FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interactive 3D Dense Surface Exploration in Immersive Virtual Reality

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Christian Kößler
Matrikelnummer 00928004

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag. Dr. Hannes Kaufmann
Mitwirkung: Dipl.-Ing. Dr. Annette Mossel

Wien, 28. Jänner 2021

Christian Kößler	Hannes Kaufmann

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interactive 3D Dense Surface Exploration in Immersive Virtual Reality

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Christian Kößler
Registration Number 00928004

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Univ.Prof. Mag. Dr. Hannes Kaufmann
Assistance: Dipl.-Ing. Dr. Annette Mossel

Vienna, 28th January, 2021

Christian Kößler	Hannes Kaufmann

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Christian Kößler
Otterweg 15/3/3, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 28. Jänner 2021

Christian Kößler

Acknowledgements

Ich bedanke mich bei meinen Betreuern Hannes Kaufmann und Annette Mossel, die mir ermöglicht haben diese Diplomarbeit an der Interactive Media Systems Group auf der TU Wien zu verfassen. Ganz besonderer Dank gebührt hierbei Annette Mossel, die wegen der mehrjährigen Fertigstellungsdauer viel Geduld aufbringen musste und dennoch die Betreuung nicht abgebrochen hat.

Ich danke allen User Study-Teilnehmern, dass sie sich die Zeit zum Testen genommen haben.

Weiters bedanke ich mich speziell bei meiner Lebensgefährtin Janine, die mich über die ganze Zeit hinweg bedingungslos unterstützt und mich mehrmals vor dem Aufgeben bewahrt hat.

Außerdem bedanke ich mich bei meinen Kindern Jonathan und Josephine, die mich in den letzten Wochen der Fertigstellung täglich ermahnt haben, dass ich jetzt weiterschreiben sollte.

Zudem möchte ich mich noch besonders bei meinen Eltern Elisabeth und Gerhard bedanken, die mir mit ihrer Unterstützung das Studium überhaupt erst ermöglicht haben.

Kurzfassung

Detaillierte 3D-Rekonstruktionen real existierender Umgebungen werden benutzt, um Aufgaben der realen Welt (beispielsweise Ferninspektionen) zu lösen. Dabei ist nicht nur das Betrachten von Szenen erforderlich, sondern auch die Fähigkeit, mit der Umgebung zu interagieren, zum Beispiel die Selektion eines benutzerdefinierten Teilbereichs der Rekonstruktion für die spätere Verwendung. Problematisch bei großen 3D-Rekonstruktionen kann sein, dass Objekte aufgrund von Szenengeometrie oder Rekonstruktionsartefakten ganz oder teilweise verdeckt werden. Da dem aktuellen Stand der Technik Ansätze für das Behandeln von Verdeckungen in Umgebungen fehlt, die aus einer oder mehreren (großen) kontinuierlichen Oberflächen bestehen, schlagen wir die neuartige Technik *Large Scale Cut Plane* vor, die eine Segmentierung und anschließende Selektion von sichtbaren, teilweise oder vollständig eingeschlossenen Teilbereichen einer großen 3D-Rekonstruktion von einem weiter entfernten Standpunkt ermöglicht. Ein immersives Virtual-Reality-Setup, bestehend aus einem Head-Mounted-Display, einem Fortbewegungsgerät (omnidirektionales Laufband) und einem 6DOF-Hand-Tracking-Gerät werden mit der *Large Scale Cut Plane*-Technik kombiniert, um das Verständnis von 3D-Szenen und natürlichen Benutzerinteraktionen mit diesen zu fördern. Wir präsentieren außerdem Ergebnisse aus einer AnwenderInnenstudie, in der wir die Leistung und Verwendbarkeit unserer vorgeschlagenen Technik im Vergleich zu einer Basis-Technik untersuchen. Unsere Ergebnisse deuten darauf hin, dass die Cut-Plane Technik für große Umgebungen hinsichtlich Geschwindigkeit und Präzision der Basistechnik überlegen ist, während eine Verbesserung der Benutzeroberfläche notwendig ist.

Abstract

Dense 3D reconstructions of real-world environments become wide spread and are foreseen to act as data base to solve real world problems, such as remote inspections. Therefore not only scene viewing is required but also the ability to interact with the environment, such as selection of a user-defined part of the reconstruction for later usage. However, inter-object occlusion is inherent to large dense 3D reconstructions, due to scene geometry or reconstruction artifacts that might result in object containment. Since prior art lacks approaches for occlusion management in environments that consist of one or multiple (large) continuous surfaces, we propose the novel technique *Large Scale Cut Plane* that enables segmentation and subsequent selection of visible, partly or fully occluded patches within a large 3D reconstruction, even at far distance. An immersive Virtual reality setup consisting of a Head-Mounted Display, a locomotion device (omni-directional treadmill) and a 6DOF-hand-tracking device are combined with the Large Scale Cut Plane technique to foster 3D scene understanding and natural user interactions. We furthermore present results from a user study where we investigate performance and usability of our proposed technique compared to a baseline technique. Our results indicate Large Scale Cut Plane to be superior in terms of speed and precision, while we found need of improvement of the user interface.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

Recent advances in virtual reality (VR) hardware have made it feasible to create immersive virtual reality environments, where a user is able to naturally walk into a virtual world and to interact with it. There exist numerous input devices like motion trackers and Omnidirectional Treadmills (ODTs), as well as VR-output devices like head mounted displays (HMDs). Those devices can be combined with the power of high-performance computers to generate realistic and immersive VR-environments, such that a user is able to explore a part of the real world without physically visiting that place and overcoming limitations which would be subject to a real exploration. The computing power of modern VR-hardware enables not only the visualization of manually created 3D worlds, but also makes it possible to visually represent arbitrary computer-generated worlds, e.g. Dense 3D Surface Reconstructions.

The creation of such dense 3D reconstructions of large real-world environments have been subject of intensive research during the last years due to the emerge of commodity hardware – such as Microsoft Kinect[Mic13] – and powerful structure from motion algorithms [RM12, NZIS13]. Both research and industry areas see a tremendous promise and benefit in using these models. Especially dense reconstructions of indoor environments can serve as core data for future real-world problem solutions, such as 3D scene understanding for inspections, simulation and training within realistic environments or search and rescue. Therefore, not only exploration is required but also user interaction with the reconstructed environment, such as highlighting areas of interest or selecting arbitrary parts of the reconstructed model (*target patches*) for subsequent manipulation. However, when an observer explores such a large reconstructed environment, the geometrical properties of the environment cause occlusion of patches, while in addition reconstruction errors can result in fully occluded and inaccessible patches, independent from viewport changes of the observer. If important target patches are hidden from view, correctness

and efficiency of target selection will suffer. To provide sufficient target selection despite the mentioned occlusion issues a proper strategy – called occlusion management – has to be chosen.

Such occlusion management strategies have been tackled by prior art introducing a range of different approaches [ET08], such as *Multiple Viewports*, *Virtual X-Ray* or *Volumetric Probes*. While these approaches aim at occlusion management in 3D environments that consist of individual objects and tackle the environmental properties object interaction, density and complexity, they are impossible to apply (i.e. volumetric probes) or are not sufficient (i.e. virtual x-rays) for large dense 3D reconstructions since those are comprised of dense 3D point clouds (a disconnected set of points in 3D space) or dense continuous surfaces (3D meshes, a connected set of triangles in the 3D space). Furthermore, for target patch selection in a 3D reconstruction, segmentation of the environment is a pre-requisite as it defines which vertices should be extracted from the 3D reconstruction. Automatic segmentation approaches, such as [ZCW+03, ITHG12, NL13] can subdivide a 3D reconstruction by relying on its distinct geometric properties which mitigates specific segmentation needs. Since they usually incorporate the entire model for their calculations, they result in increased processing time, especially for large scale reconstructions. To enable more specific segmentations, approaches such as [VTHLB15] can be employed that perform the segmentation from a pre-defined starting point. However, these approaches require the target patch to be visible to the observer. In case of (partly) occluded target patches, existing segmentation techniques usually try to solve this problem by enabling users to navigate to the area of interest using mouse and keyboard-based interaction techniques. However, this approach might result in a time consuming search task and does only hold true in case those parts of interest are not occluded by other geometry, such as reconstruction artifacts.

1.2 Motivation

To overcome limitations of prior work in occlusion management and segmentation, both regarding its applicability to large 3D surface reconstructions is the motivation of this thesis. We explore a new interaction technique that provides means for target discovery and access in large 3D reconstructions while preserving the spatial relation between the target and its context. We combine our new technique – *Large Scale Cut Plane* – with an Immersive Virtual Reality (VR) setup to foster spatial 3D scene understanding – through stereoscopic and egocentric viewing – and enable natural human computer interaction through means of natural walking and gestural input. Thereby, *Large Scale Cut Plane* enables segmentation and selection of visible, partly or fully occluded patches within a large 3D reconstruction, even at far distance from the user's current view point. The proposed technique comprises two phases, as shown in Figure 1.1. In an indication step, users cast a vertical virtual plane from their current view port through the 3D reconstruction; an interactive *CutPlane Preview* is generated that set parts of the reconstruction transparent to provide users with an x-ray view through the scene. Upon confirmation of the cut, users are presented the *CutPlane Visualization*, that resembles

the *CutPlane Preview* and provides users an enhanced scene view as well as navigation possibilities to ease (occluded) target patch discovery. They can travel along the plane cut to further inspect the reconstruction by looking into the 3D scene, as if the cut plane would be a window. At each moment, they can access the target patch by the technique's 3D segmentation and selection algorithms. To change the cut plane or upon target patch access, they can return to their original viewport from where they started the interaction.

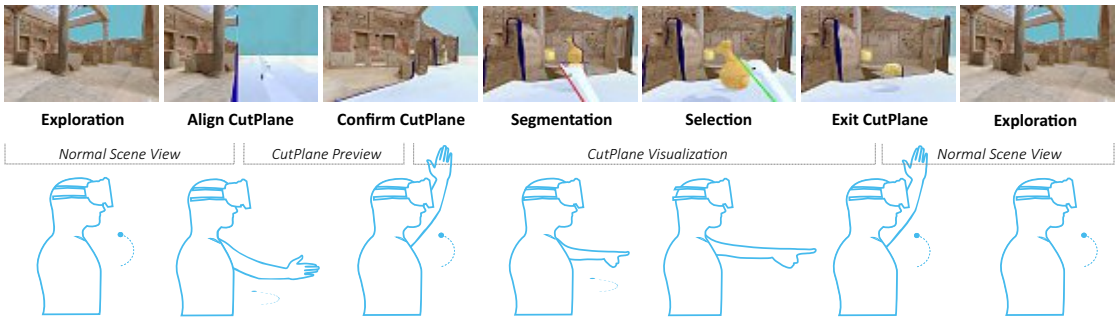


Figure 1.1: Large Scale Cut Plane.

1.3 Contribution

In summary, the contribution of this thesis is to explore occlusion management in dense large 3D reconstructions while incorporating immersive VR and extending the state-of-the-art with the following contribution points:

1. **Large Scale Cut Plane** The occlusion management technique *Large Scale Cut Plane* that provides means for target patch discovery and access in dense large 3D reconstructions while preserving the spatial relation between the target patch and its context. The technique is based on the visualization design patterns *Virtual X-Ray* and *Multiple Viewports*([ET08]).

2. **Segmentation Algorithm** As a pre-requisite to the selection step, segmentation of the 3D reconstruction is needed. We develop a segmentation technique that fits the needs regarding processing time especially for large scale dense reconstructions.

3. **Combination of VR devices** The aforementioned VR setup consisting of a HMD, an ODT and a hand gesture recognizing motion suit, which is used to bring the user into the Immersive VR environment, needs to be combined as frictionless as possible. The tracking information (meant in the sense of 3DOF – three degrees of freedom – positional and 3DOF rotational tracking, in combination also called 3D pose tracking), which is provided by all of the devices has to be transferred to correct behavior of the avatar in the virtual world.

4. **User Study** A user study that explores occlusion management in dense large 3D reconstructions by statistically evaluating the novel *Large Scale Cut Plane* technique

in terms of ease of use and performance and compare it with a state-of-the-art technique as baseline.

The work of this thesis resulted in a high quality publication ([MK16]).

1.4 Structure

This thesis is structured in the following way: the state-of-the-art in mesh segmentation, selection, 3D reconstruction methods, VR hardware and software is reviewed in Chapter 2. Afterwards, Chapter 3 covers the methodology, where the concepts of the developed *Large Scale Cut Plane* technique are explained along with the used methods for segmentation and 3D pose tracking. Implementation details of the used approaches, technical difficulties and algorithmic details are presented in Chapter 4. Chapter 5 describes the construction, implementation and results of an experimental user study, targeting the comparison of the *Large Scale Cut Plane* technique with a common baseline method. Finally, the thesis ends with a conclusion in Chapter 6.

CHAPTER 2

State-of-the-Art

The content of this Section deals with related State-of-the-Art work in the specific domains of the thesis, which are VR-hardware, software, interaction techniques in immersive virtual reality, mesh segmentation methods and methods for surface reconstructions.

2.1 VR-Hard- and Software for Immersive Exploration

2.1.1 VR-Hardware

VR-Input Devices

To enable locomotion in the virtual world, two natural walking approaches exist. In the first solution, the user walks around in a real room and gets tracked by a 6DOF tracking system, which leads to a complex and expensive setup. Also, the multi-user abilities are an uncertainty and a collision-preventing system is necessary. The second approach is an Omnidirectional Treadmill (ODT), which means, the user's position remains fixed, although he or she is walking or running (like in a classical treadmill). The advantage of this approach is that no positional tracking is required, the solution occupies less space and multi-user support is given by using one device per user and no physical collisions can occur. Several hardware vendors and institutes run active developments in the field of ODTs. One of them is the U.S. Army Research Lab, which provides an ODT for soldier training, it is not intended for public usage and costs more than $ 50.000 [CKL04]. The possible movements are walk, run, jump and crouch, which all current ODTs have in common, and additionally a lay-down movement. Two other solutions, the Virtuix Omni [Vir13] and the Cyberith Virtualizer [Cyb13] (see Figure 2.1) additionally provide the ability to sit.
To enable not only movement but also interaction with objects, specialized VR-input hardware (3DOF or 6DOF), which is suitable to track hand/arm and/or body gestures can be used. Devices like joysticks, gamepads or 3D mice are available, where orientation

and position is changed relative to the current state, whereas devices like the Razer Hydra [Raz15] or a 3D pen deliver absolute values. Modern consumer VR solutions include tracking components of their own. The HTC Vive offers two 6DOF input trackers and Oculus provides its own 6DOF input devices for the hands. Another class of VR-input devices are full body capturing systems. They work either with optical tracking like the Motion Suite [XSe15], or with a network of IMUs (Inertial Measurement Units) like the Perception Neuron [Neu15] (see Figure 2.2) or the PrioVR suit [Pri15]. If it is not possible or desired to have a device attached to the user, another tracking-approach would be optical markerless detection. Systems like MS Kinect [Mic13] or Structure.io [Str15] work with the RGB and depth image of a scene and use image processing, feature extraction and skeleton detection to track the hand/arm/body of the user.

Figure 2.1: The Virtualizer ODT

Figure 2.2: Perception Neuron Motion Suit

VR-Output Devices

For VR experiences a range of HMDs (head mounted displays) are available. HMDs are displays which provide a much higher field of view compared to a normal 2D computer screen. They provide a stereoscopic view by showing separate images for both eyes. Two devices are targeted for usage on a PC, namely the Oculus Rift [Ocu13] and the HTC Vive [HTC13], they both have a HD-resolution of 1080x1200 per eye and a refresh-rate of 90 Hz. A device with similar specifications is the PlaystationVR [Son15] from Sony, it provides a refresh rate of 120 Hz and is officially compatible with the Playstation 4 only. Solutions for the mobile market are available too, like the Samsung Gear VR [Sam15], the Zeiss VR One [Zei15], Google Cardboard [Goo15] or the Oculus Go. All mobile solutions use attached smartphones as processing unit and display, except the Oculus GO, which is a standalone VR-headset with integrated processing unit.

Figure 2.3: For PC usage: HTC Vive

Figure 2.4: For mobile usage: Oculus GO

VR-Tracking

Coupling the physical movements from the real world with the virtual movement in VR is necessary to achieve a full-immersive VR experience. Modern VR hardware makes it possible to track position and orientation of the user. By processing this tracking information, the virtual point-of-view in a visualization is automatically adjusted to the user's movement in the real world. Especially, accurate tracking of the head is essential to prevent symptoms like headache or nausea . All non-mobile solutions provide 6DOF positional and orientational tracking, whereas the mobile devices currently provide only 3DOF orientational tracking.
For 6DOF tracking, the used techniques are a combination of inertial sensors and optical methods. For instance, the Vive tracking system uses two laser emitters, placed across the room, which send out infrared laser signals. Photo diodes, which are built into the headset and the two hand controllers receive those signals and create accurate 6DOF information. Tracking updates solely based on this information would be not fast enough, therefore the inertial measurement units, which are built into the headset and the hand-controllers are used as realtime primary tracking information. Built up drift errors from those inertial measurements then get corrected in recurring intervals by the laser tracking method.

2.1.2 Software

Both market-dominating game engines Unity [Uni18] and Unreal Engine [Unr18] are already capable of building and simulating virtual worlds, integrate physically accurate collision detection systems, and creating visual effects while still running with high performance. With mass adoption of consumer-ready VR hardware the 3D engine vendors began to natively integrate the support of VR functionality into their systems, which enables a low entry point into building 3D VR experiences. For VR-devices, which

have no native support from the engine vendors, SDKs or APIs exist to integrate them into the VR application.

2.2 Interaction Techniques in Immersive VR

3D selection in virtual environments consisting of individual 3D objects have been studied in great detail [AA13] and several fundamentally different approaches exist. Virtual hand techniques [BKLP04a, PBWI96] enable direct object selection by either applying a linear or non-linear mapping between the physical and the virtual hand's position. In contrast, pointing techniques cast a virtual object into the scene for selection, such as a ray [BKLP04b] or a cone [LG94]. Pointing techniques might allow the selection of partly occluded objects, depending on the cast object and the distance to the target object. For instance, raycasting tends to be imprecise at large distances where cone-casting is better suited. Modifications of the pointing selection metaphor, such as Flexible-Pointer [OF03] work with bending of rays to improve the selectability of partly occluded objects.

Occlusion management techniques tackle the problem of partly and fully occluded target objects in 3D environments by providing means for object discovery, access and/or spatial relation. According to the taxonomy of [ET08], prior art can be categorized into Volumetric Probes, Projection Distorters, Virtual X-Ray, Tour Planners and Multiple Views. Expand [CWJ12] is an example of a volumetric probe technique as it uses a cone-cast to separate all casted 3D objects in an abstract 2D manner to provide the visualization of occluded objects. Other examples that suit the Volumetric Probe category are Depth Bubble Cursor [RN10], Depth Ray and 3D Bubble Cursor [VGC07]. Virtual X-Ray techniques make target objects visible by turning occluding surfaces invisible or semi-transparent. The work of [HPGK94] introduces the idea of cutting planes that specify the position and orientation of a slice through the virtual object. Since no viewport change is integrated, this technique is limited to individual small virtual objects. Multiple View techniques are well known from 3D modeling applications, such as Maya or Blender3D. These techniques provide different views onto the 3D scene, while these views can be separated or integrated into each other. DrillSample [MVK13] interlocks volumetric probes with multiple (double integrated) views to allow the selection of fully occluded target objects in mobile AR while preserving the spatial scene context to provide a very high object disambiguation. Another occlusion management approaches that uses a double integrated view is Starfish [WGCB12].In this technique the 3D pointer is surrounded by a starfish-shaped surface and the end-points of the branches enclose the nearest objects. For immersive 3D reconstruction environments, [LBLS14] proposes a bi-manual interaction technique by using a set of pinch gestures inspired by smartphone-based interaction to select, manipulate and annotate the surrounding 3D reconstruction. Figures 2.5 and 2.6 show examples of the Multiple Views and the Virtual X-Ray pattern.

