Reconstructing Office Environments in VR from Point Cloud Data Sets

Bachelor Thesis
1st December 2020 - 1st June 2021

by Moritz Jenny, 15-931-306

Supervisors:
Prof. Dr. Renato Pajarola
Prof. Dr. Thomas Fritz
Lizeth J. Fuentes Perez

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

Cover Image:

Visualized segment of point cloud data.

Excerpt from the Stanford S3DIS dataset [Arm+16].

Zusammenfassung

Die virtuelle Realität ermöglicht ein weitaus immersiveres Erlebnis als eine Visualisierung auf einem 2D-Bildschirm. Darüber hinaus kann die Darstellung von Daten, die über die drei räumlichen Dimensionen verteilt wurden, auf eine sehr intuitive Art und Weise wahrgenommen werden. Ziel dieser Arbeit ist es, die Darstellung von segmentierten Punktwolken zu veranschaulichen sowie Möglichkeiten zur Interaktion in VR zu präsentieren. Zudem soll der Prozess, welcher die Rohdaten eines 3D Scans in eine VR Applikation integriert, automatisiert werden. Das immersive Erlebnis wird insbesondere dadurch verstärkt, indem der Benutzer sich physisch im entsprechenden Raum befindet, von dem aus dessen gescannte Daten betrachtet werden. Das Endergebnis wird erreicht, indem ein neuronales Netzwerk mit einem Datensatz trainiert wird, der Scans von Büroräumlichkeiten beinhaltet. Nachdem weitere Daten eines Punktwolken-Scans aus einem Büro vorverarbeitet wurden, können sie mit Hilfe des trainierten Netzwerks segmentiert werden. Schließlich werden die segmentierten Daten in einer Game-Engine gerendert, damit diese in der virtuellen Realität betrachtet werden können und mit ihnen interagiert werden kann. Somit kann die Segmentierung visualisiert werden und dient als Grundlage für interaktive Elemente, die eine inspirierende oder spielerische Erfahrung schaffen.

Abstract

Virtual reality enables a far more immersive experience than a visualization on a 2D screen. Moreover, the representation of data distributed over the three spatial dimensions can be perceived in a very intuitive way. The goal of this work is to illustrate the representation of segmented point clouds as well as to present possibilities for interactions with point clouds in VR. Furthermore, the process which integrates the raw data of a 3D scan into a VR application shall be automated. In particular, the immersive experience is enhanced by having the user physically located in the corresponding space from which their scanned data is viewed. The final result is achieved by training a neural network with a dataset that includes scans of office spaces. After preprocessing the data from a point cloud scan of an office, it can be segmented using the trained network. Finally, the segmented data is rendered in a game engine so that it can be viewed and interacted with in virtual reality. Thus, the segmentation can be visualized and serves as the basis for interactive elements that creates an inspiring or game-like experience.

Contents

List of Figures

1 Introduction

In the last decades a subfield of visualization methods in computer graphics called virtual reality (VR) has emerged and gained popularity. On the one hand, VR is characterized by the immersive stereoscopic visualization of virtual environments and on the other hand by the multitude of interaction possibilities [TH20].

VR technology is a specific form of visualization that focuses on immersion and influencing the perceived environment. Bryson [Bry96] explains the use with “We want to create the effect of interacting with things, not with pictures of things”.

One way to represent the real world in VR is to visualize scanned point cloud data. A 3D scanner scans an object or an environment and generates data that represents the space in points with corresponding positions. The ability to then interact with the scanned data in VR while physically being in the same space opens up new interesting ways to perceive a space or an environment. Especially in these times of a pandemic, when people spend a lot of time at home, such excursions into other worlds are welcome. The project described here is intended to fulfill this notion.

The immersive aspect of virtual reality can be used to create simulations, represent art or architecture, or engage users in games. Finally, there are also scientific visualizations, which can be realized with the help of VR. This work does not target exactly one category, but rather points to some aspects of the scientific category and other aspects of the entertaining segment.

One idea that inspired the project was the concept of an enhanced desktop environment, which offers the viewer the possibility to step into a customized workplace and operate from there. Another inspiring idea was the notion of a work oasis that resembles some of the real physical properties of an environment the user is located in. A work oasis describes a creatively inspired place, in which workers could take a break and go after some easy tasks in order to be able to focus later on. A project that incorporated this idea was Jan Gugler’s master thesis [Gug21], that translated a living room into VR, where several interactions could be performed.

Many of the existing approaches require a large amount of manual labor to create the virtual environment. Therefore the ultimate goal of this project is to start off with a LIDAR scan and end up with a VR application, where the manual steps during the implementation of the VR application have been minimized. The final application includes interactive elements that allow the user to visualize segments of the point cloud, interact with the environment in a playful manner and to activate different themes.

In order to develop this project, I am looking at the following two research questions:

- **RQ1**: What are the challenges to realistically recreate a physical environment from point cloud data in a virtual setting with interactive elements?
- **RQ2**: To what extent and how can the implementation of a VR application be automated?

The structure of the milestones that led to the final product of this work is listed below.

- Plan out the project and get familiarized with the existing work. From there, the possible options of the end product are analyzed and elaborated.
- Set up environments in order to use Object Detection [Mat+14] and PGCNet [Sun+20] for the segmentation and also get familiar with the implementation of the polygonial surface reconstruction.
- Create a Unity scene and integrate the VR HMD HTC Vive Pro in the scene.
- Train and test the PGCNet with data to calculate the segmentation and then prepare the processed data for Unity.
- Elaborate the complete pipeline to integrate the raw data into the Unity Scene after the segmentation.
- Implement interactive elements in Unity such as controls to change the appearance of the data.

1.1 Outline

Based on the mentioned aspects and the given requirements, the structure of this document is built as follows. Chapter 2 is dedicated to the related work concerning the data processing part and also the virtual reality relevant aspect of the work. The end product is categorized and justified based on existing solutions in this chapter as well. In chapter 3 the components of the utilized methods of other work are discussed. It also describes the technical solution which allowed to achieve the end result, including data processing procedures for the realization of the pipeline of this work. In section 3.4, the VR and data integration as well as the design of the Unity application is described. Chapter 4 shows the achieved results and illustrates the challenges in context with the research questions stated in the introduction. The results are then further discussed regarding their limitations and concluded in chapter 5 including possible future approaches. Finally chapter 6 concludes this thesis with acknowledgements.

2 Related Work

Figure 2.1: The PointNet segmentation process described by Qi et al. [Qi+17a]. Left: Object classification. Middle: Part segmentation. Right: Semantic segmentation.

