Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

3D Scan Data Annotation in VR

Bachelor Thesis

2023
Martin Purgát

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

3D Scan Data Annotation in VR

Bachelor Thesis

Study programme:	Applied Computer Science
Field of study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	RNDr. Zuzana Berger Haladová, PhD.
Consultant:	Mgr. Lukáš Gajdošech

Bratislava, 2023
Martin Purgát

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Martin Purgát
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: 3D Scan Data Annotation in VR
Anotácia 3D skenov vo VR

Anotácia: Neurónové siete sú veľmi efektívny nástroj strojového učenia na spracovanie obrazových dát, napríklad na odhad 6D polohy v 3D skenoch. Na tréning robustnej neurónovej siete potrebuje dostatočne veľký dataset. Trénovacie dáta môžu byť anotované užívateľom, ale anotačný proces býva zvyčajne neefektívny a dosiahnuť požadovanú presnosť anotovania je ťažké. Táto práca sa zameriava na preskúmanie existujúcich anotačných techník a implementáciu online nástroja pre otestovanie možných výhod anotovania týchto dát vo VR.

Cieľ: Analyzujte existujúce techniky na anotovanie dát pre odhad 6D pózy
Preskúmajte pridanú hodnotu virtuálnej reality v procese anotácie
Preskúmajte existujúce algoritmy spracovania mračna bodov (t. j. registrácia viacerých zobrazení, detekcia kľúčových bodov...), vyberte si jeden a vylepšite ho poskytnutím používateľského vstupu vo VR
Implementujte online nástroj na testovanie poloautomatických prístupov pre anotáciu štruktúrovaných mračien bodov vo VR

Literatúra: Lubos, P., Beimler, R., Lammers, M., & Steinicke, F. (2014, March). Touching the Cloud: Bimanual annotation of immersive point clouds. In 2014 IEEE Symposium on 3D User Interfaces (3DUI) (pp. 191-192). IEEE.

Wirth, F., Quehl, J., Ota, J., & Stiller, C. (2019, June). Pointatme: efficient 3d point cloud labeling in virtual reality. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 1693-1698). IEEE.

Blank, A.; Baier, L.; Zwingel, M.; Franke, J.: Augmented Virtuality Data Annotation and Human-in-the-Loop Refinement for RGBD Data in Industrial Bin-Picking Scenarios. In: Herberger, D.; Hübner, M. (Eds.): Proceedings of the Conference on Production Systems and Logistics: CPSL 2022. Hannover : publish-Ing., 2022.

Hodaň, T., Sundermeyer, M., Drost, B., Labbé, Y., Brachmann, E., Michel, F., Rother, C., & Matas, J. (2020). BOP Challenge 2020 on 6D Object Localization. European Conference on Computer Vision Workshops (ECCVW).

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Martin Purgát
Study programme:	Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: 3D Scan Data Annotation in VR

Annotation: Neural networks are a very efficient machine learning tool for image processing tasks, for example, 6D pose estimation in 3D scans. In order to train the neural network in a robust way, an extensive training dataset is needed. The training dataset can be manually annotated by a user, however, the annotation process is usually inefficient and the desired precision of the annotation is hard to achieve. The scope of the thesis is to explore existing annotation techniques and implement an online tool to test out possible advantages of data annotation in VR.

Aim: Analyze existing data annotation techniques for 6D pose estimation
Research the added value of virtual reality in the annotation process
Examine existing point cloud processing algorithms (i.e. multi-view registration, keypoint detection…), pick one, and improve it by providing user input in VR
Implement an online tool to test out semi-automatic approaches for annotation of structured point clouds in VR

Literature:

Lubos, P., Beimler, R., Lammers, M., & Steinicke, F. (2014, March). Touching the Cloud: Bimanual annotation of immersive point clouds. In 2014 IEEE Symposium on 3D User Interfaces (3DUI) (pp. 191-192). IEEE.

Wirth, F., Quehl, J., Ota, J., & Stiller, C. (2019, June). Pointatme: efficient 3d point cloud labeling in virtual reality. In 2019 IEEE Intelligent Vehicles Symposium (IV) (pp. 1693-1698). IEEE.

Blank, A.; Baier, L.; Zwingel, M.; Franke, J.: Augmented Virtuality Data Annotation and Human-in-the-Loop Refinement for RGBD Data in Industrial Bin-Picking Scenarios. In: Herberger, D.; Hübner, M. (Eds.): Proceedings of the Conference on Production Systems and Logistics: CPSL 2022. Hannover : publish-Ing., 2022.

Hodaň, T., Sundermeyer, M., Drost, B., Labbé, Y., Brachmann, E., Michel, F., Rother, C., & Matas, J. (2020). BOP Challenge 2020 on 6D Object Localization. European Conference on Computer Vision Workshops (ECCVW).

Declaration of honour: I hereby affirm, that I have completed this work independently, using only the cited resources and drawing upon the guidance provided by my supervisor and consultant.

.....

Acknowledgments: I am expressing my sincere gratitude to my supervisor, RNDr. Zuzana Berger Haladová, PhD., whose expertise, understanding, and guidance were instrumental in completing this thesis. My gratitude also goes to my consultant, Mgr. Lukáš Gajdošech, for his valuable advice and help. Lastly, I would like to thank my family, whose unconditional support and encouragement made this work possible.

Abstrakt

S narastajúcim nasadzovaním umelej inteligencie v praxi sa úmerne tomu zvyšuje aj potreba kvalitných dát. Niektoré typy dát sa však získavajú náročnejšie než iné a jedným z nich sú aj priestorové dáta pre odhad polohy objektu. Priestorové dáta sú obzvlášť známe pre ich náročnú manipuláciu v tradičnom desktopovom prostredí. Takéto dáta sú však nevyhnutné pre mnoho aplikácií v oblasti automatizácie a robotiky. V tejto práci predstavujeme webovú aplikáciu využívajúcu virtuálnu realitu určenú na uľahčenie 6D anotácie v RGB-D dátach. Náš prístup využíva intuitívnu povahu virtuálnej reality a transformuje tradične namáhavý proces 6D anotácie na užívateľsky prívetivý. Navrhované riešenie umožňuje používateľom priamo interagovať s mračnami bodov vo virtuálnom prostredí, čím umožňuje efektívnu anotáciu objektov v 3D priestore. Ďalej sme tiež experimentovali s možnosťou zlepšenia existujúcich algoritmov pre zarovnávanie 3D modelov poskytnutím vstupu od človeka v reálnom čase. Na záver sme pre zhodnotenie efektívnosti a použiteľnosti aplikácie vykonali používateľskú štúdiu porovnávajúcu rýchlosť a presnosť anotácie v našej aplikácii s anotáciou v tradičnom desktopovom prostredí.

Kľúčové slová: 6D, 6D anotácia, mračná bodov, RGB-D, virtuálna realita, web, ICP, 3D UI, UX, human-in-the-loop, human-centered computing

Abstract

As the reliance on artificial intelligence expands, the need for data correspondingly rises. Nevertheless, certain data types, such as those frequently used for object pose estimation, prove difficult to acquire. Particularly, spatial data poses challenges when handled within a conventional desktop environment. However, object pose estimation remains crucial in numerous applications within automation and robotics. In this thesis, we present a web application using virtual reality dedicated to facilitating the 6D pose annotation of RGB-D point clouds. Our approach leverages virtual reality's immersive and intuitive nature, transforming the traditionally laborious process of 6D pose annotation into a user-friendly experience. Proposed solution enables users to interact directly with point clouds within a virtual environment, allowing for practical object annotation in 3D space. Furthermore, we also experimented with the possibility of improving existing algorithms for aligning 3D models by providing input from a human in real-time. Finally, to evaluate the application's performance and usability, we conducted a user study comparing the speed and accuracy of 6D pose annotation in our VR environment with annotation in a traditional desktop environment.

Keywords: 6D, 6D pose annotation, point cloud, RGB-D, virtual reality, web, ICP, 3D UI, UX, human-in-the-loop, human-centered computing

Contents

Abbreviations

2D Two Dimensional

3D Three Dimensional

6D Six Dimensional

API Application Programming Interface

AR Augmented Reality

GJK Gilbert-Johnson-Keerthi

GPU Graphics Processing Unit

GUI Graphical User Interface

HCI Human Computer Interaction

HIL Human In The Loop

HIL-ICP Human In The Loop Iterative Closest Point

HMD Head Mounted Display

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICP Iterative Closest Point

IVE Immersive Virtual Environment

PCL Point Cloud Library

PnP Perspective-n-Point

RGB Red Green Blue

RGB-D Red Green Blue Depth

SDK Software Development Kit

SFC Single File Component

UI User Interface

UX User Experience

VR Virtual Reality

Introduction

The world around us shaped us into the beings we are today. Millions of years of evolution have taught us to recognize patterns and make assumptions about the world in a split second. We are here today not because of our ability to question the cosmos but because of our ability not to. We can truly appreciate this fact when we try to describe any of the things we take for granted. For instance, to instantly recognize an object's position and orientation in space with only a glance. Determining an object's position and orientation is also necessary for software applications, ranging from robotics to augmented reality and object recognition. It is, in fact, a cornerstone of many automation advances of recent years. However, computers do not see the world just like we do. Instead, they see numbers, where we see colors and meaning. Consequently, explaining what position and rotation mean in a language they would understand is extraordinarily challenging. Luckily, even though we do not know how to tell them, we know how to show them. However, we still need to show them a lot, like a really, really lot of examples. To produce this many examples requires an astonishing amount of time, effort and can seem like a never-ending task.
Therefore, in this thesis, we have decided to look at this problem and find ways to perform this task more efficiently and make it more enjoyable.
To this end, we implemented a web application utilizing virtual reality.
In the first chapter, we dive into the location estimation problem and unravel its intricacies. We will discuss the challenges of this task and the approaches that have been used to solve it. Moving on to chapter two, we will look at the problem ahead of us and try to plot the right strategy to solve it. Next, we will review some existing solutions to learn as much as possible from them. In chapter four, we will delve into the how-tos of our proposed solution, discuss the technologies we used, and explain the obstacles we encountered and how we overcame them. Finally, we are going to put our solution to the test. We will see how it holds up in the face of real users, and their feedback will help us define the course of future action.

Chapter 1

Theoretical background

"Geometry does not teach us to draw these lines, but requires them to be drawn; for it requires that the learner should first be taught to describe these accurately, before he enters upon geometry; then it shows how by these operations problems may be solved."

-Isaac Newton

1.1 Point cloud

A point cloud is a collection of points in 3D space characterized by three Cartesian coordinates describing each point. This kind of representation is common for 3D data, and it is used in many applications, including robotics, augmented reality, and object recognition. Depending on the sensor and the observed scene, point clouds may differ in resolution, structure, and density. In general, we categorize them into two types:

- **Structured point clouds** Structured point clouds are produced by arranging the points in a predictable way, similar to pixels in a 2D image. The point cloud can be easily represented as a regular 3D array or a matrix. They are often easier to process and analyze, since the regular grid structure allows for efficient algorithms and data structures to be used. Structured point clouds can be obtained, e.g., from depth sensors that use structured light or time-of-flight techniques.

- **Unstructured point clouds** A point cloud is unstructured when it lacks a stable, consistent structure, meaning there is no apparent connection between neighboring points[65]. They can capture more detailed and complex geometry and textures, and are in general more suited for applications that require a high level of detail, such as cultural heritage preservation, geospatial mapping, or VR. These are often obtained from 3D scanners using various photogrammetry techniques.

Pointclouds are in general prone to contain various imperfections[22], such as noise, outliers, and non-uniform sampling. In this thesis, we will be predominantly concerned with the structured point clouds that are generated by RGB-D sensors, which are devices capable of

capturing a scene's color and depth information. Their representation is typically the same as an image. However, in addition to RGB information, each pixel contains a depth value representing the distance from the camera to the point in the scene. Point clouds generated by such sensors are often referred to as *depth maps*, and their resolution is determined by that of the sensor.

