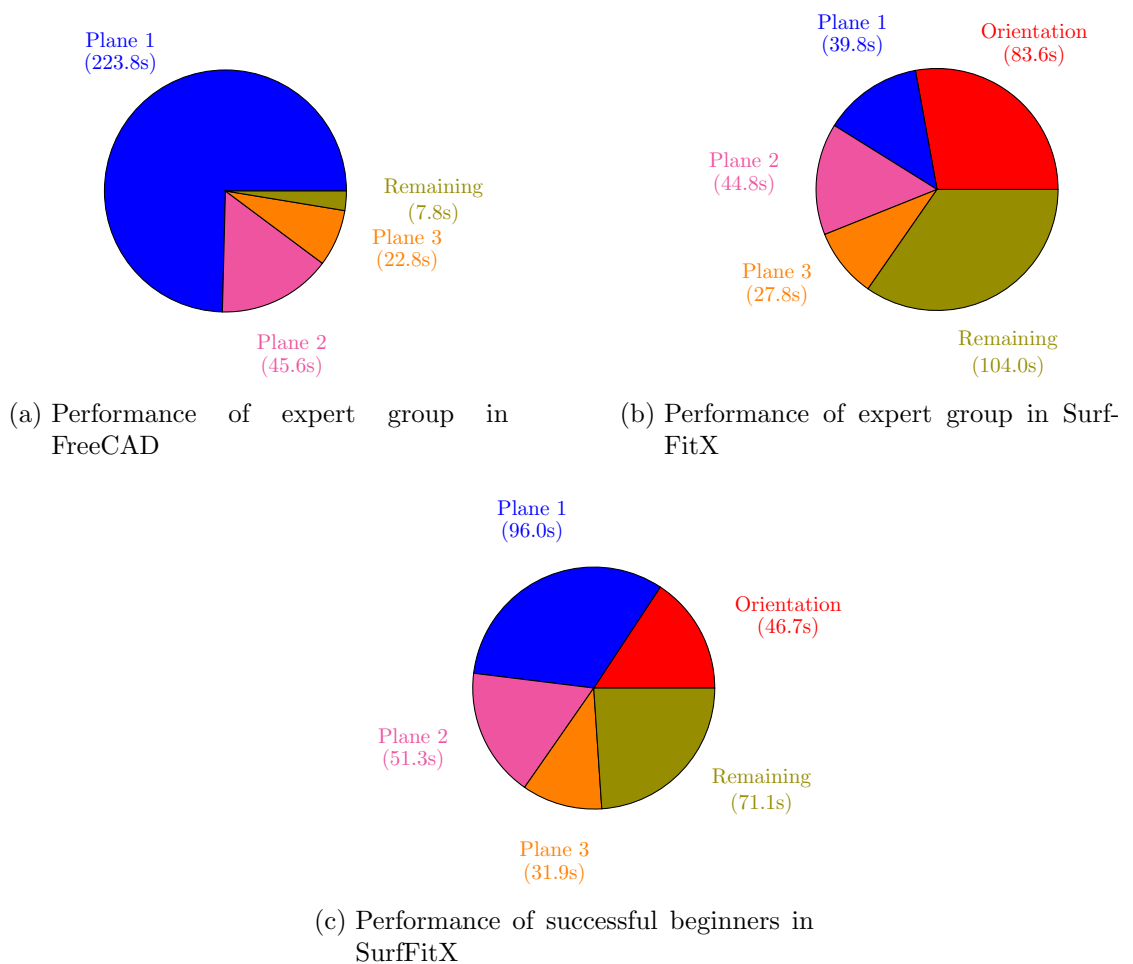


Figure 7.8.: **Comparison of time usage in part one** – We compare the experts in FreeCAD (Figure 7.8a) and SurfFitX (Figure 7.8b). For the successful beginners we only evaluate the performance in SurfFitX (Figure 7.8c).

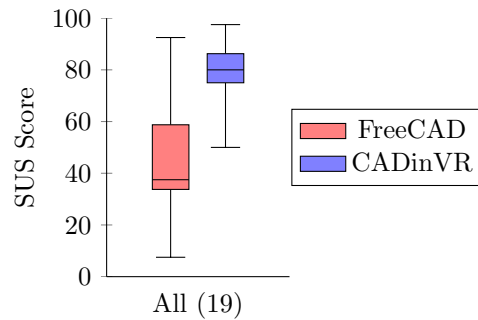
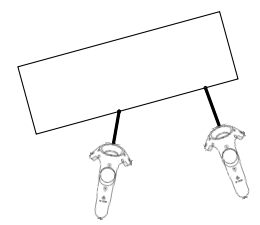


Figure 7.9.: **Comparison of the SUS scores in a boxplot** – The SUS score of SurfFitX system is higher than the score of FreeCAD. Comparing the average SUS scores of all participants in the both systems we measured a p-Value of  $10e - 7 < 0.05$ , the result is significant.

### 7.2.2. System Usability Scale

SurfFitX obtained a average SUS score of 80.9, whileFreeCAD scored 44.6 points. In general a higher SUS score corresponds to better usability (see Figure 7.9).

For better readability of the figures in this section we reformulated the negatively formulated SUS items ( $q_2, q_4, q_6, q_8, q_{10}$ ) in a positive manner and adjusted the SUS scores per item correspondingly. Therefore, higher scores per item always correspond to better usability. SUS item  $q_2$  – “I found the system unnecessarily complex” – is, for example, reformulated to  $q'_2$  “I did not find the system unnecessarily complex”. The score is adjusted to  $q'_2 = 5 - q_2$ . For the positive items ( $q_1, q_3, q_5, q_7, q_9$ ), we keep the original evaluation procedure  $q_{1,3,5,7,9} - 1$  (see Table 7.1).

For the single SUS items the average scores of SurfFitX are always higher than those of FreeCAD (see Figure 7.10). If we take a look at the differences between FreeCAD and SurfFitX we see: For some items the difference is high –  $q_2$  “I did not find the system unnecessarily complex” and  $q_8$  “I found the system not cumbersome to use” – for other items the difference is low –  $q_4$  “I think that I would not need the support of a technical person to be able to use” and  $q_6$  “I thought there was no inconsistency in this system”.

If we compare the single SUS items among the expert’s and beginner’s group we do not observe statistically significant differences.

$q_1$	I think that I would like to use this system frequently
$q_2$	I did not find the system unnecessarily complex
$q_3$	I thought the system was easy to use
$q_4$	I think that I would not need the support of a technical person to be able to use this system
$q_5$	I found the various functions in this system were well integrated
$q_6$	I thought there was no inconsistency in this system
$q_7$	I would imagine that most people would learn to use this system very quickly
$q_8$	I found the system not cumbersome to use
$q_9$	I felt very confident using the system
$q_{10}$	I did not need to learn a lot of things before I could get going with this system

Table 7.1.: **The ten SUS items.** – For easier readability in this section we reformulated the SUS items in a positive manner. The userstudy was conducted using the original questionnaire given in subsection A.2.2.

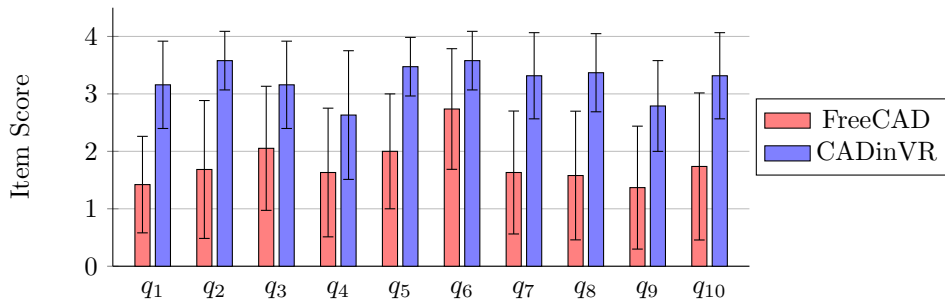
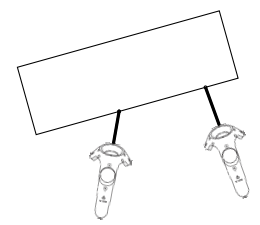


Figure 7.10.: **Average ratings of the SUS items** – Average rating of the different SUS items in both systems for all participants (19). Comparing the items for all participants we obtain p-Values smaller than 0.05. Therefore the measurements are statistically significant.





### 7.2.3. Interview

In the interview session we obtained specific feedback regarding FreeCAD and SurfFitX, and we were able to talk about the tasks in detail. Additionally we received ideas for possible improvement, as well as comments regarding the potential use and possible pitfalls of CAD in a VR environment.

In the following subsection we summarize interesting or helpful statements from the participants. Numbers in brackets indicate that multiple participants gave identical or similar statements. Statements without a number are statements that have only been made by a single participant.

**Comparison of FreeCAD and SurfFitX:** Comparing FreeCAD and SurfFitX, a majority of the participants (19 out of 20) stated that they preferred SurfFitX. The reasons why the participants did not like FreeCAD belong to the following classes:

- **Conceptual problems:** Many participants (12) stated that FreeCAD was not intuitive. They described the way of interacting with the system as being formal and abstract, especially the concept of 2D working planes (8) and the rotation mechanism (7) were problematic.
- **Background knowledge:** Some participants (3) stated that as a beginner they had to learn a lot before being able to use FreeCAD. One reason was the overwhelming user interface with many buttons (2); another reason was the necessity for learning new concepts. But also modeling experts stated that coming from a different system (e.g. 3DsMax, Catia, Inventor, ArchiCAD, NX) it was often hard to learn the different controls, even though the concepts are usually similar.
- **3D visualization:** Some participants (3) had problems with the 3D visualization in FreeCAD. They stated that it was hard to imagine a 3D object if it was only visualized with 2D means on a computer screen.

**Feedback regarding the tasks:** All participants stated that the task was basically simple. A majority of the participants (18) stated that learning how to solve the task was easier in SurfFitX:

- It was easier to learn how to solve the task, because less steps were needed to solve the task, and each step consists of less substeps. In general there less preliminary knowledge was needed and the user had to learn less for achieving the same result (8).
- For solving the task the user did not need to use abstract concepts, since the interaction was intuitive and natural. This made learning easier (6).
- The *graphical user interface* (GUI) was simpler and had less elements (3).

With respect part one (positioning of planes and creation of cross sections) a majority (19) stated that the positioning of the planes was easier in SurfFitX, because the whole workflow is more intuitive and less formalized. One participant stated that positioning of the planes was good in both systems, but slightly easier in FreeCAD.

A slight majority of the participants (11) stated that the orientation in 3D space and positioning of the camera was good in both systems. Anyhow, some participants (5) first had to get used to the way of controlling the camera in SurfFitX. Some participants (3) explicitly stated that the orientation in FreeCAD was easier, due to the better overview.

We received the following additional feedback regarding camera control and positioning of the planes in SurfFitX:

- Some participants (4) stated – even though for rough positioning they prefer VR – that exact positioning was hard in SurfFitX and easier in FreeCAD.
- Moving the head was more intuitive than using a 3D mouse<sup>4</sup>.
- In FreeCAD the dataset is always centered, this makes orientation easier than in SurfFitX.
- With the *head-mounted display* (HMD) controlling the camera and the Vive controllers grabbing the object, a parallel positioning of camera and object was possible in SurfFitX. In FreeCAD these steps could not be done at the same time, but had to be done sequentially.
- In SurfFitX it was sometimes hard to obtain a feeling for the depth, large objects that were far away seemed like smaller objects that were closer to the point of view.
- In SurfFitX it was easier to work while standing, because one can move around freely and look at the object from different positions.

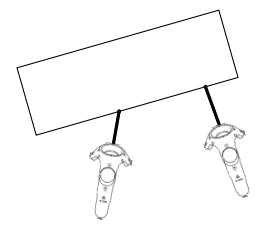
In part two (building a loft surface from existing contours), the largest part of the participants stated that both systems were equally good (9). Many participants preferred SurfFitX (7), because it was more direct and simple to select the planes without a context menu. Some participants (4) stated that FreeCAD was better, because the context menu helps with the selection and the laser pointer selection concept in VR makes exact selection difficult.

Some remarks about part two:

- The context menu was especially important in complex scenes, because it made selection of specific parts easier and also visualized the order of selection (4).
- Implementation of a proper, context sensitive visualization in SurfFitX could provide the context while still keeping the interface simple.

---

<sup>4</sup>Feedback from experienced user who usually uses a 3D mouse. All experiments in this study have been performed without a 3D mouse.



- The evaluation of the resulting loft surface was easier in SurfFitX.
- In SurfFitX the direction connection of the menu to the controller was confusing.
- SurfFitX was very specialized and therefore it was easy to create a good interface.

Finally we also asked the participants whether solving the tasks was more enjoyable in one of the two tools. The largest part stated that they only enjoyed working with SurfFitX (10), because it was intuitive and fast to learn, while the interaction with FreeCAD was very abstract and frustrating. Nearly as many participants enjoyed working in both systems (9). One participant did not enjoy working with any of the two systems.

We received the following comments regarding personal satisfaction while working in a virtual environment:

- VR was fun, because it was something new (5).
- The 3D impression in VR gave the possibility of moving the body and the feeling of being involved, which made it more enjoyable (4).
- People wearing glasses might have problems with wearing VR hardware.
- Working in a VR environment was fun; working in a desktop environment was work (2).

**Feedback for improving SurfFitX:** We asked the participants for feedback regarding possibilities for improvement of SurfFitX. We shortly summarize interesting feedback below.

- **Camera and orientation**

- A mechanism for teleporting would give the user the possibility to look at the model from a different perspective without touching the coordinate system or the model.
- Showing a bounding box of the model or limiting the field of view would help with respect to orientation.
- Use a special visualization if user is inside the object.
- Option for fitting the model to the field of view for improved orientation (3).

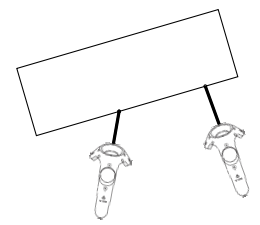
- **Visualization**

- Use different colors for different types of objects.
- Visualize order of selection when defining the loft (2).

- **Menu and context**

- Show tooltips directly on the controller in order to help beginners.
- An (optional) context menu or model tree would be helpful to support the user when selecting objects (3).

- Instead of using a GUI or context menu, improve visualization with context sensitive filters that make selection easier (3).
  - Possibility of voice control.
  - Menu selection using the touchpad, not by pointing on the corresponding entry using the second controller. Remove the menu from the controller and use a *head-up display* (HUD) or virtual tablet instead (2).
  - Develop objects providing intuitive contextual behaviour, like for example a basket for collecting objects or a stack for selecting objects in a certain order.
- **Accuracy**
    - Implement the functionality to scale down controller movement (e.g. rotation or translation of the controller reduced to 10% for more exact positioning).
    - Filter hand tremor.
    - Add 2D working planes for more exact positioning.
    - Possibility to deactivate single transformations like rotation, translation or scaling.
  - **Object interaction**
    - For close field interaction it would be more intuitive if the user was able to grab objects by touching them with the controller – additionally to pointing at them with the laser beam (2).
    - For far field interaction provide an additional transformation mode, where the object is first selected and then the motion of the controller is mirrored, this allows us to rotate the object in any direction – with the currently implemented transformation mechanism the only possible rotation is around the axis of the laser beam (2).
    - Instead of selecting large objects – like the mesh – by pointing on them, use smaller labels for reference in order to avoid a selection by accident.
    - Provide a two handed rotation mode.
    - Selection button difficult to press, better use trigger for selection and grip button for grabbing.



