Master's thesis

NTNU
Norwegian University of Science and Technology

Anders Mølster Hopland

Utilizing VR for Point Cloud Labeling

Master's thesis in Informatics (MIT)
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022

NTNU
Kunnskap for en bedre verden

Anders Mølster Hopland

Utilizing VR for Point Cloud Labeling

Master's thesis in Informatics (MIT)
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022

Norwegian University of Science and Technology

NTNU

Kunnskap for en bedre verden

Abstract

Autonomous vehicles need a solid understanding of the world for safely navigating dynamic environments. One sensor for obtaining such data is the LiDAR, which has unparalleled performance in all conditions. Supervised neural networks need massive amounts of training data to learn semantic visual cues for scene understanding. However, creating such high-fidelity training data is both time consuming and costly [1]. Failing to provide such data can make erroneous classifications propagate to the final model, which will reduce accuracy [2]. Hence, creating enough training data is a significant bottleneck in data-driven AI [3].

This thesis presents a novel VR application called PCLabel.VR. PCLabel.VR allows users to easily navigate a point cloud and label individual points using a label sphere. By offloading labeling to the GPU, it supports real-time labeling of point clouds with over 10 million points. This thesis examines whether utilizing VR for point cloud navigation and labeling makes the process faster, as VR provides both immersion and enhanced perception of visual depth.

In order to investigate this, two experiments were performed. First, 8 participants were introduced to PCLabel.VR, and were instructed to localise 4 misclassifications in a point cloud while being timed. The same experiment was then performed in the open-source desktop application Cloudcompare. Afterward, an extended SUS questionnaire was presented to the users. All participants localized the misclassifications faster in PCLabel.VR than in Cloudcompare, many by a large margin. PCLabel.VR got a SUS score of 76.25, with 68 being above average. Based on the questionnaire and the comparison with a desktop application, it is argued that it is easier to navigate and understand a complex point cloud in a VR environment than in a desktop environment. More user tests are needed to evaluate whether labeling is easier in VR, but results from the questionnaire indicates that users found it easy to label in PCLabel.VR.

Acknowledgment

I would like to express my deep gratitude to all the people that made this thesis possible. First and foremost, I would like to thank my supervisors, Gabriel Kiss and Frank Lindseth, who have provided excellent guidance while writing this thesis. Without their help, I would not have explored autonomous vehicles or gotten access to a great dataset created by an autonomous LiDAR. Furthermore, I would like to thank my employer Tenklabs, who has introduced me to computer graphics and VR. A debt of gratitude is owed to the friends who have helped me during my thesis. I have gotten excellent help with participation in the experiments and proofreading the thesis. All professionals working with point clouds who have answered my emails have been of tremendous help. Finally, I would like to acknowledge the support of my family, with a special thanks to my parents.

A.M.H

Contents

Abstract i
Acknowledgment ii
Contents iii
List of Figures vi
List of Tables vii
Listings viii
Glossary ix
Glossary ix
1 Introduction 1
1.1 Motivation 2
1.2 Goals and Research Questions 2
1.3 Contributions 2
1.4 Thesis Structure 3
2 Background and Related Work 4
2.1 Theory 4
2.1.1 LiDAR 4
2.1.2 Point Clouds 4
2.1.3 The Ouster Sensors 4
2.1.4 Normal Estimation 6
2.1.5 Unity 7
2.1.6 Virtual Reality 7
2.1.7 SteamVR 8
2.1.8 Graphics Processing Unit 8
2.1.9 System Usability Test 9
2.2 Point Cloud software 11
2.2.1 Cloudcompare 11
2.2.2 Pointly 11
2.2.3 3D survey 12
2.2.4 VRMesh Studio 12
2.2.5 Orbit Bentley 3DM 12
2.2.6 Autodesk Recap 12
2.2.7 Autodesk Civil 3D 12
2.3 Related works 13
2.3.1 SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences 13
2.3.2 Point Cloud Interaction and Manipulation in Virtual Reality 13

2.3.3 Shooting Labels: 3D Semantic Labeling by Virtual Reality . . . 14
2.3.4 Interaction and Locomotion Techniques For The Exploration of Massive 3D Point Clouds in VR Environments . . . 15
2.3.5 NASA PointCloudsVR . . . 16
3 Methods . . . 17
3.1 Overview of PCLabel.VR . . . 17
3.2 Requirements . . . 17
3.2.1 Functional Requirements . . . 18
3.2.2 Non-functional Requirements . . . 18
3.3 Point cloud preparation . . . 19
3.4 VR interaction . . . 19
3.5 Controllers . . . 20
3.6 Point Cloud interaction . . . 21
3.7 Rendering the points . . . 23
3.8 Image cylinder . . . 24
3.9 UI . . . 25
3.10 Normal estimation . . . 27
3.11 File read / write . . . 28
4 Results . . . 30
4.1 First phase . . . 30
4.1.1 The application . . . 30
4.1.2 Evaluation . . . 30
4.2 Second phase . . . 32
4.2.1 The application . . . 32
4.2.2 Evaluation . . . 32
4.3 Third phase . . . 34
4.3.1 The application . . . 34
4.3.2 Evaluation . . . 34
4.3.3 Expert Statements . . . 37
5 Discussion . . . 40
5.1 First phase . . . 40
5.1.1 The participants . . . 40
5.1.2 Results . . . 40
5.2 Second phase . . . 41
5.2.1 The participants . . . 41
5.2.2 Results . . . 41
5.3 Third phase . . . 42
5.3.1 Test between a VR and a desktop application . . . 42
5.3.2 Questionnaire . . . 43
5.3.3 Expert statements . . . 47
5.3.4 Research questions . . . 48
5.3.5 Fulfillment of requirements . . . 49

6 Conclusion and Future Work **51**
6.1 Conclusion 51
6.2 Future Work 52
6.2.1 A study with more participants 52
6.2.2 An experiment testing labeling accuracy 52
6.2.3 More interaction with experts 52
6.2.4 Porting the application to Oculus Quest 52
6.2.5 SLAM based merge of multiple point clouds 53
Bibliography **54**
A Appendix **59**
A.1 User manual 59
A.1.1 Overview 59
A.1.2 Prerequisites 59
A.1.3 Basic interaction 59
A.1.4 The menu 59
A.1.5 Time series data 60
A.1.6 Display modes 60
A.1.7 Performance considerations 60

List of Figures

1 An ouster OS0 sensor [4] . . . 5
2 An ouster LiDAR frame [5] . . . 5
3 Sample normal vectors for a surface [6] . . . 6
4 A man using a HMD and VR controllers [7] . . . 8
5 CPU and GPU die [8] . . . 9
6 One of the point clouds used in this thesis displayed in Cloudcompare . . . 11
7 Solution architecture for proposed solution in [9] . . . 14
8 User classifying a car pillar in [9] . . . 14
9 User scaling a point cloud [9] . . . 16
10 Diagram of system architecture . . . 17
11 Image accompanying LiDAR scan . . . 19
12 Oculus controller layout [10] . . . 21
13 A user labeling a traffic light in PCLabel.VR . . . 22
14 Depth enhanced and non depth enhanced shading . . . 24
15 Image cylinder displayed behind a point cloud. The same objects are displayed in both the point cloud and the image . . . 25
16 First VR menu version . . . 31
17 Second VR menu version . . . 33
18 Final VR menu version . . . 38
19 Final desktop version . . . 39

List of Tables

1 Functional requirements of PCLabel.VR 18
2 Non functional requirements of PCLabel.VR 18
3 Extended System Score questionnaire 35
4 Questionnaire results part 1 35
5 Questionnaire results part 2 36
6 Questionnaire results part 3 36
7 Timings for desktop and VR comparison 37
8 System Usability Score results 39
9 Fulfillment of non functional requirements 49
10 Non-functional requirements 50

Listings

3.1 Vertex shader labeling 21
3.2 Fragment shader 23
3.3 Registering UI callbacks 25
3.4 Button interaction VR 26
3.5 LAS structs 28

Glossary

VR Virtual Reality

CPU Central Processing Unit

RGB Red, Green, Blue. The three base colors

GPU Graphics Processing Unit

UV X and Y coordinates in a texture

API Application Programming Interface

FPS Frames Per Second

LiDAR Light Detection and Ranging

HMD Head Mounted Display

UI User Interface

UX User Experience

SUS System Usability Score

AI Artificial Intelligence

1 Introduction

Autonomous vehicles need a solid understanding of the world for safely navigating dynamic environments. One sensor for obtaining such data is the LiDAR, which has unparalleled performance in all conditions. As the LiDAR creates its own pulses, it will perform well even in environments with little or no light [2]. Furthermore, relying on a single sensor is a safety risk, as sensors can fail, and the quality of the data from a sensor depends on external conditions such as weather [11]. LiDAR would, give better data than a stereo camera when there is a lot of glare [12]. Using LiDAR for distance measurements could thus be of tremendous value. Supervised neural networks used in autonomous vehicles need massive amounts of training data to learn semantic visual cues for scene understanding. With the need for fine-grained labels, like for example determining which points belong to a person, the creation of such high-fidelity training data is both time consuming and costly [1]. Failing to provide such data can make erroneous classifications propagate to the final model, which will reduce accuracy [2]. This makes the need for high-quality training data a significant bottleneck for advancing the development of autonomous vehicles utilizing LiDAR for understanding the world [3]. Some work has already been done in this area, with SemanticKitti being a great example [13]. In this paper, a large dataset of labeled point clouds is presented with the goal of advancing the development of advanced models for semantic segmentation. Ramirez *et al.* look into utilizing VR for classifying point clouds in their paper Shooting labels [3]. Using virtual simulations to develop training data is another approach to overcome the expensive approach of manual labeling. Generating training data using simulations has been explored by Vishnyakov *et al.* in the paper Semantic Scene Understanding for the Autonomous Platform [14]. In this paper, they utilize the Unreal Engine Game Engine to create 3.500.000 labeled point clouds with corresponding images.

Several approaches have been used for point cloud labeling. These range from fully manual to nearly fully automatic approaches. A manual approach would force the user to assign each point a label, using tools such as a brush to accomplish the task. A manual approach such as this is e.g., used in Semantic 3D [1]. A more automatic approach would, for example, let the user select a region, and it would then grow in a smart way. This process would be repeated until the point cloud was labeled. Pointly is an example of such an application [15]. Automatic semantic segmentation has come a long way, but errors could still occur. An almost fully automatic such as this would still need human intervention to verify the correctness of the segmentation. Romero-Jar'en *et al.* did in their paper Automatic Segmentation of Point Clouds in the Architecture Environment get global 90% accuracy with their approach [16]. Even though this is a great accomplishment, classifications closer to ground truth would still be preferred for training neural networks.

This thesis will explore whether it is possible to present a complex point cloud structure in a VR environment. Furthermore, it will investigate whether it is easier to gain an understanding of a point cloud in a VR environment than in a desktop environment. Finally, it will examine whether an inexperienced user can perform accurate point cloud segmentation/labeling using such a solution. Using a VR headset would provide the user with immersion and visual depth perception. Previous research has shown improvements in the ability to reason about complex 3D models in a VR environment instead of a desktop environment [17] [18] [19]. This thesis presents an application called PCLabel.VR for viewing and interacting with a point cloud in Virtual Reality. PCLabel.VR is based on Unity and SteamVR, and will work on most VR headsets. By utilizing the GPU for labeling, point clouds with over 10 million points can be interacted with in real time, without the need for any acceleration structures. A user study is performed to evaluate the usefulness of utilizing VR for point cloud labeling. First, an experiment is performed where the user will localize 4 misclassifications in both PCLabel.VR and in Cloudcompare. Cloudcompare is a free open source application for viewing point clouds. This experiment will be performed 3 times in each application, and all attempts will be timed. After this, an extended System Usability Score questionnaire is answered.

1.1 Motivation

After spending much time in VR environments, I found it much easier to get an overview of complex scenes. Compared to desktop applications, I find navigating a 3D scene much easier in a VR environment. Point clouds are especially difficult to navigate and interact with, as they do not have solid surfaces. Because of this, I found it interesting to investigate whether a VR application would make this process easier.

1.2 Goals and Research Questions

VR provides both immersion and the ability to perceive visual depth, which could make analyzing and interacting with point clouds more efficient and ergonomic in VR than on a 2D screen. This thesis aims to explore the utility of using VR for interacting with point clouds. This leads to three research questions (RQs):

1. Is it possible to present a complex point cloud structure in a VR environment?
2. Is scene understanding easier in a VR environment than in a desktop environment? To what degree would an inexperienced user benefit from using a VR environment?
3. Can an inexperienced user perform accurate point cloud segmentation/labeling in a VR environment?

1.3 Contributions

This masters thesis has the following contributions:

- Presenting a novel application called PCLabel.VR for point cloud labeling in VR. The application does labeling on the GPU and is thus able to support over 10 million points in 90fps without the use of any space partitioning data structures.

- Evidence that gaining an an understanding of a complex point cloud in a VR environment is easier than in a desktop environment. This is supported by performing an experiment with 8 participants comparing a desktop application and PCLabel.VR. All the users also answered a questionnaire.
- Comments from industry experts on the usefulness of utilizing VR for point cloud interaction.

1.4 Thesis Structure

This thesis starts by outlining relevant theory and related works in Chapter 2. After that, implementation details of PCLabel.VR is outlined in Chapter 3. The results from experiments, the questionnaire and correspondence with industry experts is presented in Chapter 4. Afterwards, the results from Chapter 4 and the research questions outlined in Section 1.2 is discussed in Chapter 5. Finally, this thesis ends with a conclusion in Chapter 6.

2 Background and Related Work

This chapter will provide the reader with the necessary background to read this thesis. First, relevant theory for this thesis is presented. Afterwards, a thorough exploration of point cloud software is presented. Last but not least, relevant research for this thesis is explored.

2.1 Theory

2.1.1 LiDAR

A point cloud can be created by using one or more LiDAR sensors. LiDAR stands for Light detection and ranging. It works by sending out laser beams and then calculating the time of flight. As the LiDAR sends out laser beams and thus generates its own energy, it is an active sensing system. Depending on what the laser beams hit, multiple reflections might be returned. This returned energy distribution is called a waveform, and the amount of energy returned is called intensity. LiDAR data can either be stored as waveforms or as discrete values. Discrete values would be points in a world space. LiDAR data can be saved in several different file formats, including .las, .laz, and .pcd [20] [21].

2.1.2 Point Clouds

A point cloud consists of several samples in a 3D coordinate system. These samples consist of x, y, and z coordinates. In addition to this, other data such as color in RGB, scan direction, classification, and intensity can be associated with each point. Point clouds have a wide range of applications, ranging from rendering and modeling, depth sensing and perception for e.g. autonomous vehicles and surveying [22].

2.1.3 The Ouster Sensors

The Ouster sensors are LiDAR sensors developed by the company Ouster [23]. An image of an Ouser sensor can be seen in Figure 1. The Ouster sensor requires a computer with a Gigabit Ethernet connection and a 24V power supply [24]. The sensor stream can then be visualized in Ouster Studio or through custom drivers. Ouster defines two coordinate frames, the LiDAR coordinate frame, and the sensor coordinate frame. The LiDAR coordinate frame is a point cloud-centric frame of reference and follows the right-hand rule [24]. "The Lidar Coordinate Frame is defined at the intersection of the lidar axis of rotation and the lidar optical midplane (a plane parallel to Sensor Coordinate Frame XY plane and coincident with the 0° elevation beam angle of the sensor)" [24]. The Ouster Sensors output 3 types of fixed resolution images, signal images, depth images and ambient images. These different images can be seen at the top of Figure 2. In this figure, we can also see the accompanying point cloud for the images.