1.2 Back projection: from 2D to 3D

RGB-D is a data format that combines the RGB color channels with a depth value, the distance from the camera to the corresponding point in the scene.
Given this information and knowing the intrinsic parameters of the camera, we can compute the 3D coordinates in the camera frame of each pixel in the depth image. The intrinsics define the relationship between the camera's 3D coordinate system and the 2D image plane. They include the focal length f_x, f_y and the optical center c_x, c_y.
The focal length is the distance between the camera's image plane and the focal point, which is the point where the light rays converge after passing through the lens.
There are two focal lengths f_x, f_y to accommodate the fact that the pixel density may not be the same in both directions, horizontal and vertical. Therefore when converting values from metric units to pixels, we need to differentiate between the two.
The optical center is where the camera's image plane intersects with the camera's optical axis.
The process of converting a pixel to a 3D point is relatively straightforward.
To begin, we must convert the pixel coordinates to normalized image coordinates. These coordinates are defined as:

$$u = (x - c_x)/f_x$$
$$v = (y - c_y)/f_y$$

We assigned a vector to each pixel that points from the camera's center to the corresponding point in the scene. Normalized image coordinates have the property that a point at a distance of 1 unit from the camera center is projected to a point at 1 unit from the origin in the normalized image plane. To obtain the 3D point, we multiply the normalized coordinates by their corresponding depth value denoted as d.
Using homogeneous coordinates, we will get the following:

$$X = u \cdot d$$
$$Y = v \cdot d$$
$$Z = 1 \cdot d$$

In order to generate a point cloud representing the 3D structure of the scene observed by the camera, we repeat this process for every pixel in the image.

1.3 Euclidean space

Euclidean space is an n-dimensional affine space over the real numbers, where $n \in \mathbb{N}$ [5]. However, our main focus will be on the three-dimensional Euclidean space, denoted as $\mathbb{E}^3$. Incidentally, the Cartesian coordinates can be identified with $\mathbb{E}^3$, which we are familiar with also as an abstraction of the physical space making it an ideal candidate[1] to guide our further reasoning about the point clouds.
Each point P in the affine Euclidean space can be identified with a point in the Cartesian coordinate system:

$$P \equiv \{(x, y, z) \mid x, y, z \in \mathbb{R}\}$$

Given two points P_1, P_2, we define a *bound vector* $\vec{v}$ as the difference between the point P_1 and P_2:

$$\vec{v} = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1) \in \mathbb{R}^3$$

Bound vectors are characterized by their initial and terminal point, which in our case are denoted as P_1 and P_2 respectively.
The origin of the Cartesian coordinate system O is located at point $(0, 0, 0)$. If we consider a bound vector whose initial point is at the origin O, we get a vector characterized only by its magnitude and direction. We will refer to such object as a *free vector*. A set of all free vectors forms an associated vector space $\vec{E}^3$ to $\mathbb{E}^3$, which is equipped with an inner product defined as:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$$

where a_i and b_i are the components of the vectors $\vec{a}$ and $\vec{b}$ respectively. Therefore, for any point in the Cartesian coordinate system, there exists a unique free vector whose components are the coordinates of that point. Since the Cartesian coordinate system is also orthogonal, it forms a natural basis for the vector space $\vec{E}^3$.
Additionally, we can define a *cross product* between two vectors $\vec{a}, \vec{b} \in \mathbb{R}^3$ as:

$$\vec{a} \times \vec{b} = \begin{vmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{vmatrix} \in \mathbb{R}^3$$

which defines a vector that is orthogonal to both $\vec{a}$ and $\vec{b}$.
These operations allow us to compute the distance between two points, areas, volumes, and other geometric quantities and metrics, which we are going to use extensively in the following chapters.

[1] Strictly speaking, physical space does not conform to Euclidean geometry, and Euclidean space approximates it only over short distances (with respect to the gravitational field's strength), as implied by Einstein's theory of general relativity[40].

1.4 Rigid object

In the context of Euclidean geometry, an object can be thought of as any geometric shape or structure that is defined by a set of points and the relationships between them, which can be expressed in terms of distances and angles. The object can be described by its shape, size, orientation, and position relative to other objects or reference points.
Formally, an object is defined as an affine subspace of $\mathbb{E}^3$ and its associated vector space $\vec{E}^3$, sometimes also referred to as *linear variety*.
Furthermore, we define the deformation of an object as a transformation that does not preserve the size or shape of the object. Objects that can not be deformed are called *rigid objects*.
This assumption implies the following property of rigid objects:

For any two points $P_1, P_2 \in P$, where P is the set of points that define the object, the distance between them is constant.

This statement implies that the size of a vector and the cross product of two vectors must remain the same, which means that their dot product also remains the same.

1.5 Euclidean isometries

The transformations of Euclidean space that preserve distances and angles between points and vectors are called *euclidean isometries*. In other words, an isometry is a transformation that does not change the size or shape of the transformed objects. Examples of Euclidean isometries include translations, rotations, and reflections. All Euclidean isometries form a group, denoted as $E(n)$, where $n \in \mathbb{N}$ is the dimension of the space. However, our main focus in this thesis is on rigid objects. Any transformation of a rigid object describes a so-called *rigid body motion*, a combination of a translation and a rotation.

$$t(\vec{v}) = R\vec{v} + T$$

where R is an $n \times n$ matrix and T is a translation vector.
As we mentioned earlier, such transformation must preserve the cross and dot product of any two vectors, which results in the following:

$$R^T R = I \wedge \det(R) = 1$$

Where I is an identity matrix.
It implies that R is an orthogonal matrix, and since the determinant of R is 1, it excludes the possibility of reflections. Another way to see why reflections are impossible is that they do not preserve the handedness of objects. Therefore, we will be only concerned with a subgroup of $E(3)$. This subgroup is called a Special Euclidean group, denoted as $SE(3)$, also known under a more general term *Lie group*. It defines a *proper rigid transformation* that preserves

the size and shape of the object, but may change the object's position and orientation in space.

1.6 6D pose

The origin of the Cartesian coordinate system is referred to as the *world origin*, and in combination with the basis vectors of the vector space, it forms a frame of reference, also named a *world frame*. However, since the position of the sensor defines our origin, we will refer to it as a *camera frame*. Next, we define an *object frame* as a frame of reference attached to the object. Its origin is usually located at the object's center of mass, but it can be any point of the object. As we have seen, rigid objects' only degrees of freedom are their position and orientation in space. Consequently, their position is sufficiently described by the object frame's position and orientation with respect to the camera frame. It is defined by the following six parameters regarding the camera frame, also called a *6D pose*:

- **Position** 3 coordinates of the object's origin
- **Rotation** 3 Euler angles that define the object's orientation

1.6.1 Position

Let $t = (t_x, t_y, t_z)$ be the position of the object's origin in the camera frame. Then the $\vec{t}$ is a linear displacement of the object's origin from the camera frame's origin.

1.6.2 Rotation

The object's orientation is defined by the Euler angles α, β, and γ, each representing a rotation around the corresponding axis.
All rotation transformations form a group, denoted as $SO(3)$. This group is closed under composition. Hence, any two rotations can be composed into a single rotation. However, the composition is not commutative or nonabelian, so the rotations' order matters.
While there exists a variety of methods for representing rotations, the matrix representation is the most common.

The transformation can be represented by a $4x4$ homogeneous transformation matrix that encodes both the object's translation and rotation.
The following homogeneous transformation matrix then represents the 6D pose of the object:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t^T} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}$$

where $\mathbf{R}$ is a rotation matrix and $\mathbf{t}$ is a displacement vector.
By representing an object's position and orientation in this way, it becomes possible to perform further computations fairly efficiently.

In summary, the 6D pose of an object consists of 6 parameters that define its position and orientation. We demonstrated one possible way to represent these parameters, but it is worth noting that there are other representations whose suitability will depend on the particular application.

1.7 6D pose estimation

Pose estimation is a problem of determining the position and orientation of an object in the world, given a sensor observation. It may be a single image, a video, or, as in this thesis, a point cloud.
There are many challenges that we are faced with when trying to solve this problem algorithmically. Among common obstacles are cluttered scenes, occlusions, sensor noise, variations in lighting conditions, object appearance, and more.
According to Nejatishahidin et al.[41], there are four ways to define this problem:

- **Classification** This approach discretizes the rotation space and casts the 3D rotation estimation into a classification problem. From there, we can refine the pose estimate using, e.g., an ICP algorithm.
- **Regression** It treats pose estimation as a regression problem, where the goal is to predict a continuous output (the pose parameters) from the input data.
- **2D-3D correspondences** When working with 2D images, it is possible to find correspondences between 2D points in the image and points in the model and then using these correspondences to estimate the pose.
- **3D-3D correspondences** It is similar to the previous class of solutions, but instead of utilizing 2D points, we work with 3D points.

There are many ways to conceptualize the pose estimation problem, and each of them involves using different methods and algorithms to solve the problem. In this thesis, we will take a closer look at the 3D-3D correspondences approach.

1.8 Ground truth data

Ground truth data is a set of data representing the world's actual state. It is usually in the form of some metadata accompanied by a dataset. Regarding 6D pose estimation, this could entail a collection of labels carrying information about the 6D poses of objects within the dataset.
The ground truth data is also referred to as *annotation*, and the process of obtaining it is called *annotating*. It is a crucial step in the creation of dataset. Ground truth data can be obtained in several ways. According to Krig et al.[30], we can distinguish between:

- **Synthetic** This data is generated from computer models or renderings with predefined labels.

- **Real- produced** The data is produced by implementing a preconceived scenario in the real world.
- **Real- selected** Data selected from the real world, e.g., generated by a sensor deployed in production.
- **Machine-automated annotation** Feature analysis and learning methods extract features from the data and produce the annotations.
- **Human annotated** A specialized software can be used which can, with the help of a human, determine the precise location of features or objects in a scene.
- **Combined** Any combination of the above methods.

1.9 6D pose annotation techniques

There are several ways to annotate 6D poses, and all options mentioned in the previous section can be applied. The most common way to annotate a 6D pose is to use specialized software that allows the user to label the objects in the image manually. Users can paste the 3D model of the object into the image and adjust its position and orientation until it matches the object in the image. Alternatively, they can select corresponding points in the image and the CAD model, and the software will compute the 6D pose (e.g., using PnP algorithm[48]). Another common step in the annotation process is to preprocess the data by auto-generating the annotations algorithmically and then manually correcting them. Often datasets contain images of some scene taken from multiple viewpoints, which can be leveraged to create annotations for one viewpoint and then project them to the other viewpoints using the known camera parameters.
Another popular technique is to use fiducial markers, unique markers designed to be easily detectable[61]. They are placed on the object so that the pose can be recovered. However, this technique can be more time-consuming and not feasible when dealing with objects with complex geometry or in real production environments.
A comprehensive review of 6D pose annotation techniques is beyond the scope of this thesis, but in general, it usually consists of some combination of manual and automatic annotation.

1.10 Virtual reality

VR is a technological advancement that places users within a simulated setting. It has witnessed increased recognition and commercial utilization in recent periods. Lewis et al.[34] dates the inception of VR back to Morton Helig, a notable figure in cinematography, who secured a patent for the Head-Mounted Display (HMD) in 1960. This innovative device could present images, generate sounds, and create air currents. Helig's subsequent development, the *Sensorama Simulator* in 1962, was another apparatus trying to simulate reality.

Further progression in VR was achieved by Ivan Sutherland's *Sword of Damocles* in 1968. It was the first HMD that facilitated the rotation of the user's virtual viewpoint in alignment with their physical head movements. Its semi-transparent feature also classified it as the first system incorporating augmented reality elements. From 1970 to 1990, VR was primarily utilized for simulations and specialized training. However, the 1990s saw considerable shifts with the announcement of Sega VR and the deployment of Virtuality's multiplayer VR systems. The 2010 prototype of the *Oculus Rift* marked another significant advancement in VR technology. This device, offering a 90-degree field of view, established a new standard in the realm of HMD-based VR.
The concept of VR can be dissected into four fundamental components[2]:

- **Virtual environment** Virtual environments are digitally designed spaces that provide means of interaction between humans and machines. They further enhance human perception by extending visual information to 3D and facilitating interaction with the displayed data, which is beneficial in tasks like teleoperation in surgical simulations.

- **Immersion** VR immersion has two main components. There's immersion itself, which refers to the extent to which a VR system can accurately reproduce sensory experiences, and secondly, presence, which describes the personal psychological response of the user to the VR system[55]. The higher the level of immersion and presence is, the more seamless transition between real and virtual realms becomes.

- **Sensory feedback** Through VR, individuals can have an embodied experience, like maneuvering a simulated aircraft, with the system offering immediate perceptual responses.

- **Interactivity** A VR experience necessitates a closed feedback loop, allowing the user to engage with the digital environment, which can react with the type of interaction based on the simulated scenario.

As VR technology continues to evolve, several research areas have emerged, focusing on enhancing user experience and addressing potential challenges. A significant area of research is locomotion in VR, which refers to the ability to move around in the VR environment. Various techniques have been developed[34], including redirected walking[50], walking in place[33], joystick walking, and teleportation style movement. Each technique has advantages and potential drawbacks, and their development and refinement continue to be a focus of ongoing research. Another popular research topic is the prevention of injuries[3] and sickness caused by VR.