## 7.3. Expert Interview

We conducted an interview with the CAD expert Manuel Biedermann working at Siemens, Corporate Technology. He is a master student writing his thesis on the topic *Simulation-driven design for additive manufacturing* [77], targeting the development of a design process for additive manufacturing of parts with a simulation driven approach.

During the interview we demonstrated the prototype SurfFitX. Additionally we demonstrated the VR sketching and drawing software Google Tiltbrush [11].

In the following we summarize the results of the interview.

### 7.3.1. Today's engineering workflow

The design of engineering parts is an interdisciplinary task, where multiple experts from different fields such as 3D modeling, numerical simulation, and mechanical engineering cooperate. Exchange and generation of knowledge and results happens on a variety of different channels: One important way of communication are hand-drawn sketches, because it is simple, fast and no special tools are required for the creation of a sketch. CAD is a core technology for the creation of detailed 3D models, because it is used for visualization, specification, validation and manufacturing of the part.

An intrinsic problem of sketches in the context of complex 3D parts is the restriction to the two dimensional sheet of paper: On the one hand sketches always have to be interpreted in order to transform them into three dimensional models. On the other hand, whenever CAD geometry is validated a comparison to the sketches happens. This process is vulnerable to error in both directions: the sketch might be misinterpreted if it is translated into CAD or important features are not visualized when a comparison of the CAD model and the sketch happens. With established technology – for example by using a CAD tool – sketching in 3D is possible, but it requires expert knowledge, interferes with the creative process and is time intensive; therefore this approach is not feasible.

In [77] the application example of a twister is demonstrated: Fluid that originates from a circular tube has to be distributed to a ring shaped domain. Additionally the fluid has to exit the ring shaped domain with high tangential velocity. The fluid has to be "twisted". A part that fulfils these requirements is designed in [77] through an simulation driven approach. Here drawings have been used for the development of concepts and a CAD model has been created for validation and manufacturing of the part (see Figure 7.11).

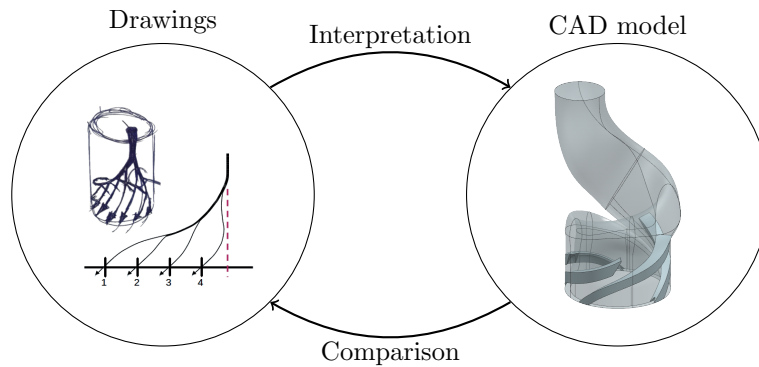


Figure 7.11.: **The interface between 2D sketches and 3D model** – Interpretation of the sketch can lead to misinterpretation, reduction of the 3D model can remove important features. Examples from [77].

### 7.3.2. Sketches and drawings in virtual reality

Using VR technology the user can directly create schematic concepts in 3D, these drawings can be imported into a CAD environment for further processing. Additionally the 3D CAD model can be visualized in the VR sketching environment.

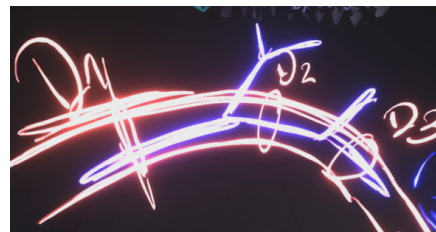
We identified the following places, where VR technology can improve the design process:

- fast sketching in 3D due to intuitive user interaction
- more consistent interface between sketching and 3D modelling
- possibility to involve non-experts into the design process
- using multiuser VR scenarios for collaborative – and possibly remote – design
- 3D visualization, validation and annotation of complex parts

As a part of the interview we created concepts and sketches in VR using Tiltbrush [11] (Figure 7.12).

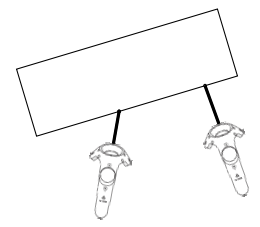


(a) Sketch of twister concept.



(b) Sketch with annotations.

Figure 7.12.: **Concept created in VR** – Using VR technology we create a sketch of complex 3D geometry in a fast and intuitive way (Figure 7.12a). Annotations can be added easily (Figure 7.12b). Created using Tiltbrush [11].



## 8. Discussion, conclusion and future directions

In the previous chapter we presented the resulting geometry that has been created using *Virtual Reality Surface Fitting Extension* (SurfFitX), we summarized the user study and we presented the outcome of an interview with a *Computer-aided design* (CAD) expert. In this chapter we now discuss these results and draw a final conclusion.

We first evaluate the design workflow that has been developed in the scope of this thesis. Here we used the prototype SurfFitX for the reconstruction of topology optimized parts in a *virtual reality* (VR) environment (section 8.1).

In section 8.2 we interpret the results of the userstudy, where we evaluated SurfFitX in comparison to the desktop CAD tool FreeCAD.

In section 8.3 we discuss the possible role of VR technology in the context of CAD on the basis of the feedback that we received in the expert interview (section 7.3).

We summarize our work and give a conclusion in section 8.4. Finally we show up future directions in section 8.5.

### 8.1. The design workflow

In section 7.1 we presented example geometry that has been created using the workflow proposed in chapter 3.

We observed that the quality of the consistent level set strongly depends on the underlying data:

If the resolution of the grid is fine compared to the smallest features, important features are conserved and the geometry remains connected. Smoothing removes noise and creates a smooth mesh that is consistent with the boundary conditions (subsection 7.1.1).

If features are on the scale of the mesh resolution, we loose important features. The smoothing algorithm fastly removes noise, but fine features are lost as well (subsection 7.1.2).

We additionally observe that with a higher number of smoothing steps a larger amount of volume is lost. This effect is due to the mean curvature flow that we apply to the level set: This flow does obviously not conserve volume and no convergence happens. Therefore, we have to stop the smoothing procedure manually. A volume conserving and converging scheme is preferable.

Using the prototype SurfFitX we can easily extract contour shapes from the mesh. The contour shapes approximate the contour well. However, the closed contour shapes are only  $C^0$  continuous and therefore kinks appear where the start- and endpoint of the

contour meet. This deficit is due to *Open CASCADE Technology* (OCCT)’s approximation algorithm, which does not support higher continuity constraints for closed curves. Sketch guiding lines is done fast and efficiently. The guiding lines approximate the shape very well and they can be positioned at arbitrary positions on the shape.

The resulting contours, guiding lines and loft surfaces are successfully exported in `.step` file format and they can be reused in NX. We remodel the contours and therefore make them smooth using NX. The guiding lines are used for modeling of regions that cannot be approximated with loft surfaces.

We summarize that the automated creation of boundary condition consistent level set data greatly helps if the resolution of the dataset is fine enough. The smoothing algorithm is able to create a smooth mesh, but has to be improved with respect to convergence criteria and volume conservation.

The prototype SurfFitX helps the user to extract geometry features from mesh geometry in a fast and intuitive way. We can import the extracted geometry into professional CAD software. However, we still have to remodel curves and surfaces due to unsatisfactory continuity of the surfaces and curves extracted using OCCT: The contour curves only provide  $C^0$  continuity and the resulting loft surfaces show kinks.

Guiding lines and contour curves provide guidance for remodeling the shape. At the same time the user can disable single guiding lines in order to concentrate on selected features. This is a great improvement compared to remodeling the shape on the basis of an unstructured mesh geometry, where contour and guiding curves for the creation of surfaces have to be extracted manually by sampling the mesh. Additional effort is necessary for partially hiding the mesh. This often leads to excessive camera adjustments or manual definition of cutting planes in order to obtain a good view onto interesting regions.

## 8.2. User Study

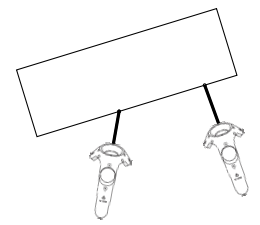
In this section we elaborate the results of the user study (section 7.2) from the perspective of the following propositions:

- A VR tool can provide a simple and more intuitive user interface.
- Learning how to solve 3D design tasks is easier in a VR environment.
- VR technology helps to make 3D design tools more accessible.
- VR technology is especially useful for designing concepts.

### 8.2.1. Simple and intuitive user interface

VR technology tries to support the natural behaviour and movement of the user: If the user moves the head the camera is positioned correspondingly; the controllers exactly follow the motion of the user’s hands. We observed that users experience this as a more natural user interface than using the desktop computer input devices keyboard





and mouse (see subsection 7.2.3). In the context of CAD users evaluated the natural way of controlling the camera and moving objects positively.

Comparing a VR system to a desktop system, we observe that the users stated that the VR system is simple (Figure 7.10; *System Usability Scale* (SUS) item  $q_2$ ), the various functions are well integrated (Figure 7.10; SUS item  $q_5$ ) and the system is built in a consistent way (Figure 7.10; SUS item  $q_6$ ). On average the VR system obtained a higher SUS score than the desktop system (Figure 7.9).

Anyhow it should also be kept in mind that the high level of immersion in a VR system sometimes leads to an overhead for orientation (in Figure 7.8 see time for orientation and positioning of the dataset) and that the evaluated VR system is a CAD system with a very narrow toolset (only two tools exist) and therefore a low level of complexity.

All in all we summarize that a VR tool can provide a simple and more intuitive user interface than a CAD tool.

### 8.2.2. Easier learning experience

The participants of the user study had to solve the same modelling task in a VR and in a desktop environment. None of the participants had a high level of expertise in any of the two provided tools and therefore all participants first had to learn how to use the system.

In general it was easier to learn the workflow in the VR environment and the users felt more confident in using the system after the explanation (Figure 7.10 SUS item 9).

While the desktop system is abstract and formal, the VR system provides a more intuitive and natural way of interacting with objects: Abstract concepts like rotation around an axis or working planes do not even have to be introduced in the VR environment, less substeps are needed to perform an action and the user does not have to be used to visualize 3D content on a 2D computer screen (see subsection 7.2.3). Therefore, less things have to be explained and understood (Figure 7.10 SUS item  $q_{10}$ ). This saves time (see Figure 7.5) and makes it easier to understand how the system works (Figure 7.10 SUS items  $q_3$  and  $q_4$ ).

Also experts that are already used to CAD technology felt more comfortable with a VR environment than with a CAD system they are not used to. Different CAD systems have different paradigms of user input and even an expert can easily get confused (subsection 7.2.3).

Therefore, we see that learning how to solve 3D design tasks is easier in a VR environment than in a desktop environment. Anyhow, we should keep in mind that professional CAD programs like NX [1] or AutoCAD [2] usually provide a better user interface than FreeCAD. A comparison of SurfFitX to a professional CAD program should be considered.

### 8.2.3. Better access to 3D design tools

In our user study more participants were able to successfully complete part one of the task using VR technology, while only a small group (which we consider as experts) was able

to successfully complete part one in both environments (Figure 7.6). We also observed that the participants mostly enjoyed working in VR and felt more involved, while using the desktop environment was often associated with working (subsection 7.2.3).

This means that VR technology helps to make 3D design tools more accessible and it therefore democratized high tech software that can be used in the product development [78].

#### 8.2.4. Concept design

The arguments mentioned above make VR technology a great tool for conceptual work: with possibility to explain the tools very fast and the high rate of success, also users unexperienced in 3D modelling can contribute to a concept for solving a 3D problem.

We claim that working in a VR environment also supports creativity, because it is easier to use (Figure 7.10 SUS item  $q_3$ ), faster (Figure 7.8), less cumbersome and frustrating (Figure 7.10 SUS item  $q_8$ ), and users are willing to use the system frequently (Figure 7.10 SUS item  $q_1$ ).

Anyhow, if we turn away from designing a concept and want to provide means of accurate construction, traditional CAD has many advantages: Some participants already stated that for accurate working they prefer traditional CAD (subsection 7.2.3) and for complex parts the advanced GUI elements of traditional CAD provide guidance (subsection 7.2.3 comments on part 2 of the task). In more traditional tasks with less degrees of flexibility and freedom the benefits of VR are not as clear (Figure 7.7).

We summarize that VR technology greatly improves solving tasks with many degrees of freedom, where accuracy is less important than creativity and ease of use; here VR technology has many advantages over traditional CAD. For constructive tasks, where accuracy and control play a major role traditional CAD is preferable.

### 8.3. The role of virtual reality in computer-aided design

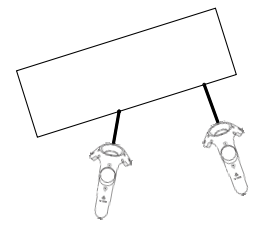
In section 7.3 we described the potential of VR technology in the context of CAD. VR technology provides great possibilities for drawing as well as fast and collaborative development of concepts. VR is accessible to beginners and intuitive.

Traditional CAD provides a wide range of advanced and highly accurate tools for the construction of detailed models and educated experts know how to work with professional CAD tools in a fast and efficient way. Today's CAD software is mature and looks back on 40 years of history<sup>1</sup>. This shows: Translating the toolset of professional CAD tools into a VR environment and the development of an intuitive user interface in order to obtain a system that can replace today's CAD technology is a task that is out of reach in the scope of a thesis.

Traditional CAD is the optimal tool for accurate modelling and for the creation of 2D technical drawings due to perfectly fitting output and input devices [20, 79], it

---

<sup>1</sup>The initial release of the predecessor of Siemens NX was in 1973, Catia has been released in 1977 and AutoCAD in 1982.



	CAD in VR	CAD on Desktop
flexible and fast modelling	+	-
accurate construction	-	+
develop a concept	+	-
refine model	-	+
fast to learn	+	-
highly skilled experts	-	+
intuitive working experience	+	-
powerful modelling tools	-	+
real world impression	+	-
good overview and control	-	+
intuitive input device in 3D	+	-
exact input device in 2D	-	+

Table 8.1.: **Comparison of two systems** – We compare CAD in VR and on desktop systems.

provides a good overview over parts, and an advanced *graphical user interface* (GUI) with context menus that help the user navigate through complex workflows. VR technology supports intuitive and fast sketching, provides an intuitive and immersive experience, and simple workflows that can be learned easily. We summarized the respective strengths of traditional CAD and VR in Table 8.1.

Similar to [20] we propose a dual approach, where the VR and the desktop interface can be used in parallel. This hybrid system should still provide enough flexibility such that the user can choose between solving a task either in the desktop or the VR environment. The underlying data and the presentation of the model should be kept consistent across the interfaces in order to provide a consistent user experience and the overhead for switching between the VR and the desktop environment should be kept at a minimum (see Figure 8.1).