Figure 1: An ouster OS0 sensor [4]

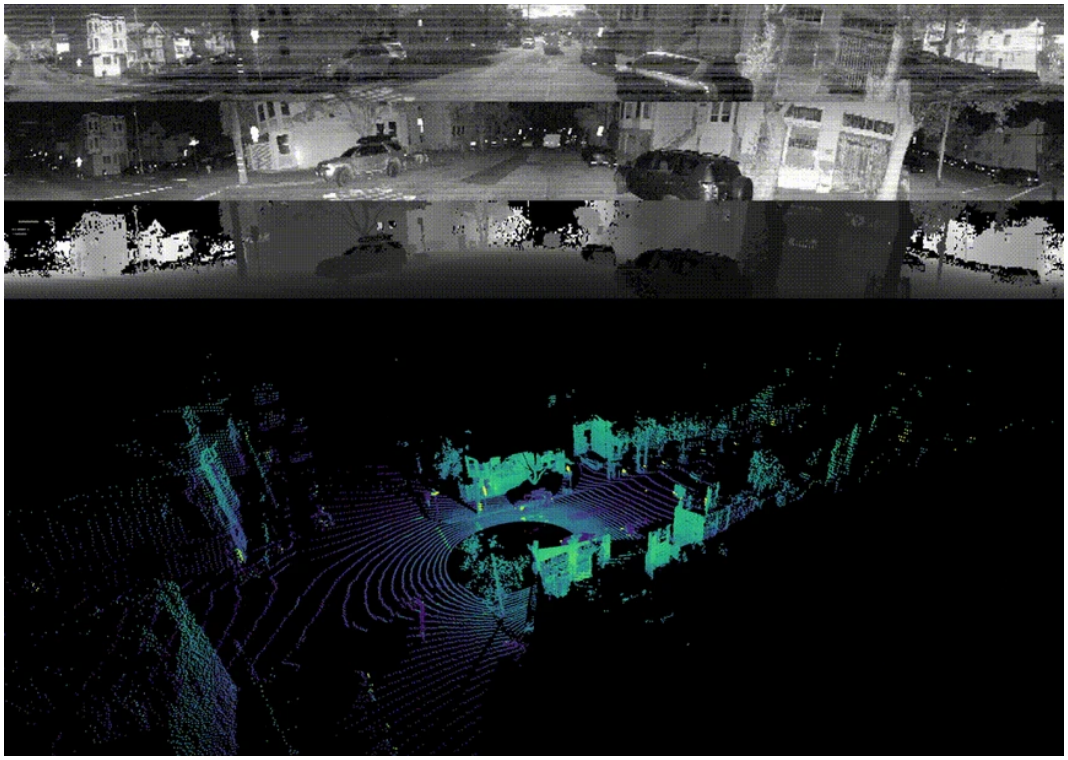


Figure 2: An ouster LiDAR frame [5]

2.1.4 Normal Estimation

A surface normal is by WolframAlpha defined as "The normal vector, often simply called the "normal," to a surface is a vector which is perpendicular to the surface at a given point." [25]. For example, in figure Figure 4 we can see multiple normal vectors on a curved surface. Each arrow represents a normal vector at a point on the surface. As the surface of the point cloud is sampled from an unknown surface, correctly estimating point normals can be challenging. There are several different ways to estimate a point normal. Plane fitting was proposed by Hoppe *et al.* in their paper "Surface Reconstruction from Unorganized Points" in 1992. The proposed method works by finding the k nearest neighbors to a point and then using the total least squares to estimate a normal [26]. If the point cloud has little to no noise, the Big Delauney Ball (BDB) method can be used. The BDB method for point normal estimation is a Voronoi-based method proposed by Amenta *et al.* [27]. Neural networks can also be used to estimate point normals, and can yield better results than more traditional normal estimation techniques. Wang *et al.* did in 2020 introduce a self-attention-based normal estimation network that was able to outperform all existing normal estimation strategies by a large margin [28].

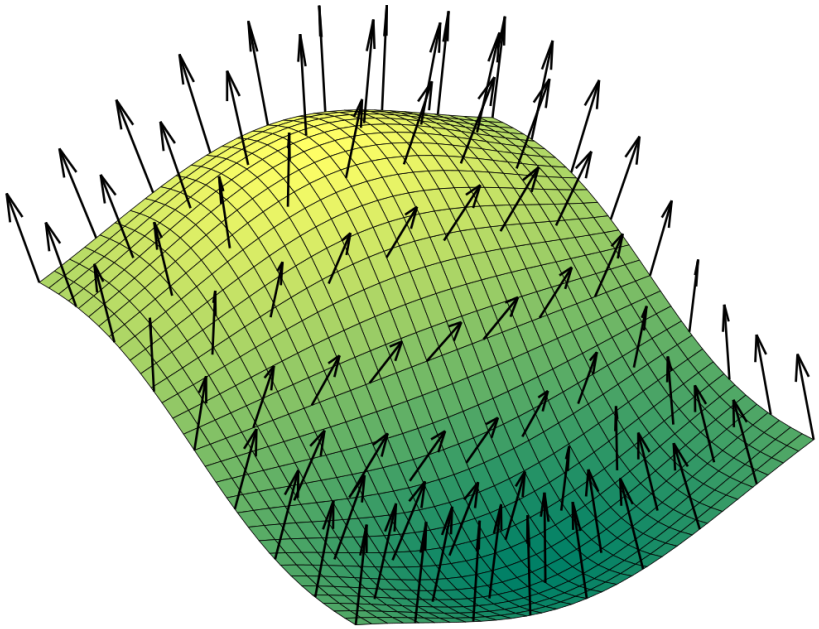


Figure 3: Sample normal vectors for a surface [6]

2.1.5 Unity

Unity is a cross-platform game engine targeting several platforms. These include Windows, Linux, MacOS and Android. Both 2D and 3D applications are supported. It is used in various settings, including games, film, animation and cinematics, automotive, transportation and manufacturing, architecture, engineering and construction, gambling, edtech, and digital twin [29]. Unity has .Net as its default scripting target, with Mono being its runtime. As a game engine, Unity contains several building blocks and tools which make it easier to create games and other interactive media. The building blocks include render pipelines, networking, VR integration, and scripting with an extensive library. Tools include a powerful editor, a package manager, a profiler, and built-in support for importing several kinds of assets [30] [31] [32].

2.1.6 Virtual Reality

Virtual reality can have several different meanings. The Merriam Webster dictionary defines virtual reality as "an artificial environment which is experienced through sensory stimuli (such as sights and sounds) provided by a computer and in which one's actions partially determine what happens in the environment" [33]. In this thesis, I will use a similar definition. In the context of this thesis VR is defined as a computer-generated virtual world a user can interact with and experience like it was a real world. The most common way to explore such a virtual world is through a Head Mounted Display, abbreviated as an HMD. An HMD is a device that is attached to head and have either one or two screens in front of the eyes and head motion tracking sensors. A HMD can also have other sensors, including sensors for tracking the controllers, an accelerometer, and a gyroscope [34]. The most popular HMD today is the Oculus Quest 2 [35].

Figure 4: A man using a HMD and VR controllers [7]

2.1.7 SteamVR

SteamVR is a set of tools for VR, making it easy to develop and experience cross-platform VR applications. SteamVR builds on the OpenVR API, which unifies multiple vendor-specific APIs under one API. This is done by dividing OpenVR into two layers, the driver and the application. When building a new headset, the vendor would write a driver complying with the OpenVR specification. The OpenVR application talks to SteamVR, and SteamVR talks to the OpenVR driver [36].

2.1.8 Graphics Processing Unit

Graphics processing units, abbreviated as GPU, are specialized pieces of hardware for running massively parallel computations and have higher processing throughput and memory bandwidth than a CPU. The GPU was initially developed to accelerate 3D graphics, but has since gotten many other use cases as the GPUs became more programmable. These use cases include neural networks and mining of cryptocurrencies. Today, CPUs do not typically have more than 16 cores, each being a fast general purpose core well suited for performing single-threaded operations very quickly. By contrast, a GPU consists of thousands of slower specialized cores, which can increase throughput on tasks that can be split into separate chunks to be distributed onto the many cores [37]. In figure Figure 5 we see a CPU and a GPU die. Here we can see that the CPU dedicates more of the die to sophisticated control logic and cache. The GPU dedicates more of the die to compute units.

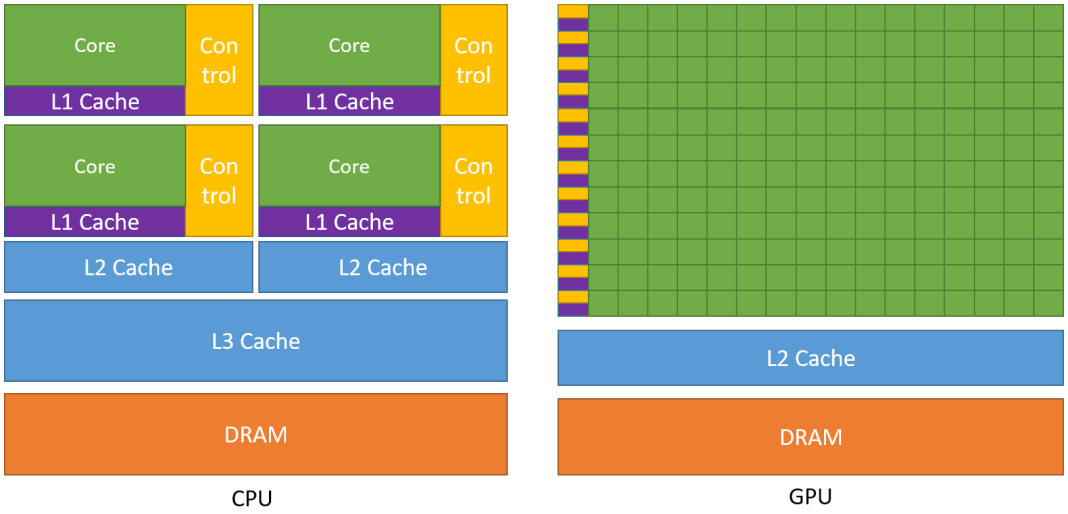


Figure 5: CPU and GPU die [8]

2.1.9 System Usability Test

The System Usability Test, abbreviated as SUS is a quick and easy way to measure applications and systems usability created by John Brooke in 1986. The SUS consists of 10 questions where the user should state to which degree they agree with the statement. To do this, the user can choose between 5 answers ranging from strong agree to strong disagree. Since the SUS was developed more than 30 years ago, it has been battle tested in a wide range of applications. Therefore, it is a reliable tool for gathering quantitative data of an application or a system. Furthermore, given the simplicity of the SUS, it is easy to distribute it using e.g. an online survey which makes it an excellent tool for gathering many data points [38].

A drawback of using a SUS is that it is a technology agnostic tool, and "The disadvantage of a technology agnostic instrument is that it can omit important information that is specific to an interface type." [39]. This means that evaluating an application or a system solely based on SUS scores from respondents might not give a complete picture. It can, for instance be hard to understand why a user agrees or disagrees with a statement. E.g. getting feedback that users finding the system unnecessarily complex does not help pinpoint which parts of the system need to be fixed. Basing the research solely on a SUS score will thus only say something about whether what you are testing is usable, but not tell you anything about what users find difficult [40].

Below is the 10 standard questions to calculate a system usability score [41].

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.

5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

2.2 Point Cloud software

Several point cloud software solutions exist, which differ both in price and functionality. During this research, a thourough analysis of existing point cloud software has been conducted. The most interesting software solutions have been downloaded and tested, and many more were explored but not tested. Below is a brief of the different software solutions that have been analyzed. The programs were tested between August 25th 2021 to September 30th 2021. Therefore, pricing and capabilities might change from when this research was performed.

2.2.1 Cloudcompare

Cloudcompare is a free and open-source point cloud program. Initially, it was a program for comparing two point clouds or between a point cloud and a mesh. It is now a more versatile point cloud software suite with a wide range of capabilities. Cloudcompare also has many plugins such as RANSAC feature detection and CANUPO feature detection [42]. During this research, Cloudcompare has been utilized as a point cloud viewer, and for converting between different point cloud formats. To the best of my knowledge, Cloudcompare does not contain functionality for manual labelling. An image of a point cloud loaded into Cloudcompare can be seen in Figure 6

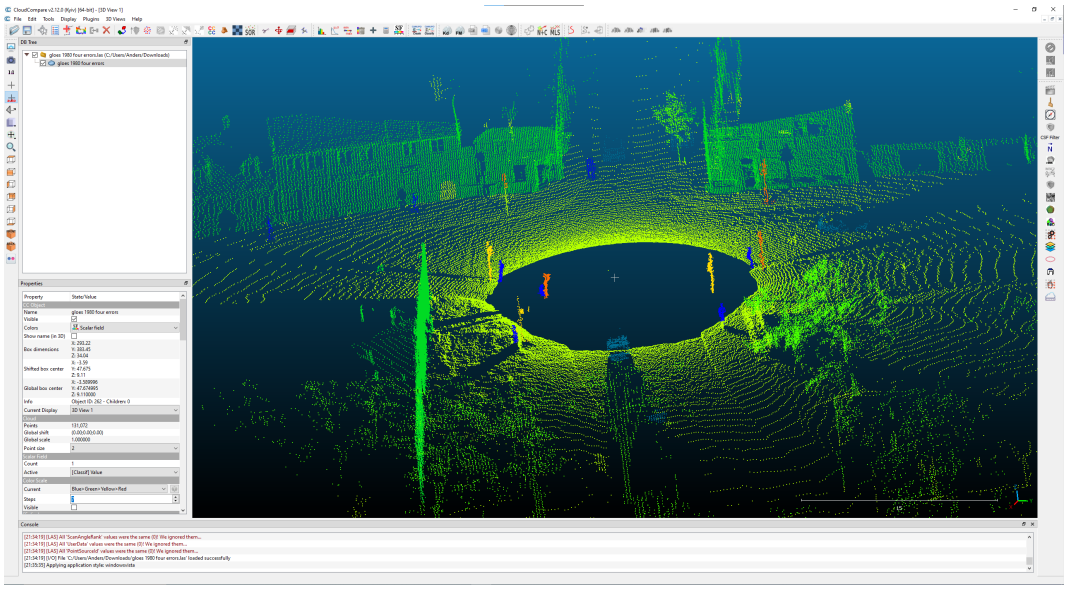


Figure 6: One of the point clouds used in this thesis displayed in Cloudcompare

2.2.2 Pointly

Pointly is a 3D point cloud classification suite. It markets itself by letting the user “create training data for neural nets the easy way”. Pointly is a web-based application, and it is easy to get started. It has an intelligent feature detection system, combining manual feature selection with an intelligent AI-based assistant helping the user grow the selected area, and clean up misclassifications. Pointly was tested quite extensively and yielded excellent results regarding feature detection. Some lag spikes were however present while working with big point clouds. Pointly is

quite expensive, costing 65$ per month for a license, and pricing for a student license was not found [15].

2.2.3 3D survey

3D survey is a program for digital surveying. It is flexible, with supported input formats ranging from point clouds generated from either LiDAR or photogrammetry to CAD models. CAD models can be produced from point clouds. I downloaded a trial version of 3D survey. Its primary focus area seems to be 3D surveying. The feature detection tool was a bit cumbersome to use, but gave good results. A small area was selected, and it smartly grew the area, which yielded good results. It did also contain manual selection which worked well. It was possible to apply for a free student license.

2.2.4 VRMesh Studio

VRMesh is a software suite for point cloud classification, feature analysis, extraction, and point cloud to mesh functionality. In this research, a trial version of VRMesh was downloaded and tested. Loading in a point cloud was fast. Marking different classes manually worked quite well, and the surface tracing was good even on sparse data. Unfortunately, I did not find a student version of the software, and it was costly, costing either 3000$ a year, or 500$ a month [43].

2.2.5 Orbit Bentley 3DM

The Orbit Bentley 3DM portifolio offers programs for point cloud operations. These include importing point clouds and auto extracting features. These auto detections seem to be targeted at mapping out roads, including auto detect pavement separation and auto detect bridge clearance. During this research, the program was downloaded. Unfortunately, a prompt asking for a license appeared upon installation, and such a license was not acquired. Orbit Bentley 3DM was thus not tested. Pricing info can be obtained by asking Orbit Bentley for a quote [44].

2.2.6 Autodesk Recap

Autodesk Recap stands for Autodesk Realitycapture, and is a software program to import laser scans and visualize them. It was downloaded and tested, and although it supports a wide variety of file formats, it was limited in functionality. It did not contain good manual or automatic feature detection / analysis tools [45].