This chapter deals with related work in the context of this thesis and shows the current state of knowledge regarding the segmentation of point cloud data and the visualization part. The first section 2.1 shows a few examples of how virtual worlds can affect us and what methods have been used to produce positive effects. The section about the point cloud segmentation 2.2 gives a brief overview of different state of the art segmentation methods. In section 2.3, technical work of point cloud visualizations is cited, but also work that looks into the effects on the well-being and the performance of VR application users in general. The methods that are specifically used in this work are then transitionally explained within the next chapter.

2.1 Aiming for an Enhanced Experience

A very interesting aspect of virtual reality is the strong immersion and thus the associated power over the affected perceptions. This makes it possible to create a work oasis or to distract oneself briefly with playful activities. This is well demonstrated in Jan Gugler's [Gug21] work, in which the application "Holoft" was developed. With Holoft, virtual work oases can be created which can be put together by the user. Studies conducted with Holoft have shown that users enjoyed engaging with the VR application during breaks and after work and that similar effects as seen in traditional work oases could be evoked.

This positive effect on the well-being of a person has been generally used for relaxation or in office environments. Ruvimova et al. [Ruv+20] studied the effect of VR environments in different office situations on people's performance. They tried to use VR to prevent the distractions that can happen in an open office situation and help people to stay in flow. A study was conducted with subjects in which they compared open and closed office situations with VR situations. The virtual settings consisted of a scene on the beach or a virtual office. The results showed that while subjects in a closed office preferred to work without VR, in an open office they favored the virtual environment

Figure 2.2: A beach scene of the work of Ruvimova et al. [Ruv+20] that shows a working environment embedded in an exotic surrounding.

in all respects examined. The implementation of the beach scene can be seen in Figure 2.2. The notion of transforming an environment into a themed virtual environment, in which some elements remain familiar had an inspiring effect for this project. This ultimately also led, together with other examples, to the decision to implement different themes in this work.

Berki [Ber19] examined the advantages of a desktop environment that is supported with VR. A number of test participants were provided with an ordinary desktop environment and another group was provided with a VR desktop setup with a permanent display of additional information in order to solve tasks. As the experiments showed, the test candidates with the VR headset and the additional information were able to achieve better performance than the users with the conventional setup. That a demand for such types of applications exists is evident in the success of the commercially available “Virtual Desktop” software. Virtual Desktop allows users to create different environments around their own desktop. There are themes that still display the screen on a monitor, localized in a tidy room. However, there are also themes that project the desktop without borders in an outdoor location such as grassland or a forest for example.

But there are also applications that are not meant to only be used in an office to enhance work performance, but rather serve simply to relax. Soyka et al. [Soy+16] investigated a VR application designed to alleviate stress and improve stress management. They combined known breathing techniques with an underwater scene and studied the effect of the environment on its own and in combination with the known breathing techniques on the stress behaviour of the participants. For example, jellyfish helped to guide the breaths with rhythmic movements. Soyka et al. [Soy+16] showed in interviews that subjects found the VR-assisted methods more enjoyable and that they were more likely to use them when doing the exercise at home.

In conclusion, these methods all try to bring the user into a new world in which his environment is generously replaced or significantly changed. A significant reason seems to be the change or the reorientation in a clearly structured environment that motivates for users to move into a VR landscape. Especially in times like these, when a pandemic brings drastic restrictions to everyday life, a switch into a completely new world might be very welcome. This work was inspired by the aforementioned creations of different atmospheres and I tried to follow the feeling of diving into another world with sceneries like the underwater world.

2.2 Point Cloud (Semantic) Segmentation

There is currently a great increase in the possibilities to scan objects from the real world in three dimensions. It is possible to capture objects with laser-based, structured light, contact-based scanners or even with an image only based method called photogrammetry. Therefore, it is possible to scan objects with a smart phone camera. Smart phones nowadays are also capable of displaying them as well as share them, which is shown with the attempted creation of the social network named “Display.Land” for example. Display.Land lets users scan and share 3D objects or environments. However, raw point cloud data is unstructured, making it difficult to implement interactive elements due to the lack of specific object references.

Interactions with point clouds greatly enrich the application potential for this type of data. A key aspect of preparing this data for use, whether for interaction or for analysis, is the segmentation of the data. This means that a point cloud scan can be divided into areas according to certain characteristics, for example to isolate objects. In computer graphics the general problem to segment meshes and point clouds in a logical way has been thoroughly researched. The segmentation of a mesh resembles very similar methods like the segmentation for a point cloud and can therefore be considered as well.

Nguyen and Le [NL13] describe how the general approach to segment meshes is to build a graph from the input mesh, and cluster the graph to produce a segmentation. They point out Shamirs [Sha06] work which lists several methods that have been proposed to approach the problem of partitioning techniques used on boundary meshes: convex decomposition, watershed analysis, hierarchical clustering, region growing, and spectral clustering. Nguyen and Le mention that many of those methods have also been used to segment point cloud data. The region growing approach has been the most widely used method. However, certain methods have particular limitations with respect to different types of point clouds. As Zhang et al. [Zha+19] states, the traditional methods for point cloud segmentation are limited by the prior assumed knowledge and the parameter adjustment is very difficult.

With the rise of deep learning in the field of semantic segmentation of data in general, the results of meaningful segmentations have been strikingly improved. The following part describes prominent types of point cloud datasets and point cloud segmentation methods with regard to the work at hand.

2.2.1 Point Cloud Datasets

Point cloud datasets typically consist of a certain set of points in space which represent an unorganized ”cloud” like structure. Each point has at least one set of x, y, and z coordinates and can optionally have other properties such as color, orientation, or label. However, many other features can also additionally describe a point. The raw data of a 3D scan are point clouds and have different structured characteristics depending on the acquisition method.

Therefore, scans taken by a fixed 360 degree laser scanner may show “star-like” structures, since only the parts visible to the scanner could be imaged. However, a point cloud generated with photogrammetry and pictured from different sides can be more detailed in angular parts of an object due to the many view directions and thus is a topologically more accurate representation of the original object. It is common for scanned point clouds to be disorganized, chaotic and incomplete. This means there are usually regions where the data is missing, which is also called undersampling or occlusion artifacts. This problem can be solved in with different methods [Qui+15] [WO07] [Jun05], but is computationally expensive and is not considered for the scope of this work.

For this work, indoor scans of a laser scanner are used, which also contain color information. Since the scans are only indoor scans in simple rooms and the scan is taken approximately from where the viewer is located, the incomplete parts are rather poorly visible in the representation of the scan.