## 8.4. Conclusion

In this work we proposed an approach for the automatized generation of level set data of topology optimized geometry that is consistent with existing boundary geometry. We implemented the prototype SurfFitX that provides VR tools for contour extraction, loft surface creation and fast guiding line sketching. We evaluated VR technology in the context of CAD by, firstly, performing a user study that quantified the performance of users solving an identical task in SurfFitX and FreeCAD and, secondly, by interviewing a CAD expert about the potential role of VR in professional CAD.

Finally we ask the following question: Does the functionality that has been implemented during the work on this thesis improve CAD? We implemented three tools: One supports the extraction of contours from faceted geometry, another tool is used for the

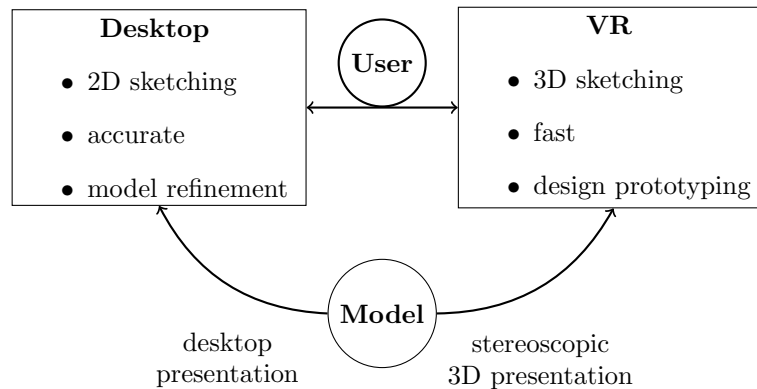


Figure 8.1.: **Hybrid CAD system** – We propose a hybrid system using VR and desktop technology for CAD. Switching between the interfaces should be fast and easy and a consistent model has to be maintained.

creation of loft surfaces from closed curves and finally we provide a tool for the creation of guiding lines.

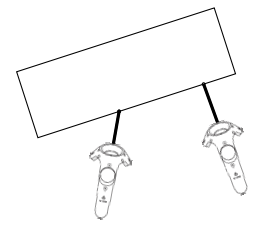
In the section 7.2 and section 8.2 we showed that beginners profit from the functionality that has been implemented in VR. They mainly profit from VR due to a more intuitive interaction with geometry. However, in the interview we experienced that experts are much harder to impress: They know how to achieve the same result using traditional CAD software and they do not necessarily profit from the VR implementation. Additionally they usually criticize the limited toolset of our VR implementation and the quality of the generated geometry.

On the one hand, a tool for the creation of loft surfaces already exist in a similar form in traditional CAD and usually the quality of professional tools is superior to our prototype implementation: As an example continuity constraints can be set easily. Here experts do not see an advantage in using our prototype.

On the other hand, the tools for fast sketching and contour extraction are custom tools that improve the workflow. We can easily extract features from mesh geometry and reuse them in professional CAD tools. A better integration of these tools into existing CAD software would be preferable in order to further enhance these tools.

This leads us to the following conclusion: If we want to improve CAD by introducing new functionality in VR, we have to provide innovative VR tools that even improve the workflow of CAD experts<sup>2</sup>. For the creation of innovative VR tools we, firstly, have to carefully think about applications where a real demand for new tools exists and, secondly, provide a user interface that does not suffer from the drawbacks of VR, namely inaccuracy and a limited degree of flexibility compared to professional CAD tools. Here a seamless integration of VR technology into existing workflows is crucial in order to provide a better user experience where no compromises have to be made.

<sup>2</sup>That VR technology definitely has the potential to improve today’s CAD workflow has been demonstrated in section 7.3 and section 8.3.



## 8.5. Future directions

Revising the feedback given during the interview sessions of the user study in subsection 7.2.3 we see many potential ideas for improvement of our tool:

The usability could be improved, by adding context based menus or introducing improved metaphors for rotation and scaling. We propose an additional interaction mode for improving rotation like for example the Voodoo Dolls technique[80]. We would like to provide the possibility to fit the scale of the object to the user's field of view to prevent loss of orientation, this is for example implemented in FreeCAD [4]. With respect to CAD-specific functionality we propose adding interaction techniques, that help to suppress hand tremor and therefore improve accuracy. We could, for example, decouple the six degrees of freedom for positioning and selection by introducing an interaction metaphor similar to Balloon Selection [81].

By directly integrating the topology optimization and the level set framework into SurfFitX, we could greatly improve the user experience and give the designer more freedom by providing a fully interactive application (see [69]). Boundary condition shapes and the optimization domain for the topology optimization could be defined directly in VR and geometry reconstruction happens in an iterative process. Finally this would lead to a fully integrated topology optimization design assistant that could greatly improve product development [78].

With respect to the algorithms we propose the use of a more advanced CAD kernel than OCCT. Some example kernels are mentioned in subsection 2.2.1. With a more advanced kernel we can fit  $C^n$  continuous, closed curves to point data and therefore improve the creation of contours, and provide a more stable working environment<sup>3</sup>.

In the current implementation of the level set smoothing algorithm we have to manually set the maximum smoothing time and find a good trade off between a smooth and conservation of features. By introducing a volume preserving flow like the min/max flow [82], this deficit could be finally eliminated. Additionally we propose using the level set data for contour extraction for a faster and more robust point sampling<sup>4</sup>.

In order to be able to find innovative new CAD tools, where VR technology can really revolutionize CAD and not only beginners, but also experts can profit, we propose to talk to CAD experts about their respective workflows and needs. Possible fields for application of VR technology are free form modelling, sketching in 3D and visualization of complex parts.

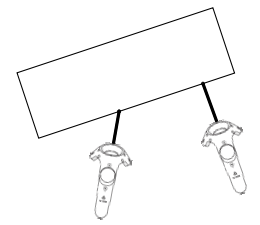
Finally we propose to perform another user study after the aforementioned improvements have been implemented. With the data from the user study of this thesis we could easily quantify the improvement. A larger group of experts would be needed in order to obtain statistically significant results for the comparison of beginner's and expert's user experience. Additionally we should compare SurfFitX with a professional CAD tool instead of FreeCAD, since professional CAD tools usually provide more advanced user interfaces and a better user interaction.

---

<sup>3</sup>In OCCT Boolean operations on primitive bodies, for examples, sometimes do not work properly.

<sup>4</sup>Similar to the algorithm used for the creation of guiding lines

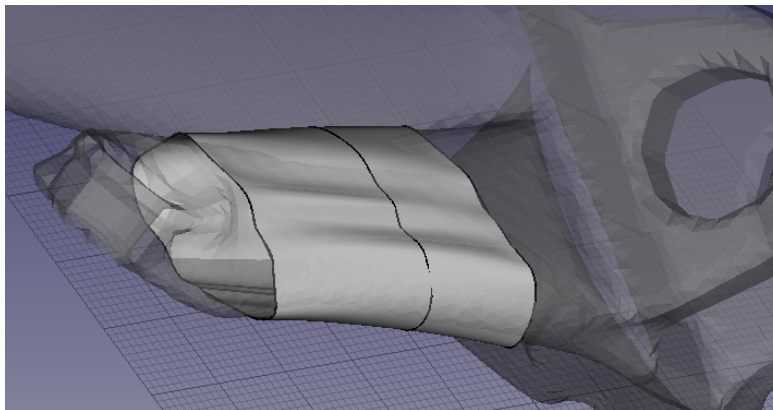




# A. User study: Material

## A.1. The task

### A.1.1. Userguide FreeCAD



#### Preparation

- prepare `task.vtk` and `task.stl`
- make sure that the correct vtk path is set.
- open FreeCAD
- close all unnecessary GUI elements. We only need: ComboView, Workbench, Draft mod Tools, Loft Tools
- open New Document
- import `task.vtk`
- open Lofting Workbench

#### Explanation

- Show Main Windows
- Show Document Tree

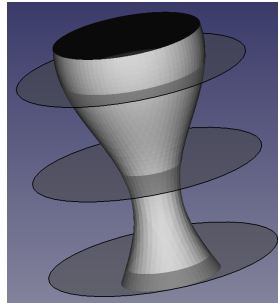
- Show Workbench Selection
- Show Toolbar
- Show panning, rotation (even though it is not needed in the sample dataset!) and zooming of the view
- Selection and deselection of objects using document tree and main window.
- Hiding and showing of objects using **space**

## Rules

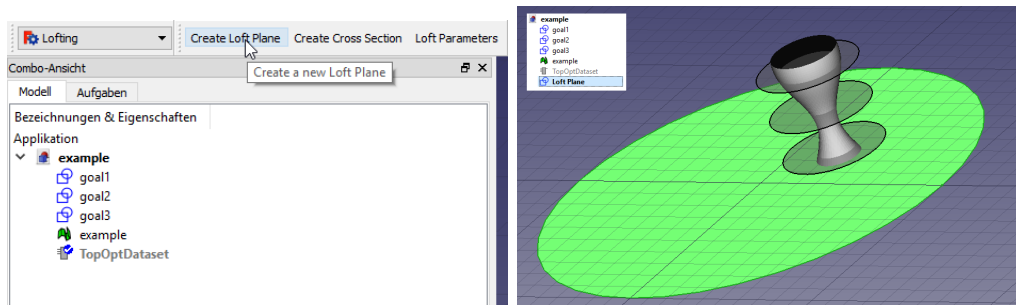
- Before start you have 2 minutes for getting started and asking questions.
- You should try to solve the task without asking questions.
- For steps (1.) - (5.) of the workflow the working time is restricted to 5 minutes.
- The guiding planes are just a assisting visualization, usually they do not exist and finding the right plane is part of the job. Therefore the loft planes have only to be positioned roughly at the same place.
- If the task has not been completed in this time, we can use a checkpoint result.

## Workflow

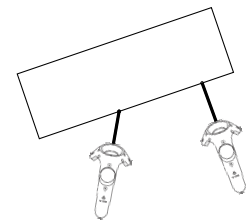
1. position the camera.








2. create a new loft plane by selecting **Create Loft Plane** in the toolbar.

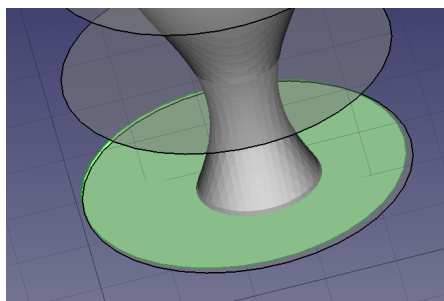




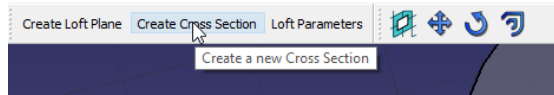


3. Transform the loft plane to obtain the demanded cross section. Take care that the center of the loft plane lies **inside the dataset**. The following tools in the toolbar should be used, points are selected in the main window by pressing **left+ctrl**:

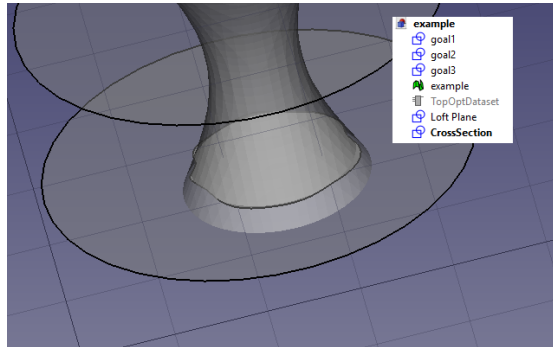
- **Select working plane (camera plane):**
  - a) Make sure that no items are selected.
  - b) Hit  in the toolbar.
  - c) In the dialog hit **View** to set the new working plane.
- **Select working plane (on existing plane):**
  - a) Make sure that no items are selected.
  - b) Select an existing plane with **ctrl + left**
  - c) Hit  in the toolbar.
- **Translation in the working plane:**
  - a) Select the translated object from the document tree.
  - b) Hit  in the toolbar.
  - c) Select a starting point in the main window.
  - d) Select an end point in the main window.
- **Rotation around axis perpendicular to the working plane:**
  - a) Select the rotated object from the document tree.
  - b) Hit  in the toolbar.
  - c) Select the rotation center in the main window.
  - d) Define the starting orientation by selecting a point.
  - e) Define final orientation by selecting a point.
- **Scaling in the working plane:**
  - a) Select the scaled object from the document tree.
  - b) Hit  in the toolbar.
  - c) Define the scaling by selecting a point.



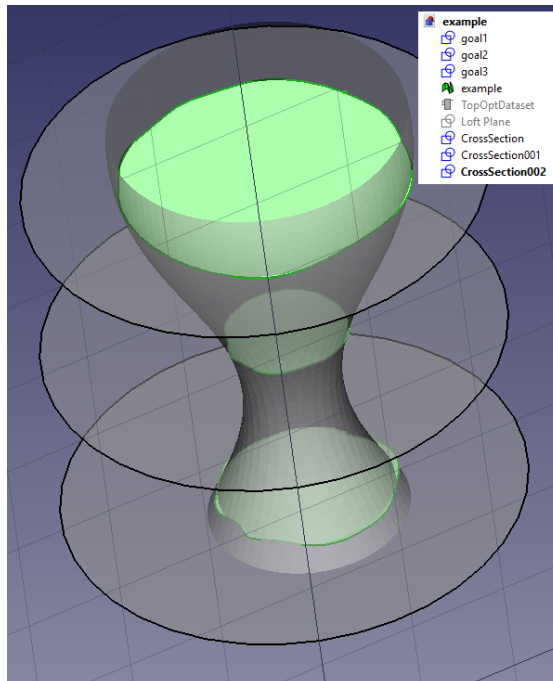
4. Create a cross section by selecting the loft plane and hitting **Create Cross Section** in the toolbar.

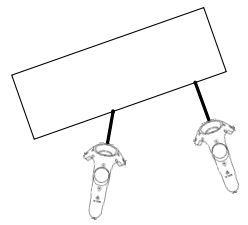


5. You may want to check the cross section by hiding the loft plane.



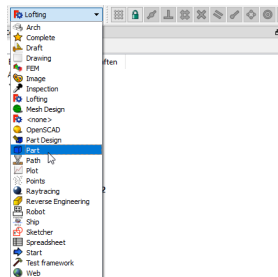
6. Create 2 more cross sections.






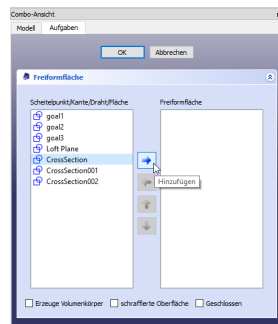
7. Create a loft through the cross sections.

a) open Part workbench.

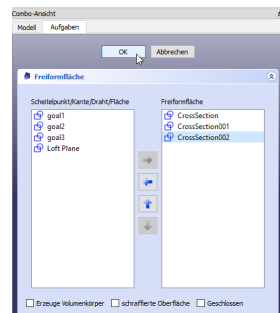


b) Hit  in the toolbar.

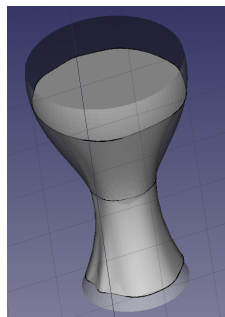
c) In the dialog select the cross sections in the right order.