1.11 Conclusion

In this chapter, we have delved, among other things, into the intricacies of point clouds, rigid objects, and 6D pose. We have explored how point clouds, structured and unstructured, are a natural representation of 3D data and how back projection allows us to compute 3D

coordinates from 2D images. We have also examined the properties of Euclidean space and how Euclidean isometries preserve distances and angles in transformations. Furthermore, we have discussed the challenges and approaches in 6D pose estimation, the importance of ground truth data, and various techniques for 6D pose annotation. Lastly, we have touched on virtual reality, its components, and emerging research areas. As we move forward, these foundational concepts will guide us in exploring possible solutions to questions, we will be faced with, in the following chapters.

Chapter 2

Problem analysis and goals

"A problem well stated is a problem half solved."

-Charles F. Kettering

2.1 Problem statement

The demand for precise and efficient 6D pose annotation of RGB-D point clouds has been driven in recent years by the expanding need for various robotics, augmented reality, and autonomous driving applications. Despite this, developing a practical and efficient annotation system remains a challenging task, and to this day, precise 6D pose annotation is an exceptionally time-consuming and tedious process. Various methods have been proposed e.g.[63],[18] or [29] to address this issue. These approaches range from analytical methods to semi-automatic and fully automatic solutions that leverage machine learning algorithms. While many of these approaches have significantly accelerated the annotation process and enhanced the precision of annotations, the dimension reduction resulting from projecting 3D data onto a 2D plane (computer screen) poses significant challenges in the annotation process and is a major obstacle for professionals with limited experience using 3D software.

2.2 Problem analysis

VR technology offers a natural solution to the problem mentioned above. It allows users to interact with 3D data directly within a 3D environment, thereby overcoming the limitations resulting from 3D-2D projection. This interaction can mimic real-world experiences but is not bound by the laws of physics. Therefore, VR not only facilitates more natural interactions than a desktop environment but can also easily overcome the physical world's constraints.
The concept of naturalism in 3D user interfaces (UI) is explored in depth in the article by Bowman et al.[9]. The roots of 3D UI research can be traced back to the 1960s, with focused research emerging in the 1990s alongside growing interest in VR. Two distinct design approaches for 3D interaction techniques have formed over time. The first approach prioritizes high-interaction fidelity, striving to create a realistic and immersive user experience. The

second approach, on the other hand, focuses on developing "magic" techniques, which aim to enhance usability and performance in three fundamental 3D UI tasks: travel, selection, and manipulation. Natural techniques have improved performance in complex manipulation tasks[57]. Furthermore, it has been demonstrated that it is possible to design hyper-natural techniques that feel natural to the user and offer high levels of precision[59].
Moreover, 3D interaction methods utilizing 6-DOF (degrees of freedom) input devices have been found to outperform traditional desktop devices, particularly in applications that require extensive manipulation[36]
Franzluebbers et al.[17] conducted a comparative study to evaluate the performance and user preferences between a VR system and a traditional 2D system for tasks involving counting and annotation. The study revealed that participants could complete the counting task significantly faster when using the VR system, although the precision variance was comparable for both systems. Interestingly, participants reported higher accuracy and efficiency when using the VR system. Moreover, the participants favored the VR system for both tasks. These findings suggest that VR systems not only hold the potential for enhancing efficiency in annotation tasks but also offer a more engaging user experience.
An experiment conducted by Gruchalla et al.[21] further underscores the advantages of VR. In this study, participants were tasked with path editing in an Immersive Virtual Environment (IVE) and a traditional desktop setting. The results indicated that tasks were completed more quickly and accurately in the IVE. The data also suggests that VR may be advantageous for spatially complex tasks.
While VR offers significant advantages, it also presents several challenges that must be addressed. These include the absence of tactile feedback, the potential for motion sickness and disorientation, and the need for users to familiarize themselves with new interaction techniques. Motion sickness is the most critical, as it can render the application unusable for some individuals. The exact cause of visually induced motion sickness remains unclear, but it is currently hypothesized to result from a sensory conflict between the visual and vestibular systems of the brain[42]. Research by So et al.[56] suggests that reducing navigation speed within an IVE can help to mitigate it.
In summary, while VR technology holds considerable promise for enhancing the annotation process, it is crucial to thoughtfully consider the challenges associated with the current generation of VR technology during the design phase. It will help to minimize any negative impact on the user experience and ensure the most effective utilization of this promising tool.

2.3 Goals and solution proposal

The primary objective of this project is to create an intuitive interface within an IVE that allows users to interact with point cloud data and create 6D pose annotations of objects located in the point cloud. We aim to investigate the benefits and challenges of using VR technology for 6D pose annotation. We hope this interface will offer researchers and practitioners across

various fields a more natural method for collecting ground truth data, which in turn could facilitate an easier adoption of recent advancements in AI within their respective domains.
We propose developing a web application that leverages VR technology to accomplish this goal. This application will enable users to upload and visualize a dataset in an IVE. In the IVE, the user can navigate, manipulate models, and annotate the 6D poses of objects within the point cloud.
Additionally, the application will try to minimize the negative effects of VR technology in the three main interaction tasks by utilizing the following techniques:

1. **Travel** Users can navigate the IVE using the Point and Teleport locomotion technique[10]. This technique, which operates at zero speed, effectively eliminates the issue of motion sickness. Although it offers slightly less control over movement, it is an ideal choice for this application, as the IVE can be designed so that complex movement will not be required for the annotation task.

2. **Manipulation** Users can manipulate models using a 6-DOF input device, such as a VR controller. This interaction method, often called the virtual hand metaphor, is considered the most natural way of interacting with objects in VR[45] and takes full advantage of 3D UI.

3. **Selection** To overcome the limited reach inherent to the virtual hand metaphor, users can select objects by pointing at them with a laser pointer attached to the VR controller. This "magic" interaction method, known as the virtual pointing metaphor[46], extends the user's reach within the IVE to all visible parts of the scene.

Furthermore, to address potential privacy concerns, the application will be designed to process all user-uploaded or generated data locally on the user's device, eliminating the need for server-side processing. It ensures that user data remains private and secure.
Lastly, the application will be developed using modern web technologies, making it compatible with all major web browsers and VR headsets. This broad compatibility allows for greater accessibility and ease of use, facilitating the application's availability across various platforms.

Chapter 3

Related works

"It's not where you take things from — it's where you take them to."

-Jean-Luc Godard

In this chapter, we will explore several projects that are relevant to this thesis. While data annotation in VR is a relatively new field of research, numerous solutions have already been proposed. By examining these projects, we hope to identify common challenges, effective strategies, and potential areas for innovation in our application. However, it's important to note that my ability to provide a comprehensive review of these projects is somewhat limited. Despite my best efforts, I was unable to personally test all of these applications, and therefore I refrained from any qualitative commentary. As such, the analysis provided in this chapter is primarily based on the information made available by the authors of these projects, supplemented by additional resources found online.

3.1 PointAtMe

PointAtMe[62] is a web-based VR application for the annotation of 3D point clouds, specifically LIDAR scans. Since this application aims at the datasets used in autonomous driving, it assumes temporal coherence of the data. This allows the application to use the data from the previous frame to initialize the annotation in the current frame.
User is presented with a 3D scene containing a point cloud and a set of tools for annotation. Above the point cloud, a screen with an RGB representation of the point cloud is displayed to help the user better orient themselves in the scene. The central tool to control the application lies in the user's hand, which is represented by a virtual hand.
The user creates the annotations by inserting a bounding box around the object of interest. These can be further accompanied by more information provided by the user.

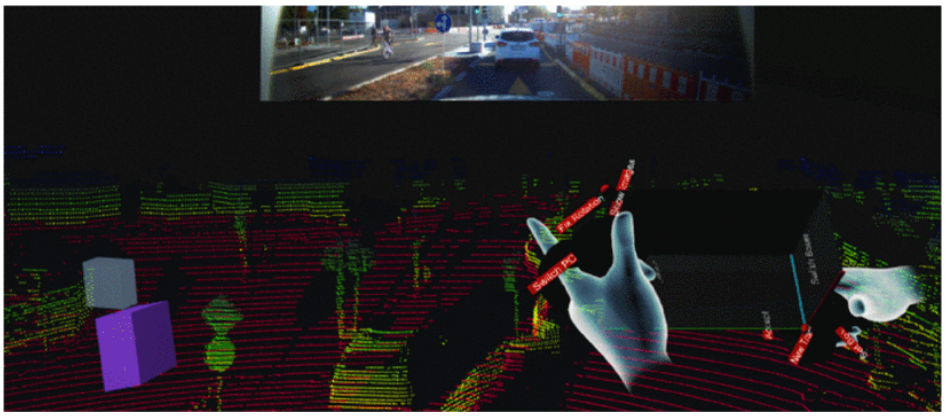

Figure 3.1: PointAtMe annotation tool[62]

3.2 Potree

Potree[53] is a WebGL[37] based library to render point clouds developed by Markus Schütz. This open-source project allows users to visualize massive point clouds in a web browser and stands behind many web-based applications in the field of the cultural heritage or archeology, where 3D scans often times reach staggering sizes. There are many notable initiatives that use Potree for rendering the scans of excavation sites or whole geographical regions.

Figure 3.2: Web rendering of a scan of the Neuchatel region using Potree[53]

3.3 Genuage platform

Genuage[8] is a VR application for the visualization, analysis, and annotation of multidimensional point clouds, meaning each point contains besides its position in 3D space some

additional information. It is an open-source project built in Unity. The application is designed to be used specifically with point clouds generated by super-resolution microscopy images, each containing millions of points. It takes advantage of dual interface consisting of a desktop mode, where user can upload and download data and a VR mode for visualization and interaction with the data. Among its features are various quantification and annotation tools.

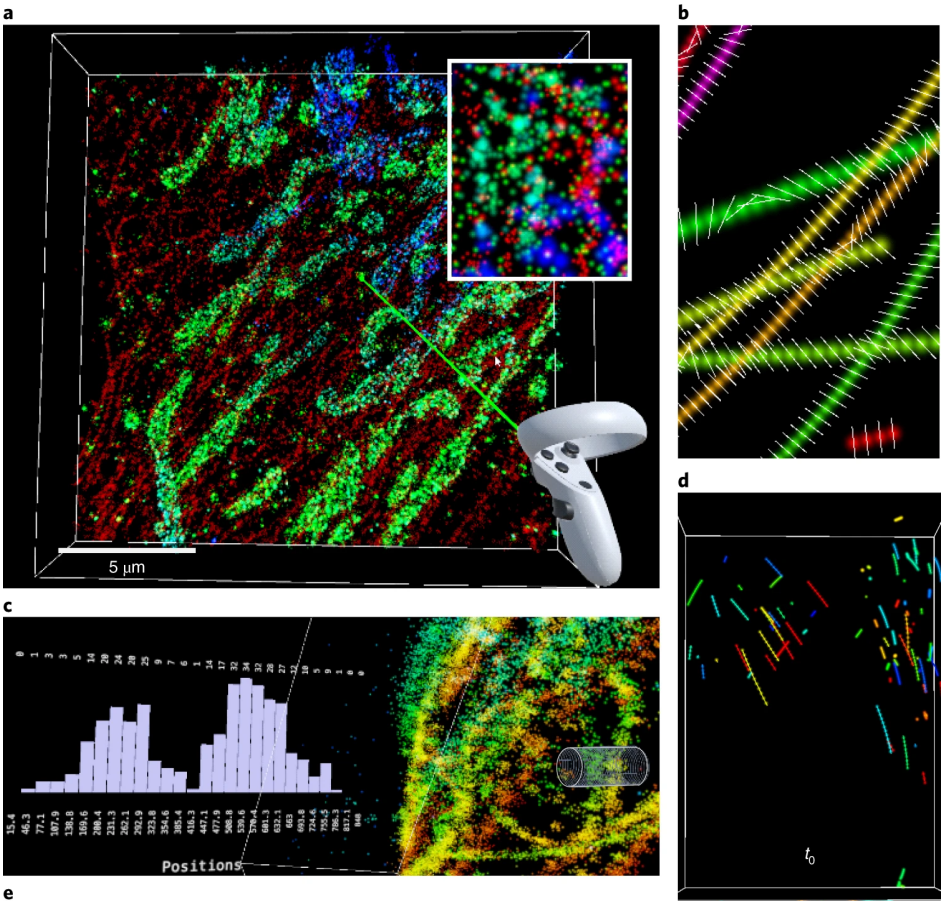

Figure 3.3: Genuage platform[8]

3.4 Conclusion

In this chapter, we have examined several projects that are relevant to this thesis, namely PointAtMe, Potree, and the Genuage platform. Both, Genuage and PointAtme, offer a unique approach to annotating 3D point clouds and their visualization. PointAtMe is a web-based VR application designed explicitly for annotating LIDAR scans. The application provides an RGB representation of the point cloud to aid user orientation and uses a virtual hand as the primary tool for controlling the application. Both elements play important roles in enhancing the overall user experience.