The Polygon File Format (.ply) used for this project was developed at Stanford University and its flexibility makes it a suitable 3D scanner format. The file can be saved in binary or ASCII format, which is quite advantageous to read and debug the files with a text editor, but also to use them efficiently in the binary format, which can be read in significantly faster.

There are some public datasets that have different properties, such as indoor and outdoor scans with different labels. Depending on the purpose of a project, the characteristics of a dataset play an essential role. Some datasets might be suitable for the training of self driving cars and others may be used for architectural studies. Since this project makes use of a semantic segmentation of an indoor scene, some indoor datasats would have been eligible. The Cornell RGBD dataset [Ana+11] has 52 indoor scenes with 27 labeled object classes. Also the ScanNet dataset [Dai+17] consisting of a large amount of 1513 reconstructed indoor scenes was considered. For this project, which is ultimately consisting of indoor office spaces, the Stanford S3DIS dataset was used, which contains 272 room scans and has 13 labeled object classes. It’s focus is mainly on office spaces with similar character.

It is common for these datasets to contain several million points and therefore they can take up a large amount of storage space.

2.2.2 Traditional Methods

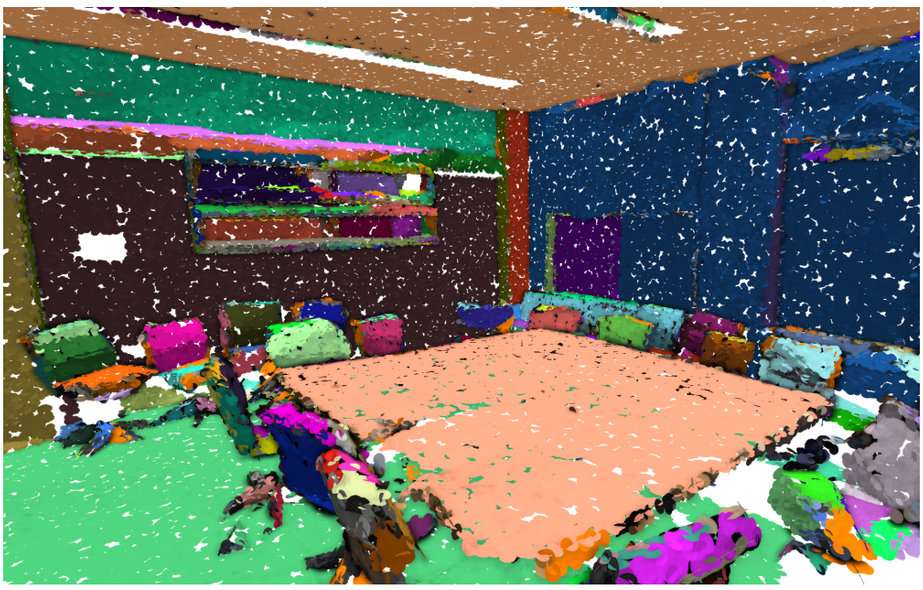

Figure 2.3: The method presented by Mattausch et al. [Mat+14]. The extracted planar patches with the seeded region growing method are colored in different color tones to visualize the segmentation of the patches throughout the whole scene.

Many traditional methods that derived from image segmentation and mesh segmentation are used for point cloud segmentation with several variations. Prominent traditional methods like edge based, region based and attribute based segmentations are described as follows. Edge based segmentation methods are used for images as well as for point cloud data. The basic idea is that edges are assumed for nearby points that show large differences in defined features. This then serves as the basis for the segmentation. However, these approaches are very prone to errors with uneven and noisy point clouds [NL13].

Region based methods inspect information from adjacent points. As Nguyen and Le [NL13] describe how these methods are more accurate than edge based approaches, but the definition of the number of segments or the determinations of region boarders can cause problems. They distinguish between seeded-region and unseeded-region methods. In seeded-region methods, a certain set of seed points is selected and then neighboring points are connected based on specific features in an iterative manner. The connected points then represent a segment. Tóvári and Pfeifer [TP05] describe the possibility to connect the neighboring points based on normals and to generate fitting planes. Seed points can be chosen randomly or according to certain criteria. However, they mention that the choice of points can determine the quality of the segmentation and that this may be a laborious task in order to achieve good results. The methods that are used to generate the patches in the preprocessing of this work are using region growing as well. They are a part of the implementation of the work of Mattausch et al. [Mat+14] and the results are presented in the Figure 2.3. The process of the patch extraction is addressed in detail in subsection 3.2.2.

The unseeded-region methods work without selected points and start with a large segment containing all points. The segment is then subdivided until the criteria for further division are no longer present. Chen [CC08] describes this method on an architectural object where different planar regions are segmented and then reconstructed. The difficulty with this method is to define the criteria for a subdivision and to choose the correct weighting so that no over or under segmentation takes place.

Attribute based methods are founded on the similarity of computed attributes on points. Basically, two steps are necessary. Filin and Pfeifer [FP06] describe how certain attributes are calculated in the first step, such as normal vectors of the points. After that, the segmentation based on certain similarity criteria can take place. Consequently, a disadvantage of this method is the high computational intensity with a large number of points.

Despite all the possibilities to segment data, the method used for this project [Mat+14] was suitable in particular, because it is optimized for indoor scenes. The environments, that contain man-made objects tend to have many flat surfaces. This helps to find planar patches with the region growing method.

2.2.3 Deep Learning Methods

Semantic point cloud segmentation was originally achieved via adapted general machine learning methods and has been advanced with specific deep learning for a few years now as shown in the review by Zhang et al. [Zha+19]. It points out that deep learning methods can be divided into two different categories. One category is called indirect and the other category is called direct point cloud segmentation methods.

Indirect methods are based on a simplification or a transformation of the point cloud raw data to then apply an appropriate deep learning method to the processed data and to draw conclusions about the composition of the original raw data from the resulting information of the corresponding networks. Most of the work on indirect methods describe using voxels [MS15] or multi view images [Su+15] as the input for the network.

The work of Su et al. [Su+15] demonstrates how the multi view method can be used to assign points to a class. The idea is that several 2D images of the scanned object from different perspectives are given into a convolution network. The classifications are then made based on the images and then used for the 3D data. It is mentioned that this method is only useful for single objects and is not suitable for large scenes, such as a complete indoor scan of a room would be.

The direct methods describe the process where the point cloud data is given directly to the network and thus the segmentation is done. The foundation for this approach was laid with the PointNet proposed by Qi et al. [Qi+17a], which takes unordered point cloud data as direct input. Figure 2.1 illustrates intermediate and end results of PointNet. While PointNet manages to obtain global features of the data, which has a beneficial effect on the classifications, it fails to process local information or to process relationships between neighboring points. This can lead to information loss when dealing with large amounts of data.