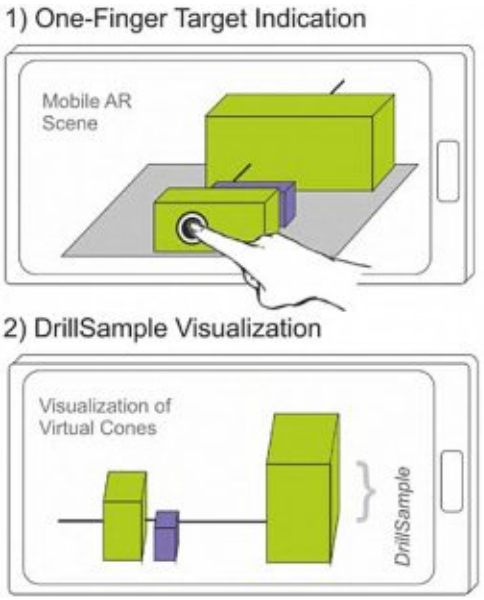


Figure 2.5: Multiple views: Drill sample [MVK13]

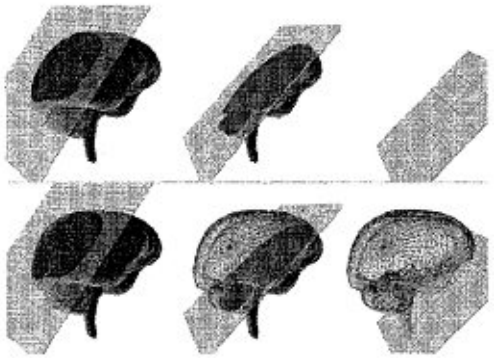


Figure 2.6: X-Ray Visualization [HPGK94]

2.3 Methods for Mesh Segmentation

2.3.1 Automatic Mesh Segmentation Methods

Automatic mesh segmentation methods try to achieve the segmentation of 3D data without any user interaction, only based on properties of the input data.

Learning 3D Mesh Segmentation and Labeling

The approach of Kalogerakis [KHS10] uses a data-driven approach for simultaneous segmentation and labeling of parts in 3D meshes. It formulates an objective function based on a Conditional Random Field model. The objective function is learned from a collection of labeled training meshes. The algorithm uses hundreds of geometric and contextual label features and is capable of learning different types of segmentations for different tasks without requiring manual parameter tuning. Since it is a data-driven approach, it needs labeled data for training the system. Results of this method are shown in Figure 2.7a.

Real-Time and Scalable Incremental Segmentation on Dense SLAM

The method of Tateno [TTN15] works on 3D point clouds. It is a real-time segmentation method and automatically creates segments of the cloud on the fly while the Simultaneous Localization And Mapping (SLAM) process is performed. It incrementally merges segments while each input depth image is processed, the method is able to handle

arbitrary complexity of the input data. Since the segmentation process is fully automatic, the user has no influence over the created parts, an example outcome of the method is presented in Figure 2.7b.

(a) Data-driven method by [KHS10] (b) SLAM-method by [TTN15]

Figure 2.7: Automatic mesh segmentation results

2.3.2 Manual Mesh Segmentation Methods

Manual mesh segmentation methods rely on the user's input to generate the separate mesh parts. The advantage of such methods lies in improved control over the whole segmentation process and therefore more accurate results. The following approaches are state-of-the art mesh segmentation methods. All of them need computational preprocessing of the input data. After that pre-processing step the user draws paint strokes on the surface to signalize the method a preferred location of the segmentation boundary. This input is then processed by each method's particular algorithm and the segmentation result is presented to the user. How paint strokes are interpreted by the several methods is shown by Figure 2.10, which is presented by the comparative interactive segmentation study of [FML12].

Easy Mesh Cutting

The method of Ji [JLCW06] is based on a region growing algorithm. A stroke is made on the desired foreground, and one stroke on the desired background, then all vertices that lie a sketch region are painted as "F" or "B". Afterwards the algorithm uses an improved isophotic distance (as defined by [PSH+04]) to further mark vertices, until all vertices are labeled. For performance reasons, the input mesh is simplified in an offline pre-processing step, and the painting algorithm works with the simplified data. After the segmentation step, hierachical correspondence between the simplified and the original mesh is used to achieve the segmentation results for the original mesh. A result of the method is shown at Figure 2.8a.

Cross Boundary Brushes

The approach of Zheng [ZT10] works with harmonic fields, which are generated using a sparse linear system. A Harmonic field is a scalar- or vector-valued field that is defined by a harmonic function to produce isolines on a surface, as shown in Figure 2.8b. Those

isolines are generated as smooth connected loops on the surface and are thus suitable as candidates for a segmentation boundary. The method lets the user draw two types of strokes, a part-brush stroke is used to segment out semantic part-components and a patch-brush stroke can be drawn to segment out flatter surface patches (Figure 2.9). When a sketch is drawn across boundaries, the algorithm chooses an isoline and performs the segmentation.

iCutter

The iCutter-method from Meng ([MFL11]) uses a similar base algorithm as the Cross-Boundary-Brush approach. The authors also generate a harmonic field consisting of isolines, which are created by a harmonic scalar function. The main difference to the Cross-Boundary method is a different paint stroke handling. The desired position of the cut is indicated by drawing a stroke along the boundary instead of crossing it (see Figure 2.8c). Only part-type segmentations are possible with this method, patch-type cuts are not supported.

Shape Diameter

The work of Fan[FLL11] is based on the shape diameter function of a mesh. The shape diameter function (SDF) [SSCO08] measures the diameter of an object's volume in the neighborhood of each point on the surface, and provides a mapping from volumetric information onto the surface boundary. The SDF is calculated by fitting a Gaussian Mixture Model (GMM) using the greedy Expectation-Maximization (EM) algorithm ([VL02]). The advantage of this method is the possibility of a progressive segmentation update while the user is drawing. This is achieved by two steps. When the user draws a stroke at the beginning of the user interaction a global optimization is triggered and an initial foreground region is obtained. The second step involves a progressive local optimization based on the ongoing paint stroke information and the segmentation boundary is updated interactively. The interactive process is shown in Figure 2.8d.

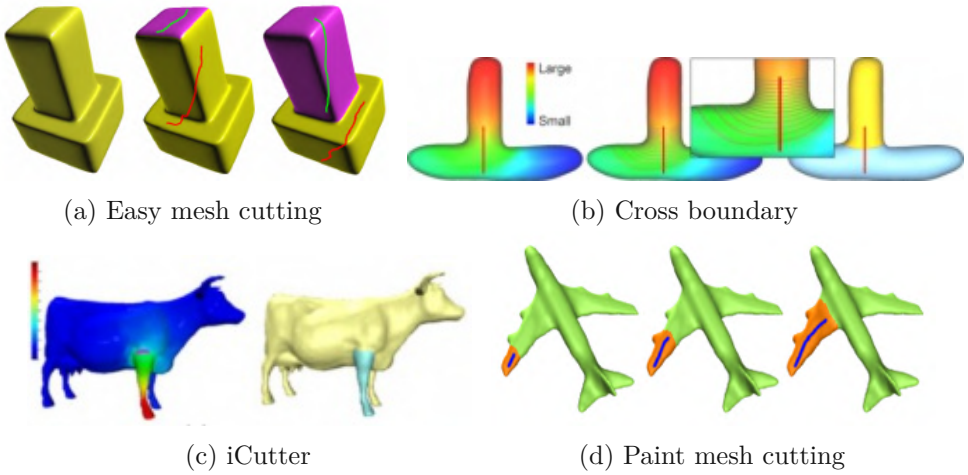
A thorough comparison of this work can be found in the comparative study of Fen[FML12]).

2.4 Methods for Surface Reconstruction

A 3D Surface Reconstruction method generates a virtual representation of a real scene, based on that scene's input data. Different types of input are used by several reconstruction methods. Structure from motion uses a set of 2D images, Kinect Fusion uses the real-time depth data from the Kinect camera, and Laser scanner methods able to produce accurate 3D point cloud data from the laser measurements.

2.4.1 Structure From Motion (SFM)

SFM reconstruction estimates a 3D structure from a set of 2D images which are taken from the desired object from overlapping but different perspectives ([WBG+12]). The



(a) Easy mesh cutting (b) Cross boundary

(c) iCutter (d) Paint mesh cutting

Figure 2.8: Interactive mesh segmentation results

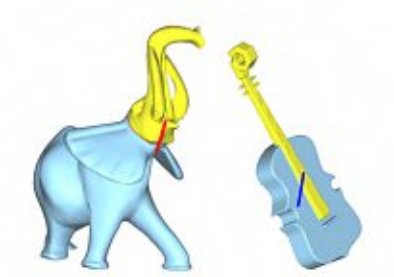


Figure 2.9: Part-brush and patch-brush of the cross boundary brush method.

method is based on finding correspondences between the input images, so-called "features". A widely used feature-detector is SIFT (Scale Invariant Feature Transform) [Low04] which transforms an image into a collection of feature vectors, each of one is invariant to image translation, scaling, and rotation. These properties make it possible to use images that have different distances and angles to the target object. After detecting features in all source images, those features are matched against each other, and outlier matches are filtered out with a RANSAC (random sample consensus) algorithm [FB81]. After matching and applying optimization algorithms, such as bundle adjustment, the resulting feature trajectories are used to reconstruct the camera position and rotation of each input image. The feature points have now a 3D position and are used for the reconstructed 3D object. SFM can be done in three ways. In incremental SFM, found relations are added one by one to the collection, in global SFM all poses get solved at the same time, and out-of-core SFM computes parts of the reconstruction independently and then combines the parts into a global reconstruction. An example for SFM can be found in Figure 2.11.

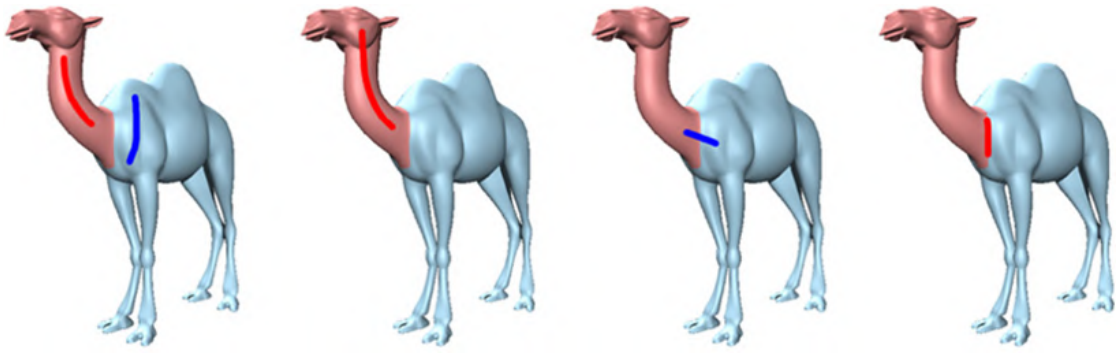


Figure 2.10: Comparison of the resulting boundary by paint strokes on different methods.

Figure 2.11: Structure from motion [SCS+10]

2.4.2 Kinect Fusion

The release of the Microsoft Kinect RGB-D system enabled the possibility of a low-cost live reconstruction of a scene. Kinect Fusion is a system for accurate real-time mapping of complex and arbitrary indoor scenes in variable lighting conditions using only a moving low-cost depth camera and commodity graphics hardware [NIH+11]. The technique fuses all of the depth data streamed from a Kinect sensor into a single global implicit surface model of the observed scene in real-time. In this method, the current sensor pose is simultaneously obtained by tracking the live depth frame relative to the global model using a coarse-to-fine iterative closest point (ICP) algorithm, which uses all of the observed depth data available. Both tracking and integration is performed on the GPU. One disadvantage of Kinect Fusion is the limitation of the algorithm to room sized scenes due to memory constraints. However, there exist several methods, which extend the Kinect Fusion algorithm to overcome this space limitation. An example for a Kinect Fusion reconstructed scene is shown in Figure 2.12.

2.4.3 3D Laser Scanning

A 3D laser scan is a method that measures distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3D representations

of the target. It it also known as LIDAR (light detection and ranging). An advantage of the technology is the ability to scan over large distances. An resulting scene from LIDAR scanning can be found in Figure 2.13.

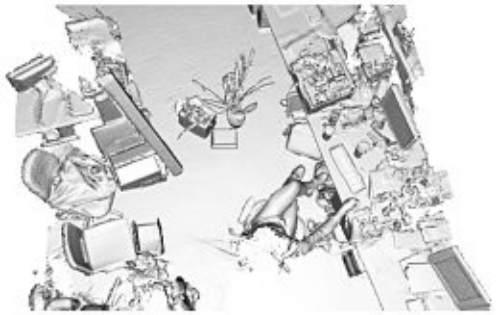


Figure 2.12: Kinect fusion example [NIH+11]

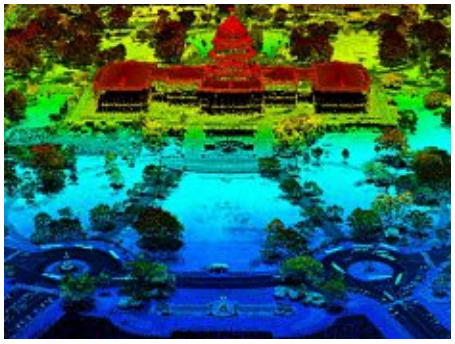


Figure 2.13: Lidar example [GIS18]

CHAPTER 3

Methodology

This chapter describes the approaches and algorithms to create the 3D exploration, segmentation and selection research prototype, which acts as a base framework for the new *Large Scale Cut Plane* technique. The concepts are described in an implementation-agnostic way, concrete technical details and specific platform requirements are described later. The technical specifics, such as used hardware/software technology, APIs, SDKs and the practical implementation details of the developed concepts are described in Chapter 4. This chapter is structured the following way. Firstly, certain design requirements are stated, and secondly concepts and solutions to those design requirements are presented.

3.1 Design Requirements

For fulfillment of the aims of the thesis, especially to create a feasible selection of visible, partly or fully occluded patches of large dense 3D surface reconstructions, several requirements have to be satisfied.

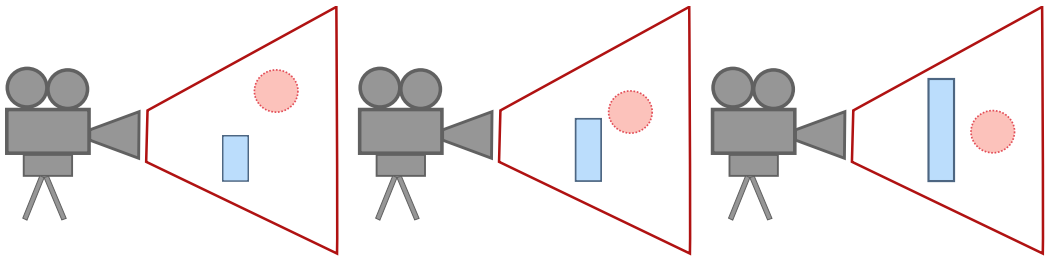
3.1.1 VR Environment

The usage of a reconstructed real world scene is inherent from the problem definition in Chapter 1. To give the user a feeling of presence, the scene has to fulfill a certain degree of realism. The choice, what kind of scene is used, and the chosen technology to get a desired level of realism is crucial for the perceived presence the user wants to find herself in. Not for the developed techniques in this thesis itself, but as requirement for the planned user study, which is also a contribution to this thesis, the spatial structure of the environment needs to be in a certain condition, such that users are forced to use tools to be able to discover and select arbitrary target patches, intuitively and despite partial or full object occlusion. A scene that contains different natural occlusion objects like walls and obstacles is necessary. Its proportions should be in spatial proximity to

the real world, such that the user does not get distracted by spatial inconsistencies. As other criteria, the used environment has to be displayable at interactive frame rates.

3.1.2 Optimized Selection of Target Patch

Within the virtual environment, users shall be provided with means to discover and select arbitrary target patches, intuitively and despite object occlusion up to containment (which means that the whole target patch is covered behind an occluding object). An illustration of the different occlusion types is given at Figure 3.1. The suggested methods have to be designed in such a way that they explore different possibilities of solving the proposed selection tasks. The design space of all theoretically possible techniques is narrowed down with respect to the given technical restrictions like available hardware and limited processing capacity.



(a) *Not occluded* (b) *Partly occluded* (c) *Fully occluded*

Figure 3.1: Occlusion types (occluder: blue box, occluded object: red circle)

3.1.3 Interaction Steps

The requirement to provide the user access to semi- or full occluded target patches in the environment depends on the interaction tools that are given to the user. A complete interaction task is defined as the composition of two subsequent steps: 1) the segmentation of a part of the scene to indicate the desired patch and 2) the selection of the before-segmented patch. Possible preceding user interactions like adapting the view and locating the target are explored to optimize the selection process. Algorithms for all sub-tasks need to be designed and implemented, with the criteria to perform at run-time with interactive frame rates.

3.1.4 Preceding Step for a Selection Task

To be able to independently select parts of the environment it is necessary that the scene consists of individual objects. Since the scanned and reconstructed mesh is usually one connected set of triangles, it is required to separate different mesh parts from the remaining data. This process of splitting 3D data is called mesh segmentation and can be tackled in various ways. There exist several approaches which are commonly used in interactive desktop applications. Segmentation is done based on one or more criteria.

Those criteria can be defined completely by the user's input, like manually drawing a segmentation boundary. It can be based on features, that are determined automatically by processing the input mesh (e.g. calculate and use mesh-defining properties like curvature). Segmentation can also be done by combining the manual and automatic approach, e.g. by manually refining the result of an automatic segmentation process. Different existing methods of mesh segmentation are explained in Section 2.3. In our case, the precondition of the usage of large meshes from a scanned real environment exists. The development of a fitting segmentation technique for this particular problem space is needed. A description, how our segmentation approach tackles the problem, is found in Section 3.3.3.

3.1.5 Navigation

To provide an immersive and natural experience, not only the environment itself has to meet certain requirements (as described in Section 3.1.1), but also the navigation inside said space has to happen in an intuitive and pleasant way. Changing the viewport performed by walking and turning/repositioning the head has to happen in a free manner.

3.1.6 Accessibility

Ease of use of the provided navigation and interaction methods is crucial for the acceptance rate of the proposed method in this thesis. Users should be able to perform accurate patch segmentation and selection, the provided tools should facilitate a precise way of working. The techniques have to be designed carefully with typical users indispositions like nausea or dizziness in mind.

3.2 Developed Concepts

This section proposes concepts and approaches to satisfy the design requirements, as stated in Section 3.1. The section is structured the following way: First, it is explored, how the virtual environment is created 3.2.1, then the design choices for the used types of hardware and interaction tools are explained 3.2.2, afterwards the optimized selection of a target patch 3.2.3 is discussed, which leads to the design of the *Large Scale Cut Plane* 3.3. Finally, a suitable segmentation method is proposed (3.3.3).