The Genuage platform is designed to visualize, analyze, and annotate multidimensional point clouds. This platform features a dual interface, with a desktop mode and a VR mode. Taking advantage of both interfaces is the right approach for this type of application.

Lastly, Potree, on the other hand, is a WebGL-based library that allows the visualization of massive point clouds in a web browser and highlights the role of data structures and algorithms in the development of efficient point cloud rendering.
The common shortcoming of these projects seems to be the lack of an appealing immersive environment, which could make it undesirable for users to utilize these applications for prolonged periods of time.

Chapter 4

Solution and implementation

"Sometimes, the elegant implementation is just a function. Not a method. Not a class. Not a framework. Just a function."

-John Carmack

4.1 Algorithms and data structures

The subsequent section will describe the algorithms and data structures relevant to the implementation. We will also present an evaluation covering the pros and cons of the algorithms implemented, along with an analysis of their appropriateness for our specific use case.

4.1.1 K-dimensional tree

The K-dimensional tree[13], or a K-d tree, is a unique data structure specifically designed for managing points in a K-dimensional space. It is a binary tree constructed by recursively partitioning the space into two half-spaces along a selected axis. The axis selection cycles through the dimensions of the space.

The partitioning point is chosen as the median of the points, with smaller values stored in the left subtree and larger values in the right subtree. Opting for the median as the partitioning point ensures the tree stays balanced, although there are alternative strategies for its selection.

A key advantage of this data structure and its common use case is to facilitate nearest neighbor search in logarithmic time, significantly enhancing the efficiency of such operations compared to a naive implementation.

Use-case analysis While K-d trees possess certain characteristics that render them less suitable for high-dimensional and non-uniformly distributed data, they are highly efficient for use in low-dimensional Euclidean space with static data. Their utility in performing nearest-neighbor search is particularly noteworthy.

These attributes make K-d trees an excellent fit for our specific use case since we plan to utilize them for just that, operating on a relatively small subset of the point cloud.

The construction of a K-d-tree is outlined in Algorithm 1. The time complexity of this algorithm is generally $\mathcal{O}(n \log^2 n)$, depending on the sorting algorithm employed, where n

represents the number of points in the dataset. The space complexity stands at $\mathcal{O}(n)$, as each point in the dataset must be stored within the tree.

Algorithm 1 K-d tree construction

```
1: function BUILDKDTREE(points, depth)
2:     if points is empty then
3:         return null
4:     end if
5:     axis ← depth mod k
6:     Sort points based on the selected axis
7:     median ← ⌊length(points)/2⌋
8:     node ← new Node(points[median])
9:     node.left ← BUILDKDTREE(points[0 : median], depth + 1)
10:    node.right ← BUILDKDTREE(points[median + 1 :], depth + 1)
11:    return node
12: end function
```

4.1.2 Octree

An octree[64] is a tree data structure specifically devised for storing points within a 3D space. The construction of an octree begins by defining the bounding box encapsulating the space containing the points. The space is then recursively divided into eight sectors, octants, each representing a node within the tree - hence the term "octree."
The leaves of the tree represent the points within the space, with each leaf containing $min(k, n)$ points, where k denotes the maximum number of points a leaf can hold. Therefore, the octant is subdivided until it contains k or fewer points.
This data structure finds extensive application within computer graphics due to its efficiency in facilitating collision detection and ray tracing operations.
Use-case analysis Octrees demonstrate high efficiency when dealing with large volumes of data. It was also evidenced by Potree[53], where the octree structure facilitated the Level of Detail (LOD) rendering of billions of points.
Depending on the specific use case, a potential drawback of octrees is their susceptibility to sparsity in cases of non-uniform data distribution. The octree will be particularly useful in our case for detecting collisions between the point cloud and objects manipulated by the user.
The construction of an octree is outlined in Algorithm 2. Generally, the time complexity of such a naive approach to construct an octree is $\mathcal{O}(n^2)$, where n represents the number of points in the point set. The theoretical space complexity of an octree is $\mathcal{O}(n)$.

4.1.3 Z-order curve

The construction time of an octree can be significantly optimized to $\mathcal{O}(n \log(n))$ by leveraging the Z-order curve[12]. This function represents a space-filling curve that effectively maps a

Algorithm 2 Octree point insertion

```
1: function INSERTPOINTINTOOCTREE(point, octree)
2:     if octree is empty then
3:         octree ← new Octree(point)
4:     else
5:         octant ← find octant of point in octree
6:         if octant is larger than k then SPLIT_OCTANT(octant)
7:             INSERTPOINTINTOOCTREE(point, octant)
8:         else
9:             octant ← add point to octant
10:        end if
11:    end if
12: end function
```

multidimensional space onto a one-dimensional space. An essential characteristic of this curve is its ability to maintain information about the locality of points within the multidimensional space. It is achieved in a manner that effectively couples with an octree structure built from these points.

As a result, once the points are mapped onto the one-dimensional space, they can be sorted, and the octree can be constructed in linear time.

We first need to map the points to positive coordinates. Subsequently, we rescale the points to 2^k, where k denotes the maximum depth of the octree.

Next, we calculate each point's Z-order value or the Morton code. It is achieved by interleaving the bits of the coordinates, starting from the most significant bit and proceeding to the least significant bit.

Let us consider, for instance, a point P located in a 3D space with the following coordinates:

$$P = (1, 7, 2)$$
$$P_x = 1 = 001_2$$
$$P_y = 7 = 100_2$$
$$P_z = 2 = 010_2$$

The Morton code for this point would then be:

$$P_{morton} = 010001100_2 = 140_{10}$$

Subsequently, we sort the points based on their Morton code values and construct the octree. The Z-sort simplifies the construction of an octree in two ways. Firstly, points within the same octant are mapped to adjacent positions within the sorted array. Secondly, the octants can be differentiated by calculating the side length of the smallest bounding box encompassing both points. The interval containing the points is bounded on both sides by a larger side

length of the respective bounding box. It provides us with complete information about the final structure of the octree.

A significant advantage of utilizing the Z-order curve is its ease of implementation and parallelization.

Use-case analysis The Z-order curve proves itself highly beneficial in indexing multidimensional data. It also facilitates the execution of range queries in logarithmic time when using UB trees. In our scenario, we can employ the Z-order curve to construct the octree in $\mathcal{O}(n \log(n))$ time. It represents a significant improvement, mainly when the number of points in the point cloud is significant.

Algorithm 3 Construct an octree using Z-order curve

```
1: function constructOctree(points, k)
2:     zOrderValues ← empty array
3:     minPoint ← find minimum of each dim in points
4:     points ← points + |minPoint|
5:     Rescale points to 2^k
6:     for each point in points do
7:         zOrderValue ← calculate Z-order value of point
8:         zOrderValues ← zOrderValues + zOrderValue
9:     end for
10:    sortedPoints ← Sort points based on zOrderValues
11:    root ← construct Octree from sortedPoints, zOrderValues
12:    return root
13: end function
```

4.1.4 Gilbert–Johnson–Keerthi distance algorithm

The Gilbert-Johnson-Keerthi (GJK) distance algorithm[19] is a method specifically designed for detecting collisions between two convex sets. This algorithm operates based on the Minkowski sum[23] of two sets, A and B. The definition of the Minkowski sum is as stated below:

$$A \oplus B = \{a + b | a \in A, b \in B\}$$

In the GJK algorithm, a form of Minkowski difference is used[1], which is defined as:

$$A - B = \{a - b | a \in A, b \in B\} = A \oplus (-B)$$

The GJK is an iterative algorithm that aims to ascertain whether the origin is within the Minkowski difference between two shapes. If the origin is indeed within this difference, it proves that the shapes are colliding.

The algorithm initiates by calculating the support point of the Minkowski difference in an arbitrary direction. This support point is the point that lies furthest along the given direction.

[1] Not to be confused with a direct inverse of the Minkowski addition.

With each iteration, the algorithm determines if the origin lies inside the object formed by the support points- the so-called simplex. The specific strategy employed for this point selection is beyond the scope of this thesis. Please refer to the original paper[19] for a more detailed understanding of the algorithm.

Use-case analysis The GJK algorithm is highly efficient for detecting collisions between convex shapes. Even though the initial objects' intersection test is usually performed using the bounding volumes; the GJK algorithm can perform more precise collision detection. This algorithm can also be used to insert geometric volumes into the octree. In general, the time complexity of the GJK algorithm is considered to be $\mathcal{O}(\log(n))$, where n is the number of vertices of the shapes. It should be noted that various factors can influence the actual performance of the GJK algorithm. These include the specific shapes of the input objects and their relative positions and orientations. However, the GJK algorithm often performs well and is widely used in physics engines and computer graphics.

4.1.5 Nearest neighbor search

Nearest Neighbor Search (NNS), or similarity search is an optimization problem aimed at determining the point(s) in a set that are most similar or proximate to a specified point.
In other words, given a set of points within a metric space, NNS aims to locate the point in the set closest to a specified query point. The definition of "closest" is domain-dependent and can be decided using various distance measures. These measures can range from Euclidean or Manhattan distance to more complex ones such as cosine similarity or edit distance.
Several algorithms can solve the NNS problem:

- **Brute-force search** The most straightforward strategy would involve calculating the distance between the query point and every other point in the set, subsequently returning the point that registers the shortest distance. While this approach is very straightforward, it is unsuitable for large datasets because it has a time complexity of $\mathcal{O}(n^2)$, where n is the number of points in the set.

- **Space-partitioning data structures** To enhance the efficiency of NNS, data structures such as K-d trees, octrees, R-trees[4], and ball trees[14] can be employed to partition the space and reduce the search time. These methods typically work by dividing the space into regions and eliminating regions from the search that are further away than the currently known nearest neighbor.

- **Hashing-based methods** Techniques such as Locality-Sensitive Hashing[54] aim to hash similar points into the same "bucket." By doing so, the search can be limited to only a few buckets, thus reducing the time complexity.

- **Graph-based methods** Methods such as K-nearest neighbor graph[24] or Navigable small world graph[35] represent the point cloud as a graph, where each point is a node and edges connect close points. The search is then performed by navigating through the edges of the graph.

Use-case analysis Given the specifics of our problem, we already utilize space partitioning data structures to detect collisions between the point cloud and objects in the environment. Consequently, it would be beneficial if we could employ the same data structures to expedite the nearest neighbor search as well.

We will focus on the K-d tree data structure and its application in solving the NNS problem. On average, the algorithm performs in $\mathcal{O}(\log(n))$ time, where n is the number of points in the set.

It is a back-tracking algorithm, as seen in the Algorithm 4 that details its implementation. It is also frequently employed in many implementations of the ICP algorithm to identify the point pairs in two point clouds due to its efficiency and simplicity. While it does have some limitations when dealing with high-dimensional or unbalanced data, given the nature of our data, we can leverage its properties to find correspondences between the point clouds in the ICP algorithm.

Algorithm 4 Nearest-neighbor search using k-d tree

```
1: function NEARESTNEIGHBOR(node, target, depth = 0)
2:     if node is null then
3:         return null
4:     end if
5:     dim ← depth mod k
6:     nextBranch ← null
7:     oppositeBranch ← null
8:     if target[dim] < node.point[dim] then
9:         followingBranch ← node.left
10:        oppositeBranch ← node.right
11:    else
12:        followingBranch ← node.right
13:        oppositeBranch ← node.left
14:    end if
15:    closest ← closerNode(target, NEARESTNEIGHBOR(followingBranch, target, depth +
   1), node)
16:    if |target[dim] − node.point[dim]| < DISTANCE(target, closest.point) then
17:        closest ← closerNode(target, NEARESTNEIGHBOR(oppositeBranch, target, depth+
   1), closest)
18:    end if
19:    return closest
20: end function
```

4.1.6 Voronoi diagram

A Voronoi diagram[1] represents a spatial partitioning into regions based on the points' distance to a specific subset of points within the space. For instance, given a set of points called

sites or seeds, the Voronoi diagram divides the space so that each region contains exactly one site. Each region in a Voronoi diagram is a Voronoi cell. The boundaries between cells, formed by line segments and vertices, are equidistant to the nearest sites. Consequently, every point within a given region is closer to the corresponding point than to any other site. This characteristic makes Voronoi diagrams an efficient tool for nearest-neighbor queries. To find the nearest point (from the given set of points) to an arbitrary location in the space, one must identify the Voronoi cell into which the location falls. The point associated with that cell is the nearest neighbor.

There are various algorithms for computing the Voronoi diagram. However, implementing this in a 3D space is challenging, and many potential pitfalls must be avoided to achieve a correct result. For more detailed information about the implementation, please refer to the paper by Ledoux[32], who provides a comprehensive explanation of the problem in much-needed detail.