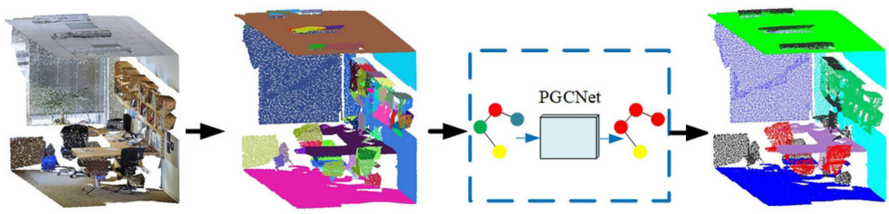

Figure 2.4: The pipline that allows PGCNet [Sun+20] to segment point clouds. From left to right: The raw input data, patch segmentation, PGCNet training and testing, the segmented output.

There are several subcategories of methods based on PointNet that attempt to incorporate local features of points and their neighbors. For example, PointNet++ [Qi+17b] includes several local features, but fails to consider relationships between sampling points. Promising results have been obtained with graph convolutional neural network (GCNN), which can give a graph with dependency information into a network. Wang et al. [Wan+19] show how GCNN in combination with PointNet manages to incorporate local neighborhood information. A disadvantage of this method is, however, that the scalability is not very good due to the many relationships between points involved. Thus, many methods are limited to more local point clouds due to their computational intensity.

This problem is addressed by PGCNet [Sun+20] in order not to have any losses in this respect for large indoor scans as it is the case for example with the S3DIS dataset. Generally, PGCNet treats pre-processed patches that combine points in a plane as input graph nodes. This process is visualized on Figure 2.4. PGCNet can be trained efficiently and perform good semantic segmentations. Since PGCNet is used in this thesis, it is explained in more detail in section 3.3.

2.2.4 Data Driven Understanding

An interesting extension of segmentation is shown by Nan, Xie and Sharf [NXS12], who present an algorithm for segmentation and subsequent reconstruction. They try to solve the problem of incomplete and noisy scans by reconstructing the scene with a template fitting step. The segmentation is based on iterative region growing and increasing classification of likelihood. The templates represent previously made meshes that correspond to the classifications appearing the scene. Therefore, a scene with whole objects drawn as meshes is produced. The result of their method is seen in Figure 2.5 This method yields interesting results, but does not yet take into account the contextual information of the surrounding objects and could thus be improved in the future.

Figure 2.5: The method presented by Nan, Xie and Sharf [NXS12]. Left: The raw scan of an indoor scene, Middle: The segmentation of the scene into meaningful objects, Right: The reconstruction achieved with templates.

Shi et al.[Shi+16] present another method for semantic classification of objects by combining several methods and dividing the segmentation into two phases. In the first phase, the objects are segmented using traditional methods. Primitive fitting is used to divide the objects. A second phase then allows the semantic segmentation of the objects through the classification by a network trained on a database containing manually built and labeled meshes.

These approaches show how diverse the segmentation methods can actually be and what can be done with segmentation information. The method presented by Nan, Xie and Sharf [NXS12] is interesting in particular for this project, because it would solve many occurring problems with point cloud data. The complete replacement of point cloud data with objects would solve problems by closing holes in scans, provide a mesh based basis for the physics engine in Unity and decrease the size of the data. I tried to incorporate the notion of replacing point clouds with objects in the final application as an experimental feature. The corresponding method is described in chapter 3.

2.3 Point Cloud Visualization in Virtual Reality

Due to the large increase of 3D scans and the option to display point cloud data, i.e. 3D scans, the opportunities to integrate these into a wide variety of applications have increased. This increase was made possible by the improved computing power of various devices in the mobile and desktop area in recent years. It has driven the development of systems that support the management and visualization of large datasets of this type.

Considering some of the challenges associated with these types of data, such as the relatively high demands of graphical processing power as well as the required storage capacities, developers are challenged to create optimized applications on which the data can be managed and visualized. Thus, possibilities were investigated to implement web applications [DAD19], server systems [CPP17] or database structures [EVH20], which are specialized to handle point cloud data.

One way to experience 3D scans in an intuitive way is by viewing the point clouds in virtual reality. The implementation of an application that can be used with a VR HMD in conjunction with a point cloud rendering capability must therefore be reconciled. Figure 2.6 shows a human model inside a point cloud environment [Ado21][mar17].

Figure 2.6: A representation of how a user might feel being inside a point cloud environment.

There is of course the possibility to implement such applications from scratch by incorporating the head tracking of the HMD with a rendered virtual environment [BSN14] to then experience a 3D scan visualized as a point cloud in VR. However, game engines are very suitable for such tasks, as they generously support the rendering of virtual environments and the use of virtual reality with plugins and additional available systems as Virtanen et al. show [Vir+20]. Game engines are additionally designed to create heavily interactive applications, be it in the entertainment or professional field. As a result, it is possible to create games and even working environments in VR.

The following two subsections 2.3.1 and 2.3.2 address the technical properties of different types of applications. They list exemplary realized point cloud projects in VR either utilizing game engines or other frameworks and show what purpose they serve. They also show where the difficulties and challenges are situated regarding interactability, usability and implementation.

2.3.1 Standalone Applications

Bruder, Steinecke and Nüchter [BSN14] present a project in which they scan a room with a robot and then render it using the 3DTK point cloud library. Challenges are mentioned which concern performance, since no continuous rendering was provided by 3DTK directly. This can lead to points not being represented dense enough due to their quantity and their constant size. This leads to representations that appear very sparse and leaky upon closer inspection. Additionally, it mentions how interactability cannot only be achieved using the same methods as in polygonial based environments because there are many voids between the points. This prevents, for example, interactions where objects are selected by ray intuitively.

Bonato et al. [Bon+16] show how the problem of the lightness and noisiness of the dots could be tackled on closer inspection and how an immersive experience can be recreated from a scan. Among other methods, they use so-called splats, which represent the dots as round disks with an orientation, and can thus fill the gaps between the dots and blend the colors in between. The render engine was implemented in C++ and shaders were implemented with GLSL 4.

Standalone applications give the developers a lot of freedom in representing the virtual environment. However, it also requires much more effort to include interactive elements without an underlying physics engine or an input manager, for example, as they would be provided by a game engine.

2.3.2 Use of Game Engines

Due to the large communities around game engines like Unity and Unreal Engine, there are several tools that provide support for specific problems like performance issues in connection with point cloud data. Much of the literature dealing with point cloud applications in conjunction with VR uses Unity as the underlying engine.