2.2.7 Autodesk Civil 3D

Autodesk Civil 3D is a program for construction design. It is not a software program targeted at point cloud workflows, but does contain point cloud support. In order to import a point cloud, it first has to be converted to an Autodesk proprietary format, as it does not directly support file formats such as .ply or .las. Civil 3D is available for free using a student license [46]. Civil 3D has an extensive toolset, but not many point cloud specific tools are included.

2.3 Related works

In this section we will look at some of the related works that have been explored before and while writing this thesis. Some of this research was found after the literature review in the start of this thesis. This related works section does only outline the most influential and relevant work explored while writing this thesis.

2.3.1 SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences

In September 2019, Behley *et al.* [13] presented a large dataset of classified point clouds. This dataset could help further the development of algorithms for LiDAR-based semantic segmentation, as it provides a large body of real-world data for training and verifying models. The point clouds are generated from a 360-degree automotive LiDAR. The dataset is large, and provides 43552 full 3D scans consisting of 28 different classes. It was the largest dataset of its kind publicly available at its inception [13].

In addition to presenting a dataset, the paper also presents a point cloud labeling tool. This tool helped the research team to effectively provide fine detail labeling. The tool was based on OpenGL, and provided many annotation tools. These include different filtering methods, a brush, and a polygon tool. For faster labeling, multiple point clouds were merged using a SLAM-based approach. Changes to a point in this aggregated point cloud are reflected in all frames containing said point. Using an aggregate based point cloud could pose some challenges, with one of the most pressing ones being moving objects being in different places in different frames. This was solved by threading extra carefully when labelling them, and by assigning them a moving class [13].

2.3.2 Point Cloud Interaction and Manipulation in Virtual Reality

In July 2021, Garrido *et al.* published an article where they investigate whether interacting with a point cloud in a manipulative fashion could be easier in an immersive VR environment, as opposed to a traditional screen-based approach [9]. This was done by creating a VR application that tries to recreate common point cloud editing tools. The Unity Game Engine was chosen for implementing the VR application, utilizing SteamVR supporting multiple headsets and VRTK (Virtual Reality Toolkit) [47] for basic interactions. Pre-processing the point clouds were done by using Cloudcompare [42] for extracting individual objects from a point cloud, and Meshlab [48] for creating mesh data from the extracted objects [9]. The proposed solution architecture can be seen in figure Figure 7. An image from the program can be seen in figure Figure 8.

The development of the application was successful, and the application worked as planned. It was very intuitive to use the application for tasks such as rotating and moving one or more points. Compared to more traditional software applications such as Cloudcompare [42] and Meshlab [48], the VR application was less cumbersome to use. Immersive 3D technologies such as VR enhance depth perception, and being immersed in a virtual world and interacting directly with the point cloud by using the hands, the interaction feels more natural and straightforward. No technical user tests or system usability tests were performed during the research. This is planned

to be done in the future [9].

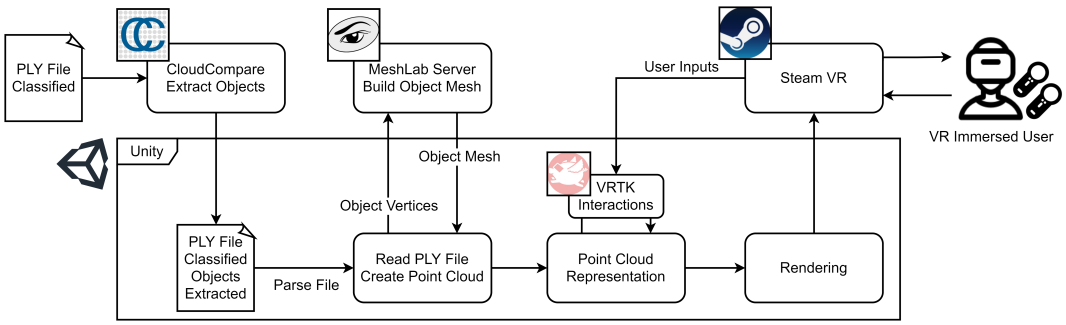


Figure 7: Solution architecture for proposed solution in [9]

Figure 8: User classifying a car pillar in [9]

2.3.3 Shooting Labels: 3D Semantic Labeling by Virtual Reality

In October 2019, Ramirez *et al.* published an article presenting Shooting Labels, a VR based tool for segmentation of dense 3D semantic segmentation. This tool is to the best of their knowledge the first VR based 3D semantic labeling tool. Shooting Labels support both mesh and point cloud data. The proposed application makes it easier to label 3D objects, and reduces the amount of expertise needed to do it successfully. By utilizing gamification, it hopes to gather a larger community for creating annotated 3D data. By having multi player integration, multiple users can label a scene, and redundancies in different users labeling can be exploited to predict the confidence of each label, and thus get better labeling than if only one user labeled a scene [3].

In order to evaluate Shooting Labels, 5 participants was tasked with labeling scenes from the Matterport 3D dataset [49]. The labels provided by Matterport was considered as ground truth data. None of the participants had any experience with 3D modeling. Combining the results of all participants gave the best results, with over 79% accuracy. By using the results from multiple users. it was possible to create an uncertainty map. This uncertainty map can be used both for improving the segmentation and be included in training data for supervised AI [3].

2.3.4 Interaction and Locomotion Techniques For The Exploration of Massive 3D Point Clouds in VR Environments

In 2018, Thiel *et al.* published an article presenting a rendering system for point clouds in VR. Making a good rendering system for a VR application has different requirements than a rendering system for desktop. A VR application typically requires 90fps for a good experience, contrasting 30-60 fps for a desktop application. The proposed rendering system is a multi-pass pipeline consisting of 3 render stages. First, a subset of the points is selected, after that, they are rendered, and finally, image-based post processing is applied. The developed application supports rendering point clouds consisting of several billion points. This is done by only showing the points that are needed [50].

Three main tools for interacting with point clouds in VR were implemented. These were a virtual measuring tape, a tool that lets the user measure the distance and area between selected points, and a gesture-based transformation tool for scaling and rotating data sets. A figure showing a user scaling a point cloud in the application can be seen in Figure 9. Navigating the scene can be done using one of several locomotion techniques. For example, a user can move using his own body, and the movement will be reflected in the virtual world. The other locomotion techniques supported are movement by joystick and movement by teleportation. The usability of the proposed application was tested by using a pilot study consisting of 8 students ranging from 18-21 years. The preferred locomotion technique was to move using the joystick. This could be because today's youth have much experience with video games and thus also experience with navigating using a joystick. As for measurement, 75% of the participants preferred the precise measuring tool instead of the virtual measurement tape. Future work will include performing a complete user study [50].

Figure 9: User scaling a point cloud [9]

2.3.5 NASA PointCloudsVR

NASA has open sourced a VR point cloud viewer on Github. The point cloud viewer is written in C++ and runs as a Windows 10 application. Utilizing OpenVR and SteamVR, it should be compatible with most headsets and have been tested on HTC Vive and Oculus Rift. It supports several point cloud formats, including .las and .ply. The program's capabilities are displaying and analyzing point clouds, with analysis tools including the ability to draw 3D lines in space [51].

3 Methods

This chapter goes in depth on how PcLabel.VR is implemented. First, a high level description is given of the various components and how they interact is presented. Afterwards, the functional and non-functional requirements of PCLabel.VR are outlined. Finally, an in depth look on implementation details is conducted. The code can be seen at https://github.com/anders-hopland/vr-pointcloud, and one of the aims of this thesis was that the code could be freely shared.

3.1 Overview of PCLabel.VR

PCLabel.VR is developed using the Unity Game Engine version 2019.4.38 and SteamVR. One of the application's goals is to be able to interact with point clouds in real-time, and it should be able to support point clouds with millions of points, but not more than 10 - 20 million points. With these limitations in mind, it is possible to do all of the processing on the GPU, and because the number of points does not exceed 20 million points, no structures for partitioning the point clouds is needed. By not partitioning the data, the system architecture can be pretty straightforward. It should however be noted that the point clouds used in this thesis have been created by a 128 x 2048 LiDAR sensor, and each frame contained 131072 points. Even if using SLAM to merge multiple point clouds, the threshold for the maximum number of points would probably still not be exceeded. A rough outline of the system architecture is shown in Figure 10. Each frame, PCLabel.VR communicate with the SteamVR driver, sending data to render and getting user input.

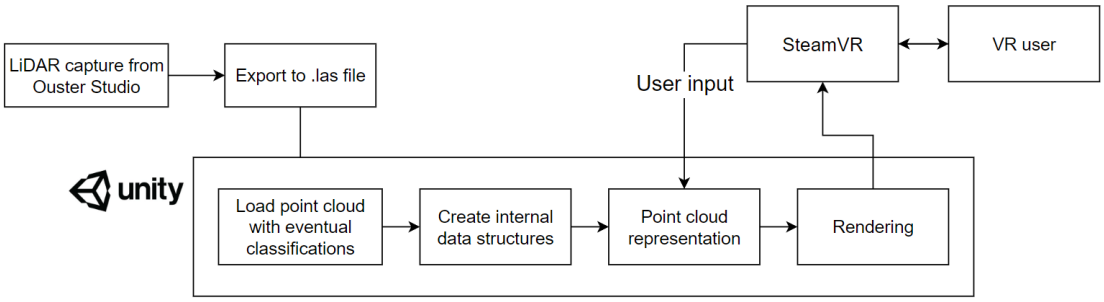


Figure 10: Diagram of system architecture

3.2 Requirements

This section goes into specifying functional and non-functional requirements for PCLabel.VR. Some requirements were decided at the start of writing this thesis, like for instance requiring

stable frame rate and ease of use. Other requirements were added after user tests in the first and second phase. An example of such an addition is the ability to show a circular image of the scene if provided.

3.2.1 Functional Requirements

ID	Functional requirement	Priority
1	A user should be able to load a point cloud from a .las file	High
2	A user should be able to navigate around the point cloud	High
3	A user should be able to zoom in on interesting areas in the point cloud	High
4	A user should be able to label different parts of the point cloud using a predefined set of labels	High
5	A user should be able to save the labels in the point cloud to a .las file	High
6	A user should be able to change the size of the label sphere	High
7	The application should be able to approximate surface normals	Medium
8	The application should be able to render both round and square points	Medium
9	The application should be able to display a color gradient based on height	Medium
10	The application should optionally display a floor	High
11	A user should be able to change the radius of the points in the point cloud	High
12	The application should be able to show a circular image of the scene if provided	High
13	A user should be able to change the radius of the circular image	High
14	The application should support time series point clouds	High
15	A user should easily be able to navigate time series data	High
16	The application should be able to work with point clouds with up to 20 million points with a Nvidia Geforce RTX 3070 or better	High

Table 1: Functional requirements of PCLabel.VR

3.2.2 Non-functional Requirements

ID	Non functional requirement	Priority
1	The application should be easy to use	High
2	The application should run smoothly at 90fps	High
3	The application should work on most commercial VR headsets	Medium
4	The application should be easy to set up on a new computer	Medium

Table 2: Non functional requirements of PCLabel.VR

3.3 Point cloud preparation

The point clouds used in this thesis were created by my supervisors when they drove a car with a Ouster LiDAR sensor around in Trondheim, Norway. This resulted in a big .pcap file and an accompanying metadata file, containing several thousand frames. I was able to view the data in Ouster Studio, which was of great help early in the development cycle [52]. Since my application only supported .las files, I had to extract relevant data from the .pcap file and save it to one .las file per frame. To accomplish this, the Ouster Sensor SDK with python bindings was utilized [53]. Using this sdk, I developed a python script for extracting each frame into a .las file. A .pcap file can also contain three images per frame, and if images were available, they were extracted as well. An example of such images is shown in Figure 11, showing both an ambient, a signal and a distance image.

Figure 11: Image accompanying LiDAR scan

3.4 VR interaction

In order to make implementing VR interaction as easy as possible, SteamVR was chosen. The reason why SteamVR was chosen is twofold. The most important one is that I already had extensive knowledge about how SteamVR works and did thus not need to spend time learning how it works. Secondly, it also supports multiple headsets, including the most popular ones, which among others are the Oculus Headsets, Valve Index, and HTC Vive [35] [54]. The Unity CameraRig Prefab was used, which integrates directly to SteamVR. This meant that controller movement and head movement was reflected without having to write any code. For each frame, it were checked whether any buttons of interest was pressed down. In the case of a button click or a button press, an appropriate action would be taken.

It was experimented with adding an abstraction layer over SteamVR, which would handle SteamVR specific logic and set the CameraRig Active. This would make changing VR backend easier. This development was an effort to port PCLabel.VR to native Oculus Quest, which do not support SteamVR. This effort was cumbersome, as all VR backend-specific code had to be hidden behind compiler flags. In addition, not all libraries could compile for that target as Oculus Quest runs on Android. For example, the native Windows file browser and SteamVR could not compile for Oculus Quest.

Getting VR events was done by polling each frame in a VR controller class, triggering the correct functions and updated parameters. By having all SteamVR specific logic in a hidden behind a

controller, switching to another VR backend system should not be to hard if needed. This could be of great use if e.g the application was to be ported to e.g native Oculus Quest or another headset which SteamVR does not support. Another benefit of having all interaction with SteamVR in one place was the ease of refactoring, and being able to add logging with ease if needed. This could be of great use in the future, as having a complete event log could make it possible to record every user session so it can be analyzed at a later time.

3.5 Controllers

The best practices described by Oculus inspired the controller layout [55]. One of the main benefits of following the best practices is to cater to what Don Norman calls cultural constraints. A cultural constraint is in this setting a norm for how an interaction should work. An example of a cultural constraint is that a closing a window on a computer is done by pressing a button in the top right corner of the window [56]. Following cultural constraints, lets users use their mental models of how to interact with the application and thus make it easier to start using the application without having to expand the mental model.

We can see a picture of Oculus controllers with arrows and names for the different controller buttons in figure Figure 12. The right trigger (IndexTrigger) is used for menu interaction and labeling point clouds. All points inside the label sphere are labeled while the trigger button is pressed. Pressing either the left or right grab button (PrimaryHandTrigger) will let the user grab the scene and move it around. Pressing both grab buttons simultaneously lets the user grab and scale the scene using a pinch-like movement. Adjusting the size of the label sphere can be done by moving the right thumbstick left or right. The VR menu visibility can be toggled by pressing the left trigger.

Other workflows were explored while developing PCLabel.VR, which would yield other controller button layouts. For locomotion, both teleportation and flying using the joystick were explored. These experiments were performed locally only by myself, and after testing the different locomotion techniques, the choice fell on grabbing the scene for moving it around. Scaling the scene was initially done by using the right trigger, but yielded poor results. Finding a good center point for scaling with the joystick was hard, and scaling up and down was cumbersome.

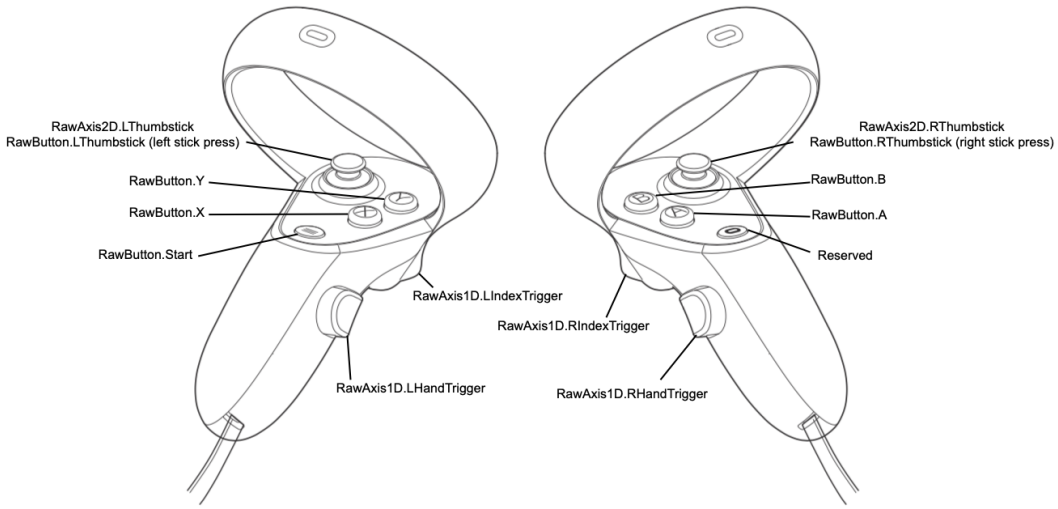


Figure 12: Oculus controller layout [10]

3.6 Point Cloud interaction

Point cloud interaction is performed in the shader code running on the GPU, which also renders the point cloud. An image of an user labeling a traffic pole can be seen in Figure 13. User interaction is done by updating shader parameters when the user performs actions. These parameters include label sphere position, label sphere radius, and current label id. Other parameters can be passed to the shader, including display mode and max and min height for creating a color gradient based on point height. Doing the point cloud interaction on the GPU has several advantages. First and foremost, the task of labeling points that are inside the sphere is an embarrassingly parallel operation, which is well suited for the GPU. Furthermore, we do not need to upload new data to the GPU for every interaction. E.g. changing the color of all points each frame would be very costly, and it could exhaust the GPU memory bus.