Use-case analysis The Voronoi diagram finds extensive application across various fields due to its natural solution to nearest-neighbor search. We may also leverage this property to expedite the nearest neighbor search in our application, specifically within the ICP algorithm. The time complexity of constructing a Voronoi diagram will depend on the specific implementation. However, the most efficient algorithms can perform in $\mathcal{O}(n^2 \log n)$ time, where n is the number of sites, and often closer to $\mathcal{O}(n \log n)$ in practice, given that the points are uniformly distributed.

4.1.7 Kabsch algorithm

The Kabsch algorithm[28] was first introduced by Wolfgang Kabsch in 1976 and is utilized to determine the optimal rotation matrix that minimizes the root mean squared deviation between two paired sets of points. This method is commonly employed in structural bioinformatics to establish the optimal alignment of two protein structures. It is because the molecules are rigid objects, and direct correspondences between atoms in the two structures are guaranteed to exist.

Given these correspondences, we can compute their cross-covariance matrix and apply singular value decomposition (SVD). As a result, we obtain a decomposition of the matrix into three separate matrices.

$$A = U\Sigma V^T$$

where A represents the cross-covariance matrix, U denotes the left singular vectors, and V signifies the right singular vectors. U and V are orthogonal matrices, while Σ is a diagonal matrix. The optimal rotation matrix R is then defined as:

$$R = VU^T$$

For a comprehensive justification of why this statement is valid, please refer to the paper by Lawrence[31], who provides an algebraic proof of the Kabsch-Umeyama algorithm[58]. Additionally, the proper formulation of the problem was first introduced by Wahba in 1965[60],

a problem now widely known as the Wahba's problem.

Use-case analysis Many algorithms build upon employing the SVD decomposition to extract data properties. In the context of this application, we use it to find the optimal rotation matrix to align the model of the annotated object with the point cloud. However, the Kabsch algorithm described in Algorithm 5 operates under the assumption that the correspondences between the two point sets are known and exist. It is different in this application, as the sampling of the point cloud is performed in an unsupervised manner. Therefore, we will utilize the Kabsch algorithm only as a subroutine within the ICP algorithm, which will be discussed in the following section.

Algorithm 5 Kabsch algorithm

function KABSCH(P, Q)
  $rotation$, $translation$
  $C_P \leftarrow \text{centroid}(P)$
  $C_Q \leftarrow \text{centroid}(Q)$
  for $i = 1$ to n **do**
    $P[i] \leftarrow P[i] - C_P$
    $Q[i] \leftarrow Q[i] - C_Q$
  end for
  $covMatrix \leftarrow P^T Q$
  $U, S, V^T \leftarrow \text{SVD}(covMatrix)$
  $d \leftarrow \text{sign}(\det(VU^T))$
  $rotation \leftarrow V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix} U^T$
  $translation \leftarrow C_Q - RC_P$
  return $rotation, translation$
end function

4.1.8 Iterative closest point

The Iterative Closest Point (ICP)[6] is a class of algorithms used to solve the registration problem in computer vision. The registration problem involves finding the transformation between two sets of points. Unlike the Kabsch algorithm, ICP relaxes the assumption that the correspondences between the two point sets are known.

Since it is generally impossible to find the optimal transformation matrix without knowing the exact correspondences, the ICP algorithm iteratively "guesses" the correspondences and then attempts to find the optimal transformation by minimizing the defined cost function.

The Algorithm 6 provides a high-level description of the algorithm.

Algorithm 6 General ICP algorithm

Require: P, Q (two sets of points), $maxIter$ (maximum number of iterations)

Ensure: P' (transformed point set), $error$ (final error)

```
iter ← 0
error ← ∞
while error > tolerance and iter < maxIter do
    correspondences ← identifyCorrespondences(P, Q)
    T ← optimize(correspondences)
    P ← applyTransformation(P, T)
    error ← computeError(P, Q)
    iter ← iter + 1
end while
return P, error
```

Each step can be implemented in various ways, leading to different variants of the ICP algorithm. The suitability of a particular variant will depend on the nature of the data and the available computational resources.

Use-case analysis We aim to align two rigid objects - an object's model and the target point cloud. We plan to employ the Kabsch algorithm as our optimization step. However, the quality of its result will be directly proportional to the quality of the correspondences between the two point sets. Since it is reasonable to assume that the actual correspondences do not exist in differently sampled point clouds, there are several strategies we can employ to identify the best possible correspondences. Generally, we can categorize them into the following groups[51]:

- **Closest distance - point to point[6]** For every point within the moving point cloud, we can identify its nearest counterpart within the target point cloud. It is the most straightforward approach and works well if the point cloud sufficiently aligns with the model. Nonetheless, this method shows susceptibility to variations in the sampling density between the point cloud and the model.

- **Closest distance- point to the plane** Identify the closest point, and compute the distance between it and the plane defined by the closest point and its k neighbors. This approach is more robust to differences in the sampling density of the point cloud and the model. However, it incurs the cost of computing the plane for each point, which may be prohibitive, especially in real-time applications.

- **Closest compatible point** This method is similar to the closest point method, but it only matches points that have a normal within 45 degrees of the source normal.

- **Normal shooting[11]** This technique identifies the intersection point where a ray, emanating from the source point and aligned along its normal, meets the destination surface.

- **Normal shooting to a compatible point** Similar to normal shooting, but it only matches points that have a normal within 45 degrees of the source normal.
- **Projection[7]** This method projects the source point onto the target point cloud, e.g., from the point of view of the target point cloud's range camera.
- **Feature matching** We may also exploit features of the model to find the correspondences. This approach is the most robust to differences in the sampling density of the point cloud and the model. Numerous methods exist to identify the features, ranging from analytical to AI-based approaches. It is usually also a go-to method in real-world applications due to its robustness and speed.

We can further filter out invalid correspondences by the following methods:

- **Distance thresholding** This method involves discarding all correspondences that exceed a certain distance threshold.
- **Statistical analysis[47]** This technique uses statistics to reject wrong correspondences, e.g., rejects all correspondences with a certain number of standard deviations away from the median.
- **Weights[51]** In this approach, weights are assigned to each correspondence and used in the transformation computation. This method can be combined with information about the sensor's accuracy, specifically the probability distribution of the distance between the actual and measured point. It is particularly relevant for depth cameras, which are known to have decreased accuracy at greater distances.
- **Neighborhood consensus[15]** This method rejects all correspondences unsupported by a certain number of neighbors based on a specific metric.

In summary, considering that our application and all data processing are executed directly in the browser and are intended to be compatible with a wide range of VR devices, the computational cost of each method is a significant consideration. As such, our baseline method will employ the nearest-neighbor approach to identify correspondences, and we will further refine this selection using statistical analysis. This approach only necessitates a little additional computation beyond what is already available, making it sufficiently fast to serve as a starting point for the human-in-the-loop version of this algorithm, which we will discuss in the following section.

4.1.9 Human-in-the-loop iterative closest point

Human-in-the-loop (HIL) algorithms are a class of algorithms that tightly couple the strengths of human intelligence with the computational capabilities of computers. The central idea is to use the computer to augment human abilities in real time, enabling the computer's actions to complement the user's actions through direct interaction. HIL algorithms typically operate

in an iterative manner, where the computer and the user engage in a feedback loop until one party decides to terminate the process. As a result, these algorithms are highly adaptable to dynamic environments and are generally less prone to errors than their traditional counterparts.

To adapt the ICP algorithm to the HIL paradigm, we first need to distinguish the components of the algorithm that can be effectively executed by the computer from those better suited for human intervention. For example, the computer excels at performing well-defined, computationally intensive tasks, such as computing correspondences - a task that would be laborious for a human. Conversely, the user can give the computer a robust initial estimate of the transformation with just a glance. Poor initial estimates are among the primary reasons for the failure of the ICP algorithm to converge. Simultaneously, the user will serve as the final arbiter of alignment quality and, therefore, has the authority to terminate the process at any time.

Overall, the HIL ICP algorithm will look as described in Algorithm 4.1.9.

Use-case analysis The HIL-ICP algorithm is a key experiment in this thesis, specifically designed for use within the IVE. The user can employ this algorithm in real-time to align the model with the point cloud, thereby creating an accurate 6D pose estimation of the model. The algorithm and the user exchange feedback through visual information. The user can see both the target point cloud and the model's point cloud within the IVE and manipulate the model's point cloud. The algorithm receives both point clouds as input, computes the correspondences, and, using the Kabsch algorithm, determines the optimal transformation between the point clouds. Subsequently, the algorithm updates the model's point cloud within the IVE. As a result, the user can visually assess the alignment outcome and determine whether further adjustments are necessary. This iterative process continues until the user achieves the desired result. While the general concept is straightforward, there are still open questions regarding the trade-offs between the quality of correspondences and the computational cost of the algorithm. Each iteration of the HIL-ICP algorithm must be fast enough to operate in real time. Hence, the algorithm must compute the correspondences quickly. However, the quality of these correspondences is crucial for the algorithm to converge to an optimal solution. Several strategies can accelerate the ICP algorithm, some of which have already been discussed.

- **Reduce the number of input points** Given our specific use case, we can filter out points from the target point cloud significantly distant from the aligned model, retaining only those within a certain distance threshold. The point cloud is already indexed using an octree, which allows this operation to perform at worst in $\mathcal{O}(n)$ time, where n is the number of nodes in the octree. However, the selection region will be considerably smaller in our scenario than the entire point cloud. Therefore, this operation will be much faster and can be used to our benefit.

- **Speed up correspondences computation** It is the most computationally intensive section relative to other algorithm parts. In the best-case scenario, we need to assign each point in the model's point cloud to a corresponding point from the target point cloud,

which makes the time complexity $\Omega(n)$. Therefore, it makes sense to optimize this part. If we are successful, we can additionally use more sophisticated methods to compute the correspondences, such as the neighborhood consensus method. Several strategies can be used to compute the correspondences, each with its trade-offs:

- **NN using K-d tree** It is among the fastest methods to compute the correspondences that can be used without much pre-computation- we still do need to construct a K-d tree first to be able to perform the nearest-neighbor search in $\mathcal{O}(\log(n))$ time.
- **NN using octree and Voronoi diagram** A method was proposed by Drost et al.[16] to use an octree in combination with a Voronoi diagram to compute the nearest neighbors in $\mathcal{O}(\log(\log(n)))$ time. The general idea is to use an octree to index the regions of the Voronoi diagram and then use the octree to find the nearest neighbor of the query point. This method performs in nearly constant time. However, it requires considerable pre-processing; furthermore, in this case, all data processing is done directly in the browser, complicating its implementation further. After some initial tests, it has become clear that this method's time and memory requirements, combined with the javascript heap memory limits imposed by major browsers, make it very problematic to implement. However, the recent release of WebGPU[20] API, which is an API aimed at the browsers for performing computations on the GPU, makes it possible to use compute shaders to perform the pre-processing in parallel and use GPU's memory directly, which could make this method potentially a viable option.
- **NN using feature space** The idea is to extract the features from both point clouds and then identify the correspondences by finding the nearest neighbors in the feature space. The quality of this method is directly dependent on the quality of the method used to extract the features. The features can be extracted using analytical methods such as the FPFH algorithm[52], which can also be used in real-time. Alternatively, a neural network may be trained to extract and match the features[44], which may be a quick method. However, it requires a time-consuming offline step in the form of training.

- **Optimization step** The computational cost of the SVD decomposition to identify the optimal rotation to align two point clouds is negligible in comparison to the size of the point cloud. It should be noted, however, that if we ever decide to optimize a different cost function, we will have to employ a different optimization method, e.g., the Levenberg-Marquardt algorithm[49].

In conclusion, the HIL-ICP algorithm will be utilized within the IVE by the user to further refine the model's pose after the user's initial pose estimation. The algorithm is designed to operate in real time, but the quality of the correspondences is constrained by the computational capabilities of the user's device. Determining the optimal strategy to balance the

quality of the correspondences and the computational cost of the algorithm remains a subject for future work.

Algorithm 7 Human-in-the-loop ICP

Require: P (target point cloud), Q (source/model point cloud), T (initial guess)

while $iter < max\_iter$ **do**
  $Q' \leftarrow T(Q)$
  $C \leftarrow$ Compute correspondences between Q' and P
  $T' \leftarrow$ Optimize T using C
  $T \leftarrow$ User interaction using T'
  if User terminates the process **then**
    break
  end if
  $iter \leftarrow iter + 1$
end while

4.2 Frameworks, libraries, and tools

During the development of the application, several frameworks, libraries, and tools were employed to aid in the development process. The following section provides a brief overview of the most important ones.