Thus, in Wirth et al. [Wir+19] work reports on PointAtMe, a labeling tool they developed. It is possible to annotate certain points by sophisticated use of the controllers. The use of Unity makes sense in this work, because it is a very interactive application, which itself acts as a tool. Therefore it is very dependent on the ease of use and the interaction possibilities.

PointXR toolbox proposed by Alexiou, Yang, and Ebrahami [AYE20] is a set of Unity applications that can host experiments in association with point clouds which can assess the performance of the state of the art MPEG compression. It also allows the user to configure different visual appearances of the point cloud with the included software package. This example shows how applications with very specific uses in this area can still use a game engine as a foundation and then devote themselves fully to their requirements.

The very popular Unreal Engine is also used for various projects for the implementation of virtual reality applications. The example of Hilfert and König [HK16] shows how the Unreal Engine is used for the implementation of an immersive system to support engineering and construction. They mention that Unity is more mature, but that Unreal Engine provides more detailed graphics.

Ultimately, there are many arguments in favor of both engines, but in practice for projects involving point clouds and VR, Unity tends to be used more often. Furthermore with the requirements of this work not being aimed specifically at graphical elegance the decision for Unity was reasonable. A significant reason for this choice was also my prior experience I had with the Unity game engine.

3 Approach

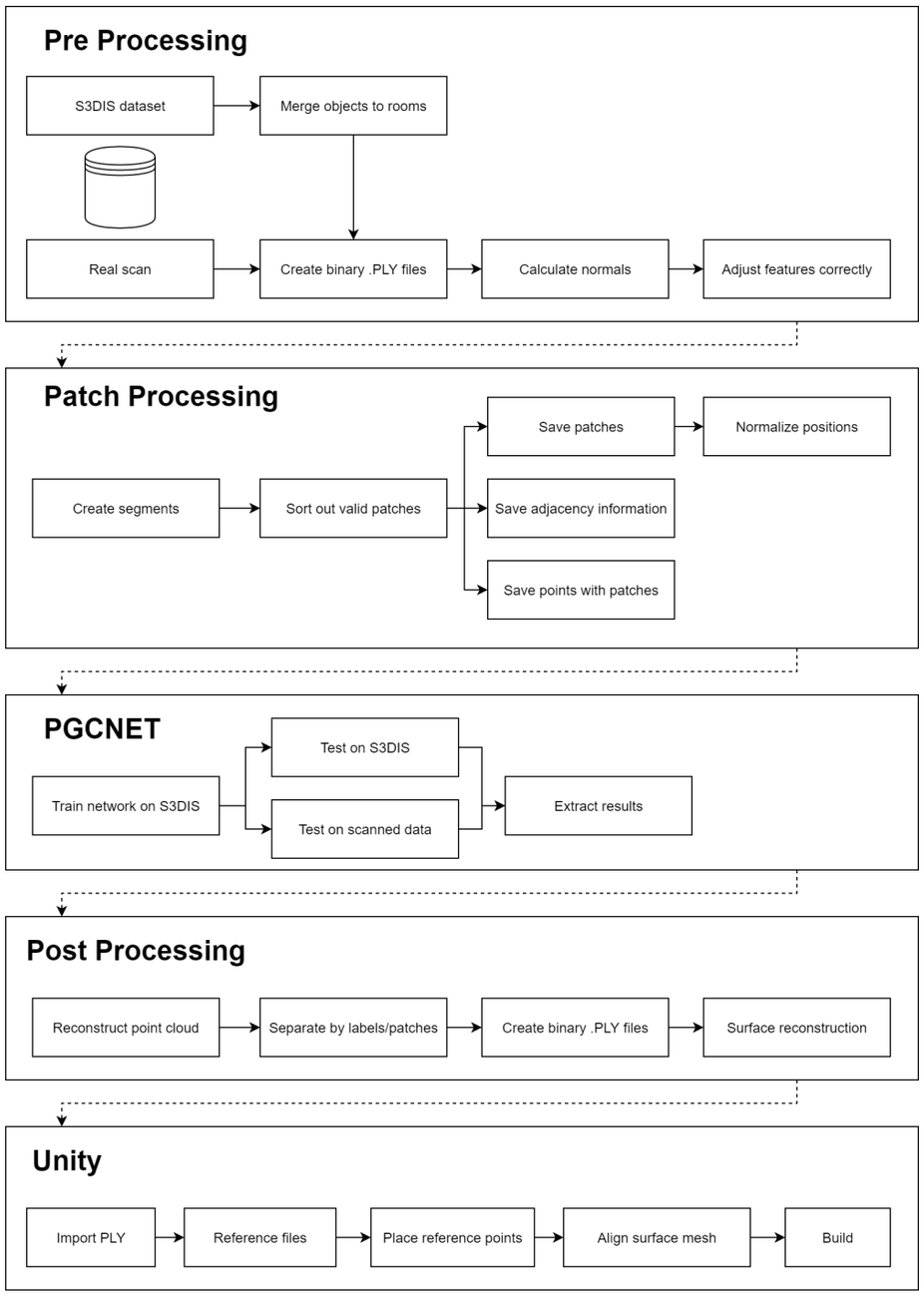

Figure 3.1: The way the data gets processed in this project. From top to bottom: The first two sections are described in section 3.2. The next two parts include the segmentation processes and are described in 3.3. The last part regarding the Unity integration is described in section 3.4.

The introduction states that the goal of this project is to implement an optimized pipeline starting from a 3D scan and ending in a VR application. The VR application itself should provide a specific set of features, which include the ability for the user to align a room, walk around freely and interact in different manners with the room. This means that the project consists of multiple parts that compose the final product.

Some decisions about what the pipeline should be consisting of were made right at the start of the project. Regarding the data processing from the raw data, I had to define methods that support the process of taking a 3D scan and putting it into a VR application. In order to make use of the unordered point cloud data, it has to be segmented semantically, so that physical bounding boxes can be generated, or that a distinction between different objects can be made. Therefore, to achieve the final result of this project, diverse processes of other work were used.

Basically, the procedures for the patch extraction and the patch adjacency detection of the data are adopted from the work of Mattausch et al. [Mat+14]. This can be seen on figure 3.1 in the second box ”Patch Processing”. The resulting patches with the adjacency information are then used in Sun et al. PGCNet [Sun+20] to train a model that can then be used to semantically segment other unknown data. This is illustrated in the third box in figure 3.1 as well. The next part of the project included the implementation of the VR application that should be easy to use and that includes interactive elements. Over the course of the project, I decided which interactive concepts should be implemented. Those requirements can be summed up to the following points:

- Alignment of a point cloud to a real physical room.
- Activating and deactivating as well as coloring different labeled elements.
- Replacing some objects with a mesh automatically.
- Placing preset objects onto specific labeled objects.
- Replacing walls, floor and ceiling with a mesh and apply a preset texture to them.
- Throwing different objects at the objects in the room.
- Transforming the room into a set of preset environments.