3.2.1 Building the Virtual Environment

To fulfill the environment design criteria requirements, as stated in Section 3.1.1, a large dense 3D reconstruction of a prior 3D scanned location is used. The chosen data representation is a point cloud, which is further triangulated to provide the technical foundation for visualization, navigation and manipulation tasks. An existing scan of a scene called *Hanghaus 2 - Ephesos* (see Figure 3.2) is used as basic environment. The scan is available as reconstructed, triangulated and texturized 3D mesh, provided by [BMAW13]. The scene's multi-room structure with narrow passages in-between is already a good fit for the spatial requirements stated in Section 3.1.1. A shot of the scene from

above, which shows the spatial properties is given in Figure 3.3.For the target-patch selection tasks (especially for the user study) additional designed target objects and obstacles are placed into the scene. An “as seamless as possible” integration-approach of those objects into the scene is crucial to preserve a high immersion level. The technical details about creating and using the scene can be found in Section 4.2.

Figure 3.2: Scanned scene: Hanghaus 2 - Ephesos

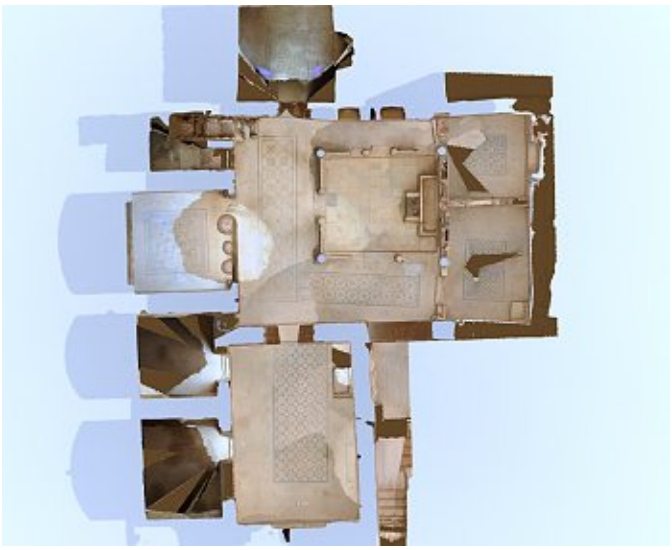


Figure 3.3: Reconstructed scene as view from above: Walls and obstacles as occlusions

3.2.2 Virtual Reality Setup

The seamless integration of different VR devices (Virtualizer (an ODT from Cyberith [Cyb13]), Oculus ([Ocu13]), Perception Neuron([Noi15])) is necessary to provide a convincing user experience. Making the correct design decisions on how those devices are intertwined and how well the input interactions are built around those systems guarantee

a purposeful applicability of the given hardware. The considerations in this section tackle the stated requirements 3.1.5 and 3.1.6.

Combination of VR-Devices

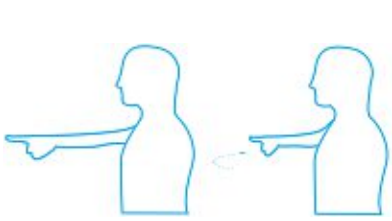
All three VR devices provide their own tracking informations. Our navigation device, the Virtualizer provides a rotation angle *alpha*, which describes the current rotation of the user around the $y - axis$. It also gives feedback when the user is sliding over its ground surface. This translational feedback is given in four directions: forward, backward, sidestep left and sidestep right. Our HMD, the Oculus Rift DK2 provides head-tracking information (both positional and rotational). The Perception Neuron delivers positional information of every single Neuron which is attached to the motion suit. To fulfill the requirement stated in 3.1.5 we chose to process the rotation of the body independently of the rotation of the head. This decoupled approach leads to a natural navigation and viewing experience. The translation information of the Virtualizer is used to move the torso in the virtual world in the correct direction. The positioning of the tracked arm and hand in the virtual scene has to ensure that it is always aligned to the virtual torso and correctly mapped to the hand in the real world to prevent conflicting information. The implementation details of how the different tracking informations are combined are described in 4.3.

Design of Input Gestures

To satisfy the requirement stated in Section 3.1.6, distinctive arm and hand gestures are designed such that they are easy to understand and respecting hardware limitations like detection accuracy of the gesture. Especially for the planned user study, where participants have to understand the gestures within a short introduction time a trade-off between understandability and necessary complexity has to be made. We decided to use a one-handed gesture setup. The given tracking hardware is in principal accurate enough to get information about individual finger positions, but has its caveats in robustness of detection (details are explained in Section 4.4). To get a robust enough detection we decided to use two distinct hand postures as base for our gestures. The first hand posture $VerticalHand$ (as seen in 3.4a) is used mainly for our selection technique. The second hand posture *Pointing* (which is shown in 3.4b) is used for pointing operations within our selection technique, as well as a controlling method for our user interface. The decision, that such an additional user interface is needed was made to allow more interactions without the need of creating more gestures, which would be harder to remember and a technical challenge to distinguish from each other. Therefore a user interface with pointing gesture enabled buttons is introduced to enable additional control possibilities like changing the current tool (from segmentation to selection), or to provide undo functionality. Since the user interface in VR cannot be made as 2D HUD (Head Up Display, like an overlay menu in classic desktop applications) it has to be designed as part of the 3D environment. We decided to implement it as overhead menu (the user interface only appears in the user's view if he leans his head up above a certain threshold,

see Figure 3.4c). This offers the advantage of not distracting the user while a task is performed, and the pointing gesture cannot be misinterpreted, also incorrect inputs are prevented that way. Since we have no additional *Accept input* gesture a button press event is triggered, if the pointing ray hovers over the button longer than a certain dwell time.

(a) *Vertical Hand* gestures



(b) *Pointing* gestures

(c) Overhead menu

Figure 3.4: Different Gestures for interactivity.

3.2.3 Optimized Selection of Target Patch

The interaction possibilities in a VR setting are different than in a common desktop setup. In a classical desktop environment, the user is able to arbitrary translate and rotate and scale a scene, until she gets a good viewpoint for a target selection task. In an immersive environment, changing the viewpoint has to happen in a careful manner to prevent disorientation, motion sickness and other illnesses. The challenge is to retain the natural movement and interaction behavior, but to enhance it with purposeful assisting techniques to optimize the desired task, which is in our case the selection of a target patch. Regarding prior art in occlusion management design pattern, we found volumetric probe techniques not applicable since target patches are not isolated upon start of the interaction task. Employing solely *Virtual X-Ray* technique – such as [HPGK94] – would not be sufficient as prior knowledge about distractors and targets is required. This violates the idea of enabling users to select arbitrary patches within the 3D reconstruction. Furthermore, *Virtual X-Ray* techniques only provide low visual depth cues which would lead to disorientation within a large environment. *Multiple Views* techniques provide high depth cues and overview to ease discovery and access, but are not capable to deal with occlusions beyond object intersection. Tour Planner techniques [AVF04] require an offline step that violates the requirement of interactive target discovery and access at

run-time. This leads to the design of our method *Large Scale Cut Plane*, which tries to provide such an optimized approach.

3.3 Large Scale Cut Plane Selection Technique

To close the gaps of prior art and to tackle the requirement stated in 3.1.2, we designed *Large Scale Cut Plane* as a combination of the *Virtual X-Ray* and the *Multiple Views* pattern, as described in Section 2.2. Thereby, we utilized the strong disambiguation strength of Virtual X-Ray that can handle object interaction up to containment, which makes up the weak disambiguation capabilities of Multiple Views. Furthermore, we obtain high depth cues by implementing a double integrated view into the Virtual X-Ray pattern and allow 3D navigation [ET08]. This is in particular important due to the large spatial extent of our intended 3D environments. We provide full depth cues across both integrated views to foster human perception and spatial understanding. Furthermore, we obtain a very high target invariance dimension as we retain – across the entire interaction period – appearance, depth, geometry and location of target and of remaining distractors. We design *Large Scale Cut Plane* as an active online interaction model to avoid offline recomputation that would violate the interactivity requirement. The technique acts in the view as well as object space for task solution, meaning that we both manipulate the viewing transform as well as the 3D environment itself, as each of them alone is not sufficient for continuous 3D surface environments. With the chosen design, Large Scale Cut Plane retains a high degree of depth cues and supports spatial relation during target discovery and access.
The Large Scale Cut Plane Technique is a composite two-step selection technique. The two steps comprise

1. an interactive plane cut through the 3D environment following the Virtual X-Ray pattern,
2. an integrated viewport change employing the Multiple View pattern to provide users with a detailed view on the virtual environment in which they can a) navigate along the cut plane, b) segment target patches and
3. select those target patches. A formal description is given in Figure 3.5.

Simple natural gestures are employed to operate our technique: we use a mid-air hand pointing gesture to align and confirm the cut plane, and a finger pointing gesture for both segmentation and selection. All gestures are illustrated in Figure 3.4.

3.3.1 Workflow of Interaction Technique

As shown in Figure 3.5, users can start the interaction process while navigating through the virtual environment. Using the mid-air arm gesture *Align CutPlane*, they trigger the vertical cut through the scene while they simultaneously can define the yaw direction θ

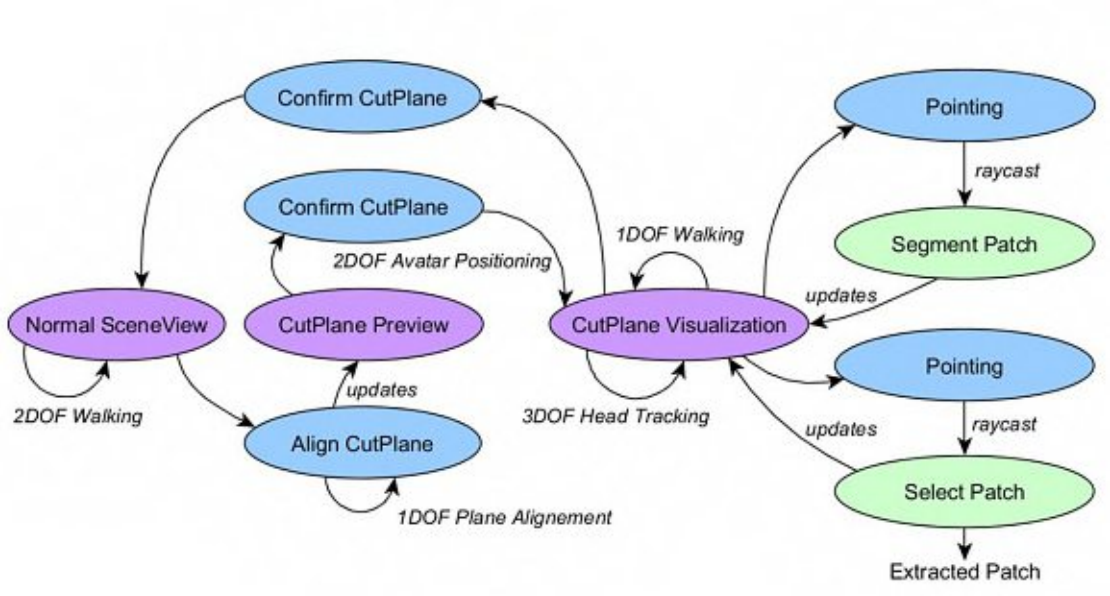


Figure 3.5: State diagram of the Large Scale Cut Plane. The viewports are coded in violet, gestures in blue and interactions in green.

of the cut plane by changing their arm's pointing direction. While doing so, users are provided with the CutPlane Preview which shows a preview on the cut scene. Therefore, the environmental 3D geometry at one side of the cut plane remains unchanged while on the other side the environmental 3D geometry is rendered semitransparent. For our algorithmic implementation, we have chosen to set the geometry transparent at the right hand side of the cut plane. This preview functionality is inspired by the Virtual X-Ray techniques that have been found to make discovery trivial and facilitates target access [ET08]. Since pure Virtual X-Ray approaches require a priori knowledge of the target to correctly remove distractors which weakens the depth perception, we incorporated the Multiple Views paradigm by integrating a second view that is presented to the user upon executing the *Confirm CutPlane* gesture.
Within this CutPlane Visualization, the cut surface of the unchanged geometry is highlighted to provide visual cues of the cut plane while the semi-transparent geometry on the other side is set to almost transparent to minimize user distraction. Next, the viewport is manipulated by translating the user's viewing component (the virtual camera) as well as the avatar's geometry along the scene's ground plane. In our implementation with a left-handed coordinate system, we have found a slight positive translation along the x- and z-axis to be effective.
Thereby, we achieve two important things: upon view port change, users are provided with

1. an overview of the cut surface

2. alignment with the cut plane so that they can directly view and explore the remaining scene geometry

Therefore, users are able to freely look around and travel along the cut plane for inspection. Thereby, hidden and occluded target areas (up to containment) can be quickly accessed, inspected and selected. Therefore, users first perform a segmentation within the visible scene geometry using the mid-air hand gesture Pointing to define a segmentation boundary that determines the target patch they want to extract. Upon satisfaction, users change the state to selection and perform the Pointing gesture to finally segment and subsequently select the target patch.
After selection, users remain in the CutPlane Visualization to perform another selection. Alternatively, they can exit the CutPlane Visualization by executing the *Confirm CutPlane* gesture. This triggers the manipulation of the viewport. An inverse translation is applied yielding a re-aligned user's viewing component and avatar geometry with their 6DOF pose upon plane cut. Furthermore, the complete virtual environment is rendered opaque again. The entire technique's workflow is illustrated in Figure 3.6, the algorithm for cut plane alignment and user repositioning is described in Algorithm 3.1.

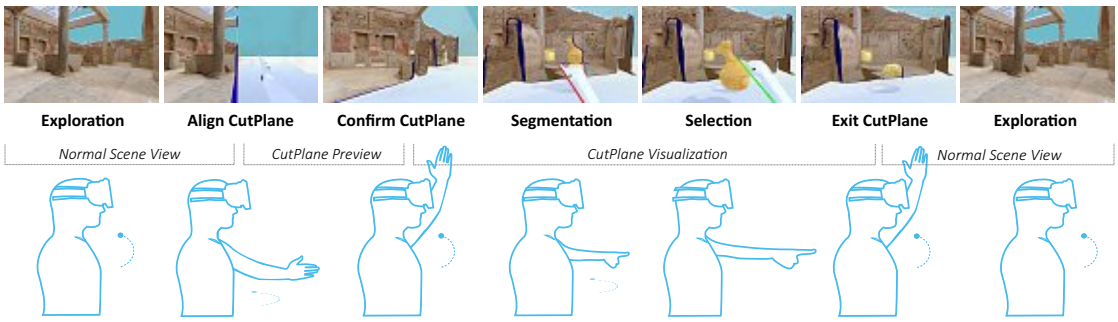


Figure 3.6: Large Scale Cut Plane interaction sequence.

3.3.2 Interaction Technique Algorithm

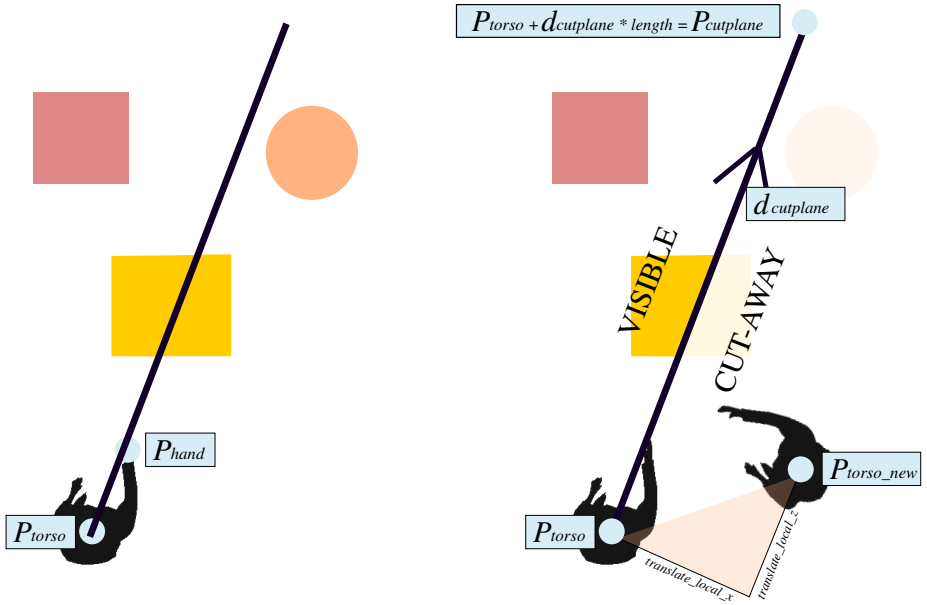
A user starts exploring the scene in the normal immersive 3D scene view. If the *Align CutPlane* gesture is performed, a plane's direction is calculated and the CutPlane Preview is rendered. Algorithm 3.1 shows the necessary calculations with user's torso position $P_{torso} \in \mathbb{R}^3$ and hand position $P_{hand} \in \mathbb{R}^3$ as input. The resulting direction $\vec{d}_{cutplane} \in \mathbb{R}^3$ together with $P_{cutplane} \in \mathbb{R}^3$ - a point that lies on the plane itself - are used to show the CutPlane Preview by rendering parts of the geometry semi-transparent, in our implementation on the right side of the cut plane. This is calculated per pixel, as described in Algorithm 3.1.

Upon execution of the *Confirm CutPlane* gesture, the cut plane is locked at the current user's position $P_{torso} \in \mathbb{R}^3$ and the manipulation of users' viewport is performed, calculating $P_{torsonew} \in \mathbb{R}^3$, as shown in Figure 3.7. In this CutPlane Visualization, travel is constrained to one degree-of-freedom along the direction of the cut plane. Thereby,

Algorithm 3.1: Cutplane-Creation and ViewChange

Data: P_{torso}, P_{hand}
Result: $\vec{d}_{cutplane}$, $P_{cutplane}$

$D_{cutplane} \leftarrow normalize(P_{torso} + P_{hand})$ $P_{cutplane} \leftarrow P_{torso} + \vec{d}_{cutplane} * length$
foreach P *in pixels* **do**
  $P.visible \leftarrow distance(P_{worldpos}, P_{cutplane}) < 0$
end



(a) *Align CutPlane* (b) *Confirm Plane, User Repositioning*

Figure 3.7: Geometric representation of the Large-Scale-Cut-Plane-Algorithm

distraction of users can be minimized as they can not get ”lost”. If the pointing gesture is executed, users can draw paint strokes onto the unchanged parts of the scene geometry by performing a ray cast with the 3D scene reconstruction. Upon finishing the pointing gesture, the segmentation algorithm computes a boundary. As prior art segmentation approaches take the entire 3D model into account, we investigated a local region growing algorithm to ensure interactive framerates even when operating within large 3D scene reconstructions. This algorithm is described in Section 3.3.3. If a correct segmentation boundary was found the user can subsequently select this indicated target patch by executing the Pointing gesture. This extracts the patch from the global 3D reconstruction model. Upon patch selection, the extracted patch is relocated to the user’s current

position for subsequent manipulation. Since manipulation is not covered within the scope of this thesis, the patch geometry remains at the user's position. Upon selection completion, users can exit the CutPlane Visualization by executing the *Confirm CutPlane* gesture. This reverts the previous view port change so that users' viewing component is re-aligned with $P_{cutplane}$. Furthermore, the transparent scene geometry is set opaque to present the normal scene view to the users.