4.2.1 WebXR

WebXR[38], an acronym for Web Extended Reality, is a web API. This innovative API enables seamless interaction using a web browser with a spectrum of augmented reality (AR) and VR devices. The WebXR Device API is the successor to the WebVR API, an experimental Web API that could only encapsulate virtual reality. It unveils novel interfaces, such as XRView and XRPose, empowering web applications to present content in VR and AR by harnessing the power of WebGL[37]. The API is part of the standards defined by two W3C groups- the Immersive Web Community Group and the Immersive Web Working Group. The application uses this API for communicating with the user's VR device.

4.2.2 Vue.js

Vue.js[66] is a modern JavaScript framework that helps build user interfaces. In contrast to its all-encompassing counterparts, it allows for a gradual adoption process. Its central functionality concerns exclusively the view layer of the application. Combined with modern tooling and support libraries, it enables the development of advanced Single-Page Applications. The framework's design philosophy lets developers describe their HTML output declaratively, directly tied to the underlying JavaScript state. It is achieved through a system that can track changes to the JavaScript state and perform efficient updates to the DOM. Vue.js also introduces a component composition and reuse system. This system is encapsulated in the

concept of Single-File Components (SFCs). An SFC encapsulates a component's template, logic, and styles in a single file. This encapsulation facilitates modularity and reusability.

4.2.3 Three.js

Three.js[43] is a popular and widely used JavaScript library that simplifies the process of working with WebGL[37]. WebGL is an API designed to render interactive 3D and 2D graphics directly into web browsers. It provides a set of abstractions and utilities that make it easier to create complex 3D scenes, animations, and interactive graphics such as:

- **Scene graph** Three.js provides a hierarchical scene graph that allows you to add and manage different elements in a 3D scene, such as lights, cameras, and meshes.
- **Camera** It supports different types of cameras, like Perspective and Orthographic, which can be used depending on the application's requirements. This project renders the scene using a `PerspectiveCamera` object.
- **Renderer** Three.js includes a variety of renderers; in our project, we use `WebGLRenderer`, which is used to render the 3D scene onto an HTML `canvas` element.

Three.js is a powerful tool that includes many more features than the ones mentioned above. Please refer to the official documentation for more information.

4.2.4 Others

Among other javascript libraries used during the development of the application are:

- **Point cloud processing** *math.js*, *pcl.js* libraries were used for point cloud processing. Namely, the K-d tree NN search was performed using *pcl.js*, and statistical analysis was performed using *math.js*.
- **File processing** *JSZip* library was used for reading and writing zip archives. *UPNG.js* library was used for reading PNG images.

For experiments and testing purposes, I used Python and the following libraries:

- **Scientific computing** *SciPy*, *NumPy*, and *Matplotlib* libraries were used for scientific computing and visualization.
- **Point cloud processing** *Open3D*, *PyPCL*, and *OpenCV* libraries were used for point cloud creation and processing.

For visualization of point clouds and creation of point clouds out of meshes, I have used:

- **CloudCompare** A 3D point cloud and mesh processing software.

Additional supporting libraries were used in the development. Please refer to the source code for complete information about the libraries and tools used in the project.

4.3 Dataset

The dataset format compatible with the application is heavily inspired by T-LESS dataset[27], and the application was developed using it. It contains a collection of RGB-D data for detecting and 6D pose estimation of texture-less objects. The dataset offers for every item two kinds of 3D prototypes: a manually engineered CAD model, and a partially automated reconstructed version. This application used CAD models, which are textureless meshes saved in the PLY format. In addition to the original format, the dataset for this project also requires the folder 'models_pcd' specified below. The dataset is organized into the following folders and files:

- **depth** A folder that contains the depth images in the PNG format. Each file name has the following format: *dddd.png*, where *d* stands for a single digit.

- **models_cad** A folder containing the CAD models of the objects in the scene in the PLY format. Each file name has the following format: *obj_dd.ply*, where *d* stands for a single digit.

- **models_pcd** A folder that contains the point cloud of each model in the models_cad folder in the PCD format. The naming convention is analogous to the models_cad folder.

- **rgb** A folder that contains the RGB images in the PNG format. The naming convention is analogous to the depth folder.

- **gt.yml** A YML file that contains the ground truth data for the 6D pose of each object in each frame.

- **models_info.yml** A YML file that contains the information about the real dimensions of the models in the models_cad folder.

An example of the dataset folder may then look as follows:

```
Dataset
├── depth
│   ├── 0000.png
│   └── 0001.png
├── rgb
│   ├── 0000.png
│   └── 0001.png
├── models_cad
│   ├── models_info.yml
│   ├── obj_00.ply
│   └── obj_01.ply
├── models_pcd
│   ├── obj_00.pcd
│   └── obj_01.pcd
└── gt.yml
```

The application expects the geometries of models, depth values, and ground truth data to be in millimeter units. For more details about the structure of the files and folders, please refer to the T-LESS dataset documentation[26].

4.4 Features and UI

The main goal of this thesis is to develop an IVE that allows users to create 6D pose annotations of RGB-D point clouds. To this end, I decided to implement the below-listed features with the intention of making the application as user-friendly and intuitive as possible. Similarly to the Ganauge platform, the application has a desktop and a VR mode. In the desktop mode, the user can upload a dataset, download the annotations, and enter the VR mode. In the VR mode, the user can annotate the dataset. It includes creating and removing annotations, changing frames, and moving around the IVE.

4.4.1 Features

- **Dataset upload** The user can upload a dataset as a zip file. The zip file must contain the folders with the specified structure. The application will then parse the dataset and display it in the 3D UI.

- **Creating and removing pose annotations** The user can create a pose annotation by choosing an object and placing it in the scene. Object selection can be made by grabbing an already present object in the scene or selecting it using a laser pointer from the model's menu in the IVE. By selecting the model, the annotation is initialized, and the user can adjust its position and rotation using the controllers as an extension of their hands. While annotating, the user can also use the HIL ICP algorithm to refine the alignment of the model with the point cloud. Once the user is satisfied with the annotation, they can confirm it by selecting the 'confirm' button above the displayed point cloud using the laser pointer. Alternatively, they can remove it by selecting the 'remove' button. Both actions will conclude the annotation process. In order to prevent unwanted selections when objects overlap in the scene, only one annotation can be created at a time. If the user wants to create the next annotation, they must confirm or remove the previous one.

- **Changing frames** If no annotation is being created, the user can change the current frame by using a laser pointer and selecting the 'next' or 'previous' button above the displayed point cloud.

- **Annotations download** The user can download the annotations as a YML file by pressing the 'download annotations' button in desktop mode. The downloaded YML file contains the annotations for each frame in the dataset in the format specified in the T-LESS dataset.

- **Travel** The user can move around the IVE by using the controllers. By pointing the laser pointer at the ground and pressing the trigger button, the user can teleport to the selected location. The orientation of the headset determines the orientation of the user.

4.4.2 User interface and experience

- **Desktop** The desktop UI contains a canvas element, where the 3D scene is rendered, and three buttons at the bottom of the screen. The UI is displayed in figure 4.1.

    1. **Canvas** The canvas element is the most dominant part of the desktop UI. It displays the 3D scene in its current state.
    2. **Upload dataset** The user can upload a dataset by pressing the 'upload dataset' button. The button opens a file dialog, where the user can select a zip file containing the dataset.
    3. **VR mode** The user can enter the VR mode by pressing the 'enter VR' button.

4. **Download annotations** Annotations can be downloaded by pressing the 'download annotations' button. The button downloads the annotations as a YML file.

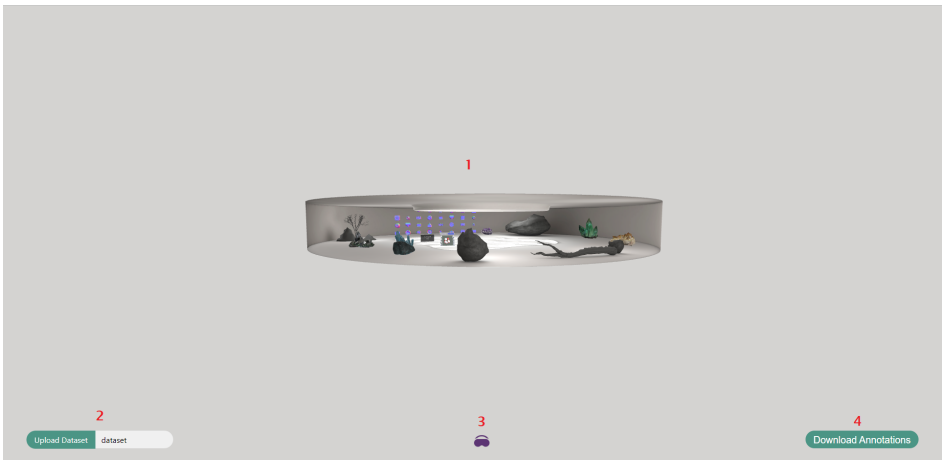

Figure 4.1: Desktop UI after uploading a dataset.

- **VR** The 3D scene represents the IVE. It takes the form of a rounded room to which the user is spawned after entering the VR. The 3D art featured in the room is authored by the artists *Karolina Renkiewicz* and *Christy Hsu*. The 3D UI has several key parts that are shown in Figure 4.2.

  1. **Point cloud** Point cloud is shown in the form of rescaled particles with the color of the corresponding pixel in the RGB image. The size of the point cloud is scaled to be comfortable for user interaction.
  2. **Models menu** The models in the dataset are displayed in a grid structure on the side of the room. By pointing the laser pointer at the model, the model gets bigger and starts rotating to inform the user that they can select it by pressing the trigger button.
  3. **Navigation buttons** Navigation buttons are located right above the point cloud. They can either change the frame or confirm/remove the annotation depending on the current context.
  4. **Annotation panel** The annotation panel is a black semi-transparent panel located by the point cloud. It contains information about the rotation and position of the current annotation.
  5. **Controllers** The controllers are represented by the mesh model of the actual controller, and the user is expected to use them during the annotation process. They can grab the model in the scene by pressing the 'grab' button on the controller if the controller collides with it and modify its position and rotation while holding it. Additionally, by pressing the 'A' button on the controller, the HIL ICP algorithm can be initiated. Lastly, the user can teleport around the scene by pointing the laser pointer at the ground and pressing the trigger button.

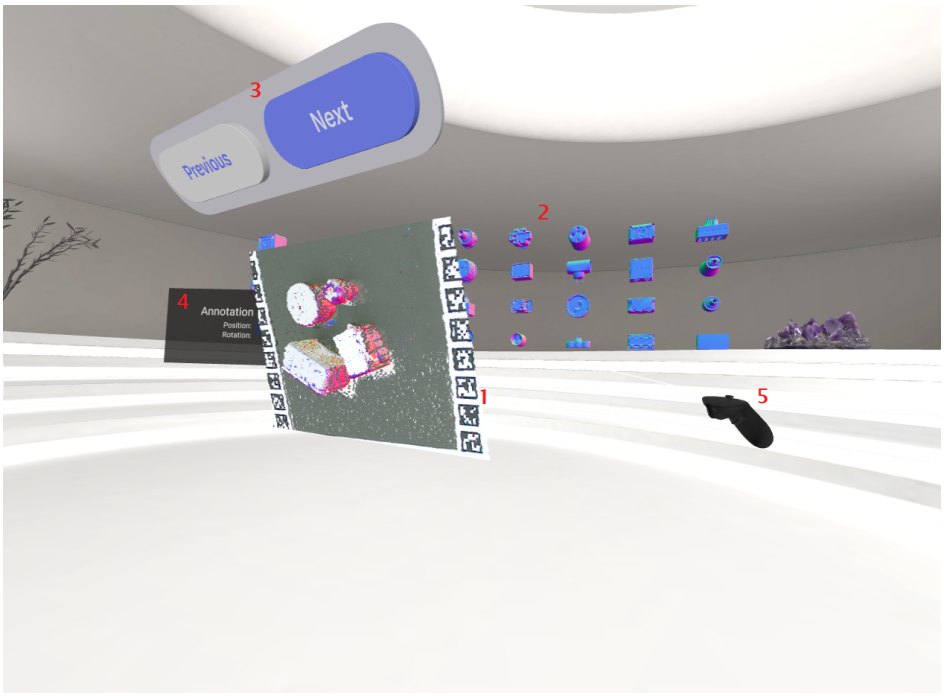

Figure 4.2: 3D UI after uploading a dataset.

4.5 Implementation design

This section provides a high-level overview of the application's implementation design.
It is beyond the scope of this thesis to provide a detailed description of the workings of all parts; therefore, for further details, please refer to the source code.
Since the application is client-side only, it is written in *TypeScript* and uses the *Vue.js* framework.