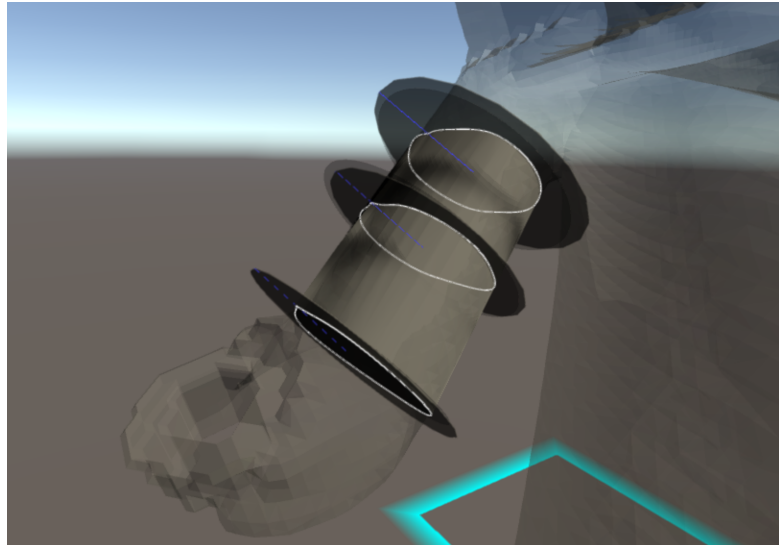
d) Hit OK.



8. Check the resulting loft by hiding the loft plane and the mesh.



### A.1.2. Userguide CADinVR

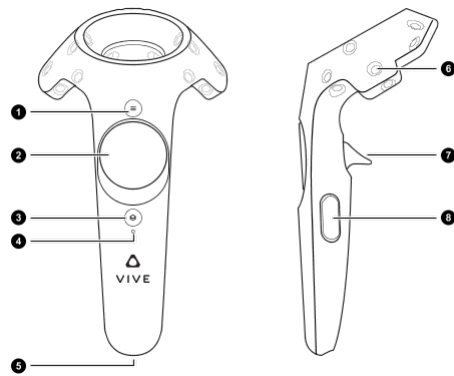
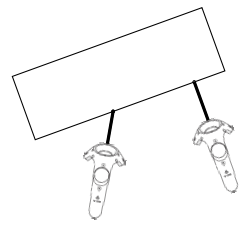


#### Preparation

- prepare `task.vtk`
- open CADinVR
- import `task.vtk`

#### Explanation

- Explain controller
- Explain head mounted device
- Show translation, scaling and rotation of objects (also far translation)
- Show translation, scaling and rotation of mesh (view)
- Explain link of objects and mesh
- Show main menu, explain scrolling and selection
- Show temporary hiding by touching object with controller
- Show selection of objects



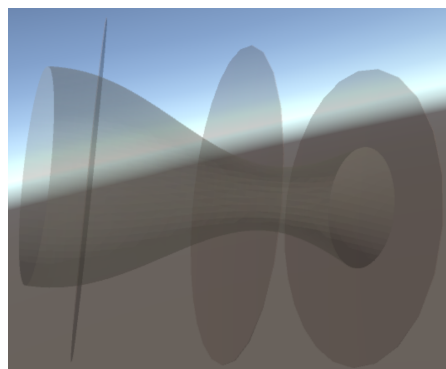
**The HTC Vive Controller:** Menu button (1), Trackpad (2), System button (3), Status Light (4), Micro-USB port (5), Tracking sensor (6), Trigger (7), Grip button (8). Picture and description from [8].

### Rules

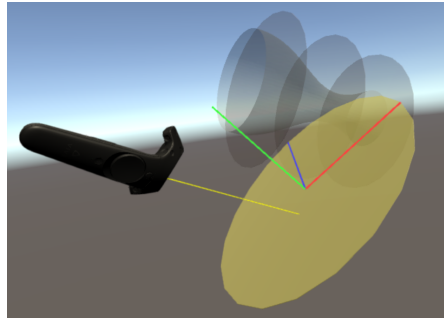
- Before start you have 2 minutes for getting started and asking questions.
- You should try to solve the task without asking questions.
- For steps (1.) - (5.) of the workflow the working time is restricted to 5 minutes.
- The guiding planes are just a assisting visualization, usually they do not exist and finding the right plane is part of the job. Therefore the loft planes have only to be positioned roughly at the same place.
- If the task has not been completed in this time, we can use a checkpoint result.

### Workflow

1. position the mesh.



2. create a loft plane tool by selecting Menu>>Lofting>>Create Loft Plane.



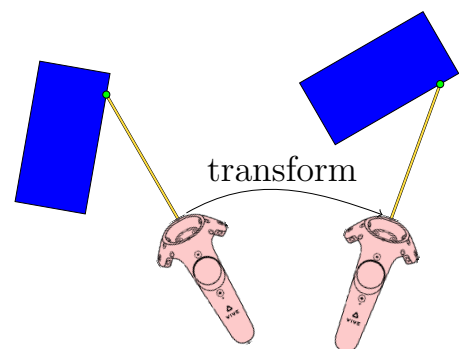
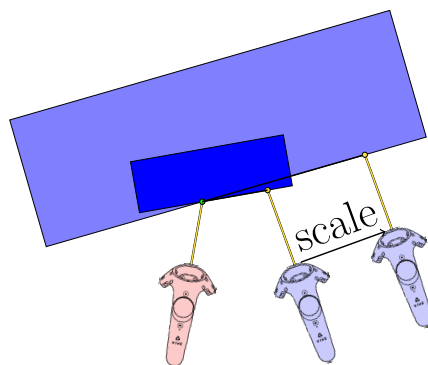
3. Transform the loft plane to obtain the demanded cross section. The following mechanisms should be used:

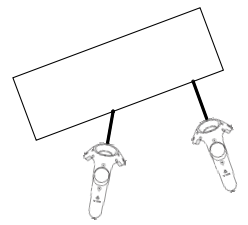
- **Translation and rotation:**

- a) Point on the translated object with one controller.
- b) Grab the object by holding the **trigger** button.
- c) Translate and rotate the object by moving the controller.
- d) Ungrab the object by releasing the **trigger** button.

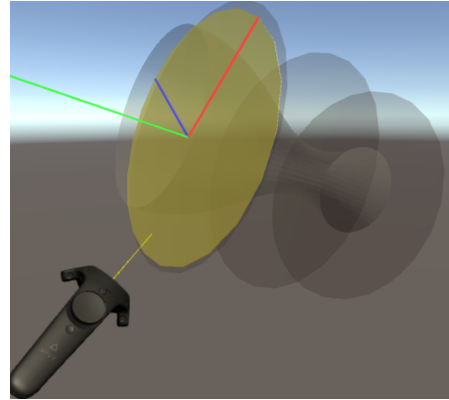
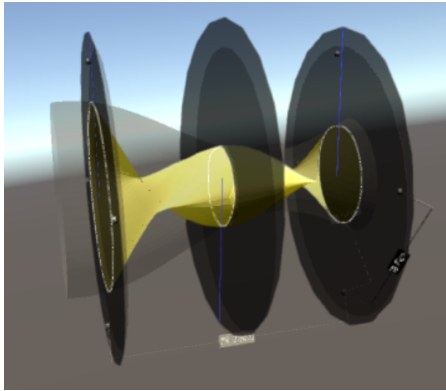
- **Scaling:**

- a) Point on the translated object with two controllers.
- b) Grab the object with two controllers by holding the **trigger** button.
- c) Scale the object by changing the distance between the controllers.
- d) Ungrab the object by releasing both **trigger** buttons.

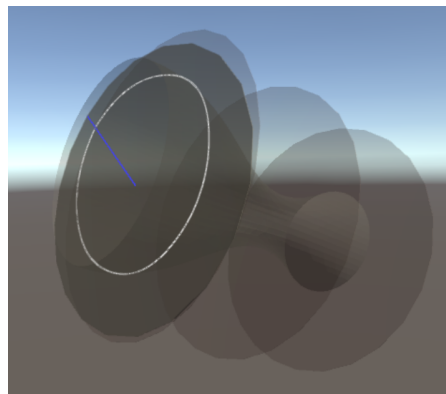




Take care that the center of the loft plane lies **inside the dataset**. Take also care that the blue line points in a similar direction for all the loft planes, otherwise we obtain a twisted shape in the end:

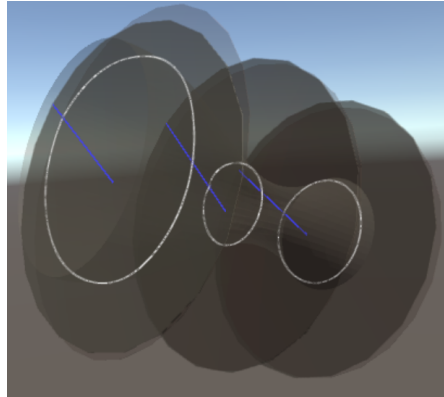


4. Create a cross section by selecting the loft plane.



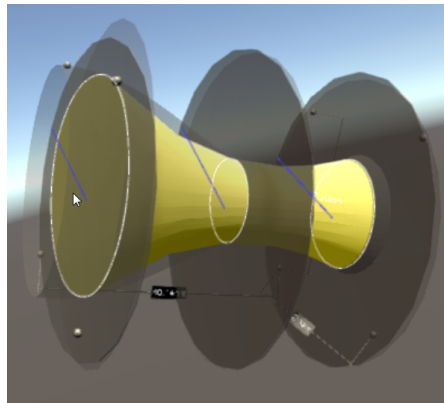
5. You may want to check the cross section. If the cross section is not satisfying, just create a new one.

6. Create 2 more cross sections.



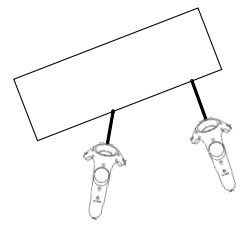
7. Create a loft through the cross sections.

- a) Select the cross sections in the demanded order.
- b) Select Menu>>Lofting>>Create Loft along Wires.



8. Check the resulting loft by hiding the mesh.





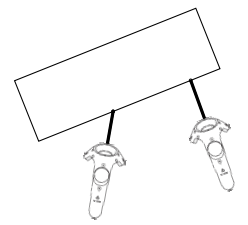
## A.2. Questionnaires and interview

### A.2.1. The Background Questionnaire

	Background	never used					expert
1	experience with 3D design tools						
2	experience with 3D design tools						
3	experience with FreeCAD						
4	experience with VR						
5	experience with HTC Vive and controllers						

### A.2.2. The System Usability Scale (SUS) Questionnaire

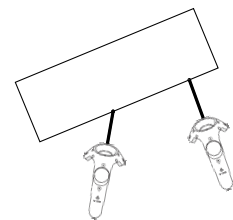
	SUS	Strongly disagree					Strongly agree				
1	I think that I would like to use this system frequently										
2	I found the system unnecessarily complex										
3	I thought the system was easy to use										
4	I think that I would need the support of a technical person to be able to use this system										
5	I found the various functions in this system were well integrated										
6	I thought there was too much inconsistency in this system										
7	I would imagine that most people would learn to use this system very quickly										
8	I found the system very cumbersome to use										
9	I felt very confident using the system										
10	I needed to learn a lot of things before I could get going with this system										



### A.2.3. Interview questions

1. Which system did you like more? / Welches System hat Ihnen besser gefallen?
2. Why did you like the other system less? / Warum hat Ihnen das andere System schlechter gefallen?
3. Do you think the task was complex? / Denken Sie, dass die Aufgabe komplex war?
4. Was it in any of the two systems easier to learn the workflow? / War das Erlernen der Aufgabe in einem der beiden Systeme einfacher?
5. In which system was the first part of the task (positioning of the planes) easier? Why? / In welchem System war der erste Teil der Aufgabe (Positionierung der Ebenen) einfacher? Warum?
6. In which system was the second part (creation of the loft) easier? why? / In welchem System war der zweite Teil der Aufgabe (Erzeugung des Lofts) einfacher? Warum?
7. Did you have fun working on the tasks using any of the systems? (no/only one/both) / Hat es Ihnen Spaß gemacht in einem der Systeme die Aufgaben zu bearbeiten? (nein/nur in einem System/in beiden)
8. Do you have any ideas how to improve the program? / Haben Sie irgendwelche Verbesserungsvorschläge für die Software?





## B. Tutorials

In this chapter we provide two tutorials with code snippets illustrating the functionality of *Open CASCADE Technology* (OCCT) (section B.1) and *Visualization Toolkit* (VTK) (section B.2) that has been used in the scope of this thesis.

### B.1. Open CASCADE Technology: Tutorial and Examples

In the following section we provide a short overview over the basic concepts of OCCT that are important in the scope of this thesis. We explain techniques for the creation and manipulation of geometry and finally we explain the file exchange mechanism of OCCT.

The following material has been compiled based on [47] and [21].

#### Topological model and geometrical model

OCCT relies on the *boundary representation* (BREP) philosophy for representing geometry. This requires OCCT to use both a topological as well as a geometric model for fully describing a shape [30, 83].

OCCT describes the composition of objects and the relationship of these using the topological model. The basic topological entity and parent class of all other topological entities is `TopoDS_Shape` (see Figure B.1). We distinguish shapes that relate to actually existing geometry<sup>1</sup> and those that are only containers for other shapes<sup>2</sup>.

Using the `BRepBuilderAPI`, we can fill these container objects: `TopoDS_Wire` is constructed with `BRepBuilderAPI_MakeWire` of multiple, connected and ordered `TopoDS_Edge` objects. A `TopoDS_Solid` is created using `BRepBuilderAPI_MakeSolid` by a set of faces that form a closed `TopoDS_Shell`<sup>3</sup>.

Using the classes `TopoDS_Iterator` and `TopExp_Explorer` we can traverse a `TopoDS_Shape` and query specific features of the shape (see Listing B.2). Using the `TopoDS_Iterator` we can iterate over all the subshapes of a shape and act in the appropriate way. The `TopExp_Explorer` filters for specific types of subshapes; we can, for example, collect all the edges belonging to a cube or iterate over the faces of a cylinder.

OCCT describes geometric entities using a geometrical model that defines the necessary geometric parameters. There is the `gp` package, that gives access to basic simple

---

<sup>1</sup> `TopoDS_Vertex`, `TopoDS_Edge`, `TopoDS_Face`

<sup>2</sup> the remaining children of `TopoDS_Shape`.

<sup>3</sup> Since OCCT uses BREP for solid body description, the `TopoDS_Solid` is actually *not* related to any geometry.

```

// a compound of different subshapes
TopoDS_Compound aCompound = ...
for (TopoDS_Iterator anIt(aCompound); anIt.More(); anIt.Next()) {
    // get the subshape
    TopoDS_Shape aSubshape = anIt.Value();
    // determine the type
    if (aSubshape.ShapeType() == TopAbs_EDGE)
        // specific treatment for edge
    else if (aSubshape.ShapeType() == TopAbs_WIRE)
        // specific treatment for wire
    else
        // do nothing for other types
}

```

Listing B.1.: **Iterating over a shape** – We iterate over all the subshapes of a compound object and treat the subshapes with respect to their type.

```

// an arbitrary shape
TopoDS_Shape aShape = ...
// the explorer is set to iterate over all the faces of the shape
for (TopExp_Explorer ex(aShape, TopAbs_FACE); ex.More(); ex.Next()) {
    // get face from the explorer
    TopoDS_Face F = TopoDS::Face(ex.Current());
    // do something
    ...
}

```

Listing B.2.: **Exploring a shape** – We iterate over all the faces of a shape. Subshapes of a different type are not considered at all.

datatypes. A `gp_Circ` object describes a circle with a given radius and its axis<sup>4</sup>. The axis defines the origin of the circle and the normal of the plane the circle lies in.