3.3.3 Interactive Segmentation within Large Dense 3D Surfaces

The segmentation algorithm has to run at interactive framerates, since lowering or even interruption of the system's responsiveness would mitigate interactive exploration and immersive user experience. It was therefore designed to solely operate in the local neighborhood of the drawn paint strokes and not on the whole dense reconstruction. In its current form, it is restricted to perform planarlike segmentation and does not involve global surface processing operations like the related interactive methods in Section 2.3. Our segmentation technique is divided into the following subtasks:

- Find a segmentation plane
- use a region growing algorithm to create a thick boundary
- use an edge thinning technique to narrow down the thick boundary
- use again a region growing technique to define the individual segments

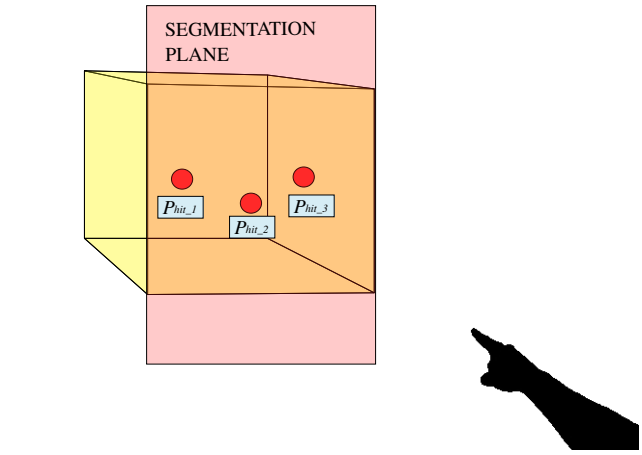
This technique tackles the requirement, as stated in Section 3.1.4.

Finding a Segmentation Plane from Hitpoints

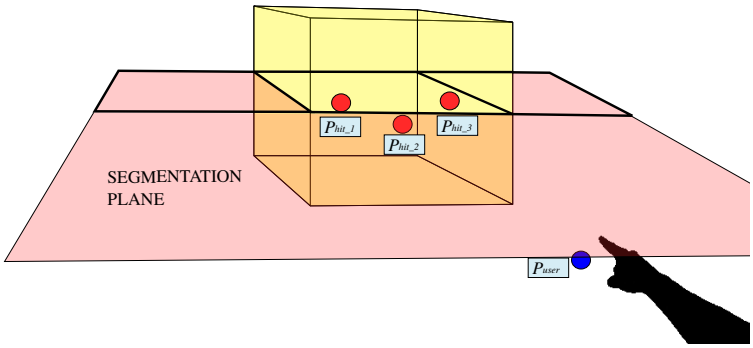
As first step, a list of geometry hitpoints is created as the result of the raycasts made by the user's segmentation paint-strokes. Then these hitpoints and the current user's position are fed into a least-squares plane-fitting algorithm; the user's position acts as a spatial reference point to ensure the correct orientation of the estimated plane. Taking the user's position into account is crucial especially in such cases when the local z-distances of the found raycast hitpoints w.r.t the user's position are nearly the same. Figure 3.8 shows this scenario. Not taking the user's position into account would create a useless nearly vertical segmentation plane, which would have never been an intended segmentation boundary w.r.t the user's paint strokes.

A Region Growing Algorithm to create a Thick Boundary

Next, the region growing algorithm as described in Algorithm 3.2 marks all vertices that lie inside a parameterizable distance to the plane. The algorithm takes the definition of the plane ($segPlane$) as input parameter, a starting vertex $ver \in \mathbb{R}^3$ (the one with the nearest distance to $segPlane$) from the set of hitpoints and a distance threshold thr,



(a) *Without user position*



(b) *With user position*

Figure 3.8: Created segmentation plane with or without user position taken into account

which influences the broadness of the resulting boundary corridor. A reasonable threshold value can be chosen, if the density of the given input data is taken into account. If *thr* is small and the vertices of the input mesh are sparse the risk for an incomplete boundary is raised, therefore the search has to start over with a larger threshold *thr*. After reaching the threshold, the algorithm returns a list of boundary candidates *boundaryList*, which represent the candidates for the final segmentation boundary, as Figure 3.9 demonstrates.

Edge Thinning Technique

A subsequent edge thinning step is employed to find the resulting thin segmentation boundary that spatially describes a closed edge-loop. Edge thinning iteratively removes those edges that have the maximal Euclidean distance to *segPlane* as long as the thinning operation does not break the boundary's connectivity constraint.

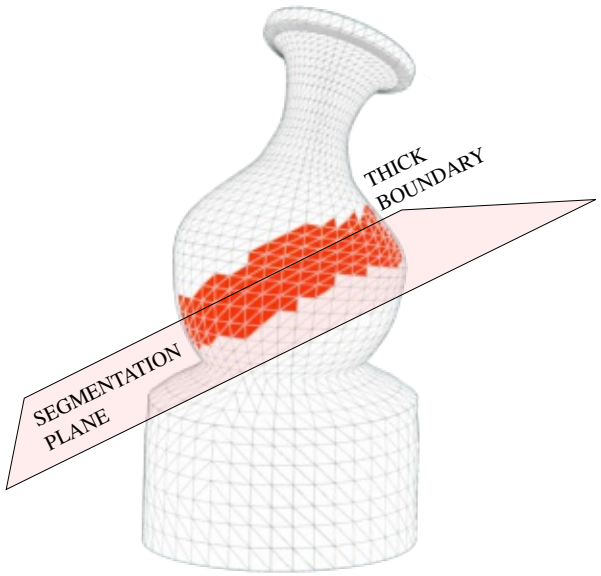


Figure 3.9: Result of the Region Growing based on segmentation plane-algorithm. The selected vertices are drawn in red (marking of vertices is continued on the invisible backfaces).

Algorithm 3.2: Region-Growing Algorithm

```
   Data: segPlane, ver, thr
   Result: boundaryList
1  foreach nb in ver.neighbours do
2      if distance(nb, segPlane <= thr) then
3          if NOT boundaryList CONTAINS nb then
4              ADD nb TO boundaryList
                RegionGrowingRecursive(segPlane, nb, thr)
5          end
6      end
7  end
```

Region-Growing to create Target Patch

The next step cuts the mesh at the found boundary into two parts, this is done again with a region-growing approach. A recursive search is started in both directions of the neighbours of the boundary, and marks them with two different ids (set_1 and set_2). If on one side all neighbours are visited and therefore all vertices at one side of the boundary are found (set_1 is full), all residual unassigned vertices are automatically assigned to (set_2). Figure 3.10 demonstrates the marking process.

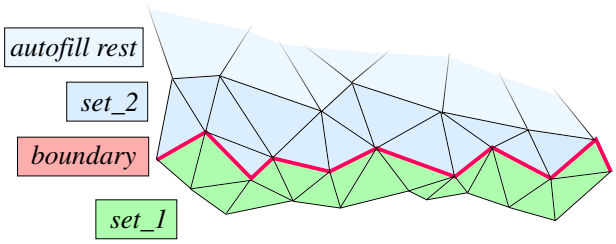


Figure 3.10: Region growing algorithm to divide mesh along the boundary, if the region growing finishes at one part (like in set_1), the remaining vertices will be automatically filled into set_2.

If the user executes another segmentation on an already logically segmented mesh, the existing boundaries and marked segments have to be taken into account. We restrict our method to only allow subsequent segmentations if the newly generated boundaries do not overlap with existing boundaries, since that would lead to many edge cases that were out of scope for this thesis. As long as the user does not apply a selection of the target patch after segmenting, the mesh does not need to be physically divided, the physical separation must happen only, if a selection task is performed, then every set of vertices has to be converted into a new mesh. The technical challenges that arise from the mesh splitting process are discussed in Section 4.6.1.

CHAPTER 4

Implementation

In this chapter, concrete implementation details, challenges and problems of the design concepts, as stated in Chapter 3, are discussed.

4.1 Employed Frameworks and Technologies

The research prototype application is implemented in C# on Windows 10 and uses the following engines/libraries/SDKs.

- Unity Engine 5.3.2 [Uni18]
- Axis-NeuronPROx64-3.6.32 (Perception Neuron SDK) [Noi15]
- CybSDK (Virtualizer Unity SDK) [Cyb13]
- Ovr-SDK-win-0.8 (Oculus Rift SDK) [Ocu13]

The following applications were used additionally:

- Maya 2014 Student Edition (for creating additional scene data) [Aut18]
- Axis Neuron 3.6.32 (for recording motion data and as tracking data source for unity) [Noi15]

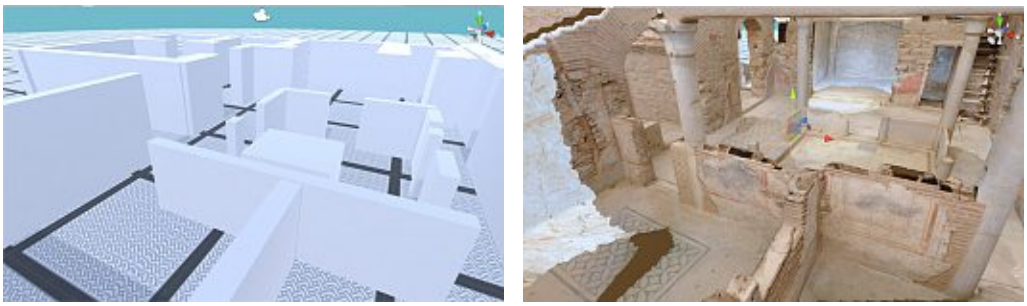
4.2 Building the Immersive Environment

The reconstructed 3D scene *Hanghaus 2 Ephesos* is available as a set of one .obj file and 90 .png-images. The model consists of ~600.000 vertices. The used Unity Engine has a

vertex limit of 65536 vertices per mesh, and thus the unity asset importer automatically splits the model into 20 different parts. Some parts of it are not exhausting the per-mesh vertex limit, but since the engine's splitting algorithm is proprietary, it can not be retraced, on what criteria the partition process is done. Figure 4.1 shows the split up mesh.

Figure 4.1: Split-up mesh from unity asset importer (parts highlighted as blue wire-frame overlay).

Since a user should be able to navigate inside the scene without walking through walls, collision objects are necessary. Unity provides the option to automatically generate colliders from imported meshes, but this option is not preferred in our case. Since the model is created from a dense reconstruction, the automatically created colliders could contain point cloud outliers which block a visually free path, they could have non-manifold geometry and would therefore be not well suited for unity's physics engine. The user could get stuck into a collider while navigating through the scene. Underlying colliders were built out of boxes and a ground plane to prevent that problem, they can be seen in Figure 4.2.



(a) *Collider scene* (b) *Original scene*

Figure 4.2: Custom-built box colliders for collision detection

For the target-patch selection tasks (especially for the user study), additional designed target objects and obstacles are placed into the scene. The added objects are designed

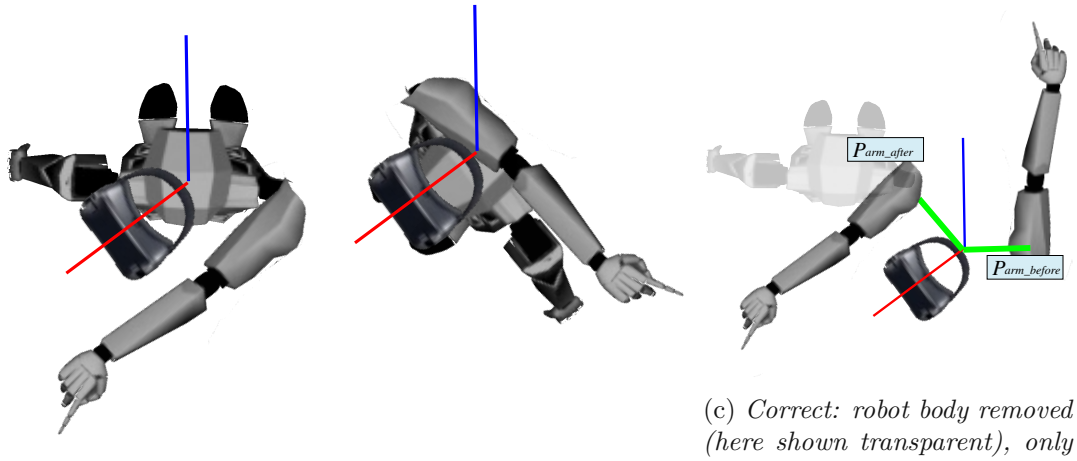
in a way to fit into the rest of the scene to preserve a high immersion level for the user. Examples are shown in Figure 4.3.

Figure 4.3: Additional target objects and obstacles

4.3 Integration & Combination of the used VR-Devices

As described in Section 3.2.2, all three VR devices deliver their own tracking information. The topic of this section is, how to use this information to get a correct mapping of physical motions into the virtual world with this special setup. The user in the unity scene is represented by a character controller. The SDK we get from the Virtualizer (referred as Omni-Directional Treadmill (ODT)) is built on top of the Oculus Rift (referred as Head Mounted Display(HMD)) SDK. This decision is reasonable, because the two devices cannot be considered independently. If a user turns around in the ODT, the HMD as well as the ODT register the movement, the control-algorithm has to decide how the rotations affect the character controller. The ODT SDK solves it by not applying its own rotation to the character controller at all. It always stays in a zero-rotated-state. To get the correct view, it is sufficient, when the camera, which is attached to the character controller is controlled by the tracking data from the HMD and the view gets adjusted properly. When a user walks forward and is in a rotated state, the ODT SDK automatically moves the character into the right direction, although the rotation itself is not applied on the character controller. The third device, the Perception Neuron (referred as PNR) further complicates the setup. As explained in more detail in Section 4.4 the SDK of the PNR controls its own character in unity (in form of a robot with bones and joints). Since we use the PNR in the one arm/hand setup mode no full-body rotations get applied to the robot. If this robot would simply be attached to the character controller from the ODT, the arm would have the correct rotation, but it would be on the wrong position and would therefore be out of view. Figure 4.4a illustrates this problem. If we would additionally apply the known rotation angle from the ODT this would be wrong too, because then we would add up the ODT plus PNR rotation, and would get twice as much rotation for the arm as needed, as Figure 4.4b demonstrates. The solution to the problem is illustrated in Figure 4.4c: The body of the robot gets visually removed, since it is not be visible in first person view anyway, and the arm gets translated, as if the shoulder starting point

would be rotated, but instead of applying the rotation, only a translation to the new shoulder starting point is applied. The correct rotation gets applied automatically from the PNR rotation and thus the arm is in its correct transformed state. The process is formally described in Algorithm 4.1.



(a) *Not correct: Robot attached without applying rotation (incorrect), blue line: original unrotated state, red line: rotated body (and HMD) in real world*

(b) *Not correct: rotation applied on robot (correct), but rotation on arm applied twice (incorrect), blue line: original unrotated state, red line: rotated body (and HMD) in real world*

(c) *Correct: robot body removed (here shown transparent), only translation applied to arm, rotation is applied from the motion suit. blue line original rotation, red line rotated body (and HMD) in real world, green line transformed with the same rotation angle*

Figure 4.4: Wrong and correct arm positioning

Algorithm 4.1: Algorithm for transforming the robot arm to the right position

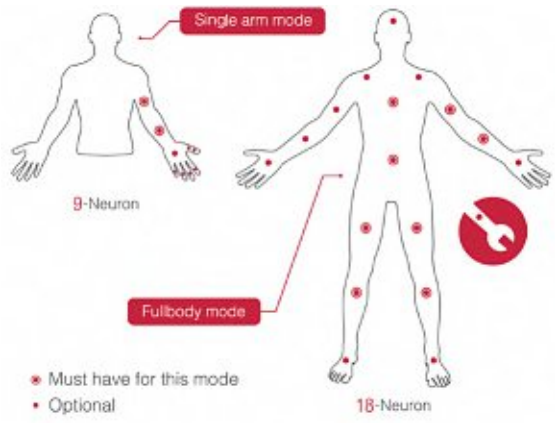
Data: $\measuredangle a_{virtY}$, $P_{arm-old}$,P_{torso}
Result: $P_{arm-new}$
1 $d_{arm} \leftarrow P_{arm-old}$ - P_{torso}
2 $P_{arm-new} \leftarrow rotate(\measuredangle a_{virtY}, Y - axis) * d_{arm}$

4.4 Implementation of Arm and Hand Gestures System

The complete gesture detection and processing system is described in the following order: First, the hardware preparation, then the software bridge from the manufacturer's software to the unity engine, next getting the tracking positions in unity, afterwards detecting gesture patterns of those positions, and finally, using that information to control our application.

4.4.1 Hardware Preparation

The motion suit Perception Neuron [Noi15], which is used for this project, has different available configurations. The two most common are full body mode and single arm mode (as shown in Figure 4.5a). Since our design consists only of one-hand gestures we setup the Neuron suit in the single arm configuration (Figure 4.5b and 4.5c) with the addition of a second neuron for the index finger to ensure increased accuracy in that for us crucial tracking area.



(a) *Comparison*

(b) *Single arm setup*

(c) *Single arm setup with additional neuron*

Figure 4.5: Used Perception neuron setup

4.4.2 The Software Interface

For getting the data into our system we use two software tools that are provided by the manufacturer of the motion suit: the software *Axis Neuron* and the *Neuron unity-integration SDK*. The software takes responsibility of the communication with the motion suit via USB cable or WIFI connection and broadcasts the data via a TCP network interface to a configurable port. The software is also capable of live recording motion data and replaying the recorded data instead of live motion data on request. We used that to test and tweak our gesture detection without the need to always put the suit on. On the unity side,the manufacturer's SDK consists of a 3D-model of a robot, which, if placed into the scene takes over the translations and rotations of the user's physical motion data. As we already discussed in Section 4.3 we programmatically removed the translation and rotation of the body such that they do not contribute to the remaining transformations. That means, our transformations begin at the shoulder component. Additionally we also removed all parts except the right arm and hand from the 3D model of the robot, so they cannot block the view, when the user and thus the robot mesh gets repositioned.

4.4.3 Obtaining 3D Pose

The transformations of the robot arm are organized as a hierarchical scene graph. Every frame, the gesture component stores the resulting world positions and local rotations from the given transformations of all body parts that are needed by the gesture detection engine. In our case, those positions are:
P_{Arm}, P_{Elbow}, P_{Hand}, $P_{Thumb1-3}$,$P_{Index1-4}$, $P_{Middle1-4}$,$P_{Ring1-4}$,$P_{Pinky1-4}$.
And one rotation angle: r_{handX}.
Figure 4.6 illustrates their location on the robot model. Those are more positions than physical neurons are available, the interpolation logic is done by the *Axis Neuron* software and works in a very robust way most of the time.

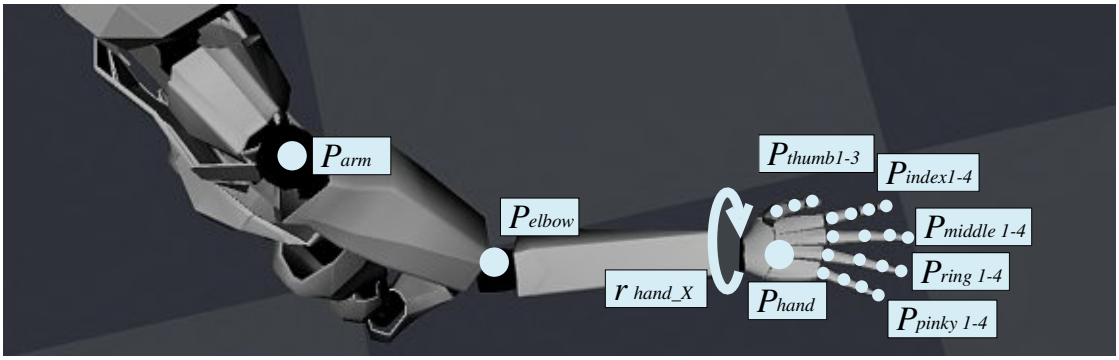


Figure 4.6: Location of the tracking points on the robot model

4.4.4 Detecting Gesture Patterns

For our two main postures $VerticalHand$ and $Pointing$ (as described in Figure 3.4) we use different angles and directions between our tracked points to detect if one of those postures is met.
For the $VerticalHand$ posture we use the following metrics: direction from arm to elbow: $d_{arm-elbow}$, direction from elbow to hand: $d_{elbow-hand}$, direction from hand to the tip of the middle finger $d_{hand-middle}$, direction from hand to the tip of the ring-finger $d_{hand-ring}$ and the local x-rotation-angle of the hand-transform r_{handX}. A $VerticalHand$ posture is detected if the directions $d_{arm-elbow}$, $d_{elbow-hand}$, $d_{hand-middle}$ and $d_{hand-ring}$ are similar enough (deviation below a certain configurable threshold) and the rotation r_{handX} is between a configurable minimum and maximum angle, which ensures that the hand must be held in vertical position to detect the posture). Algorithm 4.2 and Figure 4.7 demonstrate the detection.