One drawback of doing all computations on the GPU is that we can not support larger point clouds than what we can fit in memory on the GPU, or process fast enough not to degrade system performance. The bottleneck would most likely be processing it fast enough, as every point is tested for whether it is inside or outside the label sphere each frame. An out-of-core algorithm would need to be employed for supporting point clouds that do not fit in memory. Another drawback of doing all computations on the GPU is that all data must be present on the GPU. This could cause frame drops while going to the previous or next frame, as the data must be uploaded. A LRU cache on the GPU was implemented in order to mitigate this problem, but it would work poorly if e.g. the only looks at each frame once. Using async upload and download to the GPU to ensure that the previous and next frame always is present in GPU memory could be a better solution to this problem. only moving forward would The main logic for classifying points in the vertex shader is shown in listing Listing 3.1.

Listing 3.1: Vertex shader labeling

```
// Vertex index
int ix = v.ix;

// Get correct position from compute shader
float4 position = float4(pointBuf[ix].vert.xyz, 0);

// Check shader parameter for trigger press
if (_TriggerPress == 1) {
  // Check if point is inside label sphere
  if (distance(position.xyz, _EditPos.xyz) < _EditRadius) {
    // Write label color to compute buffer
    pointBuf[ix].classification = _CurLabel;
    }
  }

// Set output color based on label
o.col = dataBuf[pointBuf[ix].classification];
```

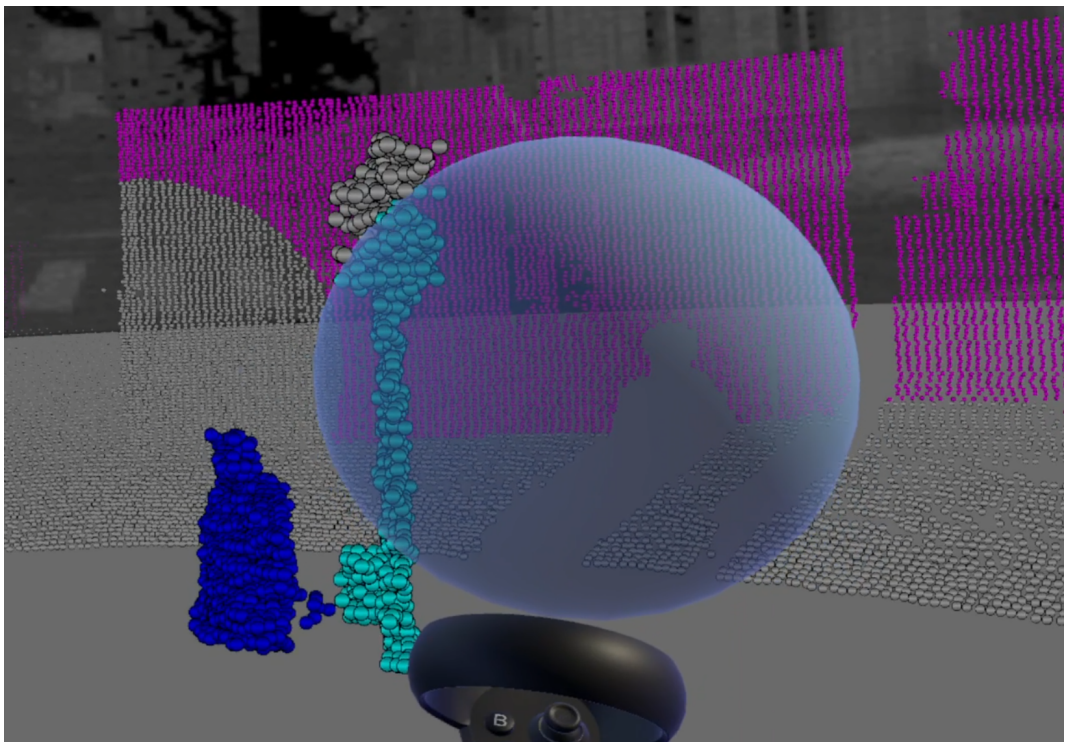


Figure 13: A user labeling a traffic light in PCLabel.VR

3.7 Rendering the points

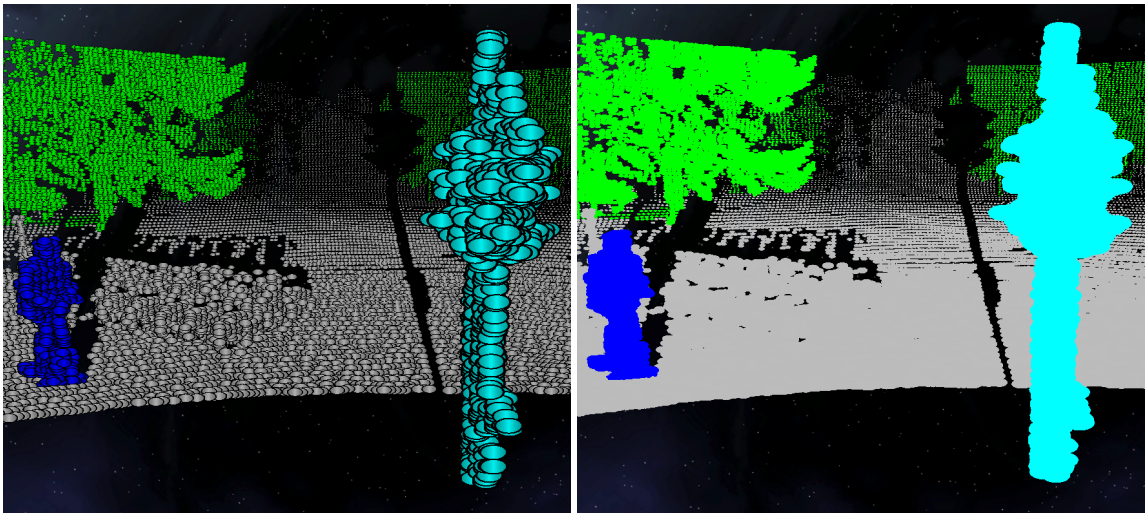
The point rendering is done in a simple shader that only utilizes the vertex shader and the fragment shader. Through shader parameters, behavior can be changed without switching to a different shader. For example, points can be rendered as both quads and circles. Creating circles from quads is done by assigning each vertex of the quad UV coordinates with values (0, 0), (0, 1), (1, 0), and (1, 1). After that, it is calculated whether each fragment's distance to the center is greater than 0.5. The complete fragment shader code can be seen listing Listing 3.2. Three color modes are supported, color from original the las file, classification color, and height map coloring. These are easy to switch between and are applied in the vertex shader.

Computing normals is expensive, and computing normals in sparse point clouds yields bad results in many cases. Therefore, billboard shading is also supported. Billboard shading is a technique where the quads are transformed to be parallel with the camera plane [57]. One problem with this is applying realistic lighting to the points. This can make it hard to separate points from each other, as we can see in Figure 14b. It should be mentioned that the separation is much easier to see in virtual reality than a desktop environment. In order to enhance depth perception, two different effects were implemented. The first one was creating a lighting effect resembling eye dome lighting, which was inspired by Kharroubi *et al.* [58]. This was done by sampling a brushed aluminum grayscale texture in the fragment shader, and then multiplying the color with that value. The other effect for enhanced depth perception was a thin black outline around each point. The impact of this can be seen in Figure 14.

Listing 3.2: Fragment shader

```
fixed4 frag(v2g i) : SV_Target {
  float4 col = i.col;

  float x = i.uv.x;
  float y = i.uv.y;

  if (_DisplayRoundPoints == 1) {
    float dist = sqrt(
      pow((0.5 - x), 2) + pow((0.5 - y), 2)
      );
    if (dist > 0.5) { discard; }

    // Black outline around circle
    if (dist > 0.42)
      { col = float4(0, 0, 0, 0); }
    }
  else {
    // Black outline around quad
    if (x < 0.06 || x > 0.94 || y < 0.06 || y > 0.94)
      { col = float4(0, 0, 0, 0); }
    }

```

```
23    // Fake lighting mask
24    float4 mask = tex2D(_MainTex, i.uv);
25    col *= mask.x;
26
27    return col;
28    }
```



(a) Depth enhanced (b) Non-depth enhanced

Figure 14: Depth enhanced and non depth enhanced shading

3.8 Image cylinder

PCLabel.VR was mainly tested on relatively small point clouds, reaching no more than 150K points. Such small point clouds generated from a 360-degree capture meant that the resolution of many objects of interest was relatively low, thus making it difficult to figure out which class they belong to. Furthermore, not having RGB colors in all point clouds, thus having only black points to work with before classification, made the task even more difficult. To lessen this problem, a user could display an image of the area in the scene. The image should be a 360-degree image, as it would be texture mapped onto a cylinder with no top and no bottom. The cylinder was created in Blender with the correct uv-s and removed the top and bottom. Finally, the normals of the cylinder were inverted, as the surface and thus the texture should face towards the center of the scene. The result of this can be seen in Figure 15, where the image cylinder displays an image behind the point cloud. The radius of the image cylinder can be changed in order to intersect objects.

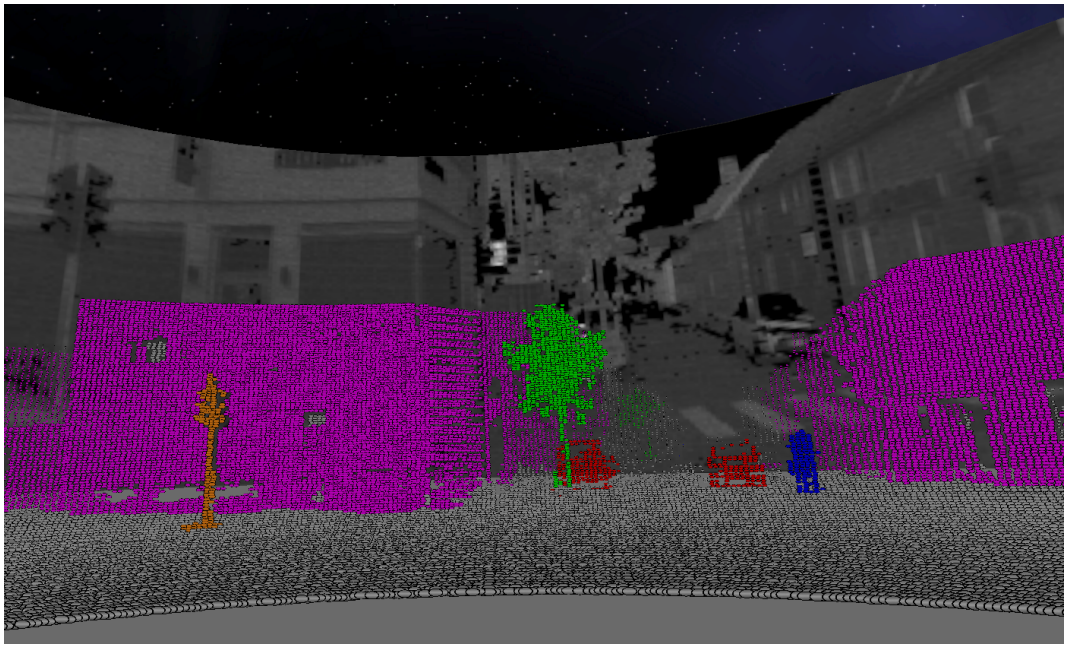


Figure 15: Image cylinder displayed behind a point cloud. The same objects are displayed in both the point cloud and the image

3.9 UI

The UI utilizes Unity's UI system. Editing the menu was thus done by using the Unity Editor and moving Gameobjects in world space. At the beginning of writing this thesis, the VR UI was made from scratch, using three-dimensional buttons which the user had to touch. This approach made it time-consuming to add new buttons, and this custom solution also lacked UI elements such as dropdown lists and check-boxes. Without such components, the menu would be cramped for space when trying to fit all the wanted functionality. By already using the Unity UI system to make the desktop menu, using the same menu system for both menus would make the development process much more manageable. By registering callbacks for buttons using code, instead of registering them in the Unity editor, it was possible to register an event for the desktop menu and the VR menu in the same way, using just one function call. An example of how callbacks were registered can be seen in Listing 3.3. This would register the same callback for the button with the GameObject name in the desktop and VR menus.

Listing 3.3: Registering UI callbacks

```
1 addListenerVrDesktop(
2   "HigherPointRadBtn", // <Gameobject name>
3   "Button", // <UI element type>
4   () => EventHandler.registerEvent(event) // <callback>
5 );
```

However, making the Unity UI system work in VR was not straightforward. The Unity UI system is made for desktop interaction and would thus not work out of the box for VR interaction. This meant that this functionality had to be implemented for VR. Figuring out which UI element the user pointed towards was done by performing a raycast from the right controller and test for an intersection with an UI element. An appropriate event with an event payload adhering to the Unity specification is triggered if a UI element is hit. These events include OnPointerIn, OnPointerOut, and OnPointerClick. A rough outline of how this was accomplished can be seen in Listing 3.4. On menu hover, a laser would be drawn from the right controller to the hitpoint on the menu. This makes it easier for the user to see where the controller is pointing.

The approach outlined in the paragraph above had some limitations. First and foremost, not all UI elements worked as expected. UI elements such as e.g. sliders did not work correctly, and the image cylinder radius did thus need to be adjusted buttons instead. Another drawback of using the Unity UI system for making the VR menu is that the menu elements are designed for desktop usage. It can be argued that a more VR-friendly menu could have been created.

Listing 3.4: Button interaction VR

```
Ray raycast = new Ray (transform.position, transform.forward);
SteamVR_Action_Boolean interactWithUI =
  SteamVR_Input.GetBooleanAction ("InteractUI");
RaycastHit hit;
bool buttonHit =
  Physics.Raycast (
    raycast,
    out hit,
    999f,
    LayerMask.GetMask ("VR_UI")
  );

if (buttonHit && interactWithUI.GetStateUp(pose.inputSource)) {
  // Submit new pointer event
  PointerEventArgs argsClick = new PointerEventArgs();
  argsClick.fromInputSource = pose.inputSource;
  argsClick.distance = hit.distance;
  argsClick.flags = 0;
  argsClick.target = hit.transform;
  OnPointerClick(argsClick);
  }
```

3.10 Normal estimation

Calculating surface normals is a compute-heavy process. It was decided to calculate the normal of a point by finding the n nearest points and then fitting a plane to the points. The plane normal should thus be the same as the point normal. This approach yielded good results when the points are in close proximity, and the variance of the point normals is pretty low. The results were bad if a point was alone or only a few points were close. Creating a fast normal estimation algorithm that yields good results is challenging task. After trying different normal estimation algorithms differing in both speed and results, an algorithm developed by Emil Ernerfeldt on the blog I Like Big Bits seemed to be the best fit [59]. This method fits a plane by minimizing the squares over the n-nearest points, and is suitable for point clouds with little noise. As described in Section 2.1.4, the best normal estimation algorithms is today based on neural networks. It was however chosen not to adopt this method, as it would potentially take a lot of time to implement. A more straightforward approach such as the one described by Emil Ernerfeldt struck the right balance between being easy to implement while still yielding good results.