In the following part, these processes are illustrated. The work steps are thus explained with the sequence how the data is processed through the pipeline in high detail.

The first section 3.1 explains the choice of the hardware and software as well as the used frameworks and languages. The first part of the implementation pipeline describes preprocessing 3.2, which explains how the files were processed and how the initial patch extraction took place. The next part describes the segmentation of the data 3.3, which contains the use of the PGCNet and the subsequent postprocessing as well as the polygonial surface reconstruction. In the following part the integration of the data in Unity and the implementation of the Unity application are described 3.4. The last section 3.5 reiterates over the whole implementation of the project in chronological order and points out special events in general.

3.1 Technical Context

This project is based on the use of different components which contribute to achieve the final result. The development and rendering took place on a Windows 10 operating system with an NVIDIA RTX 2080 GPU, 16GB DDR4 RAM and a 3.7GHz i7-8700K CPU. I chose Windows as the operating system, because I am used to it and also because Unity's most recent releases are available for Windows.

The work on "Object Detection and Classification from Large-Scale Cluttered Indoor Scans" [Mat+14], which is used for the patch extraction, is written in C++ and configured in a CMake environment. The polygonial surface reconstruction component includes implementations of a CGAL package written in C++ and works with the CMake build system. The subsequent implementation of PGCNet [Sun+20] for semantic segmentation was implemented using PyTorch. Between processing operations, Python scripts are used to summarize, split and transform the data into the appropriate formats.

I used Visual Studio for the maintenance and the development of the parts written in C++. The development of the Python scripts and the edits in the PGCNet were made with PyCharm. Unity was chosen as the game engine to build the VR application.

In Unity, the OpenVR Unity XR plugin is used, which provides the necessary sdk libraries for users to build VR applications and which are also specifically optimized for the HTC Vive Pro, which is used in this project as well. The implementations in Unity are written in C# using Visual Studio as well.

The HTC Vive Pro was used, because it has the ability to use a GPU of an external system. Therefore it is able to process demanding tasks, which consist of rendering multiple hundred thousand points. The HMD was provided by the University of Zurich for the time of the development.

3.2 Preprocessing

The starting point for this work is point cloud data from a dataset or raw data from a 3D scan. The data is either already labeled in order to train the network or it comes unlabeled in order to be segmented. Since the preprocessing hardly differs for both data types, the respective processes are described in parallel. The preprocessing includes the formatting of the file types and the extraction of the patches as well as the adjacency information. The section is describing the operations and methods that are performed on the data.

3.2.1 File Processing

The Stanford S3DIS [Arm+16] dataset used is an indoorscan consisting of 695 million points. It is divided into six areas which are divided into different rooms. The rooms are divided per object into .txt files which store the data in the form xyzrgb. A distinction is made between 13 different object types, which are as follows: bookcase, board, clutter, window, door, table, chair, sofa, ceiling, floor, wall, beam, column.

Because of this division of the data, the files are processed and grouped into larger files for later calculation, which represent whole rooms and not just individual objects. Thereby a further feature is added, which represents the label of the object.

This process is done with a python script, which was provided as part of the PGC-Net [Sun+20]. The new files, of which each file represents a room, now have 7 fields and contain the positions, colors and the corresponding label of the room. For the upcoming process of extracting the patches, normal information of all points is necessary and therefore must be calculated for each point. The normal information represents an orientation of a point and should preferably be similar for points lying in a plane.

However, in order for the files to be processed efficiently, they are first converted to .ply format and stored in binary encoding. The conversion is also done with a python script and the module plyfile, which makes the handling and parsing of .ply files very user friendly.

The normals are generated from this step on with a Matlab script, that makes use of the Matlab add-on Computer Vision Toolbox. In this case, six surrounding points were taken as reference to calculate the normal direction of each vertex with a plane fitting method. The method used for the plane fitting was introduced by Hoppe et al. [Hop+92]. The method is divided into two substeps, in which the first step defines a local area to be observed and a second step in which a contouring algorithm is used to approximate a simplical surface.

As a last step before reading the data for patch processing, the .ply files are manipulated again in such a way that the field naming and structure matches the one expected by the patch processing code. Thus an alpha value is added to the color with a python script, so that the colors are stored in the format rgba.

For data that does not originate from the dataset and directly represents a roomscan, the process is the same from the generation of the normals onwards, since laser scanners do not normally perceive normal information precisely.

3.2.2 Patch Extraction

The original work presented by Mattausch et al. [Mat+14] is segmenting indoor scenes by detecting repeated objects. They describe a process by which objects can be segmented from scans, which is achieved by finding similarities between different scans of similar structure. Previous approaches [Kim+12] use reference data, from which similarities are then compared with the scans, but this is prevented in the case of the

work described here. In order to massively optimize the process, the data is simplified and points that lie in roughly the same plane are grouped as patches and from it fitting rectangles can be obtained. These rectangles are much more efficient to handle and have certain features. The patches from this first step are then further used for the approximation process according to certain features in a high-dimensional Euclidian space. This is done in order to perform the similarity finding. After that, the patches are clustered to perform object-based segmentation, but it is mainly the first step that is most relevant for this work. Namely the patch generation with the different features together with a function that calculates the adjacency information of the patches.

The patches are generated with a greedy region growing algorithm, in which the normals and the position of the points are considered. The features are then calculated from the resulting fitting rectangle. The image 3.2 illustrates a patch with it's feature descriptors in the form it was originally used. All of the features except F6 are later used to derive the patch features for the patches used in PGCNet. Directly after the patch creation process, the adjacent patches are searched per defined distance. The adjacency information is also saved in a separate file, because it is then used by the PGCNet afterwards. The input of PGCNet is a scene graph, in which the patches represent the nodes and the adjacency data is building the edges,

$\boldsymbol{F}$	Definition
$\boldsymbol{F}_1$	Area: wl
$\boldsymbol{F}_2$	Ratio of width to length: w/l
$\boldsymbol{F}_3$	Ratio of areas: CHULL$/wl$
$\boldsymbol{F}_4$	Height of centroid of rectangle
$\boldsymbol{F}_5$	PCA normal of the points in patch
$\boldsymbol{F}_6$	Non-planarity: $d/(w+l+d)$