The second posture $Pointing$ works in a similar way. In this case the directions of the individual fingers in relation to each other are important. As long as the direction from the hand P_{Hand} to the tip of the index finger P_{Index4} is similar to the direction of the finger itself ($\vec{d}_{index1-index4}$) and the other fingers lie in orthogonal-near direction (with a theoretical angle of 90 degrees or more between index finger direction and direction

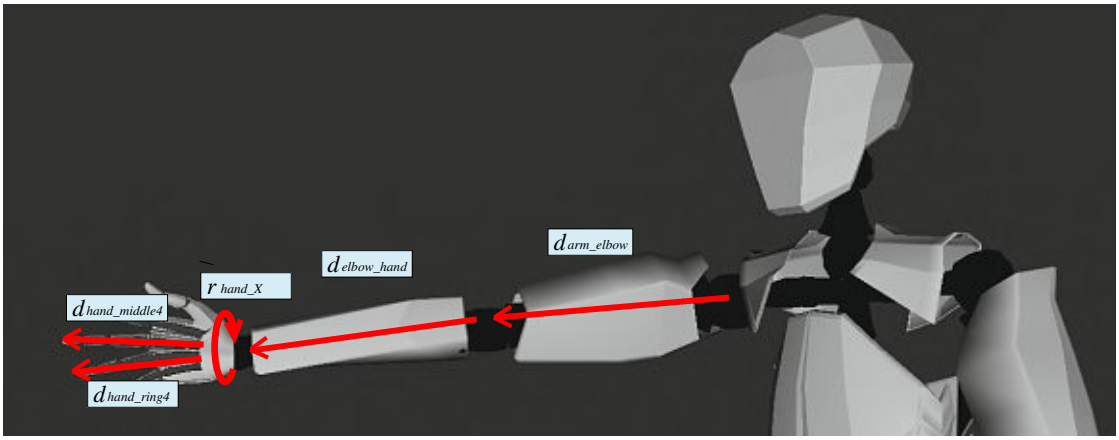


Figure 4.7: Used metrics in the Vertical Hand posture detection

Algorithm 4.2: Algorithm for detecting the Vertical Hand posture

Data: $\vec{d}_{arm-elbow}$, $\vec{d}_{elbow-hand}$,$\vec{d}_{hand-middle}$,$\vec{d}_{hand-ring}$,r_{handX}, $threshold_{gesture}$, $threshold_{angleMin}$, $threshold_{angleMax}$

Result: $isVerticalHandActive$

$dot_{armelbow-elbowhand} \leftarrow dot(\vec{d}_{arm-elbow}, \vec{d}_{elbow-hand})$
$dot_{armelbow-handmiddle} \leftarrow dot(\vec{d}_{arm-elbow}, \vec{d}_{hand-middle})$
$dot_{armelbow-handring} \leftarrow dot(\vec{d}_{arm-elbow}, \vec{d}_{hand-ring})$ $avg \leftarrow$
$average(dot_{armelbow-elbowhand}, dot_{armelbow-handmiddle}, dot_{armelbow-handring})$
$rot_{isInbetween} \leftarrow threshold_{angleMin} < r_{handX} < threshold_{angleMax}$
if $rot_{isInbetween}$ AND $avg > threshold_{gesture}$ **then**
  $isVerticalHandActive \leftarrow true$
end
else
  $isVerticalHandActive \leftarrow false$
end

of every other finger) then the *Pointing* posture is detected. Algorithm 4.3 and Figure 4.8 demonstrate the detection. The position of the tip of the index finger is also stored as input for calculating raycast directions when the gesture is used in the segmentation process.

The usage of configurable thresholds brings flexibility to the gesture detection system and can prevent inoperabiltiy of the whole application. The thresholds can be easily adjusted, if the accuracy of the used hardware is unstable, or the skills of the participating user to perfectly reproduce a demanded gesture are insufficient.

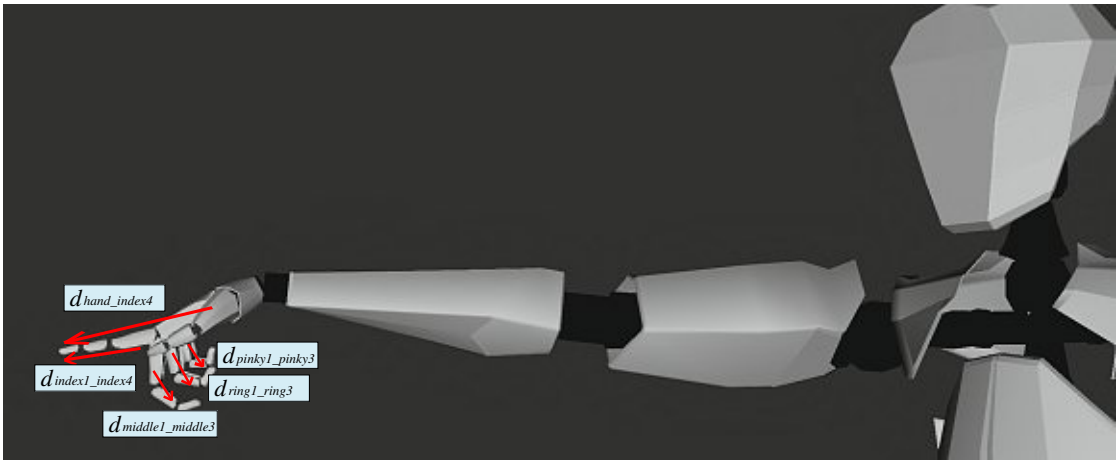


Figure 4.8: Used metrics in the Pointing posture detection

Algorithm 4.3: Algorithm for detecting the Pointing posture

Data: $\vec{d}_{hand-index4}$, $\vec{d}_{index1-index4}$,$\vec{d}_{middle1-middle3}$,$\vec{d}_{ring1-ring3}$,$\vec{d}_{pinky1-pinky3}$, $threshold_{gesture}$

Result: $isPointingActive$

$dot_{handindex4-index1index14} \leftarrow dot(\vec{d}_{hand-index4}, \vec{d}_{index1-index4})$
$invdot_{middle13-index14} \leftarrow 1 - clamp(dot(\vec{d}_{middle1-middle3}, \vec{d}_{index1-index4}))$
$invdot_{ring1-ring3} \leftarrow 1 - clamp(dot(\vec{d}_{ring1-ring3}, \vec{d}_{index1-index4}))$
$invdot_{pinky13-index14} \leftarrow 1 - clamp(dot(\vec{d}_{pinky1-pinky3}, \vec{d}_{index1-index4}))$
$avg \leftarrow average(dot_{handindex4-index1index14},$
$invdot_{middle13-index14}, invdot_{ring1-ring3},$
$invdot_{pinky13-index14})$
if $avg > threshold_{gesture}$ **then**
  $isPointingActive \leftarrow true$
end
else
  $isPointingActive \leftarrow false$
end

4.4.5 Apply Detected Patterns

The distinction between the two postures *Vertical Hand* and *Pointing* are used as a base for all gestures that are needed to interact with the system.
The gesture *Align CutPlane* uses the *Pointing* posture and the direction from torso to hand to align the CutPlane. The gesture *Commit CutPlane* to enter and exit the *CutPlane View* is also based on the *Vertical Hand* posture. Additionally, a check is implemented, such that the direction from arm to the hand ($d_{arm-hand}$) is also compared to the up-vector by building a dot-product between the two of them, as Figure 4.9

demonstrates. If a certain threshold is met, the gesture triggers the user-requested view-change.

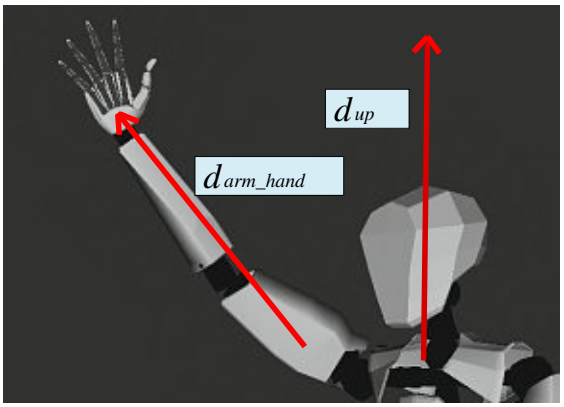


Figure 4.9: Used metrics in the commit plane gesture

4.5 Implementation of Large Scale Cut Plane Technique

The implementation details of our novel *Large Scale Cut Plane* technique are structured in the following way: In Step 1, the details of *Cut Plane Alignment* are discussed, step 2 explains the transition into the *Cut Plane View*, following with step 3, which clarifies the handling of Raycasts in the *Cut Plane View*.

4.5.1 Step 1: Cut Plane Alignment

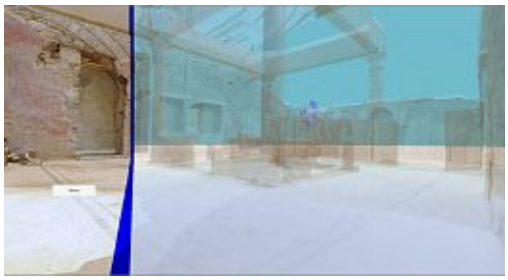
When a user performs the *Align CutPlane* gesture, an X-Ray view on the right side of the plane has to be presented to the user. The geometry on the left side of the plane should remain unchanged. The plane itself and the adjacent geometry within a certain distance to the plane have to be marked visually. An example is presented in Figure 4.10.

We decided to implement the needed x-ray semitransparent view by manipulating the pixel shader. Since the same shader is used for rendering the *Cut Plane Alignment View* and the *Cut Plane View* we need an approach that is suitable for both views. The most important difference of the two views is the viewing angle. The *Cut Plane View* allows a frontal view through the plane, therefore it is possible that front-faces of objects get sliced away, which is an issue because back-faces are usually not rendered to save computation time. Thus, for the *Cut Plane View* it is necessary to render the back faces of the model too, which would otherwise lead to the phenomenon, that sliced objects get completely cut-away in this view. The problem is illustrated in Figure 4.11a. The solution to this problem is to turn off backface culling in this case, as Figure 4.11b shows.

To take into account, that we need to render opaque and transparent parts in one view, the rendering is done in two passes: First, the model is rendered with $alpha = 1$ (opaque)

(a) *View before Align CutPlane gesture*

(b) *View while executing CutPlane Align gesture*



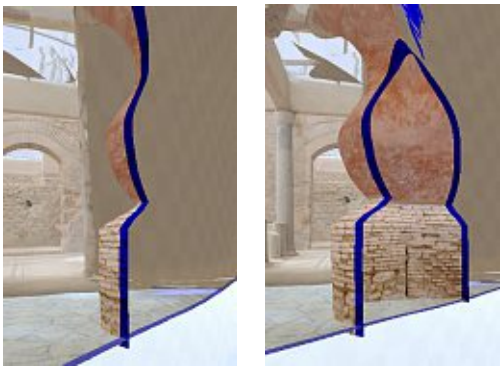
(c) *View while executing CutPlane Align gesture*

Figure 4.10: *X-Ray view in CutPlane Align Mode:left side of the geometry is untouched, right side is visualized semi-transparent, plane cut is visually enhanced on the adjacent geometry with blue coloring*

and backface culling turned off. In this iteration, every fragment that has a relative position (as seen from the viewpoint) in front of the cutplane gets discarded. Then, in a second shader pass the same geometry is rendered with $alpha = 0.3$, (semi-transparent), but this time every fragment that has a view position behind the cut plane gets discarded. Backface culling can be turned on for this pass, as there happen to be no "sliced front faces" in front of the plane. However, for this pass it is necessary to turn off the $z - test$, otherwise we would not be able to get the x-ray functionality in the *Cut-Plane Alignment view*. An enabled $z - test$ would prevent the rendering of occluded geometry at all. The blue cut-plane visualization enhancement is realized by adding blue color to the output if the fragment's world position is within a certain distance to the cut plane. Table 4.1 gives an overview over the different render options.

	1st Pass	2nd Pass
Render Style	Opaque	Semi-Transparent
Backface Culling	Off	On
z-Test	On	Off

Table 4.1: Render options of the two different passes



(a) *Problem: back faces not visible*

(b) *Solution: turn off backface culling*

Figure 4.11: *Disabled backface culling for opaque part of the cutplane*

Unity supports multiple render passes in a surface shader, and the necessary render switches can be specified per render pass, so it can handle every needed pass in one shader, which is attached to the rendering component of the 3d model. The formula, that decides if a pixel is drawn or discarded is outlined exemplary for the opaque case in Algorithm 4.4. The algorithm's input variables P_{plane1} and P_{plane2} are the world positions of two points lying on the plane, and P_{pixel} is the world position of the processed pixel. The transparent case is handled the same way, only with opposite sign.

Algorithm 4.4: Shader calculation that decides if a pixel has to be discarded.

Data: $P_{plane1}, P_{plane2}, P_pixel$
Result: $isDiscarded$

```
1 distance ← ((P_plane2.X − P_plane1.X) ∗ (P_pixel.Z − P_plane1.Z)) − ((P_plane2.Z −
    P_plane1.Z) ∗ (P_pixel.X − P_plane1.X)) if distance > 0 then
2 |   isDiscarded ← true
3 end
4 else
5 |   isDiscarded ← false
6 end
```

4.5.2 Step 2: Transition into Cut Plane View

When the user has found the desired position and rotation of the cut plane, she has to lock it in with the *Commit CutPlane* gesture. The progress, while executing the *Commit CutPlane* gesture to enter or exit the *Cut Plane View* is visually indicated by progress lines on the cutplane itself (see Figure 4.12). After finishing the gesture, a user repositioning, as outlined in Figure 3.7b is made. Since navigation in the Cut Plane View is restricted to plane-aligned movement only, a set of unity collider walls must be

generated instantly to create a small plane-aligned corridor where the user can move and the default colliders must be deactivated (see Figure 4.13).

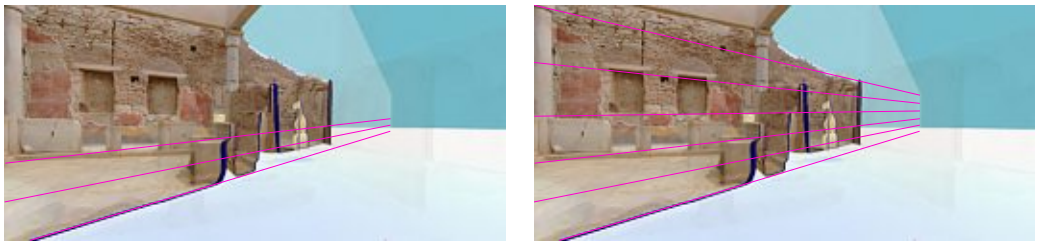


Figure 4.12: *Progress lines: Visual indication of gesture progess*

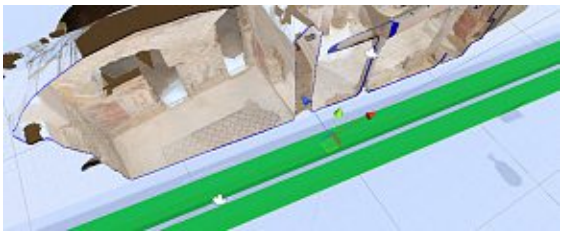


Figure 4.13: Custom colliders to restrict movement (colliders in green)

4.5.3 Step 3: Raycast in Cut Plane View

While staying in the *Cut Plane View* the user is able to draw paint strokes onto the geometry with the execution of the *Pointing gesture*. Drawing paint strokes is also possible when the comparing base line technique *Raycast* (as described in Section 5.1.1) is active. In that case, a Unity Physics Raycast is executed and the first found geometry hitpoint can be used. However, if the *Cut Plane View* is active, a straightforward Raycast is not possible any more because of two reasons: First, our Cut Plane View technically contains all the geometry from the scene, even if a part of it is rendered transparent. The first hitpoint could land on a transparent part in front of the plane, but we want to draw our paint strokes behind the cut plane. And second, even if there is no geometry in front of the cut plane we could hit a "sliced object" and Unity would ignore the hitpoint because its Raycast functionality does not take back faces into account. So we use a *Double Raycast* approach. It consists of one forward ray that stores all front-facing-hits and one inverted backward ray, that starts at the hit point with the maximal Euclidean distance to the user's position, points towards the user's position and stores all back-facing-hits, again only selecting the one with the minimal Euclidean distance to the user but behind the cut plane. Figure 4.14 explains the concept.

The resulting series of hitpoints is then used to perform a segmentation on the geometry.

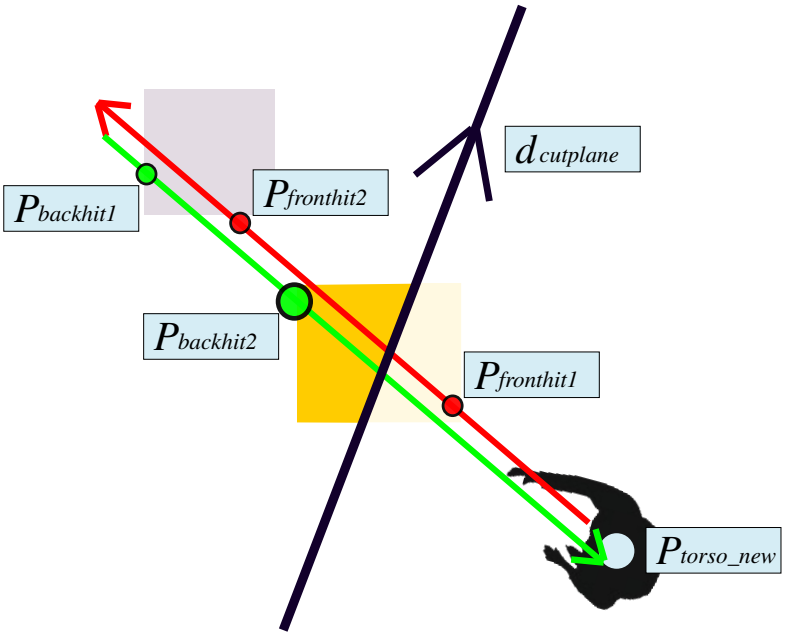


Figure 4.14: Double Raycast. Chosen hitpoint is the bigger green point from the backward ray. It is the point with the minimal Euclidean distance to the user but behind the cut plane.

4.6 Implementation of Segmentation

Our segmentation method is composed of 5 subsequent steps, as Figure 4.15 describes.