A KD-tree was utilized to speed up finding the n nearest neighbors. A KD-tree stands for k-dimensional tree and is a space partitioning data structure [60]. Implementing a fast and resilient KD-tree could take a considerable amount of time. It was therefore decided to search for open source implementations, and an implementation from the GitHub user viliwonka was chosen. This implementation is a battle tested implementation with over 300 stars on GitHub. It is optimized developed for Unity, which made it easy to include in the project. The performance is also good, with it being a thread safe implementation for 3D points. The code is licensed under the MIT license [61].

Building the tree was a single threaded task that only needs to be performed once, but querying the tree was an inherently parallel task. In order to utilize the parallel nature of querying, it was decided to spawn multiple threads that each would compute a part of the normals. Computing the normals did not require cooperation between the threads, making the code easy to write and reason about. Spawning threads was done using the .Net Threadpool, and the number of threads spawned was a multiple of the number of available cores on the machine. On a 8 core AMD Ryzen 1700X, the multi-threaded implementation was approximately 6 times faster than the single-core implementation.

3.11 File read / write

The las file format was chosen for reading and writing point cloud data to disk. Implementing a las reader and writer was quite simple, as the file format specification is pretty straightforward. There are several versions of the las file format, and the specification of file format 1.2 was chosen [62]. PCLabel.VR does not support removing or moving points, only classification of points. This lets us reuse both the header and the point structs when saving. The las structs can be seen in Listing 3.5.

Listing 3.5: LAS structs

```
public struct LasHeader
    {
    // position in header is commented behind
    public byte[] fileSignature; // 0
    public ushort fileSourceId; // 4
    public ushort globalEncoding; // 6
    public uint projectId1; // 8
    public ushort projectId2; // 12
    public ushort projectId3; // 14
    public byte[] projectId4; // 16
    public byte versionMajor; // 24
    public byte versionMinor; // 25
    public byte[] systemIdentifier; // 26
    public byte[] generatingSoftware; // 58
    public ushort fileCreationDay; // 90
    public ushort fileCreationYear; // 92
    public ushort headerSize; // 94
    public uint offsetToData; // 96
    public uint numVariableLengthRecords; // 100
    public byte pointDataFormat; // 104
    public ushort pointDataLength; // 105
    public uint numPoints; // 107
    public uint[] numPointsByReturn; // 111
    public double xScaleFactor; // 128
    public double yScaleFactor; // 136
    public double zScaleFactor; // 144
    public double xOffset; // 152
    public double yOffset; // 160
    public double zOffset; // 168
    public double maxX; // 176
    public double minX; // 184
    public double maxY; // 192
    public double minY; // 200
    public double maxZ; // 208
    public double minZ; // 216
    }

public struct LasPoint
    {
```

```
40      public Vector3 xyz;
41      public Color col;
42      public ushort instensity;
43      public byte returnNumber; // 3 bits
44      public byte numberOfReturns; // 3 bits
45      public byte scanDirectionFlag; // bit
46      public byte edgeOfFlightLine; // bit
47      public byte scanAngleRank; // -90 to + 90, should be an uchar
48      public byte classification;
49      public byte userData;
50      public ushort pointSourceId;
51      public double GPSTime;
52      }
```

4 Results

The development of PCLabel.VR presented in this thesis can be divided into three phases. The results from the three phases are outlined below. In addition, a video highlighting the core functionality of PCLabel.VR can be seen at: https://youtu.be/9ArdluqWuiU.

4.1 First phase

This section outlines the first phase of developing PCLabel.VR. This phase ran from mid august to late autumn 2021, with the goal of developing a prototype and evaluating if the prototype is useful enough to proceed with development.

4.1.1 The application

PCLabel.VR was rudimentary and only implemented the most basic functionality. A user can load a point cloud or a set of point clouds. A set of point clouds can e.g. be time series data. Such time series data could for instance come from an automotive LiDAR. The user can then switch between the point clouds using the arrows shown in Figure 16. The first VR menu design can be seen in Figure 16. Saving the modified point clouds was not implemented at this point in time. Moving around the point cloud was done by using one of the grab buttons for grabbing the scene and then moving the controller while the scene is attached. Scaling could be done using the joystick. The classification was done by painting the points, where the user can change between layers with their corresponding colors. Painting works by pressing the trigger button while having the points you want to label inside the label sphere.

4.1.2 Evaluation

The test of the first version was quite limited, as it was clear that it would not be the final version. The first user testing was done by gathering 4 participants and letting them try the application, hear what they had to say, and watch them in VR to see where they struggled. All the users had problems interacting with the VR menu. Navigating the scene was not very smooth, especially when moving over long distances. Painting worked well, and all users found it intuitive to use. It was, however, in some instances, somewhat challenging to see which points were inside the label sphere and which points were outside. This problem manifested itself primarily when the label sphere was big.

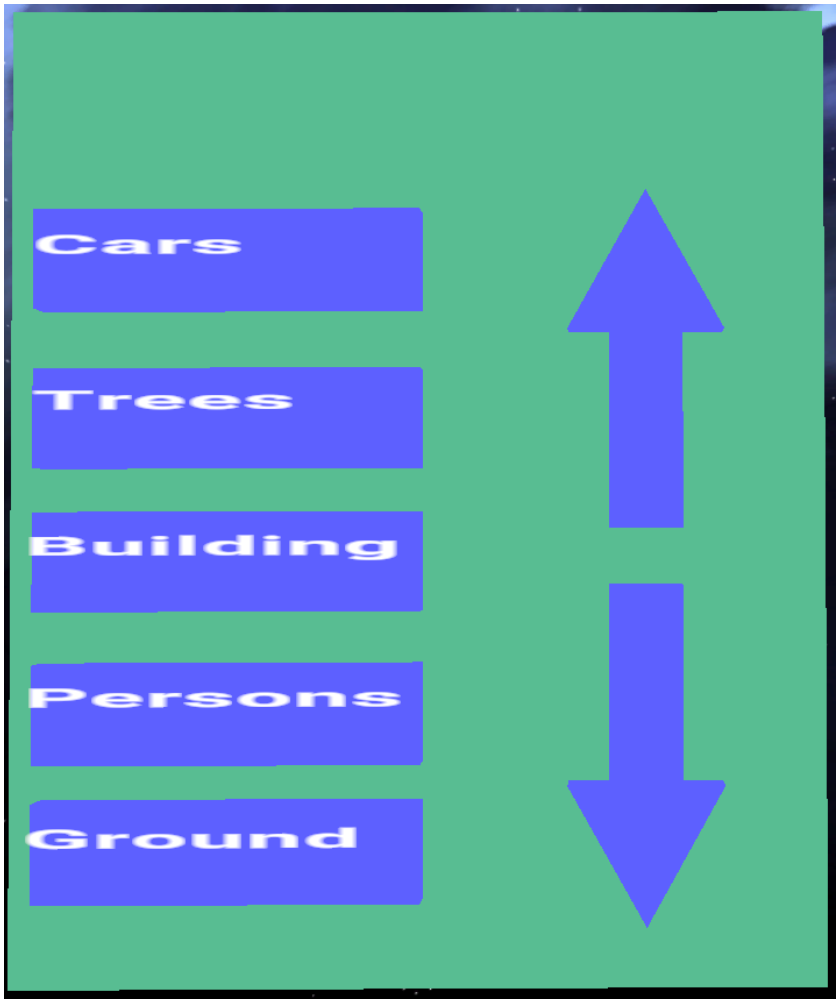


Figure 16: First VR menu version

4.2 Second phase

The second version was finished in the middle of January and brought new capabilities to the application. The main improvement was a new VR menu based on Unity's UI system. Another improvement was the addition of normal estimation for the points in the point cloud. The new menu can be seen in Figure 17.

4.2.1 The application

The application was an extended version of the application outlined in phase one Section 4.1.1. The main change, namely a new VR menu design, can be seen in Figure 17. Menu interaction is now done by pointing a laser towards the menu and pressing the trigger button on the right controller to interact. The ability to toggle between billboard shading and shading using estimated point normals was also introduced in this version. This functionality was only accessible in the desktop menu.

4.2.2 Evaluation

The test of the second version was like the first version, quite simple. 4 participants and my supervisors, Gabriel and Frank, participated in the second phase of testing. Unfortunately, Gabriel and Frank could not test the application properly, as the headset used while testing in the lab had problems with drifting. I was able to give a demo and move around while they watched as I could counteract the drifting, but doing this with no prior experience with the application was hard. The input from Gabriel and Frank during this phase was thus only based on videos and a live demo where I used the application and answered questions.

The user tests yielded promising results, and all participants found the new menu design easier to interact with than the previous version. The main frustration for all the testers was zooming and moving long distances. Two participants suggested that scaling could be done by pressing both grab buttons. All the testers found it easy to label the point cloud and found the label sphere easy to use. However, visual depth was hard to perceive, and knowing which points belonged inside the label sphere proved to be a challenge in dense regions. All testers found the estimated point normals helpful in most regions, but a significant drawback was wrong point normals in regions with few points. Furthermore, many normals were flipped, which e.g. only made the ground points visible from beyond. This caused the estimated point normals shading not to be preferred.

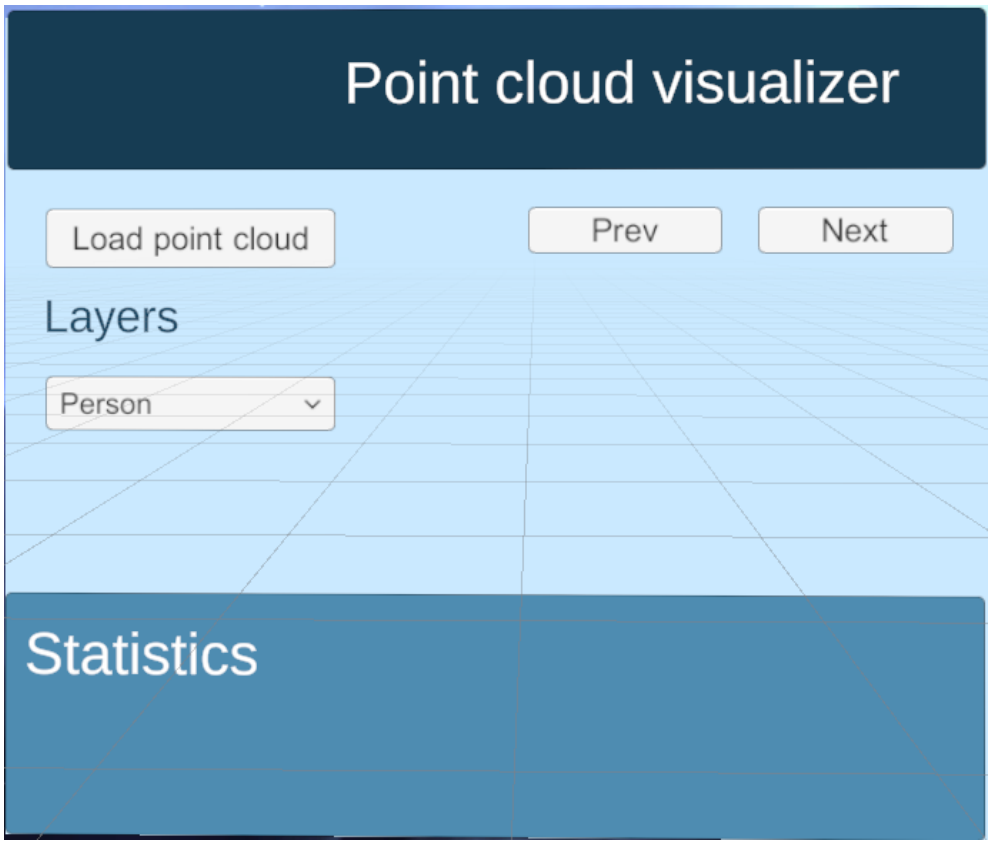


Figure 17: Second VR menu version

4.3 Third phase

The third phase was the final phase and covered the final version of PCLabel.VR. Many improvements were introduced in this phase, with changes such as scaling with two hands and better shading for enhanced depth perception being two of the most important ones. During this phase, a proper test of PCLabel.VR was conducted, consisting of a questionnaire, expert statements, and a timed task that should be performed in both Cloudcompare and PCLabel.VR.

4.3.1 The application

The application was an evolved version of the second version, introducing many new functions while keeping the core functionality intact. Saving the classified point cloud is now supported. Scaling the scene is done by grabbing with both hands. Visual depth perception was enhanced by modifying the shader rendering the points. This was done by adding: a black outline around the points and a lighting effect like eye dome lighting, a non-photo-realistic lighting technique [63]. The result of this can be seen in Figure 14a. New display modes were also developed, and the user could choose between having the point colors as original RGB color, classification color, or displaying a height map. An image cylinder was also developed, as seen in Figure 15. The final desktop and VR menus can be seen in Figure 18 and Figure 19. Besides loading and saving point clouds, the functionality is the same in the desktop and the VR menu.

4.3.2 Evaluation

In order to evaluate the last and final version of PCLabel.VR, several experiments were set up, and a user form was used. These were a System Usability Test, a comparison of PCLabel.VR to a state-of-the-art desktop application, and expert statements.

Questionnaire

The system usability score questionnaire was presented to the testers after the comparison with the desktop application was finished. The questionnaire is an extended SUS questionnaire, as shown in table Table 3. A total of 8 persons answered this questionnaire. The results had to be split over multiple tables, and can be seen in Table 4, Table 5 and Table 6.

ID	Question	Answer type
1	Do you have any experience with VR?	Yes / No
2	Do you have any experience with 3D programs (modeling, CAD, point clouds etc)?	Yes / No
3	Do you have any experience with point clouds?	Yes / No
4	Have you finished technical education on a bachelor's or master's level?	Yes / No
5	I think that I would like to use this system frequently	SUS (1 - 5)
6	I found the system unnecessarily complex	SUS (1 - 5)
7	thought the system was easy to use	SUS (1 - 5)
8	I think that I would need the support of a technical person to be able to use this system	SUS (1 - 5)
9	I found the various functions in this system were well integrated	SUS (1 - 5)
10	I thought there was too much inconsistency in this system	SUS (1 - 5)
11	I would imagine that most people would learn to use this system very quickly	SUS (1 - 5)
12	I found the system very cumbersome to use	SUS (1 - 5)
13	I felt very confident using the system	SUS (1 - 5)
14	I needed to learn a lot of things before I could get going with this system	SUS (1 - 5)
15	I found it easier to navigate the point Cloud in PCLabel.VR than in Cloudcompare	SUS (1 - 5)
16	I felt more confident using Cloudcompare than PCLabel.VR	SUS (1 - 5)
17	I did not feel confident labeling the point cloud in PCLabel.VR	SUS (1 - 5)
18	I felt that I could label the point cloud with high accuracy in PCLabel.VR	SUS (1 - 5)
19	Was there anything you would like to improve in PCLabel.VR? Please specify	Long Text
20	What did you like about PCLabel.VR? Please specify	Long Text

Table 3: Extended System Score questionnaire

ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
1	Yes	Yes	No	Yes	2	1	5	1	5
2	Yes	Yes	Yes	Yes	4	2	2	2	3
3	Yes	Yes	No	Yes	4	1	5	2	4
4	No	No	No	No	1	1	4	1	4
5	No	No	No	Yes	1	2	4	4	5
6	No	No	No	No	1	3	4	1	4
7	No	No	No	No	1	2	3	4	4
8	Yes	Yes	Yes	Yes	2	1	5	1	4
Avg					2	1,625	4	2	4,125

Table 4: Questionnaire results part 1

ID	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18
1	1	5	1	5	2	5	1	1	5
2	2	4	2	4	2	5	1	3	3
3	1	5	1	4	1	5	1	1	5
4	1	5	1	4	1	5	1	2	4
5	1	5	2	3	2	5	1	2	4
6	2	5	1	5	2	5	1	2	4
7	2	3	3	3	3	5	1	2	4
8	2	5	1	5	1	5	1	1	5
Avg	1,5	4,625	1,5	4,125	1,75	5	1	1.75	4.25

Table 5: Questionnaire results part 2

ID	Q19	Q20
1		Liked that it's very easy and fast to navigate in VR. Also thought the UI is simple and easy to use
2	Better and more descriptive VR-menu. Turn on and off classification groups. Desktop interface.	Very simple to learn VR-trigger based interaction. Decent framerates. Space-based skybox lobby. Lots of dots.
3	I would suggest locking the ground to the horizontal axis, other than that the experience was great!	Easy to navigate the 3D space in the point cloud, and very easy to seperate objects in depth
4		Could see things from many perspectives
5		Satisfying to label. Immersion.
6		Much easier to get an overview and navigate in VR
7		Was fun to be in VR and feel the immersion. I liked to label
8	Desktop support. Better VR menu.	Liked the immersion. The labeling was easy to perform. Easy to get an overview

Table 6: Questionnaire results part 3

Comparison With Desktop Application

In order to gather quantitative data on the pros and cons of point cloud labeling in VR, a test comparing labeling in a VR environment and a desktop environment was designed. The test comprised of doing the same task in both VR and desktop, and looking into which approach was fastest. The test procedure can be seen in Section 4.3.2. The results of this test can be seen in Table 7.