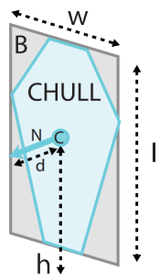

Figure 3.2: The patch features of the original method in the patch processing introduced by Mattausch et al. [Mat+14]

3.2.3 Patch processing in Practice

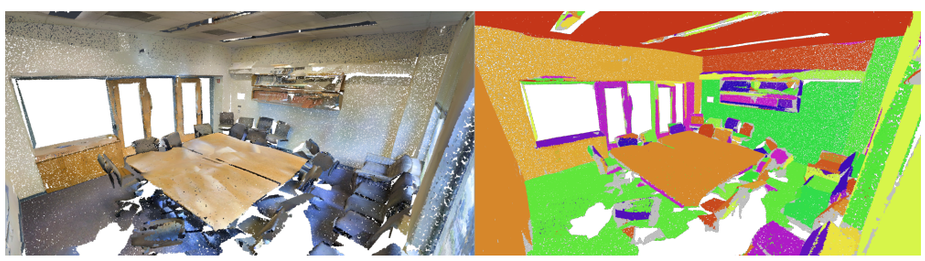

Figure 3.3: Left: The point cloud with original colors. Right: the same point cloud with randomly applied colors per patch in order to visualize the patch segmentation. This scene consists of 355 patches.

In order to utilize this method the prepared .ply files are read into the code, which is responsible for the patch processing. The patches were originally stored in .patches files and must now be stored in .txt files, so that they can be used correctly later. A total of

17 features are stored per patch, which are derived from the original patch features and point features. The 17 required features of the patches are the following: x, y, z, n1, n2, n3, height, length, ratio, area, fillratio, borderarea, width, r, g, b and label.

For example, the colors are average values of the points that make up the patches. This means that each patch has a uniform color. The code has been slightly adapted for this purpose.

Also the labels were not included in the original code from the patch processing method, so a majority voting function was implemented, which assigned the label that occurred the most in the contained points of a patch. An alternative method would have been to take the label of the seed points of the patches, however majority voting proved to provide accurate results.

Originally, only the patches that did not belong to the wall, the ceiling or the floor were used after patch processing. For this work, however, these are also important and must therefore be taken into account. For this purpose, the parameter which defines the room height, was increased to the complete room height. Thus, all patches have been used as can be seen in the Figure 3.3. Different patches are drawn in different colors so that they can be distinguished here.

The adjacency information, which is derived immediately after generating the patches, is also stored in a separate file. They are based on the evaluation of a measured distance value between patches. All adjacent patches are noted in the form of two tuples in another .txt file.

Thus all data needed for the network is available. Practice shows that a point cloud file of about 30MB can be reduced to a patch file of about 28KB. However, since the point clouds should be completely reconstructed after the segmentation process, the references of all points to their generated patches must be maintained. Therefore a third file is created, which has a patch index for each point index. The visualisation of the reconstructed patches can be seen on Figure 3.3, where all the patches are colored in a random color.

The values x, y, z as well as the length of the normal vector must be normalized for the following step, so that the spaces have a uniform structure. This is done with the help of a Python script, which reads the written patch files and sets all position data between 0 and 1 and sets the normal vectors to length 1. Therefore all files are correctly formatted and the data can be transferred to the network. In order to get a normalized value z the following formula 3.1 was used. The value x represents the value of a given dimension.

$$z = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

3.3 Segmentation

After the preprocessing step, the data can be used to train the network and also to be tested by the trained network. This testing step is then responsible for the segmentation determined by the network. Therefore, the training takes place on the labeled data and the testing can take place on unlabeled data. The following section describes how the segmentation was realized and how the method of the segmentation works.

3.3.1 PGCNet

The PGCNet, proposed by Yuliang et al. [Sun+20], embodies the core processes of the semantic segmentation of data presented in this thesis. It tries to tackle the computationally time-consuming issues of other deep learning based methods for indoor point cloud segmentation. This is basically done by treating the generated surface patches as nodes in a graph structure of which the geometric properties of the patches can be used as node features. With the help of the adjacency information the concept of the graph structure is realized by creating a Scene Patch Graph (SPG). In order to create hierarchical edge features from the created SPG, a module called dynamic graph U-Net (DGU) was utilized.

To train the network, the patches and adjacency data processed from the S3DIS [Arm+16] database were read in and trained over 200 epochs and a batch size of 8 was used. Areas two to six were used for training and area one was used as the initial testing area. The training of the models with these settings and the use of the GPU with CUDA took about an hour. The testing itself took only a fraction of that and was completed in approximately a second. The output of the testing was provided as a list of labels belonging to the tested files, determined by the network.

3.3.2 Subsequent Postprocessing

After the labeling of the patches, the point clouds need to be reconstructed. Due to the strong simplification of the data, the original points must be reassigned to the newly labeled patches. This is where the stored points-to-patches relations from the patch processing step come into play. This step is done using a Python script. Here all points of an assigned label are collected in a file and then again saved as a binary .ply file.

This means that after the postprocessing step one file per patch is generated, which then carries the label in its name. The .ply files from this step can then be used in Unity.

3.3.3 Polygonial Surface Reconstruction

Another step is the rough reconstruction of the room boundaries as a mesh. The mesh is used directly in Unity and can be used as an alternative representation of the scanned points representing the floor, walls and ceiling. For this purpose, a polygonal surface reconstruction algorithm is applied, which is provided by the CGAL library [Nan21].

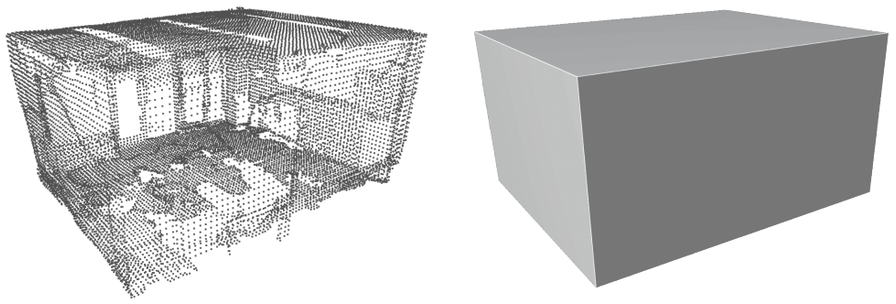

Figure 3.4: The surface reconstruction of a room after taking only the floor, walls and ceiling and downsampling to about 5% of its size.

This algorithm takes planes from the input point cloud model in a first step. In a next step candidate faces are determined, which are located between the intersections of the planar primitives of the planes. After optimizations, a subset of candidate faces is chosen that reflect the topological properties of the scan and a new mesh is formed. This basically means that from an input of floor, wall, and ceiling point clouds, a simple and watertight mesh can be generated whose topological properties are derived from the original conditions.