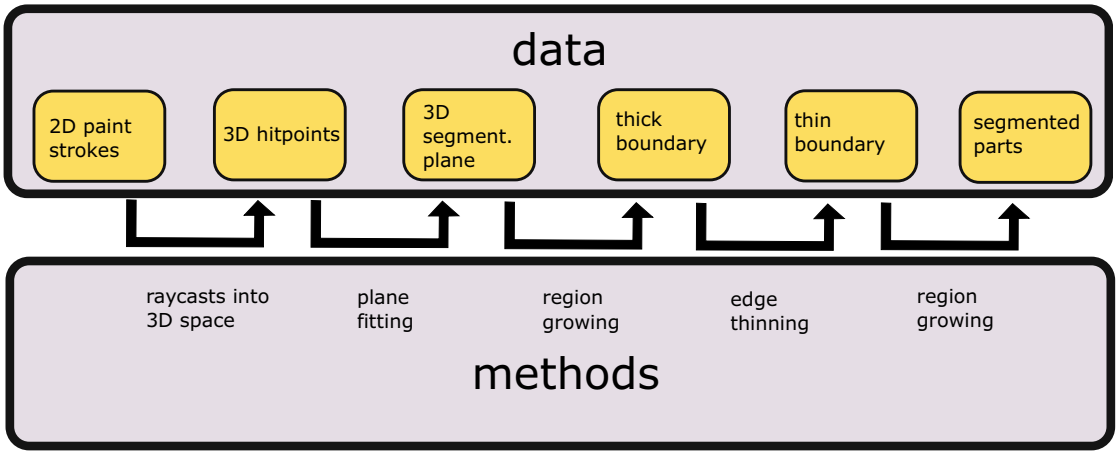


Figure 4.15: The pipeline of the segmentation method.

There are two parts in the segmentation pipeline that use some form of the region growing algorithm. Both parts, namely the creation of a thick boundary (as described in Section

3.3.3) and the marking of vertices to find the segmentation parts (7) are dependent on a suitable data structure to quickly navigate on the graph of edges and vertices. For this reason, we incorporate a pre-processing step on those parts of the environment where a segmentation at runtime should be possible. As a limitation, like stated in Section 3.2.1 unity splits the dense reconstruction into sub-meshes and thus building a connected neighbour structure is only possible inside those sub-meshes. That limits also the later segmentation in the application to specific regions of the scene. The data-structure to quickly find neighbours of vertices is built up by traversing the array of triangles that is provided by unity for each mesh. Algorithm 4.5 shows the process. This pre-processing step is done at every startup of the application, but could be sourced out to an external process which caches the neighbour graph in a file as long as the input meshes stay the same.

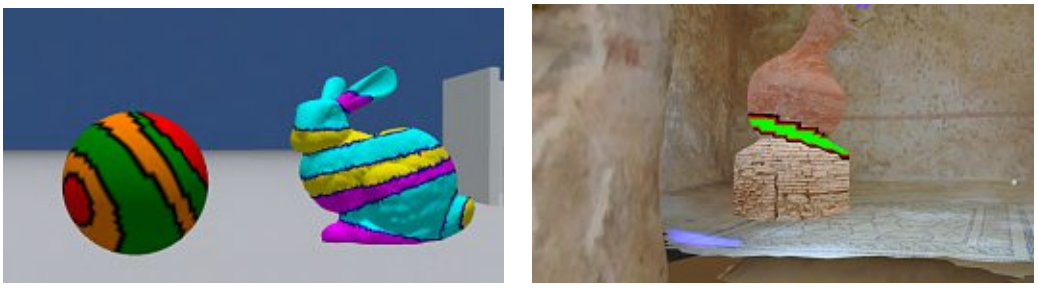
Algorithm 4.5: Create vertex-neighbour graph structure

Data: $arr_{triangles}$, P_{hand}
Result: $dic_{neighbours}$
foreach $arr_{triangle}$ *in* $arr_{triangles}$ **do**
  $dic_{neighbours}[arr_{triangle}[0]]$.add($arr_{triangle}[1]$)
  $dic_{neighbours}[arr_{triangle}[0]]$.add($arr_{triangle}[2]$)
  $dic_{neighbours}[arr_{triangle}[1]]$.add($arr_{triangle}[0]$)
  $dic_{neighbours}[arr_{triangle}[1]]$.add($arr_{triangle}[2]$)
  $dic_{neighbours}[arr_{triangle}[2]]$.add($arr_{triangle}[0]$)
  $dic_{neighbours}[arr_{triangle}[2]]$.add($arr_{triangle}[1]$)
end

With that structure in place, a region growing algorithm simply can loop through all neighbours of a vertex and call itself recursively until all vertices are visited. After the creation of a thick boundary the edge thinning process, as described in Section 7 is supposed to narrow down the boundary until it only consists of a thin edge loop. However, while performance-testing the pipeline, the iterative removal and the subsequent boundary-correctness-check after each iteration, broke in some cases the interactivity requirement as stated in 3.1.6. As it turned out, the skip of that step does not harm the overall segmentation process. The parametrizable distance threshold *thr* from section 3.3.3, which influences the width of the resulting boundary corridor in the boundary creation step can be optimized, until a reasonable thin boundary, that fits our purposes is found. Our region growing algorithm to find segmentation parts does not necessarily need a complete thinned out boundary as input. Figure 4.16 demonstrates the resulting boundaries.

4.6.1 Selection and Target Patch Creation

After a successful segmentation and the logical division of an object in segmented parts, a physical splitting of the mesh is necessary for two reasons: first, if a subsequently



(a) *Multiple segmentation boundaries on test models*

(b) *Segmentation boundary on the actual scene*

Figure 4.16: *Progress lines: Visual indication of gesture progess*

selection happens after the segmentation, we move the cut out part towards the user as result of the selection process. So this part has to be a an own detached object to enable the needed transformations on it. Second, if the user's way is blocked by geometry, a selection gesture cuts away the obstacle. The splitting itself is done easily in unity, such that a new object with an attached mesh can be created in runtime without interrupting interactivity. Though, a problem arises when creating the new colliders that are needed to update the engine's collision detection information. If a mesh-collider is created from the cut geometry, the user can get stuck on them because the engine's internal collision physics algorithms are not very robust against non-closed-objects. The solution to this problem is to automatically generate a closed box-collider out of the non-closed mesh-collider that serves as user's collision object. The mesh-collider itself is needed too, because the Physics Raycasting needed for our segmentation-paint-strokes has to work with exact geometry, the box-collider is not applicable for that. To get this combination of different colliders working without interfering each other, we define physics layers in Unity, where the Raycast only takes collision information from *layer*1 into account, the player collision indeed the colliders from *layer*2. Figure 4.17 shows a subsequent segmentation and selection process. After a part is selected, the specific geometry is cut out from the rest.

4.7 Implementation of the VR-User Interface

As described in Section 3.2.2, the user is provided with additional input possibilities in form of an overhead menu to prevent the needed creation of a new gesture for every new input action. The overhead UI is created as a set of camera-aligned unity buttons with attached colliders to recognize a raycast hit from a pointing gesture. If the raycast's direction hovers over a button longer than a defined dwell time *thr*, a certain action is executed. The change between selection and segmentation technique, as well as executing undo-commands can be performed that way. An example of the overhead UI is shown at Figure 4.18.

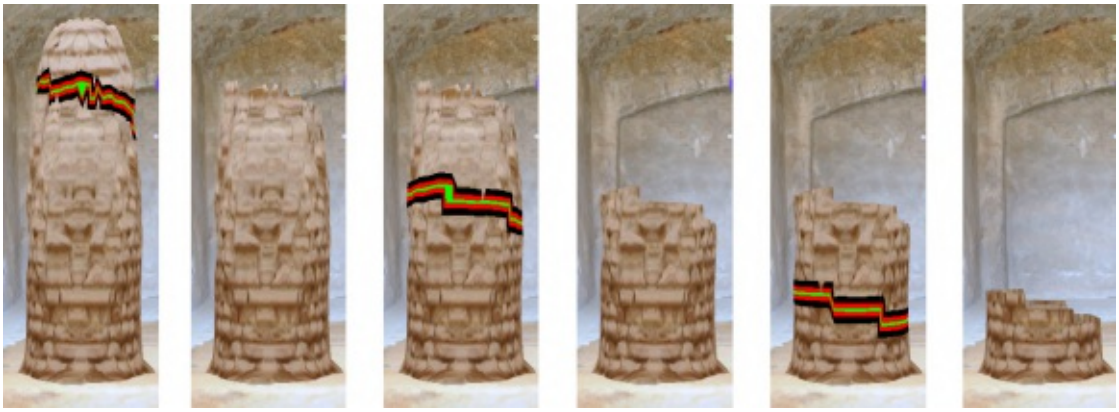


Figure 4.17: Subsequent segmentation-selection process

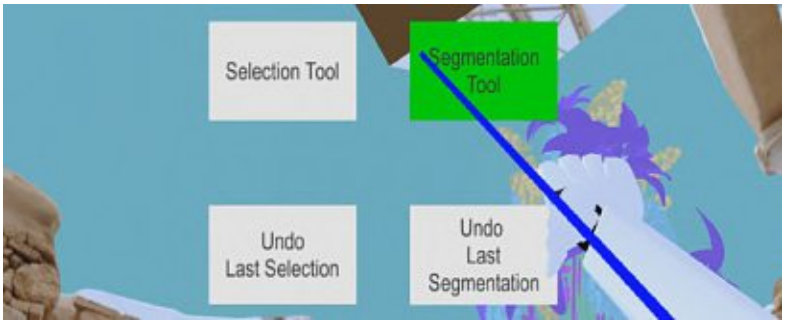


Figure 4.18: The overhead UI

4.8 Software Components

Figure 4.19 shows the software components, structured by device input components, processing in Unity, and output generation. Inside the Unity box the internal software flow of the core components is displayed.

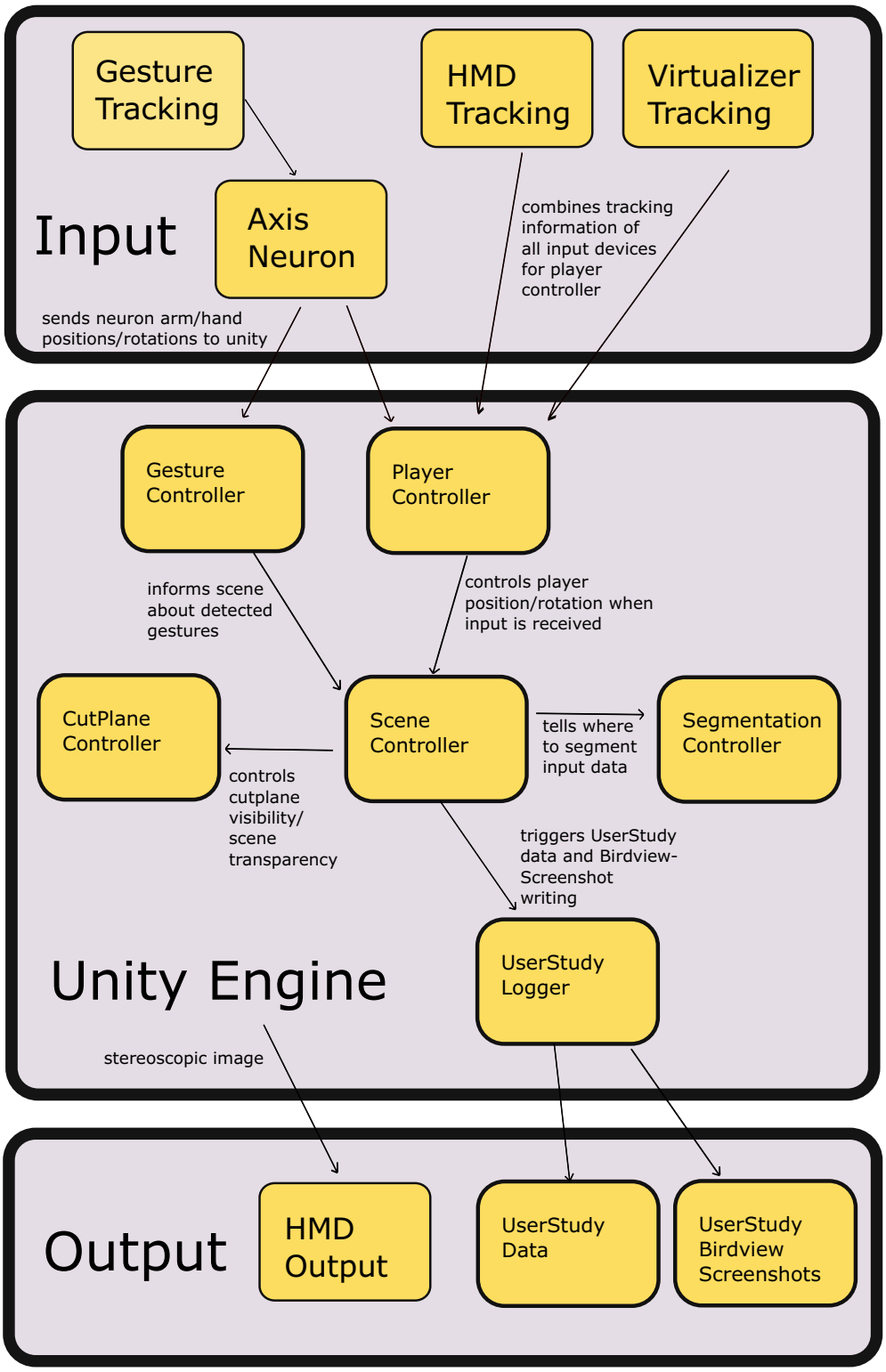


Figure 4.19: The software information flow

CHAPTER 5

Experimental Results

For a comprehensive evaluation of this thesis' work we conducted a quantitative and qualitative evaluation by comparing the Large Scale Cut Plane selection technique with the baseline technique *Raycast* – as described in Section 5.1.1 – across three different selection scenarios based on variations of patch position and visibility. As target patches, we used convex parts of the scene that can be segmented by defining a straight boundary.

5.1 Setup

5.1.1 The Raycast-Technique as Comparison

The well-studied Raycast technique [BKLP04c] was implemented as a baseline technique for comparative purposes, used for our user study. Raycast comprises the same sequence as the Large Scale Cut Plane technique, but without setting a cut-plane and execution of the transition into the geometry-reduced refinement view. Thus, it consists only of the segmentation and selection step which are both performed with the Pointing gesture.

5.1.2 Objectives

With this study, we want to investigate users' performance and perception when selecting patches from large dense 3D reconstructions while being immersed. Therefore, we evaluated both *Large Scale Cut Plane's* performance and usability compared to a state-of-the-art technique by investigating the following research questions:

1. How does the two-step technique *Large Scale Cut Plane* quantitatively and qualitatively perform compared to *Raycast*?

2. How do both techniques perform regarding users preference and level of expertise?

5.1.3 Apparatus

To conduct our experiment, we used the test apparatus as shown in Figure 5.1a. It comprises of the devices integrated into the research prototype, the Oculus Rift DK2 as HMD, the Cyberith Virtualizer as an omni-directonal treadmill to allow natural walking for virtual travel and Perception Neuron [Noi15] as motion capture suit to track users' right hand and upper limb motion.

Implementation

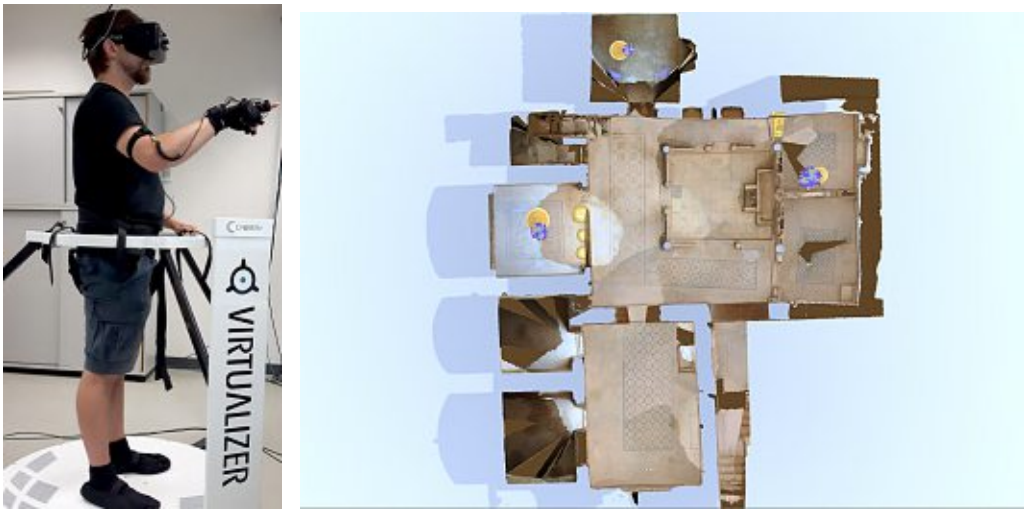
The devices of our apparatus are connected to a single workstation running Windows 8.1 featuring an Intel Core i7-4790K processor (3.10 GHz), a GeForce GTX 980Ti graphics card and 32 GB memory. On the software side we chose Unity 3D as rendering and physics engine, and integrated both Oculus' and Virtualizer's SDK for scene viewing, head orientation tracking and natural walking. We also integrated the Perception Neuron's Unity-Plugin, which provides a scene-graph including the world-position of all necessary arm and hand parts with low latency and implemented our gesture recognition on top of it. With the described setup, the dense 3D reconstruction can be explored at 60 frames per second, the segmentation of a scene part with 21000 vertices can be performed in 480 milliseconds (20ms for region growing, 180ms for edge thinning, 260ms for physical removal of the part).

5.1.4 Study Design

The study procedure for a single user consisted of four stages: 1) introduction and pre-questionnaire, 2) training phase, 3) experiment, and 4) a post-questionnaire. At stage 1, users were informed about the study and the procedure, followed by filling out a pre-questionnaire. At stage 2, users were introduced to the VR input- and output hardware as well as to the gesture set. Next, users were equipped with the VR hardware and had time to familiarize with the hardware and both selection techniques by freely interacting in a test environment, which comprised a simple Unity3D scene with some artificial virtual objects that could be segmented and selected and where objects' visibility ranged from visible to fully occluded. As soon as the user reported to feel confident, the experiment stage (3) started, comprising a randomized sequence of technique and task combinations. Therefore, the user was positioned at a pre-defined starting point within the immersive 3D point cloud while an info screen in the user's HMD gave instructions about the appearance of the target patch and the selection technique. Each task was completed as soon as the user had correctly segmented and selected the target patch. Then, the user was positioned at the starting point and was instructed with the next task. Upon the completion of all task and technique combinations, users had to fill out a post-questionnaire (5).

5.1.5 Test Environment

As test environment, we used the dense 3D reconstruction taken from [BMAW13] that comprises approx. 650 000 vertices. A bird's eye view onto the test environment is given in Figure 5.1b where the positions of the target patches for each task can be seen.



(a) Test apparatus. (b) Bird's eye view on the virtual scene.

Figure 5.1: The test setup

The model was provided as a textured mesh and covered a space of 25×34 m. After importing the mesh into Unity3D, we scaled it so that $1\,Unity3D\,unit \approx 1\,meter$. Furthermore, we manually added Unity3D colliders to the dense scene to prevent users from walking through walls. With the setup from Section 5.1.3, the test model can be explored at 60 frames per seconds. A test patch comprising 21.000 vertices can be segmented with 200ms and extracted and thus selected within 260ms.

5.1.6 Tasks

We built three task scenarios that varied in target patch visibility and position. All task scenarios were embedded in the test environment from Section 5.1.5. The target objects of the task scenarios can be seen in Figure 5.2. During all tasks, test participants were able to freely explore the scene but unable to walk or see through walls and other obstacles. To prevent users' disorientation in the large scene, we restricted segmentation and selection to several key areas that were crucial for task completion. As a guidance for the user the geometry of those areas was highlighted in yellow. The three scenarios differ in the placement of the target patch, which in every case was a vase filled with flowers. The goal of every task was to separate the vase from its pedestal with a horizontal segmentation gesture followed by the selection of the segmented part of the vase. The participants had to use both *Raycast* and *Large Scale Cut Plane* in combination with all

three scenarios. That results in total in six different tasks which the participants perform in random order. The difficulty of each task is given by the visibility and accessibility of the task's target patch.