Test procedure

1. Look at the video showing where 4 misclassifications are located
2. Measure the time it takes to locate all 4 misclassifications in the application developed in this thesis
3. Measure the time it takes to locate all 4 misclassifications in Cloudcompare
4. Repeat steps 2 and 3 three times, alternating between the application developed in this thesis and Cloudcompare

Timings for desktop and VR								
ID	**VR**				**Desktop**			
	Run 1	Run 2	Run 3	Avg	Run 1	Run 2	Run 3	Avg
1	46.0s	21.0s	20.0s	29.0s	89.0s	87.0s	65.0s	80.3s
2	25.7s	18.6s	21.0s	21.8s	95.3s	47.0s	56.2s	66.2s
3	38.0s	16.9s	17.0s	24.0s	83.0s	64.0s	73.0s	73.3s
4	90.0s	62.0s	53.0s	68.3s	150.0s	121.0s	92.0s	121.0s
5	103.3s	84.2s	59.7s	82.4s	126.6s	98.0s	88.5s	104.4s
6	76.0s	69.0s	57.0s	67.3s	103.0s	94.0s	88.0s	95.0s
7	128.1s	24.0s	25.3s	59.1s	233.4s	105.7s	95.5s	144.9s
8	31.0s	23.0s	25.0s	26.3s	51.0s	53.0s	47.0s	50.3s
Avg	**67.3s**	**39.8s**	**34.8s**		**116.4s**	**83.7s**	**75.7s**	
Avg	**47.3s**				**91.9s**			

Table 7: Timings for desktop and VR comparison

4.3.3 Expert Statements

Multiple emails were sent out to industry experts. The email contained a video showing the functionality of the application and a description of what was being researched. Furthermore, they were asked whether such an application could be of interest to them, and which applications they use to work with point clouds. Only a few recipients replied, and just one of the respondents was interested in testing the application. One respondent said that they almost exclusively use automatic segmentation, using macros for extracting data of interest. Manual work would thus only be used to verify classifications or remove points from the scan. Another respondent had previously used manual classification of point clouds. He said that he liked the ability to perceive visual depth. While doing manual classification, they had problems with not being able to perceive visual depth. This caused some misclassifications, and forced them to spend extra time verifying that the classifications were correct.

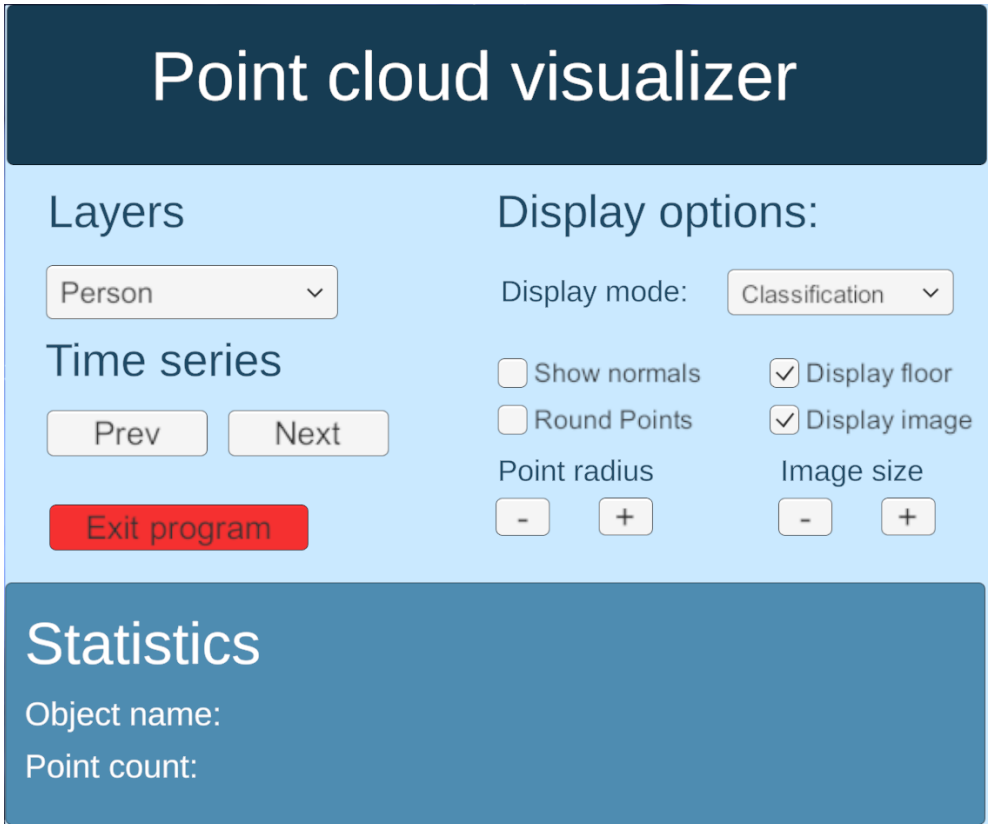


Figure 18: Final VR menu version

A longer statement from an expert user was also submitted. The full statement is: "I have tested the VR point cloud labelling tool on two occasions, one early prototype and the final version. In my opinion the use of VR glasses is very well suited for this application since VR gives a better understanding of the scene and various objects in it. Especially novice users that are not used to 3D point clouds and their visualization on 2D screens will benefit by using the app. The addition of the Ouster LiDAR images in the background, in the final prototype, further simplifies the labelling task, by giving a broader context when the user is very close to a part of the scene. The labelling tool is easy to use and based on a few cases is faster than the existing desktop applications that I have used before. The tool can be combined with a semantic segmentation algorithms that can give an initial segmentation of the point cloud such that the user will only refine the result. The ability to load a temporal series is very welcomed especially when the input is the LiDAR data generated by an autonomous vehicle. Moving from frame to frame is seamless and easily done. Finally, depth enhanced point cloud shading is a very welcomed addition to the application as it generates views that are easier to understand compared to simple colour based shading, which is most commonly employed combined with point cloud visualization".

Figure 19: Final desktop version

Question	**Score**
I think that I would like to use this system frequently	Score
I found the system unnecessarily complex	2
I thought the system was easy to use	1.625
I think that I would need the support of a technical person to be able to use this system	4
I found the various functions in this system were well integrated	2
I thought there was too much inconsistency in this system	4.125
I would imagine that most people would learn to use this system very quickly	1.5
I found the system very cumbersome to use	4.625
I felt very confident using the system	1.5
I needed to learn a lot of things before I could get going with this system	4.125
Score:	76.25

Table 8: System Usability Score results

5 Discussion

5.1 First phase

After gathering feedback in the first phase, it was evident that PCLabel.VR had significant potential. However, there were several problems with the application, with most of them coming from a bad UI / UX. Investing in VR headsets and graphics cards for running an application like this is a substantial investment. The application thus needs to be significantly better than a similar desktop application if it should gain widespread adaption. Although the application is easy to learn, getting up to speed with a new program is also costly. At this point, PCLabel.VRs state was not good enough to compete with the state-of-the-art desktop applications.

5.1.1 The participants

The participants consisted of 4 persons testing the application and coming with feedback. They all had at least some experience with VR, and 2 of them had some experience with 3D modeling software such as Blender [64]. Unfortunately, with only 4 participants, no conclusion can be drawn from the tests. However, this was not a problem as the goal of this first phase was to test the feasibility of such a program and find the major problems with the application in its current state.

5.1.2 Results

All participants saw potential in the application, but it had rough edges and was cumbersome to use. The main takeaway from these tests was that the menu was not good enough. The menu, as shown in Figure 16 was difficult to interact with and did not look pretty. The user had to touch the button with the controller to press the menu buttons. This was not implemented well enough, and bad hitboxes meant that more than one button could be hit on a button click.

Another big problem with the menu was that all the buttons for the different layers took much space, and the menu would get too big if all menu options should fit in such a design. Expandable menus or dropdowns were possible mitigations for this problem. When labeling point clouds for autonomous vehicles, SemanticKitti is e.g., using 28 different classes [13]. Having support for the same number of classes would be hard to fit without utilizing dropdown menus.

5.2 Second phase

5.2.1 The participants

The participants consisted of the same 4 persons as outlined in Section 5.1.1, and my supervisors, Gabriel and Frank. Like the first phase, the second phase did not present a finished product but would instead look into possible enhancements. This allowed using a small number of participants, with the goal being to find problem areas instead of doing an evaluation. Using a small number of participants made it manageable to have longer sessions where the application was explored and possible enhancements were discussed.

5.2.2 Results

The main goal of the test was to find the main problems with using PCLabel.VR and look into the best use cases for such a program. As the application was still undergoing extensive development, and the main focus was gathering information to enhance the product, using tools such as a SUS would not be a good time investment. A SUS would let us know whether PCLabel.VR is useful, but would not let us know what a user might find not useful about the application. Therefore, it was opted for an unstructured interview, letting the users test the application and then discuss the interface and their perception of the application.

After the tests, it was apparent that PCLabel.VR had moved in the right direction. Even though it had moved in the right direction, it still needed many improvements. After zooming with the joystick in both the first and the second phase, it was evident that redesigning how to zoom in and out was priority number one. Scaling by using the joystick made the scene drift away, and the joystick was also too sensitive. The newly introduced shading, which used point normals, was not working as well as hoped. This was caused by some points not having enough neighbors with approximately the same point normals, making fitting a plane to the closest neighbors yield bad results. This approach gave much better results when tested on denser point clouds. Normal computation by plane fitting is very compute-heavy, and the results did not justify the added load time, which could be over an hour for big point clouds. Another problem was flipped normals, which made e.g., the ground points only visible from beyond.

Point size was a nuisance point for some of the testers. Using point clouds with a non uniform point distribution, points would either be rendered on top of each other or have significant gaps between each other. For example, a person would have many overlapping points, and the ground would have much space between each point, assuming an equal point radius. This problem was amplified with unlit shading using a uniform color on each point, making it difficult to separate individual points. An example of this problem can be seen in Figure 14. A clear separation of individual points is of utmost importance for an application focusing on labeling and cleaning up individual points. In some instances, this, combined with a non-transparent label sphere, made it hard to know which points were inside the label sphere.

5.3 Third phase

This phase was the final phase in the development and culminated in a proper test. This section will look at how users found the program's usability and delve into the results from the timed task performed in both PCLabel.VR and Cloudcompare, and examine the expert statements.

5.3.1 Test between a VR and a desktop application

Initially, it was planned to support both desktop and VR interaction in PCLabel.VR. This was needed for comparing point cloud interaction in a VR environment and a desktop environment. However, it was chosen during development that the desktop application should instead be an existing application. This was decided because it could be argued that the proposed implementation of desktop interaction and feature set would favor VR interaction and thus void any results that deem VR interaction favorable to desktop interaction. Finding point cloud software with the same functionality as the proposed application was difficult, as point cloud labeling is mainly done using semi-automatic or automatic labeling. This was confirmed by talking with companies in the industry, explaining that manual labeling would be too costly to do at scale. Depending on the quality of the automatic labeling, a person could then look over it manually and verify the correctness and make minor adjustments. As a direct comparison of all the functionality of PCLabel.VR proved difficult, and it does not contain functionality that is present in other programs, it was decided to test a common denominator.

Reproducibility is essential when developing such a test. A reproducible test favors free open source applications, as other researchers will not have to potentially spend lots of money to reproduce. Furthermore, it would be easy to revert to an earlier version if significant application changes are made. With these criteria in mind, it was decided to compare PCLabel.VR with Cloudcompare. An in-depth description of Cloudcompare can be seen in Section 2.2.1. Making the test simple was also important. A simple test that evaluates a simple workflow will contain fewer variables, making the results easier to reason about. With this in mind, it was decided that the participants should locate multiple misclassifications in a point cloud as fast as possible and then compare how much time was needed in both PCLabel.VR and Cloudcompare. The test procedure can be seen in Section 4.3.2.

From the results in Table 7, we can see that all participants produced faster times in PCLabel.VR than in Cloudcompare. On average, PCLabel.VR was almost twice as fast as Cloudcompare. On average, each subsequent run was faster than the previous one, but some participants had performed the task faster in run 2 than in run 3. Other users had a bad start and then significantly improved. Participant 7 is a prime example of this, which after managing to complete the assignment once, he was able to perform it much faster in subsequent runs. One probable explanation for this is that not enough time was being spent on teaching the users the ins and outs of PCLabel.VR before the evaluation began. This effect is more profound for participants not having previous experience with 3D applications and VR, which could mean that experience in those domains is easy to transfer to point cloud applications. Creating a tutorial might help alleviate

this problem. However, it should be noted that this study had few participants, and the number of participants is not big enough to draw conclusions from. Even though Cloudcompare is one of the popular point cloud software solutions, it might be possible that other desktop applications could yield better results. In addition to better navigation, other software solutions could have tools that make it easier to make sense of a point cloud. This could, for instance, be other render modes, automatic segmentation, or the ability to hide certain points, to name a few. In the extensive test of different point cloud software solutions at the beginning of this research, Cloudcompare is amongst the best applications to the best of my knowledge. It was easy to navigate the scene, and was fast and responsive.

5.3.2 Questionnaire

The questionnaire was an extension of the system usability score questionnaire. A system usability score is a widely used system tool for measuring the usefulness of a system. Being system-agnostic, the scores can be compared across different domains. These characteristics made a SUS a good candidate for evaluating PCLabel.VR. However, by being a system agnostic test, it does not capture system-specific feedback. With this in mind, the SUS questionnaire was expanded to ask the participants how they felt PCLabel.VR performed compared to Cloudcompare, and how what they thought about labeling. This gave more data points to analyze and also provided a direct comparison between a desktop environment and a VR environment. It could, for example, be possible that the desktop application was faster than the VR application, but that the users preferred VR because of the immersion.

The system usability score of this application is 76.25, as shown in Table 8. A score over 68 is considered above average, which means that the SUS score of this application is above average with a large margin [41]. Below is a discussion regarding each question in the questionnaire. All answers from the questionnaire can be seen in Table 4, Table 5, and Table 6.

Question 1 to Question 4

50% of the participants had previous experience with VR, and the same participants did also have experience with desktop 3D applications. From the timings in Table 7, we can see that the participants with previous experience with VR also had the best times in both desktop and VR. All but one person who had experience with point clouds answered yes on questions 1 and 2. 5 of 8 participants had finished education on a bachelor's or master's level, with the remaining ones having begun on one. It was a strong correlation between the first 4 questions, and answering yes on these questions was a strong indicator for doing well on the test where desktop and VR were compared. One of the participants had participated in testing in phase 1 and 2, so it was expected that he should perform well.