4.5.1 Structure

The application is divided into vue components, which can be composed to form complex UIs. Therefore, from a structural point of view, the application can be seen as a tree of components as shown in Figure 4.3.

The tree structure is based on the dependencies of the components. *Vue.js* framework also provides primitives that allow the components to communicate with each other, hence it loosely follows the data-flow in the application as well.

4.5.2 Integration

Three key contexts must always work together: WebGL, the application, and WebXR. Let us take a look at each of them in more detail.

- **WebGL context** The WebGL context renders the 3D scene onto the `canvas` element. In the case of this application, it is not controlled directly but via the Three.js library. The context is initialized by creating a Three.js `Renderer` object, which then receives

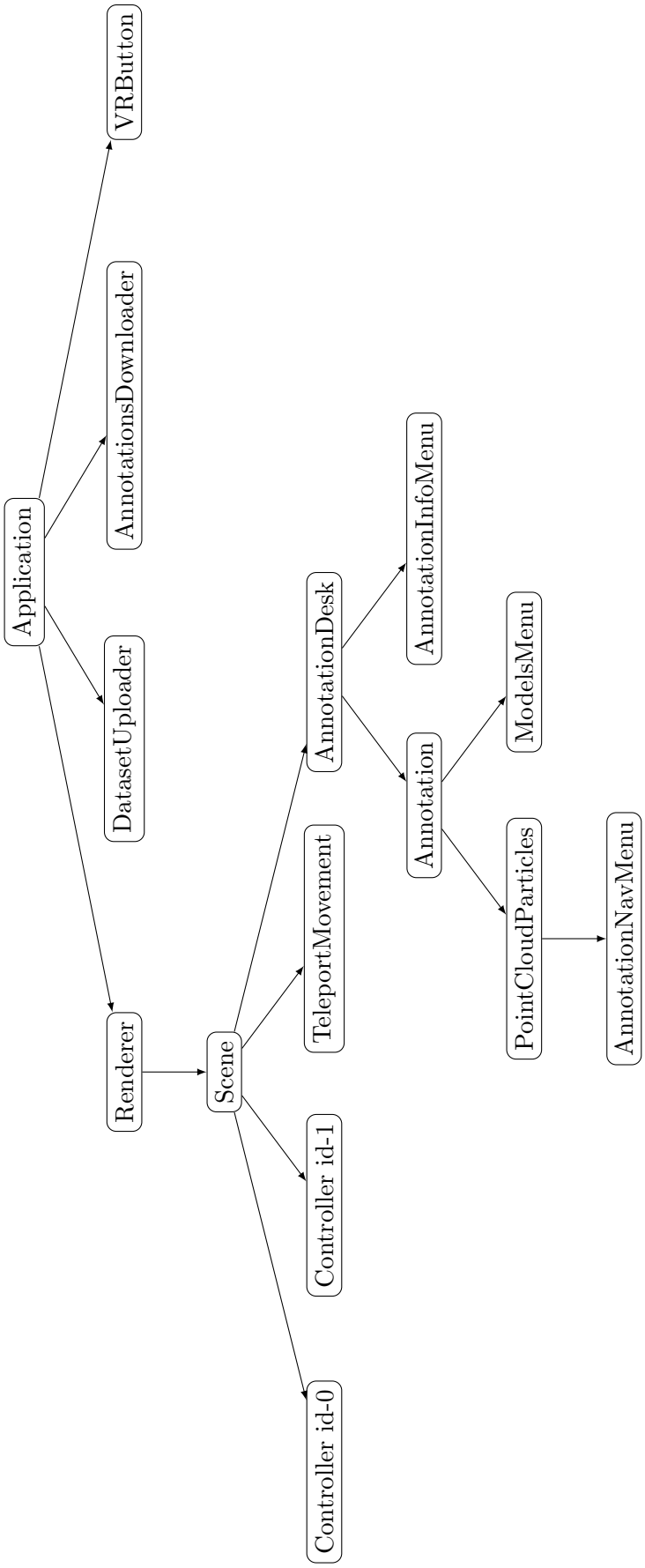

Figure 4.3: Vue component structure.

and takes control of the HTML `canvas` element. Next, we create a Three.js `Scene` object, which represents the scene graph and contains all objects that can be rendered. The `Renderer` is then updated by calling its `render` method, which takes two Three.js objects as its parameters- the `Scene` and the `Camera`. The `Scene` contains the objects to be rendered, and the `Camera` defines the perspective from which the scene is rendered. The `Camera` object is also part of the scene graph. The render method is called in the `requestAnimationFrame` callback, which is called every time the browser is ready to repaint the `canvas`.

- **WebXR context** The WebXR is primarily responsible for initiating, maintaining, and closing the session with a VR device. It is initialized by calling the `navigator.xr.requestSession` method from the global scope. The method returns a promise, which resolves to an `XRSession` object. The `XRSession` object is then used to create an `XRReferenceSpace`, which defines the coordinate system of the VR scene. The `XRReferenceSpace` can be obtained by calling the `session.requestReferenceSpace` method, which takes a string as a parameter. The string defines the type of the reference space. In this application, the type is 'local,' which means that the origin of the reference space is the position of the headset when the session is created. The `XRReferenceSpace` is then used to create an `XRRenderState` object, which in turn is used to define the rendering parameters of the VR scene.

- **Application context** The application state is sustained using *Pinia*, a state management library for *Vue.js*. In this application, it functions as a global store, which is accessible from all components. Among other things, it is responsible for storing the dataset, the annotations, and the current frame. It also facilitates inter-component communication by providing components with information about pressed buttons, selected models, and transformation matrices.

The application is served by *Vite*, a web development build tool that provides a development server. Since the *WebXR* requires a secure connection, the application must be served over HTTPS protocol. The development server is therefore configured to use a self-signed certificate, which is generated on the first run of the application and consequently any device that has access to the network can run the application.

4.6 Conclusion

That wraps up the overview of the application's implementation design. We explored the challenges of developing a VR application and the design decisions we made along the way to overcome them.
In the process, we developed a web-based VR application that allows the user to annotate 6D poses of objects in a point cloud within IVE. The application is designed to be intuitive and easy to use.
Some obstacles, such as the trade-off between the performance and robustness of the HIL-ICP

algorithm, we haven't satisfactorily overcome and require further research.
In the last chapter, we will describe and perform the evaluation of the application and discuss the results.

Chapter 5

Evaluation

"Pay attention to what users do, not what they say."

-Jakob Nielsen

In order to evaluate the application, we will conduct a small user study. The study will be divided into two parts- qualitative and quantitative.

5.1 User study

The qualitative part of the study will evaluate the application's usability, and the quantitative part will compare the accuracy and speed of creating the annotations in this application to an existing desktop solution. The study will be conducted on a small group of people, who will be tasked to correct the annotations in ten frames of the T-LESS dataset.

5.1.1 Experiment design

- **Participants** The participants will be uniformly selected with respect to their technical ability and experience, aged between 18 and 40. There will be *5 participants* in total.

- **Task** The participants will be tasked to correct annotations in ten frames of the T-LESS dataset - five in the desktop application and five in the VR application. The annotations will contain errors in position and rotation but not in the object type. Therefore, the participants will only have to correct the position and rotation of created annotations and not create new ones. The objects and frames supposed to be annotated will be different for VR and desktop. The setup will be the same for all participants.

- **Environment** The experiment will be conducted in a quiet room with a computer and a VR headset. The computer will be equipped with a monitor, keyboard, and mouse. The selected desktop application is Pose Annnotator Toolkit[39], and the VR

application is the one developed in this thesis. The VR headset will be Pico 4, and the computer will be HP Probook 470 G5.

- **Procedure** First, participants will be able to familiarize themselves with both applications and the task. Once they feel comfortable working with the respective application and confirm that they understand the task ahead, they will be able to go through the frames without annotations. Then, they will be presented with the frames with annotations and the core experiment will begin. They will start by correcting the annotations in one of the applications and then switch to the other. The order of the applications will be randomized for each participant. While annotating in the VR application, the participants will be standing and able to move around the scene. In addition, the VR session will be recorded for later analysis.

- **Quantitative measurements** The participants' performance will be measured in both applications regarding the time spent on the task and the accuracy they were able to achieve.

  - **Accuracy** There are many ways to measure the accuracy of the 6D pose estimation, and we will use the same technique as in the T-LESS dataset[25]- the average minimum Euclidean distance between points of the ground truth and estimated pose.

    $$e = \underset{x_1 \in \mathcal{M}}{\text{avg}} \min_{x_2 \in \mathcal{M}} \left\| \left(\overline{\mathbf{R}}\mathbf{x}_1 + \overline{\mathbf{t}}\right) - \left(\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{t}}\right) \right\|_2$$

    The average error $\overline{e}$ is computed both for the original and corrected annotations. The error reduction per annotation is then calculated as the difference between the average original and average corrected error. Therefore, the metric m used to evaluate the precision of the annotation process is defined as:

    $$m = \frac{\overline{e}_{\text{original}} - \overline{e}_{\text{corrected}}}{\overline{e}_{\text{original}}} \cdot 100$$

    and describes the percentage of error reduction per annotation.

  - **Time** The time measurement will start when the participant initiates the first annotation and stop when they confirm the last. Finally, the average time per annotation in seconds will be calculated.

- **Qualitative measurements** The participants will be asked to complete a questionnaire after the experiment. The questionnaire will contain questions about participants' experience with the applications and their preferences. It is part of the appendix 5.1.3. The main goal of the questionnaire is to assess which application the participants prefer and feel more comfortable working in. Each question in the questionnaire will be evaluated with a score of -1, 0, or 1. The score -1 means that the participant prefers the desktop application, 0 means that the participant has no preference, and 1 means they prefer the VR application. Since there are five questions, the maximum score is 5, and the minimum is -5. Furthermore, we will analyze the participants' behavior in the VR application from the recorded sessions and try to identify any usability issues.

- **Ethics** The participants will be informed about the study's purpose and the session's recording. This study will not process personal information besides the data generated during the experiment, and that will be used solely for the purpose of this study. The participants can withdraw from the study at any time.

5.1.2 Hypotheses

1. **H1** *The average time of creating an annotation in the VR application is lower than in the desktop application.*

2. **H2** *The average error reduction per annotation is higher in the VR application than in the desktop application.*

3. **H3** *The participants prefer annotating using VR over the desktop setting.*

5.1.3 Results

The experiment was conducted on 6 participants, and 1 participant was invalidated. First, we will look at the results of the quantitative part, then analyze the results of the qualitative part and finally discuss the shortcomings of the study and summarize the results.

- **Quantitative results** The results are summarized in the table 5.1. The results show that the participants could correct the annotations faster in the VR application than in the desktop application. Also, the accuracy of the annotations was higher in the VR application. The results confirm the first two hypotheses- **H1** and **H2**.

	VR		**Desktop**	
Participant	**Avg. time (s)**	**Error reduction- m (%)**	**Avg. time (s)**	**Error reduction- m (%)**
1	**38**	**97.740**	134	89.225
2	**42**	**98.057**	181	85.793
3	**47**	**97.883**	213	86.082
4	**44**	**97.640**	174	93.545
5	**39**	**98.074**	227	90.531

Table 5.1: Results of the quantitative study

- **Qualitative results** The questionnaire results are summarized in the table 5.2. The results show that the participants preferred the VR application over the desktop application and confirm the third hypothesis- **H3**.

Participant	Questionnaire score
1	5
2	5
3	5
4	5
5	5

Table 5.2: Results of the questionnaire administered after the experiment

Furthermore, we analyzed the participants' behavior in the VR application from the recorded session and identified some usability issues. The participants had problems with the following actions:

- **Object identification** A common issue was with identifying the objects in the point cloud. That can be attributed to the T-LESS dataset containing textureless objects that are difficult to identify if the person is unfamiliar with them.
- **Movement** The participants had problems with minor movements in the scene. When they struggled to reach some object, they had tendencies to make physical movements instead of using a controller. A possible explanation may be that the point and teleport locomotion strategy is not ideal for this, and the participants were trying to compensate for that with physical movement.
- **Annotation creation** Many users kept forgetting to confirm the annotation after creating it. That may be caused by the confirmation button's position, which is not in the user's field of view, hence creating an additional cognitive load and, consequently, unnecessary friction during the annotation process.

- **Discussion** The experiment results show that the participants could correct the annotations faster and more accurately in the VR application than in the desktop application. Furthermore, the participants preferred the VR application over the desktop application. However, the study has some shortcomings. First, the sample size is small. Second, the desktop application does not support some common UX features for 6D pose annotation, which may have influenced the results favoring the VR application. On the other hand, the objects annotated in the desktop application were easier to identify since they had texture. Lastly, some participants did not finish the experiment in the allocated time using the desktop application, therefore not annotating all the objects. To summarize, the results strongly suggest the superiority of the VR application over the desktop application and are in line with the study conducted by Franzluebbers et al. Finally, all hypotheses were confirmed, and we may conclude that the VR application is a viable alternative to the desktop application and can be used for 6D pose annotation.