Using the `Geom` package, also more complex shapes can be described: A B-Spline curve with all the necessary parameters<sup>5</sup> is described by a `. B-Spline surface is described using Geom_BSplineSurface.`

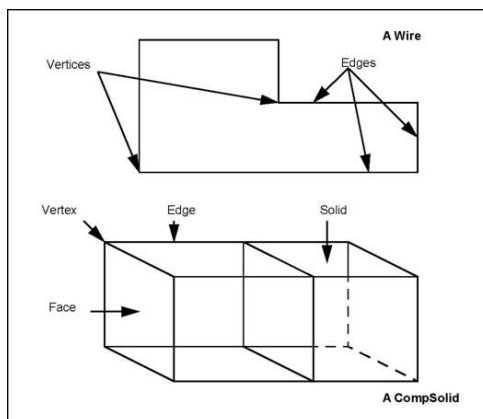
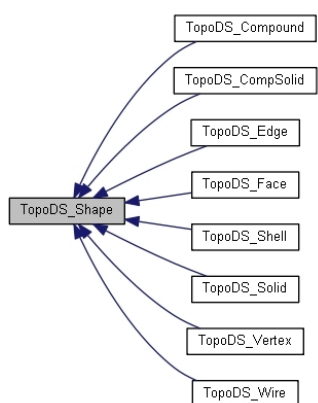
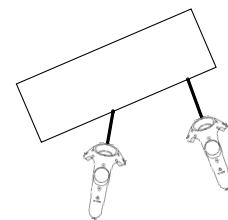
Finally the `Geom2d` package is a collection of objects used in the context of two dimensional geometry. This involves curves, that are lying on surfaces or that are restricted to a certain plane.

In order to be able to relate the representations of topology and geometry to one another, OCCT provides the following Interface:

- Using the `BRepBuilderAPI`, we can create topological descriptions from geometry. `BRepBuilderAPI_MakeEdge` for example allows the creation of `TopoDS_Edge` from a variety of different geometric objects: We can use two `gp_Pnt` objects to create

<sup>4</sup>An axis is a `gp_Ax2` holding an origin and a direction.

<sup>5</sup>knot vector, control points, degree



(a) The inheritance diagram of `TopoDS_Shape`. From [46].

(b) Example shape and components. From [21].

Figure B.1.: **OCCT topological model** – All topological entities root from `TopoDS_Shape` (Figure B.1a). We distinguish entities that are directly related to existing geometry (`TopoDS_Vertex`, `TopoDS_Edge`, `TopoDS_Face`) and those that only provide containers (all other children). Two example shapes and their components are shown in Figure B.1b.

a simple line or provide a generic `Geom_Curve` (for example a `Geom_BSplineCurve`) to create a complex curve.

- The objects provided by the topological model can be transformed to their geometric counterparts using adaptors from the packages `Adaptor3d` and `Adaptor2d`.

### Techniques for creation and modification of geometry

OCCT enables the user to create and design parts with a high level of detail. Therefore, the possibilities for creating and modifying geometry in OCCT are vast. The user can create and manipulate simple shapes like lines and circles or complex free form surfaces, that are defined by a set of boundary conditions. In the following we will review some techniques, that are used in the scope of the thesis.

**Parametric description of geometry** Geometric shapes can be created using the `Geom` and `Geom2d` package simply by providing the necessary parameters for a geometry (see Listing B.3).

With `Geom_BSplineCurve`, `Geom_BSplineSurface` and `Geom2d_BSplineCurve` the `Geom` and `Geom2d` package also provide a powerful interface for *non-uniform rational B-spline* (NURBS) modeling<sup>6</sup>, that supports

- the definition of NURBS using control points, weights, knot vector and degree,

<sup>6</sup>For corresponding theory please refer to section 2.6.

```

double radius = 5;
gp_Pnt origin = gp_Pnt(0, 0, 0);
gp_Dir normal = gp_Dir(0, 0, 1);
gp_Ax2 axis = gp_Ax2(origin, normal);
Handle(Geom_Circle) aCircleGeom = new Geom_Circle(axis, radius);
TopoDS_Edge aCircleTopo = BRepBuilderAPI_MakeEdge(aCircleGeom);

```

Listing B.3.: **Creation of a circle in OCCT** – We create a circle with radius  $r = 5$ , centered at the origin  $\vec{x} = (0, 0, 0)$  and lying in the XY-plane with the normal being  $\vec{n} = (0, 0, 1)$ . Then we convert the circle to a `TopoDS_Edge`.

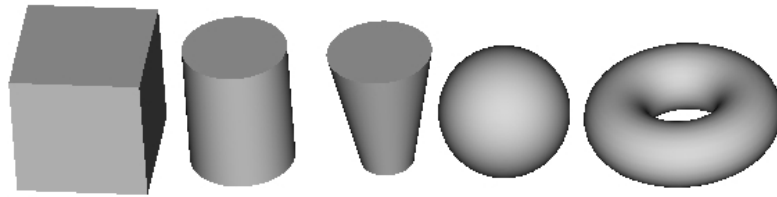


Figure B.2.: **Primitive shapes** – Box, cylinder, cone, sphere, torus (from left to right). Created in FreeCAD [4].

- the evaluation of a NURBS curves or surfaces and their respective derivatives, and
- advanced techniques like knot insertion, degree elevation insertion, curve splitting and the definition of periodic curves and surfaces.

For more types of geometry that are supported by OCCT, please refer to the documentation [46].

**Primitive shapes and boolean operations** Using primitive shapes and boolean operations OCCT can also model geometry following the *constructive solid geometry* (CSG) philosophy. Anyhow, internally the geometry is still represented using BREP and the geometric and topological model, mentioned above, is in use.

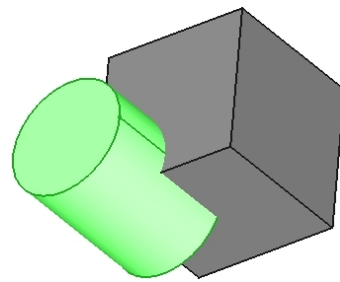
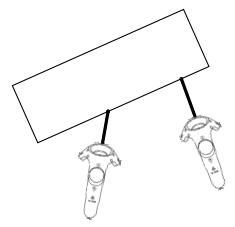
The package `BRepPrimAPI` allows the user to create the basic primitive objects box, cylinder, cone, sphere and torus (see Figure B.2).

Using `BRepAlgoAPI_BooleanOperation` the user performs the boolean operations *fuse*, *common* and *cut* (see Figure B.3). Please note that in general not only boolean operations on primitives, but on arbitrary shapes that belong to the `TopoDS_Shape` family are supported by OCCT. A thorough explanation of the algorithms can be found at [21].

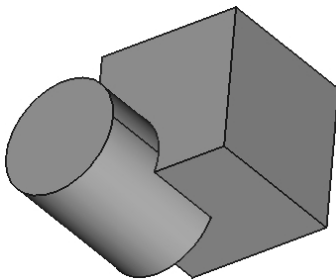
**Transformation of shapes** `BRepBuilderAPI_Transform` realizes isometric transformations, that do not change the basic nature of a geometric object<sup>7</sup>. Here we basically apply a `gp_Trnsf` to a given `TopoDS_Shape`. Using `gp_Trnsf` we translate, rotate and uniformly scale the object.

<sup>7</sup>This means, that a circle always stays a circle.

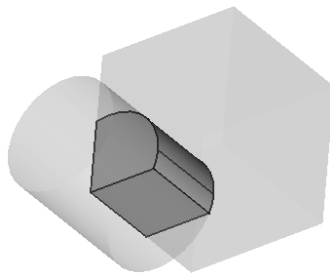




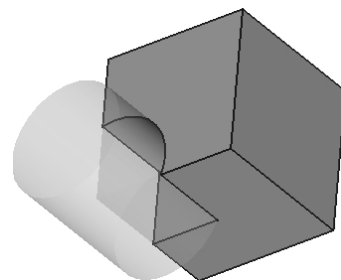
(a) Overlapping Box and Cylinder.



(b)  $\text{Box} \cup \text{Cylinder}$



(c)  $\text{Box} \cap \text{Cylinder}$



(d)  $\text{Box} - \text{Cylinder}$

Figure B.3.: **Boolean operations** – Boolean operations fuse/ $\cup$  (Figure B.3b), common/ $\cap$  (Figure B.3c), cut/ $-$  (Figure B.3d) performed on overlapping Box and Cylinder (Figure B.3a). All figures created using FreeCAD [4].

For non-uniform transformations, like scaling an object in only one direction, we have to use `BRepBuilderAPI_GTransform`. Here we apply the general transformation `gp_GTrsf`<sup>8</sup> to a `TopoDS_Shape`.

**B-Spline approximation of points** OCCT provides tools for the B-Splines approximation of arrays of points. `GeomAPI_PointsToBSpline` and `Geom2dAPI_PointsToBSpline` provide functionality for the approximation of a 1D array of points (a `TColgp_Array1OfPnt`) with a B-Spline curve in 3D and 2D, respectively. `GeomAPI_PointsToBSplineSurface` approximates a 2D grid of points (a `TColgp_Array2OfPnt`) with a B-Spline surface.

These tools provide a high level interface to advanced approximation algorithms. Customization is still possible up to a certain degree: The user can control degree and smoothness of the resulting B-Spline as well as approximation tolerance of the approximation algorithm and parametrization of the input data. Due to the documentation [46] also a variational smoothing algorithm can be applied, that tries to minimize the following criterion:

$$c_1L(\gamma) + c_2\kappa(\gamma) + c_3\tau(\gamma), \quad (\text{B.1})$$

where the weights  $c_i$  can be chosen by the user and  $L, \kappa$  and  $\tau$  denote length, curvature and torsion of the approximating curve  $\gamma$ .

```
int nPts = 40;
TColgp_Array1OfPnt samplePoints(1, nPts);

for (int i = 1; i <= nPts; i += 1)
    samplePoints(i) = gp_Pnt(...);

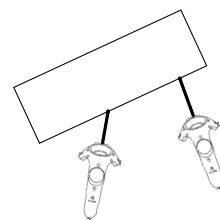
double tolerance = .01;
int minDegree = 2;
int maxDegree = 5;
GeomAbs_Shape smoothness = GeomAbs_C3;

GeomAPI_PointsToBSpline bSplineApproximator(samplePoints, minDegree,
    maxDegree, smoothness, tolerance);
Handle(Geom_BSplineCurve) bSpline = bSplineApproximator.Curve();
```

Listing B.4.: **B-Spline approximation of points** – We first define a `TColgp_Array1OfPnt` and fill it with sample points. Then we approximate the points with a B-Spline using `GeomAPI_PointsToBSpline`

**Lofting** OCCT’s loft interface `BRepOffsetAPI_ThruSections` allows the creation of complex surfaces and solid bodies. A loft is a surface that smoothly connects two or more closed or open curves. In general the curve is represented by a `TopoDS_Wire`;

<sup>8</sup>A general transformation is a linear transformation that is represented by a  $4 \times 4$  matrix. Using a non-uniform transformation we cannot guarantee that geometric entities keep their nature: a circle becomes an ellipse if we scale it non-uniformly.



therefore, the curves do not necessarily have to be constructed from a single segment (a `TopoDS_Edge`).

OCCT provides further possibilities for customization of the loft:

- Instead of providing a `TopoDS_Wire` as the first or last curve, the user can also provide a `TopoDS_Vertex`. This results in a singularity at the beginning or end of the loft.
- A solid object is created from the loft surface, by adding lids at both ends of the loft surface.
- Specifying a tolerance allows to create rather an approximation than an interpolation of the provided curves.

Some example loft surfaces can be found in Figure B.4. A code snippet for creating a loft through a set of wires is given in Listing B.5.

```
bool isSolid = true;
bool isRuled = false;
double tolerance = .01;
BRepOffsetAPI_ThruSections loftBuilder(isSolid, isRuled, tolerance);

for(int i = 0; i < n; i++)
    loftBuilder.AddWire(loftWires[i]);

loftBuilder.SetSmoothing(true);
loftBuilder.CheckCompatibility(true);

TopoDS_Shape loft = loftBuilder.Shape();
```

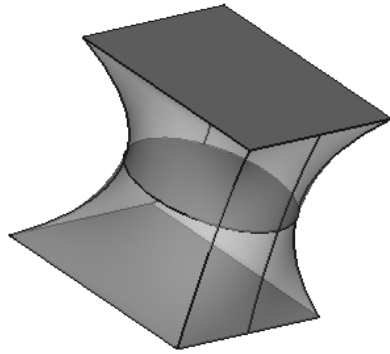
Listing B.5.: **Creation of a loft surface** – First the `BRepOffsetAPI_ThruSections` object is created, then a series of `TopoDS_Wires` is added and finally the loft surface is requested.

**Shape healing and analysis** Using algorithms for shape healing and analysis, we can, for example, detect contours of connected faces, by removing seam edges (see Listing B.6 and Figure B.5). Additionally, we can check existing shapes for consistency and fix errors, like for example edges being part of a wire that are not oriented consistently<sup>9</sup>.

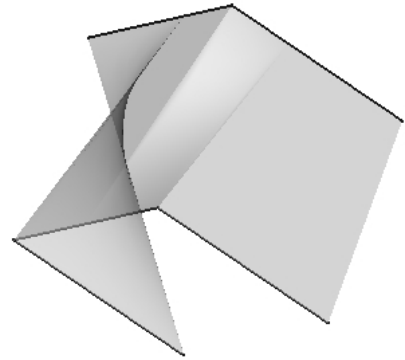
## Data exchange

OCCT not only provides powerful interfaces for creation and interaction with *computer-aided design* (CAD) geometry, but also interfaces for data exchange. This involves, On

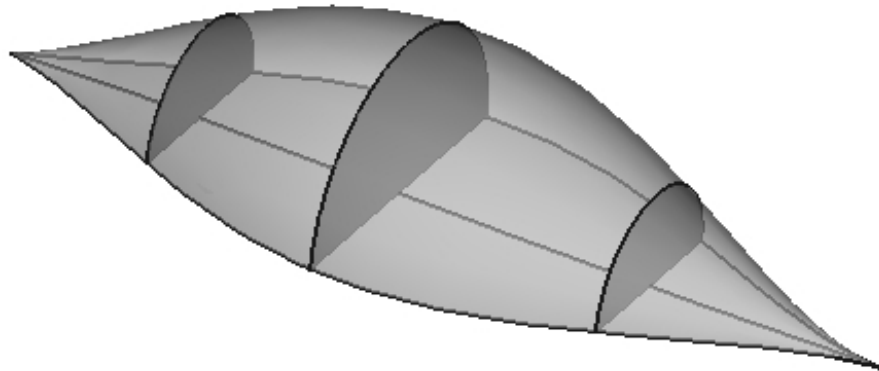
<sup>9</sup>In OCCT a wire is a collection of one or more, connected and consistently oriented edges. Edges have an orientation – start and end of the edge – and have to be oriented, such that the end point of one edge is identical with the starting point of the next one.