Questions 5 to Question 14

These questions were the questions comprising the SUS questionnaire. An in-depth explanation of the SUS can be seen in Section 2.1.9. There are several interesting takeaways from questions 5 to 14. Question 5, which states "I think that I would like to use this system frequently" did only get an average score of 2, with 5 being the highest possible score. The reason for why most

participants might not want to use PCLabel.VR frequently could be that they do not work with point cloud classification. Such an application is a niche product made for a specific use case. However, with LiDAR scanners coming into the hands of more people, the ability to view point clouds in VR can be of great interest for them. Apple has, for example, included a LiDAR scanner in their iPhone 12 Pro and iPhone 13 Pro mobile phones [65]. The users found the application easy to use and not being too complex. This can be seen from the results of 6 and 7. One reason the users did not find the system unnecessarily complex could be the limited functionality. The menu did not have any expandable areas or menus to switch between. The button layout was also very simplistic, with no buttons having more than one function.

In question 8, all users except two think that they should be able to use the application without help from a technical person. Users 5 and 7 were in the bottom three worst times in the desktop and VR comparison, and it is a correlation between performance in this test and confidence in the ability to use the application without the support of a technical person. This matches the answers in questions 13 and 14, where the same users felt the least confident in using the system and felt that they needed to learn the most before getting going with the system. The average scores of question 13 and 14 is respectively 4.125 and 1.75, with 5 and 1 being the best possible score. This shows that all users at least did not feel unconfident with the application and that they should be able to learn to use it without too much help.

The best average score from the SUS questions came from question 12, which asks the user if they found the system very cumbersome to use. The average score here was 1.5, where 1 is the best possible score. Although this is a great result, one reason why it gets such a high rating could be the simplicity of the application. Another possible explanation could be the ease of navigation, which the time trial results support.

Question 15 : I found it easier to navigate the point Cloud in PCLabel.VR than in Cloudcompare

All the participants strongly agreed that it is easier to navigate the point cloud in the proposed application than Cloudcompare. If we look at the timings in Table 7, all participants were able to finish the task faster in this application than in Cloudcompare, which is in line with these findings. This could be attributed to the user navigating with his hands.

Question 16 : I felt more confident using Cloudcompare than PCLabel.VR

All the participants strongly disagreed with being more confident with this application when compared to Cloudcompare. This is in line with question 15. The high degree of confidence experienced in this application compared to Cloudcompare could in part be caused by the simple interface of this application. Cloudcompare boasts a large amount of functionality and needs to expose all of this functionality. This could be overwhelming for a new user. If we compare the final menu of this application which can be seen Figure 18 and the Cloudcompare menu seen in Figure 6, the Cloudcompare menu is more crowded and fits much more functionality. For example, changing point radius would require multiple clicks navigating the menu to find the correct button, whereas it is accessible with a single click in this application. Another explanation for the increased confidence in this application could be the intuitive navigation in the VR application making the user feel more confident in being able to achieve their objectives.

Question 17 and 18 : Feedback on labeling

The feedback on labeling was good, with all participants either feeling positive or neutral about it. Question 17, which asks whether the user felt confident labeling the point cloud, got an average score of 1.75, with 1 being the best score. Question 18 asks if the user felt that they could label the point cloud with high accuracy and got an average score of 4.25, with 5 being the top score. Both questions got very good scores, with every participant except participant number 2 answering that they at least felt a bit confident and to some degree believe that they can label the point cloud with high accuracy. It is possible that participant 2 got a bad introduction to labeling, or said that he felt confident in starting to use the application himself before he was ready. As the participant scored well on both the timed test between a PCLabel.VR and Cloudcompare and gave good scores in the SUS, this is a probable explanation for the bad scores on questions 17 and 18.

It was no standardized way to introduce participants to labeling/segmentation. Furthermore, it was not defined any objectives for what they should label. Only evaluating labeling based on what the participants felt about a feature does not necessarily give a complete picture. Participants could, for instance, feel like they performed well, and feel like it was easy, without necessarily being able to label all points in an object correctly. In order to mitigate this problem, a quantitative labeling experiment should have been performed. It could for instance use the same point clouds as in the localization of misclassifications experiment, where 4 misclassifications should be located. However, instead of locating them, they should be labeled correctly. By having ground truth labeling, a percentage of points labeled correctly could be computed. Measuring both the number of misclassifications and the time it took for a user to complete the

labeling/segmentation would have provided much more data to analyze. Realizing that a more thorough labeling evaluation should be performed did, unfortunately happen too late to conduct a new experiment.

Question 19 and 20 : Feedback from the participants

Only 3 participants came with constructive criticism for how they wanted to improve PCLabel.VR, while the remaining 5 participants did provide an answer. Two participants said that they wanted a better VR menu, without specifying specific enhancements they wanted to see implemented. The VR menu was made with components created for traditional desktop applications, and exploring e.g. three dimensional menu components could be of interest in the future. Only 3 out of 8 participants providing any info on what they would like to improve with PCLabel.VR, combined with SUS score of 76.25, shows that most participants liked the application and did not have any suggestions on possible improvements.

When asked about what they liked about the application, a recurring theme was the immersion in the virtual world. In addition, the ease of navigation was another recurring theme. The ease of navigation means that it is easy to change perspective and thus see things from several sides. This, in conjunction with enhanced depth perception, could be one of the reasons that the participants were faster in VR than on desktop.

5.3.3 Expert statements

The expert statements were valuable and gave insight into how professionals would use such an application. The expert statement from Section 4.3.3 saying that they previously have had problems perceiving visual depth, and thus spending extra time on verification is very interesting. This shows a need for an easier way to verify that labeling/segmentation is done correctly. Another expert stated that they almost exclusively use automatic segmentation. Although automatic segmentation has come a long way, verifying the validity of the classifications would still be necessary.

The long-form expert statement in Section 4.3.3 agrees with this sentiment. He sees great potential in such an application. He thinks that it could be well suited to manually adjust the result of an automated labeling process and verify the correctness of the labeling. He also likes the image cylinder presented in Section 3.8 and thinks it makes it easier to understand the scene. It would be of great interest to see whether tools such as the image cylinder can make it easier to understand point clouds. Sparse point clouds without RGB data could be hard to reason about, and additional tools could be of great help.

5.3.4 Research questions

Research Question 1 : Is it possible to present a complex point cloud structure in a VR environment?

Based on the application developed in this thesis, it is evident that it is possible to present a complex point cloud in a VR environment. Both the comparison with a desktop application and the results from the questionnaire show that the users were faster and more confident in a VR environment. The testers liked the immersion and ease of navigation in a VR environment, and it was fast to come up to speed with the application. Previous research has also shown that it is possible to visualize complex point clouds in a VR environment [3] [9] [50].

Research Question 2 : Is scene understanding easier in a VR environment than in a desktop environment? To what degree would an inexperienced user benefit from using a VR environment?

This thesis shows evidence that scene understanding is easier in a VR environment than in a desktop environment. It was faster by a large margin to localize misclassifications in PCLabel.VR than in Cloudcompare, as shown in Section 4.3.2. It could thus be argued that scene understanding is easier in a VR environment than in a desktop environment. Furthermore, all participants felt more confident in PCLabel.VR than in Cloudcompare. It should, however be noted that this test could have favored PCLabel.VR, as discussed in Section 5.3.1. Both experienced and inexperienced users benefited greatly from using PCLable.VR, and it can be argued that an inexperienced user could greatly benefit from using a VR environment for interacting with point clouds.

Previous research has also shown that it can be easier to understand a point cloud in a VR environment than in a desktop environment. This could, for example, be seen in the paper Point Cloud Interaction and Manipulation in Virtual Reality written by Garrido *et al.* [9]. Here, they show that being in a VR environment which provides both immersion and the perception of depth while interacting directly with the point cloud by using your hands feel more natural and straightforward. However, they did not perform any system usability tests or technical user tests during this research, so these results need to be confirmed.

Research Question 3 : Can an inexperienced user perform accurate point cloud segmentation/labeling in a VR environment?

Results from both the questionnaire and the expert statements indicate that a user could perform accurate point cloud segmentation/labeling in a VR environment. As discussed in Section 5.3.2, users at least feel somewhat confident in being able to perform accurate point cloud segmentation/labeling in PCLabel.VR. One key element of point cloud segmentation/labeling is the ability to localize and then navigate to objects that should be classified. As an expert mentioned in Section 4.3.3, verifying the correctness of the labeling could be very time-consuming. With PCLabel.VR providing easier navigation and makes it easier to reason about complex point clouds, it can be argued that this will make labeling easier in a VR environment than in a desktop environment for an inexperienced user. An experiment where users, for instance, classify several objects should be employed to answer research question 3 with more confidence. Such an experiment is discussed in more detail in Section 5.3.2.

5.3.5 Fulfillment of requirements

This section looks at how well the functional and non functional requirements described in section Section 3.2 were fulfilled.

Fulfillment status of functional requirements

In table Table 9 we can see the status of the functional requirements. All requirements were fulfilled by the end of this phase.

ID	Functional requirement	Status
1	A user should be able to load a point cloud from a .las file	Fulfilled
2	A user should be able to navigate around the point cloud	Fulfilled
3	A user should be able to zoom in on interesting areas in the point cloud	Fulfilled
4	A user should be able to label different parts of the point cloud using a predefined set of labels	Fulfilled
5	A user should be able to save the labels in the point cloud to a .las file	Fulfilled
6	A user should be able to change the size of the label sphere	Fulfilled
7	The application should be able to approximate surface normals	Fulfilled
8	The application should be able to render both round and square points	Fulfilled
9	The application should be able to display a color gradient based on height	Fulfilled
10	The application should optionally display a floor	Fulfilled
11	A user should be able to change the radius of the points in the point cloud	Fulfilled
12	The application should be able to show a circular image of the scene if provided	Fulfilled
13	A user should be able to change the radius of the circular image	Fulfilled
14	The application should support time series point clouds	Fulfilled
15	A user should easily be able to navigate time series data	Fulfilled
16	The application should be able to work with point clouds with up to 20 million points with a Nvidia Geforce RTX 3070 or better	Fulfilled

Table 9: Fulfillment of non functional requirements

Fulfillment status of non-functional requirements

It is more difficult to determine whether the non-functional requirements were fulfilled than the functional ones. This is because the non-functional requirements have more nuances and cover more general goals instead of the functional requirements. Because of this, each non-functional requirement will have a brief discussion to see if it was fulfilled. The results can be seen in Table 10.

ID	Non-functional requirement	Status
1	The application should be easy to use	Based on the responses from the extended SUS schema Table 3 the application was relatively easy to use. The users found it easier to navigate the scene in the application developed for this thesis than in Cloudcompare.
2	The application should run smoothly at 90fps	No users had any negative experiences regarding performance. Most tests were performed on a top-of-the-line computer containing an Nvidia RTX 3080. In addition, the point cloud that was used in most tests did only contain 130K points, which is very easy to render for modern hardware. These two factors could have masked eventual performance problems. A known problem regarding performance is that a too big point radius would lead to lots of overdraws, which is very costly and could result in lag. Therefore, while testing larger point clouds, it was important not to use a point radius that was too big.
3	The application should work on most commercial VR headsets	Using SteamVR, the application should work on most headsets. Some controllers do not have the same layout as the Oculus controllers, so creating new mappings is therefore needed. A headset with a different controller layout than the Oculus headset is the HTC Vive [66].
4	The application should be easy to set up on a new computer	Setup on a new computer is straightforward, as the program can be downloaded as a zip file containing an executable and the needed libraries. SteamVR and headset-specific software must also be installed on the computer.

Table 10: Non-functional requirements

6 Conclusion and Future Work

This chapter looks into the conclusions drawn from the results gathered through working on this thesis.

6.1 Conclusion

This thesis has explored whether it is possible to present a complex point cloud structure in a VR environment. Furthermore, it has investigated whether it is easier to gain an understanding of a point cloud in a VR environment than in a desktop environment. Last but not least, it has examined whether an inexperienced user can perform accurate point cloud segmentation/labeling using such a solution. In order to answer these questions, the VR application PCLabel.VR has been developed. The development of PCLabel.VR was successfull, and all requirements specified in Section 3.2 was satisfied. Point cloud labeling worked well, and the application was able to display both big point clouds and time series point clouds.

In order to answer the research questions outlined in Section 1.2, two experiments were performed. 8 participants were introduced to PCLabel.VR and the desktop application Cloudcompare. After getting to know both applications, an experiment where the participants should locate 4 mis-classifications was performed, with 3 attempts in both applications. All participants had significantly better results in a VR environment than in a desktop environment. A questionnaire was also answered by each participant, consisting of the questions comprised of the questions from a SUS questionnaire, questions regarding the participant's background, and questions about which application they felt most confident using. The SUS questions gave a score of 76.25, which is a very good score considering a score of 68 is an above-average score [41]. Answers to the other questions indicate that users preferred to navigate point clouds in a VR environment instead of a desktop environment, and felt more confident in a VR environment.

Based on these results, it is evident that it is possible to present a complex point cloud structure in a VR environment. This is in line with previous research in this area [50] [9]. Furthermore, it can be argued that it is easier to gain an understanding of a point cloud in a VR environment than in a desktop environment, with compelling evidence in both the questionnaire, the experiment comparing Cloudcompare and PCLabel.VR and expert statements. A more thorough argument can be seen in Section 5.3.4. There are indications that a user can perform accurate point cloud segmentation/labeling in a VR environment from both the questionnaire and the expert statements. A better user test is however needed to evaluate to which degree this is correct.

6.2 Future Work

This section presents possible future work. All the possibilities for future work listed would improve either the PCLabel.VR or the research of this thesis.

6.2.1 A study with more participants

A study with a larger body of participants would be of great interest. This study only contained 8 participants, and a larger body of participants would make it possible to do more quantitative analysis on the data.

6.2.2 An experiment testing labeling accuracy

As discussed in Section 5.3.2, labeling/segmentation was not explored in enough detail. An example of an experiment that can be performed to better evaluate labeling/segmentation in PCLabel.VR is to have a user either fix misclassifications or segment an unlabeled point cloud manually. After labeling/segmentation has been performed, accuracy could be measured against ground-truth labeling, and the time it takes could also be measured. If misclassifications are to be used, their location should be known to the user before the experiment starts, as the question to be answered is ease labeling, not ease of localization.

6.2.3 More interaction with experts

It would be of great interest to hear what experts think about this application. This research found evidence that labeling in VR could be faster and easier in a VR environment than in a comparable desktop environment. It is however possible that expert users are so used to the traditional approach that they don't want to change. One contacted expert mentioned that he found it easy to use 2D applications, as he had a trained eye for contextualizing what he sees on the screen. Shortcuts, workflows, and eventual integrations could be looked more closely into to get a better lay of the land.

6.2.4 Porting the application to Oculus Quest

The Oculus Quest and Oculus Quest 2 are VR headsets with all the required hardware to run applications in a single package [67]. This means that a user could use it anywhere without having to connect it to a computer. Not being limited by where a powerful computer is present, it would be much easier to conduct tests as they can be performed anywhere. With the Oculus Quest 2 only costing 299$ [68], it would lower the barrier of entry for using such software. This could, in turn, make it easier for the application to gain adoption, and companies would be less wary about investing in hardware and setup.

During this thesis, porting to Oculus Quest was explored and actively started on but did not reach completion. The Unity game engine can compile applications for Oculus Quest, which is an Android target. However, it must be noted that the Oculus Quest and Oculus Quest 2 only contain a fraction of the computing power of a powerful desktop computer [69]. This would highly limit the number of points that could be rendered. Another issue with porting the application is that the Oculus Quest platform targets the OpenGLES graphics API, whereas PCLabel.VR targets DirectX11. There were some issues regarding writing to compute buffers during the vertex shader

stage, which were not solved. One probable theory is that doing this is not supported by the version of OpenGLES which was targeted.

6.2.5 SLAM based merge of multiple point clouds

As proposed in the SemanticKitti paper, labeling could be done much faster by merging multiple scans using a SLAM system [13]. As many points would be present in adjacent frames, the user would not have to label them more than once.

Bibliography

[1] Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., & Pollefeys, M. 2017. SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. In *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume IV-1-W1, 91–98.

[2] Li, Y., Ma, L., Zhong, Z., Liu, F., Cao, D., Li, J., & Chapman, M. A. 2020. Deep learning for lidar point clouds in autonomous driving: A review. URL: https://arxiv.org/abs/2005.09830, doi:10.48550/ARXIV.2005.09830.