Conclusion

This thesis presented the design and evaluation of a web-based VR application for 6D pose annotation of RGB-D point clouds. We have successfully built an application that leverages the immersive and interactive nature of VR, to improve the speed, accuracy, and comfort of annotating 6D poses. The final user study was designed to evaluate our application in these aspects. We hypothesized that the VR application would outperform a conventional desktop application. The results confirmed these hypotheses. While some usability issues were noted, including difficulties in object identification and navigation, these can be addressed in future research. The overwhelmingly positive feedback indicates that the benefits of VR may outweigh the initial learning curve, novelty factor, and other concerns typical for any new technology or interface.

We also implemented a human-in-the-loop version of the ICP algorithm. This approach also shows promising results but requires further research to address the algorithm's robustness and scalability issues.

It is also important to acknowledge the limitations of this thesis. A larger sample size and feedback from more expert participants would undoubtedly provide a more robust and nuanced understanding of the VR application's efficacy. Additionally, the solutions for the VR platform require users to have access to a VR headset and, in some cases, a VR-ready computer, which is far from ubiquitous. Moreover, the benefits presented in this thesis may be less pronounced in comparison with mature desktop solutions.

In conclusion, this thesis has made a contribution to the field of 6D pose annotation by showing that employing VR satisfactorily overcomes hurdles caused by 3D-2D projection in the annotation task, and it is a viable alternative to traditional desktop-based methods. We hope that this work will inspire continued exploration into VR applications and promote further adoption of VR across various fields of research and industry.

Bibliography

Sources

[1] Franz Aurenhammer. "Voronoi diagrams—a survey of a fundamental geometric data structure". In: *ACM Computing Surveys (CSUR)* 23.3 (1991), pp. 345–405.

[2] Aryabrata Basu. "A brief chronology of Virtual Reality". In: *arXiv preprint arXiv:1911.09605* (2019).

[3] D Baur, C Pfeifle, and CE Heyde. "Cervical spine injury after virtual reality gaming: a case report". In: *Journal of Medical Case Reports* 15.1 (2021), pp. 1–4.

[4] Norbert Beckmann et al. "The R*-tree: An efficient and robust access method for points and rectangles". In: *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. 1990, pp. 322–331.

[5] M. Berger, M. Cole, and S. Levy. *Geometry I*. Universitext. Springer Berlin Heidelberg, 2009. ISBN: 9783540116585. URL: `https://books.google.sk/books?id=5W6cnfQegYcC`.

[6] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606.

[7] Gérard Blais and Martin D. Levine. "Registering multiview range data to create 3D computer objects". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 820–824.

[8] Thomas Blanc et al. "Genuage: visualize and analyze multidimensional single-molecule point cloud data in virtual reality". In: *Nature Methods* 17.11 (2020), pp. 1100–1102.

[9] Doug A Bowman, Ryan P McMahan, and Eric D Ragan. "Questioning naturalism in 3D user interfaces". In: *Communications of the ACM* 55.9 (2012), pp. 78–88.

[10] Evren Bozgeyikli et al. "Point & teleport locomotion technique for virtual reality". In: *Proceedings of the 2016 annual symposium on computer-human interaction in play*. 2016, pp. 205–216.

[11] Yang Chen and Gérard Medioni. "Object modelling by registration of multiple range images". In: *Image and vision computing* 10.3 (1992), pp. 145–155.

[12] Open Geospatial Consortium et al. *Discrete Global Grid Systems Abstract Specification*. 2017.

[13] Christopher M De Vries and Shlomo Geva. "K-tree: large scale document clustering". In: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 2009, pp. 718–719.

[14] Mohamad Dolatshah, Ali Hadian, and Behrouz Minaei-Bidgoli. "Ball*-tree: Efficient spatial indexing for constrained nearest-neighbor search in metric spaces". In: *arXiv preprint arXiv:1511.00628* (2015).

[15] Chitra Dorai et al. "Registration and integration of multiple object views for 3D model construction". In: *IEEE transactions on pattern analysis and machine intelligence* 20.1 (1998), pp. 83–89.

[16] Bertram H Drost and Slobodan Ilic. "Almost constant-time 3D nearest-neighbor lookup using implicit octrees". In: *Machine Vision and Applications* 29.2 (2018), pp. 299–311.

[17] Anton Franzluebbers et al. "Virtual reality point cloud annotation". In: *Proceedings of the 2022 ACM Symposium on Spatial User Interaction*. 2022, pp. 1–11.

[18] Lukáš Gajdošech et al. "Towards Deep Learning-based 6D Bin Pose Estimation in 3D Scans". In: *arXiv preprint arXiv:2112.09598* (2021).

[19] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal on Robotics and Automation* 4.2 (1988), pp. 193–203.

[20] GPUWeb. *GPUWeb*. `https://gpuweb.github.io/gpuweb/`. Accessed: May 15, 2023.

[21] Kenny Gruchalla. "Immersive well-path editing: investigating the added value of immersion". In: *IEEE Virtual Reality 2004*. IEEE. 2004, pp. 157–164.

[22] Paul Guerrero et al. "Pcpnet learning local shape properties from raw point clouds". In: *Computer graphics forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 75–85.

[23] Hugo Hadwiger. "Minkowskische addition und subtraktion beliebiger punktmengen und die theoreme von Erhard Schmidt". In: *Mathematische Zeitschrift* 53.3 (1950), pp. 210–218.

[24] Kiana Hajebi et al. "Fast approximate nearest-neighbor search with k-nearest neighbor graph". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[25] Stefan Hinterstoisser et al. "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes". In: *Computer Vision–ACCV 2012: 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5-9, 2012, Revised Selected Papers, Part I 11*. Springer. 2013, pp. 548–562.

[26] Hodan. *T-LESS Documentation*. `https://github.com/thodan/t-less_toolkit/blob/master/doc/t-less_doc.md`. Accessed: May 18, 2023.

[27] Tomáš Hodan et al. "T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 880–888.

[28] Wolfgang Kabsch. “A solution for the best rotation to relate two sets of vectors”. In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32.5 (1976), pp. 922–923.

[29] Wadim Kehl et al. “Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1521–1529.

[30] Scott Krig. “Ground Truth Data, Content, Metrics, and Analysis”. In: *Computer Vision Metrics: Survey, Taxonomy, and Analysis*. Berkeley, CA: Apress, 2014, pp. 283–311. ISBN: 978-1-4302-5930-5. DOI: 10.1007/978-1-4302-5930-5_7. URL: https://doi.org/10.1007/978-1-4302-5930-5_7.

[31] Jim Lawrence, Javier Bernal, and Christoph Witzgall. “A purely algebraic justification of the Kabsch-Umeyama algorithm”. In: *Journal of research of the National Institute of Standards and Technology* 124 (2019), p. 1.

[32] Hugo Ledoux. “Computing the 3d voronoi diagram robustly: An easy explanation”. In: *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE. 2007, pp. 117–129.

[33] Juyoung Lee, Sang Chul Ahn, and Jae-In Hwang. “A walking-in-place method for virtual reality using position and orientation tracking”. In: *Sensors* 18.9 (2018), p. 2832.

[34] Christopher Lewis and Frederick C Harris Jr. “An Overview of Virtual Reality”. In: *Proceedings of 31st International Conference*. Vol. 88. 2022, pp. 71–81.

[35] Yu A Malkov and Dmitry A Yashunin. “Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[36] Ryan P McMahan et al. “Separating the effects of level of immersion and 3D interaction techniques”. In: *Proceedings of the ACM symposium on Virtual reality software and technology*. 2006, pp. 108–111.

[37] mdn. *Vue.js*. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed: May 19, 2023.

[38] mdn. *Vue.js*. https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API. Accessed: May 19, 2023.

[39] mdn. *Vue.js*. https://github.com/Bear-kai/pose_annotator. Accessed: May 30, 2023.

[40] C.W. Misner et al. *Gravitation*. Princeton University Press, 2017. ISBN: 9780691177793. URL: https://books.google.sk/books?id=SyQzDwAAQBAJ.

[41] Negar Nejatishahidin and Pooya Fayyazsanavi. “Review on 6D Object Pose Estimation with the focus on Indoor Scene Understanding”. In: *arXiv preprint arXiv:2212.01920* (2022).

[42] Matthias Nürnberger et al. “Mismatch of visual-vestibular information in virtual reality: is motion sickness part of the brains attempt to reduce the prediction error?” In: *Frontiers in Human Neuroscience* 15 (2021), p. 757735.

[43] open-source. *Three.js*. `https://threejs.org/`. Accessed: May 19, 2023.

[44] G Dias Pais et al. “3dregnet: A deep neural network for 3d point registration”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 7193–7203.

[45] Krzysztof Pietroszek. “Virtual Hand Metaphor in Virtual Reality”. In: Mar. 2019. ISBN: 978-3-319-08234-9. DOI: 10.1007/978-3-319-08234-9_178-1.

[46] Krzysztof Pietroszek. “Virtual Pointing Metaphor in Virtual Reality”. In: Oct. 2019. ISBN: 978-3-319-08234-9. DOI: 10.1007/978-3-319-08234-9_179-1.

[47] Kari Pulli. “Multiview registration for large data sets”. In: *Second international conference on 3-d digital imaging and modeling (cat. no. pr00062)*. IEEE. 1999, pp. 160–168.

[48] Long Quan and Zhongdan Lan. “Linear n-point camera pose determination”. In: *IEEE Transactions on pattern analysis and machine intelligence* 21.8 (1999), pp. 774–780.

[49] Ananth Ranganathan. “The levenberg-marquardt algorithm”. In: *Tutoral on LM algorithm* 11.1 (2004), pp. 101–110.

[50] Sharif Razzaque et al. “Redirected walking in place”. In: *EGVE*. Vol. 2. Citeseer. 2002, pp. 123–130.

[51] Szymon Rusinkiewicz and Marc Levoy. “Efficient variants of the ICP algorithm”. In: *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE. 2001, pp. 145–152.

[52] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 3212–3217.

[53] Markus Schütz et al. “Potree: Rendering large point clouds in web browsers”. In: *Technische Universität Wien, Wiedeń* (2016).

[54] Malcolm Slaney and Michael Casey. “Locality-sensitive hashing for finding nearest neighbors [lecture notes]”. In: *IEEE Signal processing magazine* 25.2 (2008), pp. 128–131.

[55] Mel Slater. “A note on presence terminology”. In: *Presence connect* 3.3 (2003), pp. 1–5.

[56] Richard HY So, WT Lo, and Andy TK Ho. “Effects of navigation speed on motion sickness caused by an immersive virtual environment”. In: *Human factors* 43.3 (2001), pp. 452–461.

[57] Ivan E Sutherland. “A head-mounted three dimensional display”. In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 1968, pp. 757–764.

[58] Shinji Umeyama. “Least-squares estimation of transformation parameters between two point patterns”. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 13.04 (1991), pp. 376–380.

[59] Steven J Vaughan-Nichols. “Game-console makers battle over motion-sensitive controllers”. In: *Computer* 42.08 (2009), pp. 13–15.

[60] Grace Wahba. “A least squares estimate of satellite attitude”. In: *SIAM review* 7.3 (1965), pp. 409–409.

[61] John Wang and Edwin Olson. “AprilTag 2: Efficient and robust fiducial detection”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4193–4198.

[62] Florian Wirth et al. “PointAtMe: Efficient 3D Point Cloud Labeling in Virtual Reality”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 1693–1698. DOI: 10.1109/IVS.2019.8814115.

[63] Yu-Shiang Wong, Hung-Kuo Chu, and Niloy J Mitra. “Smartannotator an interactive tool for annotating indoor rgbd images”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library. 2015, pp. 447–457.

[64] Kazunori Yamaguchi et al. “Octree-related data structures and algorithms”. In: *IEEE Computer Graphics and Applications* 4.01 (1984), pp. 53–59.

[65] Cheng Chun You et al. “A survey on surface reconstruction techniques for structured and unstructured data”. In: *2020 IEEE Conference on Open Systems (ICOS)*. IEEE. 2020, pp. 37–42.

[66] Evan You. *Vue.js*. https://vuejs.org/. Accessed: May 18, 2023.

Appendix A: User study questionnaire

5.2 Questionnaire

1. **In which application do you find it easier to work?**
2. **Which application is your preferred one to work in?**
3. **In which application do you think you completed your work faster?**
4. **In which application do you find your work more accurate?**
5. **In which application was achieving accuracy easier for you?**