(a) Loft through closed the closed surrounding curves of square, circle and square.

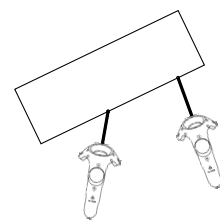


(b) Loft through two open curves build from two and three line segments.



(c) Loft through three half circular arcs and a vertex at the beginning and the end of the loft.

Figure B.4.: **Loft surfaces** – Different shapes created using OCCT's `BRepOffsetAPI.ThruSections`. All figures created using FreeCAD [4].



```
// collect all edges into the analyzer
ShapeExtend_WireData wireAnalyzer;

// several connected faces
TopoDS_Face* faces = ...
int nFaces = ...

// iterate over all faces
for (int i = 0; i<nFaces; i++) {
    TopoDS_Face aFace = faces[i]
    // get contour of face and add it to the analyzer
    for (TopExp_Explorer faceEx(aFace, TopAbs_WIRE); faceEx.More();
         faceEx.Next())
        wireAnalyzer.Add(TopoDS::Wire(faceEx.Current()));
}

// only consider outer edges. Discard seam edges.
Handle(ShapeExtend_WireData) outerEdges = new ShapeExtend_WireData();

for (int edgeId = 1; edgeId <= wireAnalyzer.NbEdges(); edgeId++)
    if (!wireAnalyzer.IsSeam(edgeId)) // discard seam edges
        outerEdges->Add(wireAnalyzer.Edge(edgeId));

// fix order/orientation of edges in wire
ShapeFix_Wire wireFixer;
wireFixer.Load(outerEdges);
wireFixer.FixReorder();

// collect edges
TopTools_ListOfShape edges;
for (int edgeId = 1; edgeId <= outerEdges->NbEdges(); edgeId++)
    edges.Append(outerEdges->Edge(edgeId));

// create Wire
BRepLib_MakeWire wireMaker;
wireMaker.Add(edges);
TopoDS_Wire theContour = wireMaker.Wire();

// do final shape fixing
Handle(ShapeFix_Shape) shapeFixer = new ShapeFix_Shape;
shapeFixer->Init(theContour);
shapeFixer->Perform();
theContour = TopoDS::Wire(shapeFixer->Shape());
```

Listing B.6.: **Shape healing and analysis** – Here we use shape analysis and healing algorithms for first removing seam edges from connected faces using `ShapeExtend_WireData`. Then we fix the orientation of the edges and create a single wire using `ShapeFix_Wire`.

the one hand, meshing of CAD geometry, and, on the other hand, reading and writing of files that comply with the *Standard for The Exchange of Product model data* (.step).

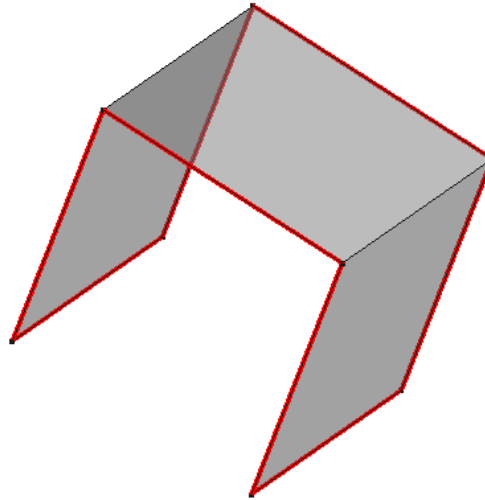


Figure B.5.: **Outer contour of connected faces** – We created the outer contour (red) of connected faces by removing the seam edges and collecting the remaining edges in a single wire.

**Geometry meshing** As soon as we want to display CAD geometry, we first have to mesh it. OCCT provides a framework for the triangulation of arbitrary faces. Therefore, we firstly extract the `TopoDS_Face` Objects from each `TopoDS_Shape` and then generate a triangle mesh for each `TopoDS_Face`.

We can generate a mesh of a shape using `BRepMesh_IncrementalMesh`, where we can also set discretization parameters like the linear and angular deflection (see Figure B.6). Using `TopExp_Explorer` we can iterate over the faces of a shape and access the Triangulation of each face using `BRep_Tool::Triangulation`. Finally we have to transform the vertices of the mesh in order to get their global coordinates (see Listing B.7). The resulting mesh is a triangle mesh, with one array of vertices and another array of vertex indices that define the triangles.

**File input and output** In order to be able to persistently save our CAD designs or read input CAD geometry, we decided to use `.step` files for data exchange. The `.step` file format is a standardized exchange format for industry applications [31, 32]. OCCT provides an interface for reading and writing `.step` files, which comply with this standard.

Using `STEPControl_Writer` we can save `TopoDS_Shapes` objects to `.step` (see Listing B.8). Reading `.step` is done using `STEPControl_Reader` (see Listing B.9).

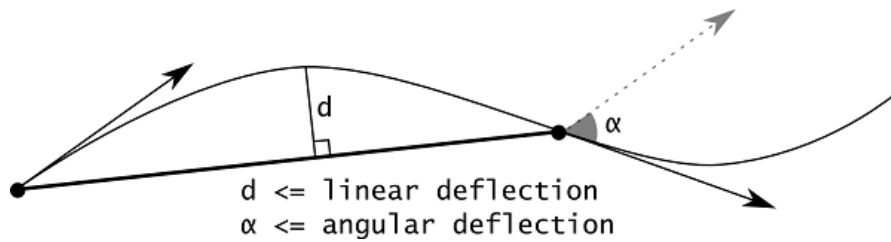
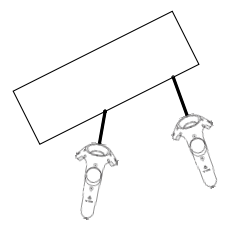


Figure B.6.: **Discretization parameters for meshing** – In the process of meshing we can specify the linear and angular deflection. “Linear deflection limits the distance between a curve and its tessellation, whereas angular deflection limits the angle between subsequent segments in a polyline.” Picture and description from [21].

## B.2. Visualization Toolkit: Tutorial and Examples

In the following section we give a detailed overview of the VTK functionality that is used in the scope of this thesis. For a thorough explanation of VTK functionality please refer to VTK’s userguide [71], the VTK textbook [72] and the reference documentation [73].

**Description of voxel data** With `vtkImageData` VTK provides a datastructure for defining uniform grids of data. Here we have to distinguish between cell data and point data. The dimension and origin of the dataset can be described as well (see Figure B.7 and Listing B.10).

**Dataset interpolation and probing** VTK provides the possibility to resample and probe a dataset. `vtkPointDataToCellData` and `vtkCellDataToPointData` allow us to transform data from point data representation to the corresponding cell data representation and vice versa by using interpolation methods (see Figure B.8 and Listing B.11). Using `vtkProbeFilter` we can sample the dataset in a certain region, like for example along a line, if we use a `vtkLineSource` (see Figure B.9 and Listing B.12).

**Contour extraction** VTK provides an implementation of the marching cubes algorithm [36] for the visualization of isocontours in datasets. Anyhow, many other methods for the visualization of isocontours and surfaces exist [37, 84–86]. Since, the drawbacks of the marching cubes algorithm do not matter in our field of application, we decided to rely on the existing marching cubes implementation of VTK. Using the `vtkContourFilter` we can extract an isosurface at a certain threshold from a dataset consisting of point data. The isosurface is represented as a set of triangles that is saved in a `vtkPolyData` object (see Figure B.10 and Listing B.13).

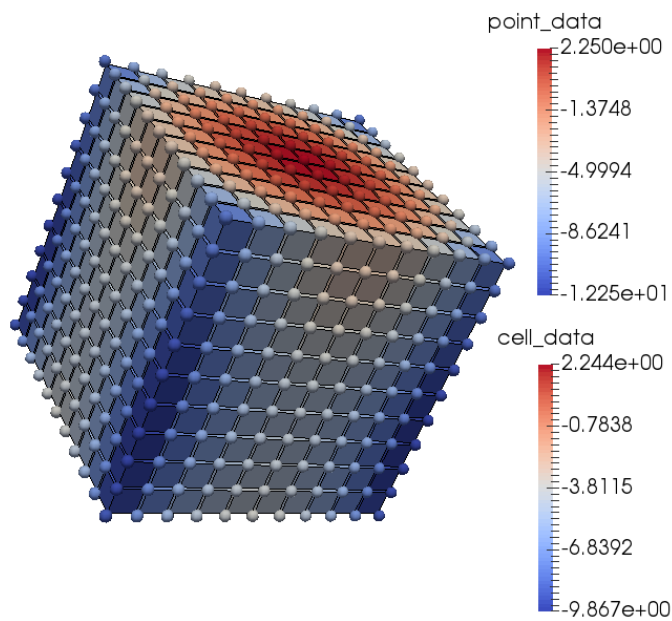


Figure B.7.: **Cell and point data in VTK** – Dataset with cell data (boxes) and point data (spheres), colored by their corresponding value. The dataset has its origin in  $\vec{x} = (0, 0, 0)$  and a size of  $\vec{s} = (5, 5, 5)$ . With a uniform resolution of  $h = 0.5$  this means that there exist 11 point data values and 10 cells in each dimension. This leads to 1331 point data values and 1000 cell data values in total. Figure created using Paraview [40].



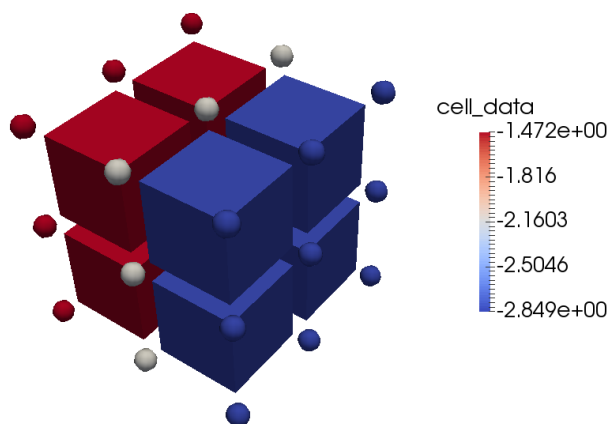
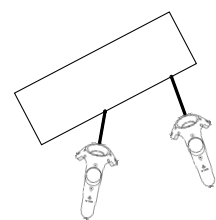


Figure B.8.: **Interpolation of cell data to point data** – The original point data (boxes) is interpolated to the corresponding point data (spheres) on the grid.

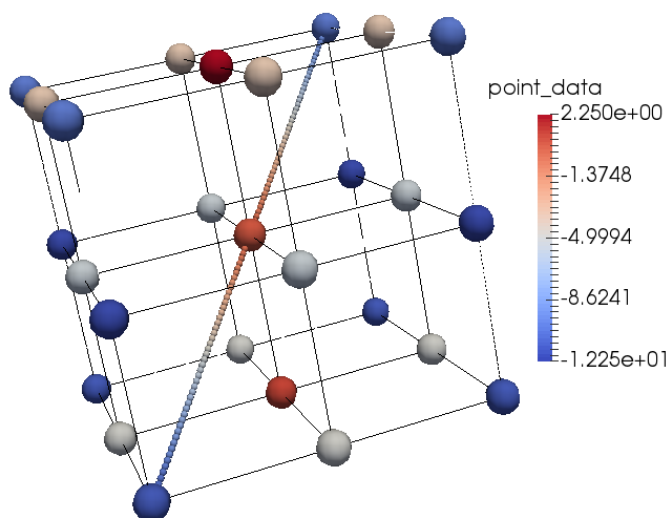


Figure B.9.: **Line probing on dataset** – We sample a coarse point dataset (large spheres) using 101 sampling points (small spheres) on a line from  $\vec{x}_0 = (0, 0, 0)$  to  $\vec{x}_1 = (5, 5, 5)$ .

```

// shape to be triangulated
TopoDS_Shape aShape = ...
// discretization variables
Standard_Real aLinearDeflection = 0.1;
Standard_Boolean isRelative = Standard_False;
Standard_Real anAngularDeflection = 0.5;
// do the triangulation
BRepMesh_IncrementalMesh(aShape, aLinearDeflection, isRelative,
    anAngularDeflection);

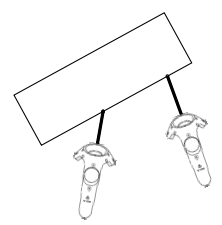
// iterate over the faces
int f = 1;
for (TopExp_Explorer ex(aShape, TopAbs_FACE); ex.More(); ex.Next()) {
    std::cout << "Face " << f << std::endl;
    f++;
    TopoDS_Face aFace = TopoDS::Face(ex.Current());
    TopLoc_Location aLocation;
    // access the triangulation of the face
    Handle(Poly_Triangulation) triangulation = BRep_Tool::Triangulation(
        aFace, aLocation);

    // array holding the vertices
    TColgp_Array1OfPnt vertices(1, (triangulation->NbNodes()));
    vertices = triangulation->Nodes();
    // loop over all vertices and apply the transformation
    for (int v = 1; v <= triangulation->NbNodes(); v++) {
        gp_Pnt aVertex = vertices.Value(v);
        aVertex.Transform(aLocation.Transformation());
        vertices.SetValue(v, aVertex);
        std::cout << "vertex " << v << ": (" << aVertex.X() << "," <<
            aVertex.Y() << "," << aVertex.Z() << ")" << std::endl;
    }
    // array holding the vertex indices of the triangles
    Poly_Array1OfTriangle triangles(1, triangulation->NbTriangles());
    triangles = triangulation->Triangles();
    for (int t = 1; t <= triangulation->NbTriangles(); t++)
    {
        Poly_Triangle aTriangle = triangles.Value(t);
        int N1, N2, N3;
        aTriangle.Get(N1, N2, N3);
        std::cout << "triangle " << t << ": (" << N1 << "," << N2 << "," <<
            << N3 << ")" << std::endl;
    }
}
}

```

Listing B.7.: **Meshing a TopoDS.Shape** – We mesh a TopoDS.Shape by first splitting the shape into its faces, and then creating a triangulation of each face. Discretization parameters can be chosen by the user.

**File input and output** VTK can be used for reading and writing .vtk files (see Listing B.1). They can be visualized using tools like for example Paraview [40] or processed



```
// first we construct a shape
TopoDS_Shape aShape = ...;
// we create a writer
STEPControl_Writer writer;
// we add the shape to the writer
writer.Transfer(aShape, STEPControl_AsIs);
// and finally write a step files
writer.Write("aShape.step");
```

Listing B.8.: **.step writing** – We write an existing `TopoDS_Shape` to a `.step` file.

```
// we create a reader
STEPControl_Reader reader;
// we read the file
IFSelect_ReturnStatus stat = reader.ReadFile("aShape.step");
// we define our Shape
TopoDS_Shape importedShape;
// we check for errors
if (stat == IFSelect_RetDone) {
    // we do some necessary preprocessing
    reader.ClearShapes();
    Standard_Integer NbRoots = reader.NbRootsForTransfer();
    Standard_Integer NbTrans = reader.TransferRoots();
    // we import the shape
    importedShape = reader.OneShape();
}
```

Listing B.9.: **.step reading** – We read an existing `.step` file.

furtherly.

If the data is already described using VTK datastructures (for example `vtkImageData`), the data can be written to a `.vtk` file using the appropriate writer. For datasets of structured points, like the example dataset from above, one has to use `vtkStructuredPointsWriter` (see Listing B.14). Reading of `.vtk` files can be done using the appropriate reader, like for example a `vtkStructuredPointsReader` for a `STRUCTURED_POINTS` `.vtk` file (see Listing B.15).