[3] Ramirez, P. Z., Paternesi, C., De Luigi, L., Lella, L., De Gregorio, D., & Di Stefano, L. 2020. Shooting labels: 3d semantic labeling by virtual reality. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 99–106. IEEE.

[4] Ultra-wide view lidar sensor os0. URL: https://ouster.com/products/scanning-lidar/os0-sensor/.

[5] Lidar as a camera – digital lidar's implications for computer vision. URL: https://ouster.com/blog/the-camera-is-in-the-lidar/.

[6] File:surface normals.svg. URL: https://en.wikipedia.org/wiki/File:Surface_normals.svg.

[7] Reality check. URL: https://www.esa.int/ESA_Multimedia/Images/2017/07/Reality_check.

[8] Cuda toolkit documentation. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[9] Garrido, D., Rodrigues, R., Augusto Sousa, A., Jacob, J., & Castro Silva, D. *Point Cloud Interaction and Manipulation in Virtual Reality*, 15–20. Association for Computing Machinery, New York, NY, USA, 2021. URL: https://doi.org/10.1145/3480433.3480437.

[10] Map controllers. URL: https://developer.oculus.com/documentation/unity/unity-ovrinput/.

[11] Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., & Weinberger, K. Q. 2018. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. URL: https://arxiv.org/abs/1812.07179, doi:10.48550/ARXIV.1812.07179.

[12] Wu, B., Wan, A., Yue, X., & Keutzer, K. 2017. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. URL: https://arxiv.org/abs/1710.07368, doi:10.48550/ARXIV.1710.07368.

[13] Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., & Gall, J. 2019. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*.

[14] Vishnyakov, B., Blokhinov, Y., Sgibnev, I., Sheverdin, V., Sorokin, A., Nikanorov, A., Masalov, P., Kazakhmedov, K., Brianskiy, S., Andrienko, , & Vizilter, Y. 08 2020. Semantic scene understanding for the autonomous platform. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B2-2020, 637–644. doi:10.5194/isprs-archives-XLIII-B2-2020-637-2020.

[15] Pointly - 3d point cloud classification made smart, fast, and accessible. URL: https://pointly.ai/.

[16] Romero-Jarén, R., Arranz, J. J., Navas-Sánchez, L., Erduran, E., Martínez-Cuevas, S., & Benito, B. 2021. Automatic segmentation of point clouds in the architecture environment. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B2-2021, 215–221. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLIII-B2-2021/215/2021/, doi:10.5194/isprs-archives-XLIII-B2-2021-215-2021.

[17] Freeman, I., Salmon, J., & Coburn, J. 05 2018. A bi-directional interface for improved interaction with engineering models in virtual reality design reviews. *International Journal on Interactive Design and Manufacturing*, 12, 1–12. doi:10.1007/s12008-017-0413-0.

[18] Satter, K. & Butler, A. 09 2015. Competitive usability analysis of immersive virtual environments in engineering design review. *Journal of Computing and Information Science in Engineering*, 15. doi:10.1115/1.4029750.

[19] Berg, L. & Vance, J. 07 2016. An industry case study: Investigating early design decision making in virtual reality. *Journal of Computing and Information Science in Engineering*, 17. doi:10.1115/1.4034267.

[20] The basics of lidar - light detection and ranging - remote sensing. URL: https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics.

[21] The pcd (point cloud data) file format. URL: https://pointclouds.org/documentation/tutorials/pcd_file_format.html.

[22] Point clouds are eating the world, one application at a time. URL: https://www.sigarch.org/point-clouds-are-eating-the-world/.

[23] Explore our sensor options. URL: https://ouster.com/products/.

[24] Software user manual. URL: https://data.ouster.io/downloads/software-user-manual/software-user-manual-v2p0.pdf.

[25] Normal vector. URL: https://mathworld.wolfram.com/NormalVector.html.

[26] Hoppe, H. 1992. Surface reconstruction from unorganized points. URL: https://graphics.pixar.com/library/Reconstruction/paper.pdf.

[27] Amenta, N. & Bern, M. 12 1999. Surface reconstruction by voronoi filtering. *Discrete Computational Geometry*, 22, 481–504. doi:10.1007/PL00009475.

[28] Wang, Z. & Prisacariu, V. A. 2020. Neighbourhood-insensitive point cloud normal estimation network. URL: https://arxiv.org/abs/2008.09965, doi:10.48550/ARXIV.2008.09965.

[29] Say hello to unity gaming services. URL: https://unity.com/solutions.

[30] Whitepaper. URL: https://web.archive.org/web/20060222000905/http://unity3d.com/whitepaper.html.

[31] Steamvr unity plugin. URL: https://valvesoftware.github.io/steamvr_unity_plugin/.

[32] Profiler overview. URL: https://docs.unity3d.com/Manual/Profiler.html.

[33] virtual reality noun. URL: https://www.merriam-webster.com/dictionary/virtual%20reality.

[34] How do common virtual reality tracking systems work? URL: https://www.mechatech.co.uk/journal/how-do-common-virtual-reality-tracking-systems-work.

[35] Steam hardware software survey: February 2022. URL: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam.

[36] Api documentation. URL: https://github.com/ValveSoftware/openvr/wiki/API-Documentation.

[37] Cpu vs. gpu: Making the most of both. URL: https://www.intel.com/content/www/us/en/products/docs/processors/cpu-vs-gpu.html.

[38] Klug, B. 2017. An overview of the system usability scale in library website and system usability testing. *Weave*, Volume 1, Issue 6, 2017. URL: https://quod.lib.umich.edu/w/weave/12535642.0001.602?view=text;rgn=main.

[39] Sauro, J. 2015. Supr-q: A comprehensive measure of the quality of the website user experience. *Journal of Usability Studies*, Volume 10, Issue 2, 2015. URL: https://uxpajournal.org/supr-q-a-comprehensive-measure-of-the-quality-of-the-website-user-experience/.

[40] Sauro, J. 2011. Sustisfied? little-known system usability scale facts. *User Experience Magazine,*. URL: https://uxpamagazine.org/sustified/.

[41] System usability scale (sus). URL: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html.

[42] Cloudcompare - 3d point cloud and mesh processing software. URL: https://www.danielgm.net/cc/.

[43] Vrmesh point cloud mesh processing software. URL: https://www.vrmesh.com/products/studio.asp.

[44] Orbit 3dm manage, extract, and share 3d mapping data. URL: https://www.bentley.com/en/products/brands/orbit-3dm.

[45] What's new in recap pro. URL: https://www.autodesk.com/products/recap/features.

[46] Civil 3d. URL: https://www.autodesk.no/products/civil-3d/overview?term=1-YEAR&tab=subscription.

[47] Vrtk - virtual reality toolkit. URL: https://vrtoolkit.readme.io/.

[48] Meshlab. URL: https://www.meshlab.net/.

[49] Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., Song, S., Zeng, A., & Zhang, Y. 2017. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*.

[50] Thiel, F., Discher, S., Richter, R., & Döllner, J. 2018. Interaction and locomotion techniques for the exploration of massive 3d point clouds in vr environments. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4, 623–630. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4/623/2018/, doi:10.5194/isprs-archives-XLII-4-623-2018.

[51] Pointcloudsvr. URL: https://github.com/nasa/PointCloudsVR.

[52] Ouster studio lidar visualizer. URL: https://ouster.com/products/software/ouster-studio-visualizer/.

[53] Ouster sensor sdk. URL: https://static.ouster.dev/sdk-docs/index.html.

[54] Steamvr. URL: https://store.steampowered.com/app/250820/SteamVR/.

[55] Oculus for developers. URL: https://developer.oculus.com/blog/tech-note-touch-button-mapping-best-practices/.

[56] Norman, D. 05 1999. Affordance, conventions, and design. *Interactions*, 6, 38–42. doi:10.1145/301153.301168.

[57] Cg programming/unity/billboards. URL: https://en.wikibooks.org/wiki/Cg_Programming/Unity/Billboards.

[58] Kharroubi, A., Hajji, R., Billen, R., & Poux, F. 2019. Classification and integration of massive 3d points clouds in a virtual reality (vr) environment. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W17, 165–171. URL: https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-2-W17/165/2019/, doi:10.5194/isprs-archives-XLII-2-W17-165-2019.

[59] Fitting a plane to many points in 3d. URL: https://www.ilikebigbits.com/2015_03_04_plane_from_points.html.

[60] 16: Kd trees. URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote16.html.

[61] Kdtree. URL: https://github.com/viliwonka/KDTree.

[62] Las specification 1.2. URL: https://www.asprs.org/a/society/committees/standards/asprs_las_format_v12.pdf.

[63] Ribes, A. & Boucheny, C. 04 2011. Eye-dome lighting: a non-photorealistic shading technique.

[64] Blender. URL: https://www.blender.org/.

[65] Your iphone pro has lidar: 7 cool things you can do with it. URL: https://www.howtogeek.com/759121/your-iphone-pro-has-lidar-7-cool-things-you-can-do-with-it/.

[66] About the vive controllers. URL: https://www.vive.com/us/support/vive/category_howto/about-the-controllers.html.

[67] Explore oculus quest. URL: https://www.oculus.com/quest/features/.

[68] Meta quest 2. URL: https://store.facebook.com/quest/products/quest-2/tech-specs/.

[69] So just how powerful is oculus quest 2, really? URL: https://uploadvr.com/oculus-quest-2-benchmarks/.

[70] Steam. URL: https://store.steampowered.com/.

A Appendix

A.1 User manual

Below is a user manual for the application presented in this thesis

A.1.1 Overview

The application is a tool that tries to make it easier to label and inspect point clouds in VR manually. It does only support the .las file format.

A.1.2 Prerequisites

Running this application requires a VR ready computer. In addition to this, SteamVR is needed. SteamVR can be downloaded from Steam [70]. The VR headset does also need a vendor specific application to run. An Oculus headset would for example need the Oculus Home application in addition to SteamVR to run this application.

A.1.3 Basic interaction

Locomotion is done through grabbing the scene using the grab buttons on the controllers. By pressing either the left or right grab button, the scene is attached to the controller and will move along the controller until the grab button is released. If both grab buttons is pressed simultaneously, a combination of zooming and grab is performed. Zooming is used when trying to get a different perspective, and when you need to move big distances. When moving big distances, a good workflow is to zoom out, move, and then zoom back in.

Labeling is done by using the sphere on the right controller. All points inside the label sphere will be labeled with the selected label when pressing the right trigger button. The size of the label sphere can be adjusted with the right joystick.

A.1.4 The menu

The application has both a desktop menu and a VR menu. Apart from saving and loading files, both menus have the same functionality. Loading a point cloud is done by pressing the Load button shown in 19 and then select the desired .las file. It is also possible to select multiple .las files, which will load them as time series data. The point clouds will be sorted alphanumerically on file name. Saving is done by using the Save button shown in 19.

Interacting with the VR menu is done by pointing a laser towards it, and then pressing the right trigger button while hovering over the desired button. The VR menu is attached to the left controller, and can be toggled on and off by using the left trigger. This can be useful while e.g navigating the scene, as the menu does not obstruct what you see.

A.1.5 Time series data

The application supports time series data, which can be useful in many circumstances. This could for example be data from self driving cars. Moving to another frame can be done by using the Prev and Next buttons on the menu. The name of the frame will be shown in the Statistics Pane. Labels can not be transferred to other frames, each frame has to be labeled individually. Going to a different frame may cause frame drops.

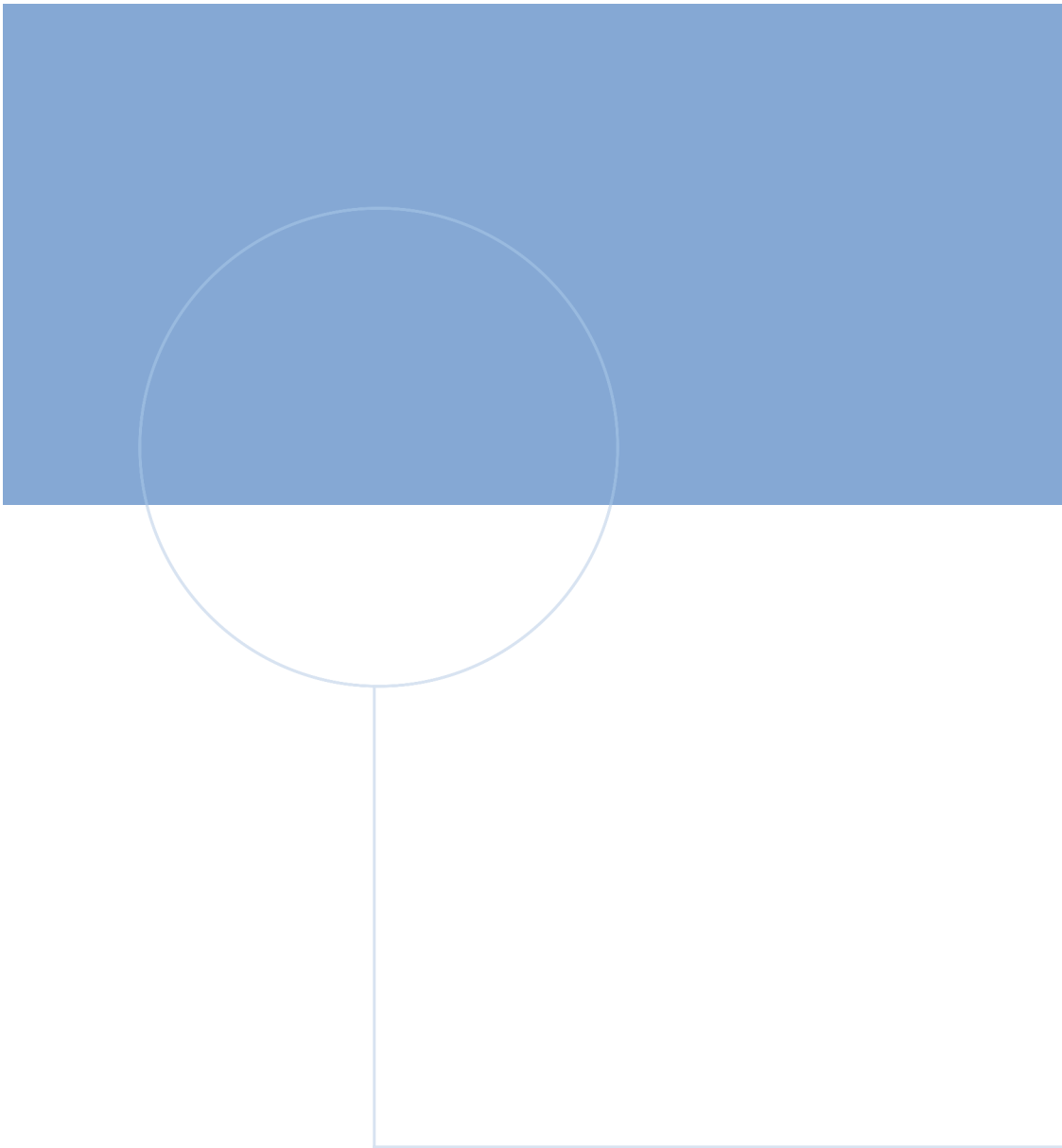
A.1.6 Display modes

The application support 3 display modes. Utilizing the different display modes can make it easier to reason about the point clouds. The display modes are original RGB color from the point cloud, color based on current label, and color based on height.

A.1.7 Performance considerations

There are a few considerations that should be kept in mind in order to get good performance and few / no frame drops. These considerations are listed below.

1. A large point radius causing overdraw is very expensive to render. If the performance is bad, try to lower the point radius.
2. Going to a different frame in time series data could cause frame drops. This is caused by uploading a new point cloud to the GPU.
3. Toggling on point normals will trigger an expensive normal calculation, that can take anywhere from multiple seconds to several hours depending on the number of points. This will freeze application until it's finished.
4. Loading a point cloud with too many points. Using modern hardware, it should support 10 million points or more. This number can vary based on both overdraw and hardware capabilities. While testing on a Nvidia RTX 3080, 30 million points did not cause any frame drops.



NTNU

Kunnskap for en bedre verden