No Occlusion - Object Fully Accessible

In this task, the vase with the flowers was not in the spatial vicinity of users, but clearly visible from the starting point. With both techniques, users had the option to first walk closer towards the target object or – since it is not occluded – try to segment and select it from far distance.

(a) Fully visible (b) Partly occluded (c) Fully occluded

Figure 5.2: Different types of target objects.

Partly Occluded - Object Partly Accessible

Users were challenged with a target patch that was partly hidden behind three stone pillars. With Raycast, it was either possible to segment and select one or more pillars to obtain a better view onto the target patch, or to try to directly segment the semi-hidden target vase between the pillars. The second approach required very precise positioning as well as segmentation. With *Large Scale Cut Plane* it was possible to induce such a CutPlane Visualization that presents the target patch fully visible and segmentable during the refinement step.

Fully Occluded - Object Not Accessible

In this task, the vase was fully occluded by a wall within a separate room that entrance was blocked by a wall; only the flowers were partly visible above the walls to indicate the position of the vase. With Raycast, users first had to unblock the entrance by cutting through the wall. With CutPlane, users could induce such a CutPlane Visualization that

removed parts of the rooms to be able to directly access the vase by traveling along the CutPlane during the refinement step.

Methods

We conducted the study using a within-subjects factorial design where the independent variable was selection technique. In a second order evaluation, we introduced user experience and favorite technique as third and forth independent variable. For analysis, we both employed quantitative objective and subjective measures that are outlined in Table 5.1.

[Task Duration]	Accumulated time across all tasks in seconds.
[Walk Distance]	Sum of distances users walked during all tasks in meters.
[Segmentation Miss]	Number of performed segmentation attempts in areas that are non-relevant for task completion.
[Selection Miss]	Number of performed selection attempts in areas that do not belong to the target patch.

Table 5.1: Objective Performance Measures.

In a post-questionnaire, we measured subjective measures as described in Table 5.2, using a 5-point Likert scale rating:

Furthermore, we collected qualitative data through users' comments and observations during the experiment as well as written comments in the post-questionnaire.

5.2 Experimental Results

The quantitative objective and subjective data gathered from the test application and the questionnaires were analyzed using repeated measures single factor ANOVA with repeated contrast and Bonferroni confidence interval adjustment. To judge for significance, we evaluated the F-ratio using both Huynh-Feldt and Greenhouse-Geisser adjustments; only if both indicated a F-ratio below the significance value $p = 0.05$, we accept this F-ratio as significant.

During the data analysis, we focused on 1) the evaluation of of all participants' data regarding the employed selection techniques and 2) we looked at data of selected participants - according to their experience - for each selection technique separately.

[Task Load]	Mean of the questionnaire's results measured with the NASA Task Load Index (TLX) [NA10]. Scale levels were: very low (1), low (2), average (3), high (4), and very high (5).
[System Usability]	Mean of the questionnaire's results measured with the System Usability Scale (SUS) [Bro96]. Scale levels were: strongly disagree (1), disagree (2), neither agree nor disagree (3), agree (4), and strongly agree (5).
[Ease of Use]	Denoting participants' degree of agreement with "Rate each selection technique in terms on how easy it was to select the target patch". Scale levels were: Very poor (1), poor (2), acceptable (3), good (4), very good (5).
[Perceived Speed]	Denoting participants' degree of agreement with "Rate each selection technique in terms on how fast you could select the target patch", with the same scale levels as above.
[Perceived Accuracy]	Denoting participants' degree of agreement with "Rate each selection technique in terms on how accurate you could select the target patch", with the same scale levels as above.

Table 5.2: Subjective Performance Measures.

5.2.1 Participants

Our test participants were found through local social media groups as well as amongst students and staff of the university. Twenty-four (24) participants (19 males – 79,2%, 5 females – 20,2%) were involved in the experiment, while all participants successfully

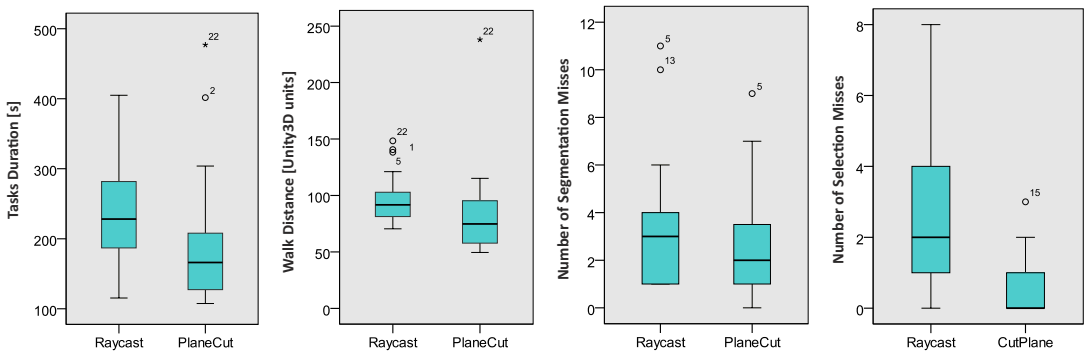
finished the it. Participants' ages ranged between 22 and 56 years (mean $\mu = 31.92$, standard deviation $\sigma = 7.87$).

We asked all participants in the pre-questionnaire about their experience with VR technology. One user indicated to have *no* (4,2%), seven *slight* (29,2%), four *somewhat* (16,7%), eight *moderate* (33,3%) and four *extreme* (16,7%) VR experience. For later evaluation, we grouped users according to their subjective experience, creating two extreme groups: *No Experts* comprises users that indicate to have *no* or *slight* VR experience (33,4%), as well as *Experts* that consists of users indicated to have *moderate* or *extreme* VR experience (50,0%).

5.2.2 Overall Evaluation

In average, users trained both handling of the VR hardware and the two selection techniques for $\mu = 399.17$ [seconds] ($\sigma = 128.331$). To finish all tasks, users required in average $\mu = 394.11$ [seconds] ($\sigma = 26.36$) where they walked in average $\mu = 179.62$ [m] ($\sigma = 11.09$) while they performend $\mu = 5{,}83$ ($\sigma = 0.91$) segmentation misses and $\mu = 2.95$ ($\sigma = 0.42$) selection misses.

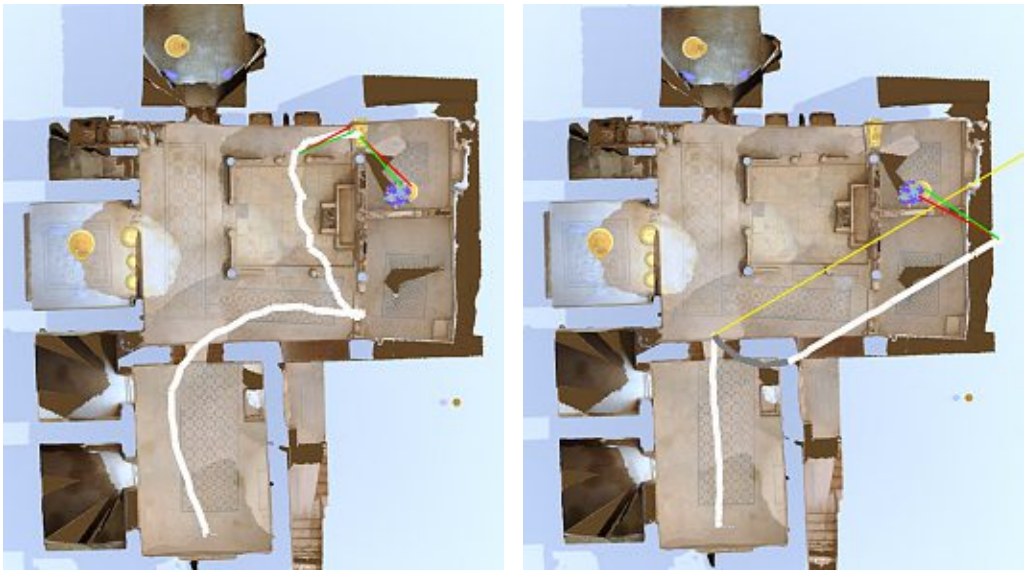
When analyzing these objective quantitative measures by technique, we found significant differences between Raycast and CutPlane in all performance measures, as shown in Figure 5.3.



(a) Task Duration (b) Walk Distance (c) Segm. Misses (d) Sel. Misses

Figure 5.3: Objective measures for both techniques across all tasks.

CutPlane outperforms Raycast in *[Task Duration]* with $p = 0.025$ (Raycast: $\mu = 239.62$, $\sigma = 76.56$ / CutPlane: $\mu = 191.57$, $\sigma = 91.43$), *[Walk Distance]* with $p = 0.031$ (Raycast: $\mu = 96.83$, $\sigma = 20.85$ / CutPlane: $\mu = 82.78$, $\sigma = 38.62$), *[Segmentation Miss]* with $p = 0.032$ (Raycast: $\mu = 3.41$, $\sigma = 2.70$ / CutPlane: $\mu = 2.41$, $\sigma = 2.22$), and *[Selection Miss]* with $p = 0.00$ (Raycast: $\mu = 2.54$, $\sigma = 2.06$ / CutPlane: $\mu = 0.41$, $\sigma = 0.77$). An example of the user's interaction in Task 3 – both with Raycast and with Cutplane – while selecting a fully occluded patch is illustrated in Figure 5.4. Travel trajectories, CutPlane interactions, segmentation (red) and selection (green) attempts are shown.



(a) Raycast (b) CutPlane

Figure 5.4: User interaction while selecting a fully occluded patch.

Analyzing the subjective data, participants reported a significant difference between Raycast and CutPlane for *[Task Load]* with $p = 0.013$. To draw further conclusions, we looked separately at the results of each question of the TLX questionnaire and found significance for *"How physically demanding was the task?"* with $p = 0.010$ (Raycast: $\mu = 2.50$, $\sigma = 1.06$ / CutPlane: $\mu = 2.00$, $\sigma = 0.83$), and *"How hard did you work to accomplish your level of performance?* with $p = 0.015$ (Raycast: $\mu = 3.00$, $\sigma = 0.78$ / CutPlane: $\mu = 2.58$, $\sigma = 0.82$). No significance was found within the other TLX questions. These results indicate that users felt to spend significantly less physical effort across all tasks when using CutPlane compared to Raycast, while feeling alike with both techniques in terms of success and confidence. This latter tendency is further backed by the results of the overall subjective measure *[System Usability]*, where no significant difference between Raycast and CutPlane was found ($p = 0.087$). We further found no significant differences between Raycast and Cutplane for the detailed subjective measures *[Perceived Speed]* with $p = 0.765$ (Raycast: $\mu = 4.12$, $\sigma = 0.79$ / CutPlane: $\mu = 4.21$, $\sigma = 0.93$) and *[Perceived Accuracy]* with $p = 1.00$ (Raycast: $\mu = 4.00$, $\sigma = 0.88$ / CutPlane: $\mu = 4.00$, $\sigma = 0.72$). In contrast, users reported a significant easier selection (*[Ease of Use]*) using Raycast compared to CutPlane with $p = 0.001$ (Raycast: $\mu = 4.54$, $\sigma = 0.65$ / CutPlane: $\mu = 3.83$, $\sigma = 0.86$).

This subjective tendency was backed by comments from eight users describing Raycast as very easy to understand and "direct". However, there was also mixed feedback for Raycast. Three users felt annoyed when they needed to use RayCast ("So in this task I cannot use CutPlane?") and two commented on the physical demands ("Oh that requires so much walking and cutting."). Furthermore, five users commented on the obstacles within their

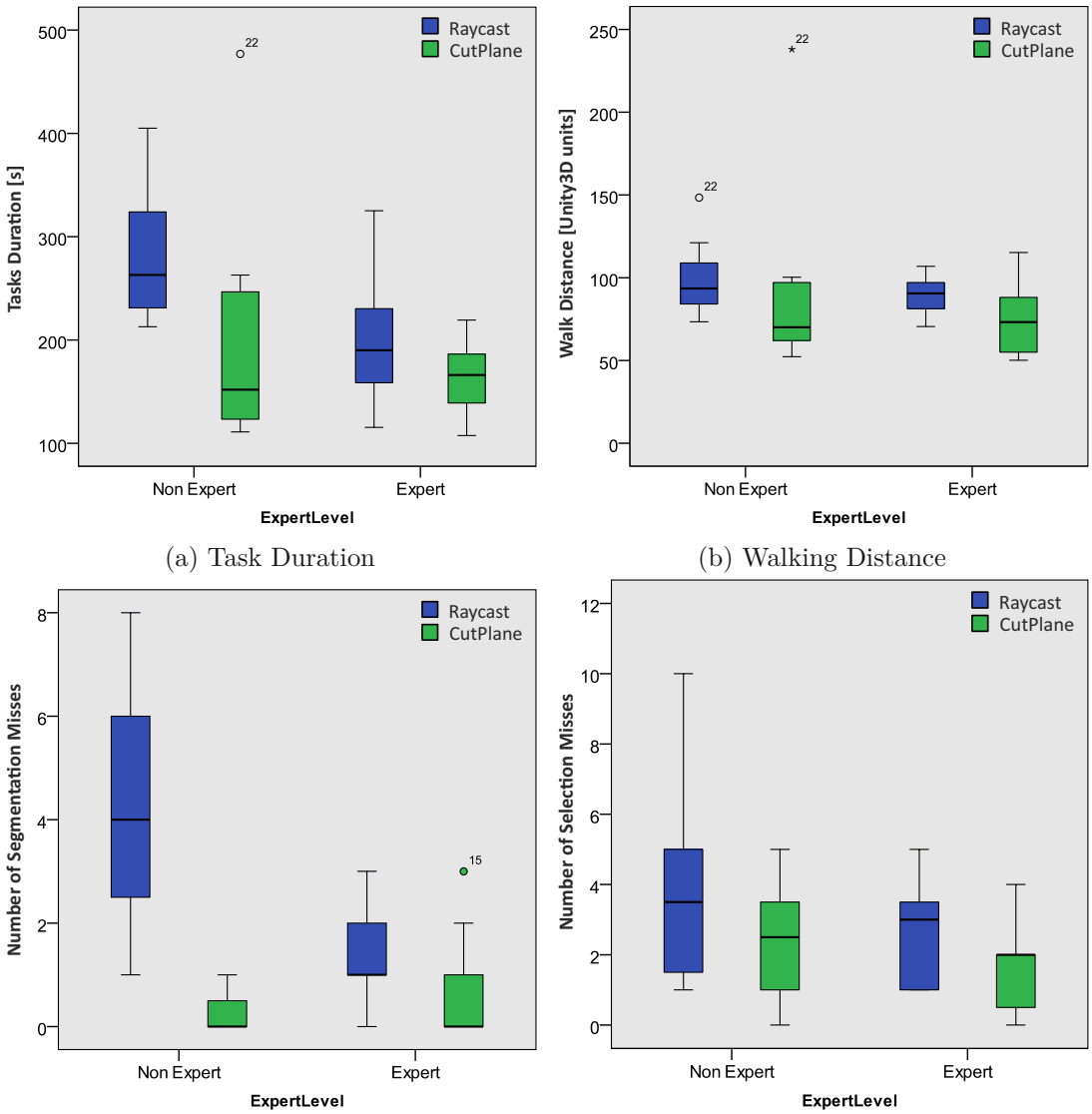
way to the target object ("Oh that is mean (the obstacle), "I cannot directly select the object. So do I need to cut through the wall first?" "Can I walk through the walls?"). Despite the clear out-performance of CutPlane regarding the quantitative measures, its qualitative feedback was indifferent as well. In general, users commented more positively about CutPlane the longer the experiment lasted ("Ah, now I understood." "Now I see the advantages of the CutPlane visualization"), which is notably since all participants reported to have understood the principles of both techniques in the training stage. Nine users commented in a very enthusiastic way on the CutPlane visualization ("That's so cool - that is kind of an X-Ray view"), while one did not find the interface intuitive ("I do not really understand the CutPlane visualization") and two asked – during the experiment – which part of the scene will be shown during the CutPlane visualization. While some users did not realize across all tasks that you can view through walls with the CutPlane Preview (4 users) and then walk through walls (which has been cut by the cut plane) within the CutPlane Visualization (1 user), ten users heavily used the CutPlane technique by indicating the plane cut from the initial position before even start to walk.

5.2.3 Evaluation on Preference & Expertise

To further analyze the above quantitative and qualitative findings, we evaluated the collected data regarding the users' experience level. Firstly, we found that Non Experts preferred Raycast over Cutplane (87,5%), while Experts favored Cutplane over Raycast (66,6%). This is notable, since both user groups had better results with CutPlane regarding all objective performance measures, as given in Figure 5.5. It can be further seen that Non-Experts achieve similar results with CutPlane across all measures as the Experts.

Analyzing the subjective measures, Non Experts reported a similar perceived *[Task Load]* for Raycast (μ = 2,43, σ = 0,43) and CutPlane (μ = 2.375000, σ = 0.54), while Experts perceived more *[Task Load]* for Raycast (μ = 2,62, σ = 0,62) than for CutPlane (μ = 2,36, σ = 0,47). Non Experts showed the tendency to clearly prefer Raycast over CutPlane in terms of *[Ease of Use]* (Raycast: μ = 4.38, σ = 0.744 / CutPlane: μ = 3.38, σ = 0.775) and *[Perceived Speed]* (Raycast: μ = 4.38, σ = 0.743 / CutPlane: μ = 3.75, σ = 0.707) but not in terms of *[Perceived Accuracy]* (Raycast: μ = 3.63, σ = 1.061 / CutPlane: μ = 3.63, σ = 0.518). Experts did not indicate a clear preference between both techniques in terms of *[Ease of Use]* (Raycast: μ = 4.56, σ = 0.669 / CutPlane: μ = 4.17, σ = 0.835) or *[Perceived Accuracy]* (Raycast: μ = 4.25, σ = 0.754 / CutPlane: μ = 4.08, σ = 0.793). However, they reported to prefer Cutplane over Raycast in terms of *[Perceived Speed]* (Raycast: μ = 4.00, σ = 0.853 / CutPlane: μ = 4.50, σ = 0.793).

When analyzing users' preference for one of the tested techniques, we further asked in the post-questionnaire about the factors that users influence during their subjective technique ranking. Therefore, they had to indicate on a 5-point Likert scale their perceived influence of *[Importance: Ease of Use]*, *[Importance: Speed]*, *[Importance: Accuracy]*, the possibility to straightforwardly select objects at far distance – referred as *[Importance: Distance]* – and the straightforward selection of fully occluded objects, referred as *[Importance:*



(a) Task Duration

(b) Walking Distance

(c) Segmentation Misses

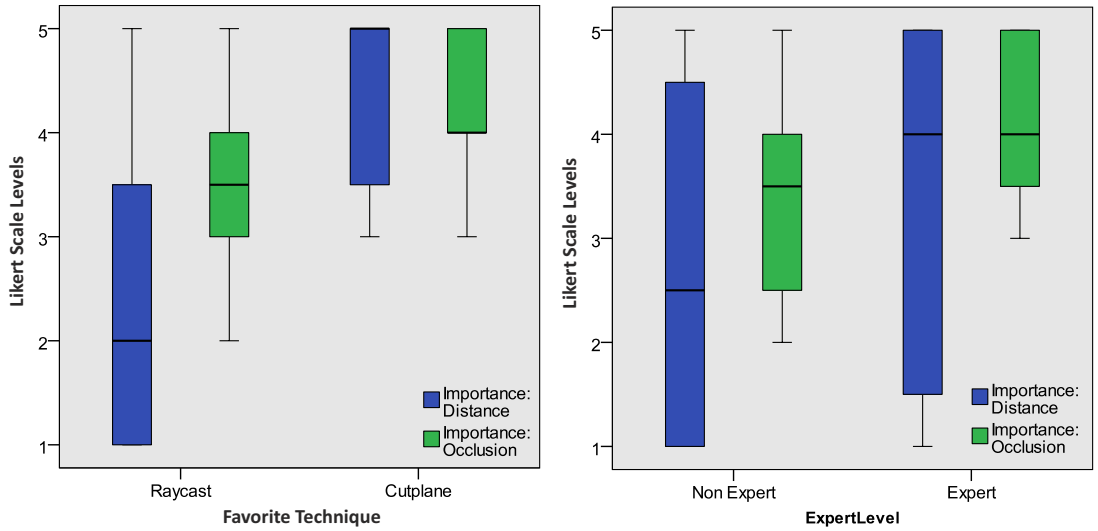
(d) Selection Misses

Figure 5.5: Objective performance measures, grouped by expertise.