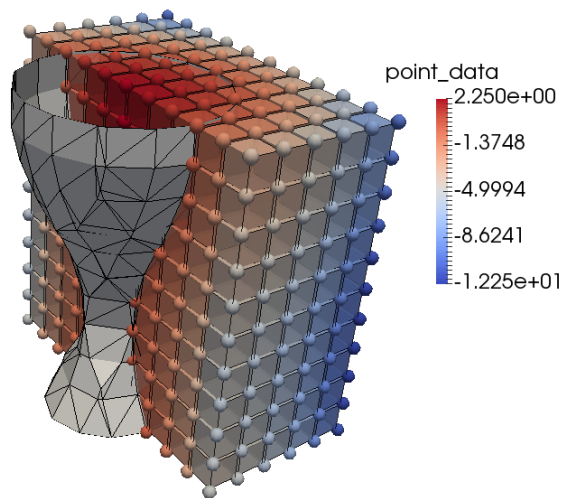
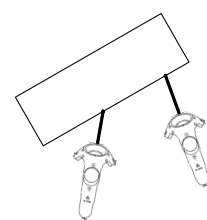


Figure B.10.: **Isosurface inside a dataset** – The isosurface divides the point dataset at the threshold value  $t = 0$ .

```
# vtk DataFile Version 4.0
vtk output
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 11 11 11
SPACING 0.5 0.5 0.5
ORIGIN 0 0 0
CELL_DATA 1000
FIELD FieldData 1
cell_data 1 1000 double
-9.3571 -7.3571 -5.8571 ... -4.37606 -5.87606
POINT_DATA 1331
FIELD FieldData 1
point_data 1 1331 double
-11.5 -9.25 -7.5 ... -6.31119 -8.06119 -10.3112
```

Listing B.1: **Example .vtk file:** A structured grid with cell and point data. The dimensions of the grid, the spacing and the origin are defined first. Then the cell and point data values are given.



```
import vtk
import numpy as np

# grid parameters
dx, dy, dz = 0.5, 0.5, 0.5 # spacing
Lx, Ly, Lz = 5.0, 5.0, 5.0 # sizing
x0, y0, z0 = 0.0, 0.0, 0.0 # origin
nx, ny, nz = int(Lx/dx + 1), int(Ly/dx + 1), int(Lz/dx + 1) # number of
    points

# initialize vtk grid
grid = vtk.vtkImageData()
grid.SetOrigin(x0, y0, z0)
grid.SetSpacing(dx, dy, dz)
grid.SetDimensions(nx, ny, nz)

# add point dataset
point_array = vtk.vtkDoubleArray()
point_array.SetNumberOfComponents(1)
point_array.SetNumberOfTuples(grid.GetNumberOfPoints())
point_array.SetName("point_data")

for i in range(grid.GetNumberOfPoints()):
    x,y,z = grid.GetPoint(i)
    point_array.SetValue(i, -1 * ((x - 2.5)**2 + (y - 2.5)**2 - (1-.5*np.
        sin(z))**2))

grid.GetPointData().AddArray(point_array)

# add cell dataset
cell_array = vtk.vtkDoubleArray()
cell_array.SetNumberOfComponents(1)
cell_array.SetNumberOfTuples(grid.GetNumberOfCells())
cell_array.SetName("cell_data")

# here we set up a filter to compute the cell centers
cellCenters = vtk.vtkCellCenters()
cellCenters.SetInputData(grid)
cellCenters.Update()

for i in range(grid.GetNumberOfCells()):
    x,y,z = cellCenters.GetOutput().GetPoint(i)
    cell_array.SetValue(i, -1 * ((x - 2.5)**2 + (y - 2.5)**2 - (1-.5*np.
        sin(z))**2))

grid.GetCellData().AddArray(cell_array)
```

Listing B.10.: **Creation of a grid with point and cell data:** – We use python VTK to first create a `vtkImageData` object. Then we add `vtkDoubleArray` objects holding samples from an implicit function as point and cell data. For generation of cell data we sample the implicit function at the corresponding cell center.

```

import vtk

grid = ...

#apply cell to point data filter
cellToPoint = vtk.vtkCellDataToPointData()
cellToPoint.SetInputData(grid)
cellToPoint.PassCellDataOn()
cellToPoint.Update()

```

Listing B.11.: **Interpolating a dataset's cell data to point data** – We use `vtkCellDataToPointData` for computing an interpolation of cell values at point coordinates.

```

import vtk

grid = ...

#Set the points between which the line is constructed.
p1=[0,0,0]
p2=[5,5,5]
# number of sampling intervals
numPoints=100

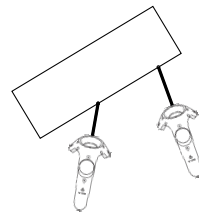
# Create the line
line = vtk.vtkLineSource()
line.SetResolution(numPoints)
line.SetPoint1(p1)
line.SetPoint2(p2)
line.Update()

# sample along the line
probe = vtk.vtkProbeFilter()
probe.SetInputConnection(line.GetOutputPort())
probe.SetSourceData(grid)
probe.Update()

for i in range(numPoints+1):
    thePoint = probe.GetOutput().GetPoint(i)
    theValue = probe.GetOutput().GetPointData().GetArray(0).GetValue(i)
    print "value at "+str(thePoint)+"="+str(theValue)

```

Listing B.12.: **Probing a dataset along a line** – We first create a `vtkLineSource` defining the line along which we want to sample the dataset, then we use the `vtkProbeFilter` for creating the samples from the dataset using interpolation methods. Finally we write the sample points to a file using `vtkPolyDataWriter`.



```
import vtk
import numpy as np

# initialize vtk grid
grid = ...

grid.AllocateScalars(vtk.VTK_DOUBLE, 1)
for z_id in range(grid.GetDimensions()[2]):
    for y_id in range(grid.GetDimensions()[1]):
        for x_id in range(grid.GetDimensions()[0]):
            id = grid.ComputePointId((x_id, y_id, z_id))
            value = grid.GetPointData().GetArray("point_data").GetValue(
                id)
            grid.SetScalarComponentFromDouble(x_id, y_id, z_id, 0, value)

threshold = 0
iso = vtk.vtkContourFilter()
iso.SetInputData(grid)
iso.SetValue(0, threshold)
iso.Update()

poly = iso.GetOutput()
# traverse the points
for pointId in poly.GetNumberOfPoints():
    print "point "+str(p_id)+": "+str(poly.GetPoint(pointId))

# traverse the triangles
polys = poly.GetPolys()
polys.InitTraversal()
pointIndices = vtk.vtkIdList()
while polys.GetNextCell(pointIndices) == 1:
    print "triangle: " + str((pointIndices.GetId(0), pointIndices.GetId
        (1), pointIndices.GetId(2)))
```

Listing B.13.: **Extracting an isosurface** – Using VTK’s `vtkContourFilter` we extract an isosurface for a given threshold from a given point dataset. `vtkContourFilter` outputs `vtkPolyData`, that holds the coordinates of the triangle vertices and the indices of the points that form a triangle.

```

import vtk

grid = vtk.vtkImageData()
...

# write file
writer = vtk.vtkStructuredPointsWriter()
writer.SetInputData(grid)
writer.SetFileName("exampleCellPoint_tiny.vtk")
writer.Write()

```

Listing B.14.: **Writing a .vtk file:** – We write a .vtk file by transferring the `vtkImageData` object `grid` to the writer.

```

import vtk

#read the vtk file as an unstructured grid
reader = vtk.vtkStructuredPointsReader()
reader.SetFileName("exampleCellPoint.vtk")
reader.ReadAllVectorsOn()
reader.ReadAllScalarsOn()
reader.Update()

# obtain the data
grid = reader.GetOutput()

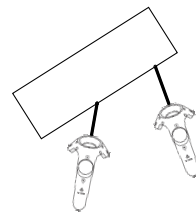
point_data = grid.GetPointData().GetArray("point_data")

for i in range(grid.GetNumberOfPoints()):
    x,y,z = grid.GetPoint(i)
    print "value at"+str((x,y,z))+ "=" +str(point_data.GetValue(i))

```

Listing B.15.: **Reading a .vtk file** – We read a .vtk file and then iterate over all the points and print the corresponding values.





```
import vtk
import numpy as np

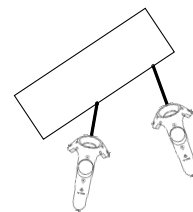
grid = ...

iso = ...

# write file
stl = vtk.vtkSTLWriter()
stl.SetInputConnection(iso.GetOutputPort())
stl.SetFileName("isosurface.stl")
stl.SetFileType(2)
stl.Write()
```

Listing B.16.: **Writing a .stl file** – We can write a `.stl` file with geometry information, by forwarding the output of, for example, the `vtkContourFilter` to the `vtkSTLWriter`.

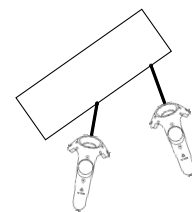




## Bibliography

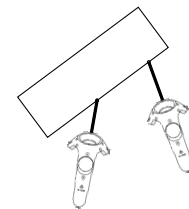
- [1] Siemens Product Lifecycle Management Software Inc. *Siemens NX*. 1973. URL: <https://www.plm.automation.siemens.com/en/products/nx/> (visited on Jan. 10, 2017).
- [2] Autodesk Inc. *AutoCAD*. 1982. URL: <http://www.autodesk.com/products/autocad/overview> (visited on Jan. 6, 2017).
- [3] Onshape Inc. *Onshape*. 2014. URL: <https://www.onshape.com/> (visited on Jan. 6, 2017).
- [4] FreeCAD. *FreeCAD*. 2009. URL: <http://freecadweb.org/> (visited on Jan. 6, 2017).
- [5] M M M Sarcar, K Mallikarjuna Rao, and K Lalit Narayan. *Computer Aided Design and Manufacturing*. PHI Learning Pvt. Ltd., 2008. ISBN: 9788120333420.
- [6] Autodesk Inc. *Tinkercad*. 2017. URL: <https://www.tinkercad.com/> (visited on Feb. 2, 2017).
- [7] Tracy Woo. *Maximizing Product Design in a Complex Manufacturing Environment*. Tech. rep. Aberdeen Group, 2016.
- [8] HTC Corporation. *HTCVive*. 2016. URL: <https://www.vive.com/us/> (visited on Nov. 26, 2016).
- [9] Oculus VR LLC. *Oculus*. 2016. URL: <https://www.oculus.com/> (visited on Dec. 11, 2016).
- [10] Simon Sua et al. “Virtual reality enabled scientific visualization workflow”. In: *2015 IEEE 1st Workshop on Everyday Virtual Reality, WEVR 2015* (2015), pp. 29–32. DOI: 10.1109/WEVR.2015.7151692.
- [11] Google Inc. *Tiltbrush*. 2017. URL: <https://www.tiltbrush.com/> (visited on Feb. 2, 2017).
- [12] Unity Technology. *Unity Editor VR*. 2016. URL: <https://blogs.unity3d.com/2016/12/15/editorvr-experimental-build-available-today/> (visited on Feb. 2, 2017).
- [13] Epic Games Inc. *Unreal Engine VR Editor*. 2017. URL: <https://docs.unrealengine.com/latest/INT/Engine/Editor/VR/> (visited on Feb. 2, 2017).
- [14] Kitware Inc. *Using Virtual Reality Devices with VTK*. 2016. URL: <https://blog.kitware.com/using-virtual-reality-devices-with-vtk/> (visited on Feb. 2, 2017).

- [15] Mindesk Inc. *Mindesk*. 2016. URL: <http://www.mindeskvr.com> (visited on Feb. 2, 2017).
- [16] Autodesk Inc. *VRED*. 2017. URL: <http://www.autodesk.com/products/vred> (visited on Feb. 2, 2017).
- [17] TechViz. *TechViz XL*. 2017. URL: <http://www.techviz.net/techviz-xl> (visited on Feb. 2, 2017).
- [18] ESI Group. *IC.IDO*. 2017. URL: <http://virtualreality.esi-group.com/> (visited on Feb. 2, 2017).
- [19] FreeCAD. *FreeCAD release notes 0.15*. 2015. URL: [http://freecadweb.org/wiki/Release\\_notes\\_015](http://freecadweb.org/wiki/Release_notes_015) (visited on Feb. 27, 2017).
- [20] Alexandre Millette and Michael J McGuffin. “DualCAD: Integrating Augmented Reality with a Desktop GUI and Smartphone Interaction”. In: *2016 IEEE International Symposium on Mixed and Augmented Reality Adjunct Proceedings*. 2016, pp. 21–26. ISBN: 9781509037407. DOI: 10.1109/ISMAR-Adjunct.2016.23.
- [21] OPEN CASCADE S.A.S. *OpenCASCADE Technology 7.0.0*. 2016. URL: <https://www.opencascade.com/doc/occt-7.0.0/overview/html/index.html> (visited on Nov. 7, 2016).
- [22] Unity Technology. *Unity*. URL: <https://unity3d.com/de/> (visited on Nov. 26, 2016).
- [23] Matthias Eck and Hugues Hoppe. *Automatic reconstruction of B-spline surfaces of arbitrary topological type*. 1996, pp. 325–334.
- [24] Saumitra Joshi et al. *CAD-integrated Topology Optimization*. Tech. rep. München: Technische Universität München, 2016.
- [25] Kitware. *VTK*. 2015. URL: <http://www.vtk.org/> (visited on Feb. 1, 2017).
- [26] Gavranovic Stefan. “Topology Optimization using GPGPU”. Master Thesis. Technische Universität München, 2015.
- [27] U.S. Department of Health and Human Services. *System usability scale*. 2017. URL: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html> (visited on Feb. 10, 2017).
- [28] General Electric and GrabCAD. *GE jet engine bracket challenge*. 2013. URL: <https://grabcad.com/challenges/ge-jet-engine-bracket-challenge> (visited on Feb. 25, 2017).
- [29] Generate and GrabCAD. *The Generate Quadcopter Challenge*. 2016. URL: <https://grabcad.com/challenges/the-generate-quadcopter-challenge> (visited on Feb. 25, 2017).
- [30] Roland Wüchner, Michael Breitenberger, and Anna Bauer. *Isogeometric structural analysis and design*. München: Chair of Structural Analysis, Technical University of Munich, 2016.