Occlusion]. Scale levels were: not important (1), slightly important (2), moderately important (3), important (4), and very important (5).

While there was no noticeable difference in influence of *[Importance: Ease of Use]*, *[Importance: Speed]* and *[Importance: Accuracy]*, users in favor of Raycast did not regard *[Importance: Distance]* as a strong influential factor on their decision making ($\mu = 2.33$, $\sigma = 1.557$), while users favoring CutPlane found that measure much more influential ($\mu = 4.00$, $\sigma = 1.537$). Notably, not only users favoring Cutplane regarded *[Importance: Occlusion]* as an important factor for their decision making, but also those in favor of

Raycast but with a less strong tendency (Raycast: $\mu = 3.42$, $\sigma = 0.900$ / CutPlane: $\mu = 4.25$, $\sigma = 0.754$). Details are given in Figure 5.6a. Furthermore, when looking at the users expertise, we found that Non Experts tend to rate both measures less important than experts, as shown in Figure 5.6b.



(a) By Favorite (b) By Expert

Figure 5.6: Influencing factors for technique ranking.

5.2.4 Discussion

Evaluating the objective performance measures, tendencies for all measures favoring CutPlane over Raycast were reported, in particular in cases of more complex selections, reflected by Task 2 and Task 3. While no significant difference between the techniques was reported on [Walk Distance] across the tasks, CutPlane outperformed Raycast in [Task Duration], significantly for Task 2 and with a strong tendency for Task 3. Thus, Raycast cannot compensate with its direct nature the additional time that is required for cut-away of obstacles in scenarios where the target patch is partly or fully occluded. In Task 1 with no object interaction, we reported a tendency favoring Raycast's direct nature over CutPlane, however not significantly. Furthermore, we found strong tendencies that Cutplane enables users to select more precisely and faster, as less segmentation and selection misses occurred when using CutPlane compared to Raycast, in particular for the complex selections in Task 2 and Task 3. Here, significant less errors could be reported in Task 2 using CutPlane compared to RayCast. The above findings can support developers and researchers when designing selection within a large immersive 3D reconstruction. We plan to conduct more research in smaller as well as even larger environments to identify thresholds at which one technique begins to outperform the other. As it was described above, qualitative feedback on both techniques was mixed. This is also reflected by the

fact, that no technique was subjectively favored over the other. In fact, tendencies of preference were found when looking at users' VR knowledge, where Non-Experts favored RayCast while Experts preferred CutPlane. This is surprisingly due to the fact, that also the Non-Experts performed significantly faster and more precise with CutPlane. However, these results suggest that Non-Experts can quickly familiarize with CutPlane which indicates that the technique's underlying properties are well integrated and proved to be beneficial for task completion. This enables Non Experts to correctly use it, however additional investigations should be conducted regarding CutPlane's user interface. Based on the comments, we found that both CutPlane Preview and CutPlane Visualization might invoke a lack of understanding. We will tackle this point in our future work. To conclude, we can suggest to rather use CutPlane for selection of visible, partly as well as fully occluded patches of dense point clouds as it was found superior over Raycast.

CHAPTER 6

Conclusion

In this thesis, the novel selection technique *Large Scale Cut Plane* was presented, that enables users to select visible, partly as well as fully occluded patches with interactive frame rates in immersive large dense reconstructions. It combines the visualization design patterns *Multiple Views* and *Virtual X-Ray*. To the authors' best knowledge, no prior art exists that combines those patterns in a scenario of user-driven selection of target patches within a large and dense 3D reconstruction that is immersively explored.

6.1 Contribution

The contribution of this thesis consists of the following items:

1. **Large Scale Cut Plane** We designed and developed the occlusion management technique *Large Scale Cut Plane* that provides means for target patch discovery and access in dense large 3D reconstructions while preserving the spatial relation between the target patch and its context. The technique is based on the visualization design patterns *Virtual X-Ray* and *Multiple Viewports*([ET08]). The *Large Scale Cut Plane* technique enables selection of visible, partly as well as fully occluded target patches while being fully context preserving during the refinement phase to allow for enhanced scene understanding.
   The workflow of the *Large Scale Cut Plane* technique required the following implementation tasks to be done:

   a) enable the navigation with the help of an ODT inside a dense reconstructed environment with correct collision detection

   b) develop the motion tracked gestures to indicate CutPlane Alignments

   c) provide a correct visualization of the CutPlane Alignment View

d) perform the transition of the user into the CutPlane View

e) implement the changed navigation behavior reduced to one-axis movement inside the CutPlane View

f) detect and process the gestures that induce painting onto the geometry to indicate segmentation boundaries

g) develop the segmentation algorithm, which works based on those raycast hitpoint results

h) implement a mesh separation algorithm to generate separate selectable objects across the found boundary

i) enable the selection of the segmented parts

2. **Segmentation Algorithm** As a necessary pre-requisite to the selection step, segmentation of the 3D reconstruction was a requirement. Several segmentation algorithms (user-input-driven and automatic methods) were evaluated and as a result we developed a segmentation technique that fitted our needs regarding processing time and flexibility, especially for large scale dense reconstructions. The segmentation algorithm was designed to operate on local raycast hitpoint results on the surface of the target patch and tried to find a segmentation boundary only by using information in spatial proximity of the given user's paint stroke.

3. **Preparation of a reconstructed Scene** To use a dense reconstructed scene with the Unity engine, some pre-processing steps had to be done. The importing of the mesh and the preparation with correct and usable collision detection objects was necessary.

4. **Partial Geometric Transparency for CutPlane Alignment View** We explored the possibilities of the Unity Engine to create the necessary visual adaptations for the CutPlane Alignment View, since it was crucial to present the user a mixture of opaque and transparent parts of the geometry at the same time within acceptable real-time rendering times. We developed and integrated our own cut-out shaders to achieve those results.

5. **Combination Of VR Devices** The VR setup consisting of a HMD, an ODT and a hand gesture recognizing motion suit, which is used to enable interaction in the Immersive VR environment, was combined in an effort to provide the user a frictionless VR experience. Natural walking inside the ODT, while turning the head into another direction, and the simultaneous execution of arm and hand gestures was made possible. All of these systems independent tracking information had to be joined in the correct way.

6. **Easy to learn and accurate Gesture System** A Gesture System with two different main gestures was developed to give the user a natural, but efficient way to interact with the VR environment. The gestures were designed in a way to provide first-time users an easy learning curve, and nevertheless provide more

experienced users accurate tools to interact with the virtual environment. The technical implementation tried to allow performing the necessary movements in an imprecise way and nevertheless provide execution of the correct action in the system. The gesture system was complemented with an overhead button-click-menu for additional tasks to prevent the necessity of learning too many gestures at once.

7. **User Study** A user study was conducted, that explored occlusion management in dense large 3D reconstructions by statistically evaluating the novel *Large Scale Cut Plane* technique in terms of ease of use and performance and compared with a state-of-the-art technique as baseline. With this user study we evaluated the technique in terms of speed, precision and ease of use, both quantitatively and qualitatively. Our promising results indicate that *Large Scale Cut Plane* outperforms the prior art technique *RayCast*, significantly for more complex selection tasks. However, is is not considered as preferred option by non experts which suggests that improvements of the user interface are required.

6.2 Future Work

The current implementation of the presented *Large Scale Cut Plane* technique explores the basic concept of the presented idea in a very specific and constrained way. It serves as a proof-of-concept of the idea to combine the patterns *Multiple Views* and *Virtual X-Ray*. The overall design space of the combination of those patterns is much larger.

6.2.1 Exploring the Design Space of the Method

As future work, we plan to conduct further research on *Large Scale Cut Plane's* user interface to explore more possibilities of its design space, obtain an in depth understanding of its current limitations and to draw conclusions on how to improve it to enable a better (subjective) understanding of the technique's core functionality for users. The extension of the following parts of the technique could be worth examining:

- Allowing multiple (instead of currently one) geometry cutting steps in sequence in advance of the selection step to further reduce the visible geometry and get a better view of the selection target
- Experimenting with a different cut-away geometry than a plane, for instance applying a cylindrical cut around the desired target patch
- Allowing the user to freely navigate after applying the cutting step instead of being constrained to the direction of the plane

6.2.2 Testing Different Environment Types and Sizes

Furthermore, we plan to evaluate selection in both smaller and larger environments to obtain findings if a threshold exists at which a one step technique such as Raycast begins

to be outperformed by our two step technique. Also the type of the used environment could be varied at a follow-up user study to measure the performance of both methods. Impacts of changes could be analyzed, e.g. if a more open space is used or in the opposite case if a more maze-like environment is used.

6.2.3 Technical Improvements of the Algorithms

Finally, we aim at improving our current segmentation technique to overcome current limitations. The usage of a plane in 3D space as segmentation base, as described in Section 3.3.3 was a design decision to keep the drawing of segmentation boundaries a simple and comprehensible task. Also, by using this technique we could implement the necessary boundary finding algorithm relatively straightforward. If the user wanted to create non-plane based segmentation boundaries, for instance cutting a hole out of existing geometry by drawing a boundary circle, the segmentation method would require a more sophisticated approach.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AA13] Ferran Argelaguet and Carlos Andujar. A survey of 3D object selection techniques for virtual environments. *Computers and Graphics (Pergamon)*, 37(3):121–136, 2013.

[Aut18] Autodesk. Maya. `https://www.autodesk.com/products/maya/overview`, 2018. Accessed: 2018-03-08.

[AVF04] C. Andujar, P. Vazquez, and M. Fairen. Way-Finder: guided tours through complex walkthrough models. *Computer Graphics Forum*, 2004.

[BKLP04a] Doug Bowman, Ernst Kruijff, JJ LaViola, and Ivan Poupyrev. Direct Manipulation: Virtual Hand Techniques. In *3D User Interfaces: Theory and Practise*, chapter 5.4.3, page 158. Addison Wesley, 2004.

[BKLP04b] Doug Bowman, Ernst Kruijff, Joseph J LaViola Jr., and Ivan Poupyrev. Interacting by Pointing. In *3D User Interfaces: Theory and Practise*, chapter 5.4.2, page 150. Addison Wesley, 2004.

[BKLP04c] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[BMAW13] Michael Birsak, Przemyslaw Musialski, Murat Arikan, and Michael Wimmer. Seamless texturing of archaeological data. In *Digital Heritage International Congress (DigitalHeritage), 2013*, pages 265–272. IEEE, October 2013. DOI: 10.1109/DigitalHeritage.2013.6743749.

[Bro96] John Brooke. SUS - A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[CKL04] Charlotte H Campbell, Bruce W Knerr, and Donald R Lampton. Virtual environments for infantry soldiers: virtual environments for dismounted soldier simulation, training and mission rehearsal. Technical report, DTIC Document, 2004.

[CWJ12] J. Cashion, C. Wingrave, and J. J. LaViola Jr. Dense and dynamic 3d selection for game-based virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):634–642, April 2012.

[Cyb13] Cyberith. Cyberith virtualizer. `http://cyberith.com/product/`, 2013. Accessed: 2018-03-08.

[ET08] Niklas Elmqvist and Philippas Tsigas. A taxonomy of 3D occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, 2008.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FLL11] Lubin Fan, Ligang Liu, and Kun Liu. Paint mesh cutting. *Computer Graphic Forum (Proceedings of Eurographics)*, 30(2):603–611, 2011.

[FML12] Lubin Fan, Min Meng, and Ligang Liu. Sketch-based mesh cutting. *Graph. Models*, 74(6):292–301, November 2012.

[GIS18] GISGeography. lidar. `https://gisgeography.com/lidar-light-detection-and-ranging/`, 2018. Accessed: 2018-10-28.

[Goo15] Google. Google cardboard. `https://www.google.com/get/cardboard/`, 2015. Accessed: 2015-11-18.

[HPGK94] Ken Hinckley, Randy Pausch, John C Goble, and Neal F Kassell. Passive Real-world Interface Props for Neurosurgical Visualization. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '94)*, 30:452–458, 1994.

[HTC13] HTC/Valve. Htc vive. `https://www.htcvive.com/`, 2013. Accessed: 2016-03-19.

[ITHG12] Yani Ioannou, Babak Taati, Robin Harrap, and Michael Greenspan. Difference of normals as a multi-scale operator in unorganized point clouds. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 501–508. IEEE, 2012.

[JLCW06] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy Mesh Cutting. *Computer Graphics Forum*, 2006.

[KHS10] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. Learning 3d mesh segmentation and labeling. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 102:1–102:12, Los Angeles, California, 2010. ACM.

[LBLS14] P. Lubos, R. Beimler, M. Lammers, and F. Steinicke. Touching the cloud: Bimanual annotation of immersive point clouds. In *3D User Interfaces (3DUI), 2014 IEEE Symposium on*, pages 191–192, March 2014.

[LG94] Jiandong Liang and Mark Green. Jdcad: A highly interactive 3d modeling system. *Computers and Graphics*, 18(4):499 – 506, 1994.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[MFL11] Min Meng, Lubin Fan, and Ligang Liu. icutter: A direct cut out tool for 3d shapes. *Journal of Computer Animation and Virtual World*, 22(4):335–342, 2011.

[Mic13] Microsoft. Microsoft kinect. `https://dev.windows.com/en-us/kinect`, 2013. Accessed: 2015-09-29.

[MK16] Annette Mossel and Christian Koessler. Large scale cut plane: An occlusion management technique for immersive dense 3d reconstructions. In *Proceedings of the 22Nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, pages 201–210, Munich, Germany, 2016. ACM.

[MVK13] Annette Mossel, Benjamin Venditti, and Hannes Kaufmann. Drillsample: Precise selection in dense handheld augmented reality environments. In *Proceedings of the Virtual Reality International Conference: Laval Virtual*, VRIC '13, pages 10:1–10:10, Laval, France, 2013. ACM.

[NA10] National Aeronautics Nasa and Space Administration. NASA TLX: Task Load Index. In *Planta Medica*, volume 35, pages 308–315, 2010.

[Neu15] NeuronMoCap. Perception neuron. `https://neuronmocap.com/`, 2015. Accessed: 2015-11-18.

[NIH+11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[NL13] Anh Nguyen and Bac Le. 3d point cloud segmentation: a survey. In *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, pages 225–230. IEEE, 2013.

[Noi15] Noitom. Perception neuron. `https://neuronmocap.com/`, 2015. Accessed: 2018-03-08.

[NZIS13] Matthias Niessner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.*, 32(6):169:1—-169:11, 2013.

[Ocu13] Oculus. Oculus rift. https://www.oculus.com/en-us/rift/, 2013. Accessed: 2018-03-08.

[OF03] Alex Olwal and Steven Feiner. The flexible pointer: An interaction technique for augmented and virtual reality. In *Proc. of ACM Symposium on User Interface Software and Technology (UIST*, pages 83–84, 2003.

[PBWI96] Ivan Poupyrev, Mark Billinghurst, Suzanne Weghorst, and Tadao Ichikawa. The go-go interaction technique: Non-linear mapping for direct manipulation in vr. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, UIST '96, pages 79–80, Seattle, Washington, USA, 1996. ACM.

[Pri15] PrioVR. Priovr. http://www.priovr.com/, 2015. Accessed: 2015-11-18.

[PSH+04] Helmut Pottmann, Tibor Steiner, Michael Hofer, Christoph Haider, and Allan Hanbury. The isophotic metric and its application to feature sensitive morphology on surfaces. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 560–572, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[Raz15] Razer. Razer hydra. http://www.razerzone.com/gaming-controllers/razer-hydra-portal-2-bundle, 2015. Accessed: 2015-11-18.

[RM12] Henry Roth and Vona Marsette. Moving Volume KinectFusion. *Proceedings of the British Machine Vision Conference*, pages 112.1—-112.11, 2012.

[RN10] D. A. W. Rosa and H. H. Nagel. Selection techniques for dense and occluded virtual 3d environments, supported by depth feedback: Double, bound and depth bubble cursors. In *Chilean Computer Science Society (SCCC), 2010 XXIX International Conference of the*, pages 218–225, Nov 2010.

[Sam15] Samsung. Samsung gear vr. http://www.samsung.com/at/promotions/galaxynote4/feature/gearvr/, 2015. Accessed: 2015-11-18.

[SCS+10] R. Szeliski, B. Curless, S. M. Seitz, N. Snavely, Y. Furukawa, and S. Agarwal. Reconstructing rome. *Computer*, 43:40–47, 06 2010.

[Son15] Sony. Playstationvr. https://www.playstation.com/de-at/explore/ps4/features/playstation-vr/, 2015. Accessed: 2015-11-18.

[SSCO08] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249, Jan 2008.

[Str15] Structure. Structureio. http://structure.io/, 2015. Accessed: 2015-11-18.

[TTN15] K. Tateno, F. Tombari, and N. Navab. Real-time and scalable incremental segmentation on dense slam. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4465–4472, Sept 2015.

[Uni18] Unity. Unity 3d engine. https://unity3D.com/, 2018. Accessed: 2018-03-08.

[Unr18] Unreal. Unreal 3d engine. https://www.unrealengine.com/, 2018. Accessed: 2018-03-08.

[VGC07] L. Vanacken, T. Grossman, and K. Coninx. Exploring the effects of environment density and target visibility on object selection in 3d virtual environments. In *2007 IEEE Symposium on 3D User Interfaces*, March 2007.

[Vir13] Virtuix. Virtuixomni. http://www.virtuix.com/, 2013. Accessed: 2016-03-19.

[VL02] Nikos Vlassis and Aristidis Likas. A greedy em algorithm for gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87, Feb 2002.

[VTHLB15] Anh-Vu Vo, Linh Truong-Hong, Debra F Laefer, and Michela Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100, 2015.

[WBG+12] Matt Westoby, James Brasington, Neil Glasser, Michael Hambrey, and John Reynolds. *'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications*, volume 179. 12 2012.

[WGCB12] Jonathan Wonner, Jérôme Grosjean, Antonio Capobianco, and Dominique Bechmann. Starfish: A selection technique for dense virtual environments. In *Proceedings of the 18th ACM Symposium on Virtual Reality Software and Technology*, VRST '12, pages 101–104, Toronto, Ontario, Canada, 2012. ACM.

[XSe15] XSens. Motionsuite. https://www.xsens.com/products/xsens-mvn/, 2015. Accessed: 2015-11-18.

[ZCW+03] Keqi Zhang, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne lidar data. *IEEE Transactions on Geoscience and Remote Sensing*, 41(4):872–882, 2003.

[Zei15] Zeiss. Zeiss vr one. http://vr-one.eu/de/VR-ONE/VR-ONE.html, 2015. Accessed: 2015-11-18.

[ZT10] Youyi Zheng and Chiew-Lan Tai. Mesh decomposition with cross-boundary brushes. *Comput. Graph. Forum*, 29(2):527–535, 2010.