- [31] STEP Tools Inc. *TheSTEPStandard*. 2017. URL: <http://www.steptools.com/stds/step/> (visited on Feb. 1, 2017).
- [32] *ISO10303-242:2014 - Industrial automation systems and integration - Product data representation and exchange - Part 242: Application protocol: Managed model-based 3D design*. Standard. Geneva, Switzerland: International Organization for Standardization, 2014.
- [33] M. Breitenberger et al. “Analysis in computer aided design: Nonlinear isogeometric B-Rep analysis of shell structures”. In: *Computer Methods in Applied Mechanics and Engineering* 284 (2015), pp. 401–457. ISSN: 00457825. DOI: 10.1016/j.cma.2014.09.033.
- [34] Gregory Nielson. “Chord Length (Motivated) Parametrization of Marching Cubes IsoSurfaces”. In: *Geometric Modeling and Processing* (2004).
- [35] Wikipedia. *Voxel @ en.wikipedia.org*. 2017. URL: <https://en.wikipedia.org/wiki/Voxel> (visited on Feb. 9, 2017).
- [36] Timothy S. Newman and Hong Yi. “A survey of the marching cubes algorithm”. In: *Computers and Graphics* 30.5 (2006), pp. 854–879. ISSN: 00978493. DOI: 10.1016/j.cag.2006.07.021.
- [37] Scott Schaefer and Joe Warren. “Dual Contouring:” The Secret Sauce””. In: *Cite-seer* (2002), p. 5. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.2631>.
- [38] Wenping Wang, Helmut Pottmann, and Yang Liu. “Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization Fitting B-Spline Curves by SDM”. In: *ACM Transactions on Graphics* 25.2 (2006), pp. 214–238. ISSN: 07300301. DOI: 10.1145/1138450.1138453.
- [39] Shiaofen Fang and Hongsheng Chen. “Hardware accelerated voxelization”. In: *Computers and Graphics (Pergamon)* 24.3 (2000), pp. 433–442. ISSN: 00978493. DOI: 10.1016/S0097-8493(00)00038-8.
- [40] Kitware. *Paraview*. 2015. URL: <http://www.paraview.org/> (visited on Feb. 1, 2017).
- [41] Wikimedia Commons. *Vector Graphics Tutorial*. URL: [https://commons.wikimedia.org/wiki/Help:Vector\\_graphics\\_tutorial](https://commons.wikimedia.org/wiki/Help:Vector_graphics_tutorial) (visited on Feb. 9, 2017).
- [42] solidThinking Inc. *Inspire2016*. URL: <http://www.solidthinking.com/Inspire2016.html> (visited on Nov. 15, 2016).
- [43] Blender Foundation. *Blender*. 2017. URL: <https://www.blender.org/> (visited on Feb. 1, 2017).
- [44] Siemens Product Lifecycle Management Software Inc. *Parasolid*. URL: <https://www.plm.automation.siemens.com/en/products/open/parasolid/> (visited on Jan. 6, 2017).
- [45] Spatial Corp. *ACIS*. URL: <https://www.spatial.com/products/3d-acis-modeling> (visited on Jan. 6, 2017).

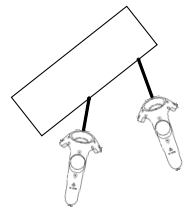
- [46] OPEN CASCADE S.A.S. *Open Cascade Technology 7.0.0 Reference Documentation*. 2016. URL: <https://www.opencascade.com/doc/occt-7.0.0/refman/html/index.html> (visited on Jan. 10, 2017).
- [47] Roman Lyngin. *Open Cascade Notes*. 2008. URL: <http://opencascade.blogspot.de/> (visited on Jan. 10, 2017).
- [48] Python Software Foundation. *Python*. 2001. URL: <https://www.python.org> (visited on Jan. 10, 2017).
- [49] Kongsberg Gruppen. *Coin3D*. 2014. URL: <https://bitbucket.org/Coin3D/coin/wiki/Home> (visited on Jan. 10, 2017).
- [50] The Qt Company. *Qt*. 2017. URL: <https://www.qt.io/> (visited on Jan. 10, 2017).
- [51] Carolina Cruz-Neira et al. “The CAVE: Audio Visual Experience Automatic Virtual Environment”. In: *Commun. ACM* 35.6 (1992), pp. 64–72. ISSN: 0001-0782. DOI: 10.1145/129888.129892. URL: <http://doi.acm.org/10.1145/129888.129892>.
- [52] Google Inc. *Cardboard*. 2017. URL: <https://vr.google.com/cardboard/> (visited on Feb. 2, 2017).
- [53] Google Inc. *Daydream*. URL: <https://vr.google.com/daydream/> (visited on Dec. 11, 2016).
- [54] Sony. *Playstation VR*. 2017. URL: <https://www.playstation.com/en-us/explore/playstation-vr/> (visited on Feb. 12, 2017).
- [55] Leap Motion Inc. *Leap Motion VR*. 2017. URL: <https://www.leapmotion.com/#112> (visited on Feb. 27, 2017).
- [56] Microsoft. *Microsoft HoloLens*. 2017. URL: <https://www.microsoft.com/microsoft-hololens/en-us> (visited on Feb. 12, 2017).
- [57] Niantic Inc. *Pokemon GO*. 2017. URL: <http://pokemongo.nianticlabs.com/en/> (visited on Feb. 12, 2017).
- [58] R. Gleasure and J. Feller. “A rift in the ground: Theorizing the evolution of anchor values in crowdfunding communities through the oculus rift case study”. In: *Journal of the Association of Information Systems* 17.10 (2016), pp. 708–736. ISSN: 15583457 15369323.
- [59] Valve Corporation. *SteamVR*. URL: <http://store.steampowered.com/steamvr> (visited on Nov. 26, 2016).
- [60] Oculus VR LLC. *Utilities for Unity 5.x Developer Guide*. 2016. URL: <https://developer3.oculus.com/documentation/game-engines/latest/concepts/book-unity/> (visited on Dec. 11, 2016).
- [61] Valve Corporation. *SteamVR Asset*. URL: <https://www.assetstore.unity3d.com/en/content/32647> (visited on Nov. 26, 2016).
- [62] Google Inc. *Get Started with the Google VR SDK for Unity on Android*. 2016. URL: <https://developers.google.com/vr/unity/> (visited on Dec. 11, 2016).



- [63] Sysdia Solutions Ltd. *VRTK - SteamVR Unity Toolkit*. URL: <https://www.assetstore.unity3d.com/en/content/64131> (visited on Dec. 4, 2016).
- [64] Albert Hwang. *Focal Point and SteamVR Adapter*. URL: <https://www.assetstore.unity3d.com/en/content/59625> (visited on Dec. 4, 2016).
- [65] Frustum Inc. *Frustum*. 2016. URL: <https://www.frustum.com/> (visited on Feb. 8, 2017).
- [66] Topology Optimization research group at DTU Mechanical Engineering and DTU Mathematics. *TopOpt DTU*. 2009. URL: <http://www.topopt.dtu.dk/> (visited on Feb. 8, 2017).
- [67] William Hunter. *ToPy - 2D and 3D Topology Optimization using Python*. 2009. URL: <https://github.com/williamhunter/topy> (visited on Feb. 8, 2017).
- [68] M P Bendsoe and O Sigmund. *Topology Optimization: Theory, Methods, and Applications*. Springer Berlin Heidelberg, 2013. ISBN: 9783662050866. URL: <https://books.google.de/books?id=ZCjsCAAAQBAJ>.
- [69] Niels Aage et al. *Interactive topology optimization on hand-held devices*. Tech. rep. Lyngby: Department of Informatics and Mathematical Modelling, Technical University of Denmark, 2012, pp. 3–4.
- [70] James Albert Sethian. *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Vol. 3. Cambridge university press, 1999.
- [71] Kitware Inc. *The VTK User's Guide*. 2010, p. 536. ISBN: 978-1-930934-23-8.
- [72] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit*. 2006. ISBN: 978-1-930934-19-1.
- [73] Kitware Inc. *VTK Documentation*. 2017. URL: <http://www.vtk.org/documentation/> (visited on Jan. 30, 2017).
- [74] J Hoschek and D Lasser. “Grundlagen der geometrischen Datenverarbeitung”. In: *BG Teubner, Stuttgart* (1989).
- [75] M Arie Kurniawan. *GE Jet Engine Bracket Version 1.2*. 2013. URL: <https://grabcad.com/library/m-kurniawan-ge-jet-engine-bracket-version-1-2-1> (visited on Feb. 25, 2017).
- [76] Valve Corporation. *The Lab*. 2016. URL: <http://store.steampowered.com/app/450390/>.
- [77] Manuel Biedermann. “Simulation-driven design for additive manufacturing”. Master Thesis. Technische Universität München, 2017.
- [78] Tracy Woo. *The Democratization of Simulation in a Multiphysics World*. Tech. rep. Aberdeen Group, 2016. DOI: [Article](https://doi.org/10.1145/258549.258778).
- [79] Ravin Balakrishnan et al. “The Rockin’Mouse: Integral 3D Manipulation on a Plane”. In: *Proceedings ACM SIGCHI Conference on Human Factors in Computing Systems* (1997), pp. 311–318. DOI: [10.1145/258549.258778](https://doi.org/10.1145/258549.258778).

- [80] Jeffrey S Pierce et al. “Voodoo Dolls : Seamless Interaction at Multiple Scales in Virtual Environments in Virtual Environments”. In: *CMU-HCI Institute* (1999), pp. 141–145. DOI: 10.1145/300523.300540.
- [81] Hrvoje Benko and Steven Feiner. “Balloon selection: A multi-finger technique for accurate low-fatigue 3D selection”. In: *IEEE Symposium on 3D User Interfaces 2007 - Proceedings, 3DUI 2007* (2007), pp. 79–86. DOI: 10.1109/3DUI.2007.340778.
- [82] Ravikanth Malladi and James a. Sethian. “Image processing via level set curvature flow.” In: *Proceedings of the National Academy of Sciences of the United States of America* 92.15 (1995), pp. 7046–50. ISSN: 0027-8424. DOI: 10.1073/pnas.92.15.7046. URL: <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=41468&tool=pmcentrez&rendertype=abstract>.
- [83] Ian Stroud. *Boundary representation modelling techniques*. Springer Science and Business Media, 2006.
- [84] Tao Ju et al. “Dual contouring of hermite data”. In: *ACM Transactions on Graphics*. Vol. 21. 3. 2002, pp. 339–346. ISBN: 1581135211. DOI: 10.1145/566654.566586.
- [85] Scott Schaefer, Tao Ju, and Joe Warren. “Manifold Dual Contouring”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.3 (2007), pp. 610–619. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.1012.
- [86] Chien Chang Ho et al. “Cubical marching squares: Adaptive feature preserving surface extraction from volume data”. In: *Computer Graphics Forum* 24.3 (2005), pp. 537–545. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2005.00879.x.

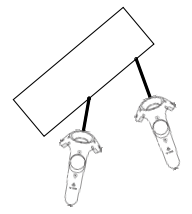




# List of Figures

1.1. Overview over CADinVR with SurfFitX . . . . .	5
2.1. Two geometric descriptions of a circle . . . . .	8
2.2. Representation of a simple shape with CSG and BREP . . . . .	9
2.3. Approximative geometry representation . . . . .	10
2.4. Conversion of geometry representations . . . . .	11
2.5. OCCT technical overview . . . . .	13
2.6. The Vive Controller . . . . .	15
2.7. Unity game object . . . . .	17
2.8. Screenshot from Interactive 2D TopOpt App . . . . .	20
3.1. GE jet engine bracket designs . . . . .	27
3.2. Design workflow realized in this thesis . . . . .	29
3.3. IDeAs input geometry . . . . .	31
3.4. IDeAs output . . . . .	32
3.5. Transformation to level sets . . . . .	35
3.6. Boolean operations on level sets . . . . .	36
3.7. Level set smoothing . . . . .	37
3.8. Consistent level set . . . . .	38
3.9. Toolset for geometry reconstruction implemented in SurfFitX . . . . .	39
3.10. Contour extraction . . . . .	41
3.11. Loft from three contours . . . . .	43
3.12. Loft connecting extracted contours and existing geometry . . . . .	43
3.13. B-spline sketching on mesh . . . . .	44
4.1. Hovering and selection . . . . .	46
4.2. Schematic drawing of translation and rotation . . . . .	48
4.3. Transformation of far away objects . . . . .	48
4.4. Schematic drawing of scaling . . . . .	49
5.1. Screenshots from the frontend application . . . . .	52
5.2. The menu . . . . .	53
5.3. Frontend requesting a mesh from backend . . . . .	55
6.1. Schematic overview over task . . . . .	58
6.2. Reconstructed part of the GEBracket . . . . .	58
6.3. "Lofting" workbench in customized FreeCAD . . . . .	59

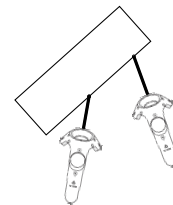
7.1. Model of the Quadcopter . . . . .	62
7.2. Topology optimized quadcopter . . . . .	63
7.3. Level set smoothing . . . . .	64
7.4. Classification of participants with respect to their performance in the both systems . . . . .	65
7.5. Comparison of time needed for teaching in SurfFitX and in FreeCAD . . .	67
7.6. Success in part one . . . . .	67
7.7. Comparison of the time needed for the second part of the task . . . . .	67
7.8. Comparison of time usage in part one . . . . .	68
7.9. Comparison of the SUS scores in a boxplot . . . . .	69
7.10. Average ratings of the SUS items . . . . .	70
7.11. The interface between 2D sketches and 3D model . . . . .	76
7.12. Concept created in VR . . . . .	76
8.1. Hybrid CAD system . . . . .	82
B.1. OCCT topological model . . . . .	101
B.2. Primitive shapes . . . . .	102
B.3. Boolean operations . . . . .	103
B.4. Loft surfaces . . . . .	106
B.5. Outer contour of connected faces . . . . .	108
B.6. Discretization parameters for meshing . . . . .	109
B.7. Cell and point data in VTK . . . . .	110
B.8. Interpolation of cell data to point data . . . . .	111
B.9. Line probing on dataset . . . . .	111
B.10. Isosurface inside a dataset . . . . .	114



## List of Tables

1.1. Summary of different design tools . . . . .	2
7.1. The ten SUS items. . . . .	70
8.1. Comparison of two systems . . . . .	81





## List of Listings

B.1. Iterating over a shape . . . . .	100
B.2. Exploring a shape . . . . .	100
B.3. Creation of a circle in OCCT . . . . .	102
B.4. B-Spline approximation of points . . . . .	104
B.5. Creation of a loft surface . . . . .	105
B.6. Shape healing and analysis . . . . .	107
B.7. Meshing a <code>TopoDS_Shape</code> . . . . .	112
B.8. <code>.step</code> writing . . . . .	113
B.9. <code>.step</code> reading . . . . .	113
B.10. Creation of a grid with point and cell data: . . . . .	115
B.11. Interpolating a dataset's cell data to point data . . . . .	116
B.12. Probing a dataset along a line . . . . .	116
B.13. Extracting an isosurface . . . . .	117
B.14. Writing a <code>.vtk</code> file: . . . . .	118
B.15. Reading a <code>.vtk</code> file . . . . .	118
B.16. Writing a <code>.stl</code> file . . . . .	119

## Acronyms

**AR** augmented reality

**BREP** boundary representation

**CAD** computer-aided design

**CAE** computer-aided engineering

**CSG** constructive solid geometry

**FEM** finite element method

**FMM** fast marching method

**GO** game object

**GUI** graphical user interface

**HMD** head-mounted display

**HUD** head-up display

**IDeAs** Interactive Design Assistant

**IGA** isogeometric analysis

**NURBS** non-uniform rational B-spline

**OCCT** Open CASCADE Technology

**SDF** signed distance function

**SIMP** Solid Isotropic Microstructure with Penalization

**.step** Standard for The Exchange of Product model data

**.stl** stereolithography

**SUS** System Usability Scale

**UEx** user experience

**VR** virtual reality

**VR** Virtuellen Realität

**SurfFitX** Virtual Reality Surface Fitting Extension

**VTK** Visualization Toolkit