The semantic segmentation process takes care of extracting these components needed for the reconstruction. In order to start the process only a conversion of the necessary .ply files into a unified .pwn file is necessary, which again is done with the help of a Python script. For performance reasons, the point clouds are reduced to 5% of the number of points beforehand. This is done by clustering decimation in meshlab. Hereby details are lost, but they are completely irrelevant for the surface reconstruction as long as the decimation keeps points in an evenly distributed fashion.

As an output a mesh file is saved in .ply format and is converted to .blend format using Blender in order to be treated as a mesh file in Unity. In Blender the object is also UV unwrapped with a cubic projection and three materials are separately applied. This helps to distinguish the ceiling, wall and floor materials later in Unity.

3.4 Unity Integration

Different plugins for well-known game engines in connection with VR are currently under discussion, because different frameworks are compatible with different manufacturers and platforms. The long-term support of the various frameworks has also to be considered by the developers. For this project, OpenXR, developed by Khronos, was chosen because it has unrestricted access to different platforms and devices and because it allows Unity's new input system to be connected to it. Specifically, OpenXR was connected to the input system using the new XR Interaction Toolkit, which was still available as a preview package in the package manager at the time this work was developed. Thus, various inputs can be kept in general and changed very easily. Deployments to devices like Facebook's Occulus should also be possible. This generally gives the project scalable and modular properties.

The basic framework of the VR application consists of a Unity scene in which the viewer can move freely with an HMD. Inputs are given with the help of two controllers in their hands. The following subsections explain how the VR application is built and how the point clouds are handled.

3.4.1 Point Cloud Handling

The point cloud data is imported in the .ply format for the use in Unity. The files that are isolated by label are used. These files are then dragged into the scene and oriented correctly. Since the coordinate system of Unity uses a different vertical axis, a rotation of 90 degrees around the x axis is necessary. In the scene the files are basically not changed anymore, because this process should be as automated as possible. The editor is then used to pass the references to the individual labeled files to a defined point cloud manager class object, which acts as a singleton.

In order to render the point clouds, a custom importer and renderer from a third party is required, as Unity does not support this feature directly. For this purpose the Pcx importer and renderer was used [Tak21]. The Pcx importer and renderer allows to render the point cloud using different container types. There is the mesh container, which is rendering with a standard Unity mesh renderer and there is the ComputeBuffer container, which renders the data using a custom PointCloudRenderer component provided by the Pcx package. The ComputeBuffer container was also used in this project for rendering.

The mesh container is needed by Unity, in order for it to treat the point cloud like a mesh and that a convex mesh collider could be applied. This allows a user to interact physically with the object for example. The ComputeBuffer container stores point data and therefore the points can be contained in a PointCloudData object. With this component I had the possibility to change the color tint of the point cloud and also to change the size of the points. The points are rendered as disks and their orientation is looking towards the camera.

The mesh from the surface reconstruction 3.3.3 is taken into the scene and aligned with the point cloud manually. It embodies the walls, the floor and the ceiling as mesh. A specific reference to this mesh is also included in the point cloud manager class. In this case the traditional Unity mesh renderer components can be used with simple materials.

3.4.2 Creating the VR application

With OpenXR, Unity provides an environment where the tracking of the HMD and the controllers work pretty much out of the box. However, the hand models are not included by default. This meant that I had to implement a system to control the VR application as a user. My idea was to use hands at the place of the controllers, which can be pointed at a panel that displays the current options a user has.

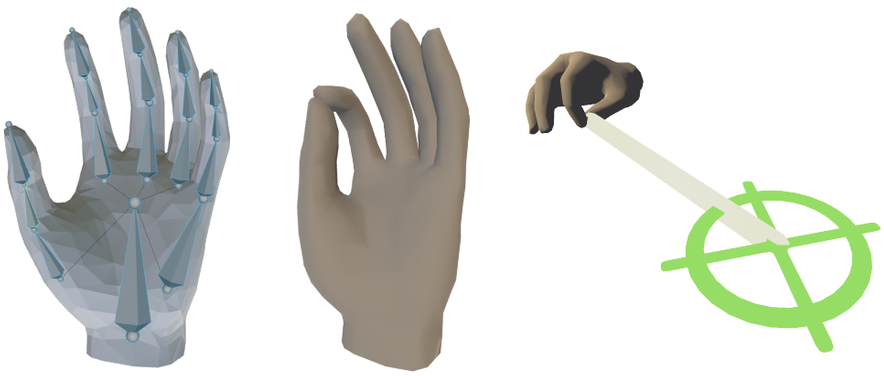

Figure 3.5: Left: The hand model [Tec17] with its rig. Middle: The “pinch” keyframe applied on the mesh. Right: The same pose of the hand during utilization.

In order to use hands as controllers I needed a mesh of a hand and the adequate Occulus hand models [Tec17] were used. These came originally in the .mb Maya format and had to be converted to the .fbx format for proper use in Unity. The rig of the models was already available and so keyframes for the gestures “pinch”, “point” and “throw” could be set with the animation tool integrated in Unity.

A rig basically represents the bone structure of a model. In this case, the hand and finger bones are represented in a simplified hierarchical structure of bones being interconnected. The mesh, which is located around the bones, adapts to the transformations of the bones and therefore different poses can be represented by putting the hand skeleton into specific shapes. The keyframes store the changed positions of the bones in the rig and can then be referenced in the context of Unity blend trees.

The blend trees are components that can be used as keyframe managers in Unity’s animator system. They allow the conversion of keyframes to a model in runtime and allow smooth blending between different frames. Thus it is possible to determine with a value from 0 to 1 how much a keyframe, respectively how much the ”point” gesture for example should be applied to a hand model. As a reference for this value, the value of the trigger button is read out via the input system and the hand gesture is adjusted depending on how hard the button is pressed. This gives the user intuitive control over the hand, as the finger movement on the trigger button is directly transferred to the hand in VR.

The hand models were then each assigned to a controller, which created the interaction basis. In order for the interface, between the user and the controller, to be designed as simple and intuitive as possible, most actions are activated with the trigger button. Likewise, it does not matter whether the user uses the right or the left hand for the interactions.

These measures contribute to the fact that the user does not have to think unnecessarily about which button she has to press and contributes to the maintenance of the immersion. To enable more than one action to be performed with the same button, I have designed the application in such a way that it can adopt different states with different modes in which specific actions can be performed with the same inputs. For example, there is the pinch or the point mode in the main menu state, which determines which current action should be executed with the trigger button.

Figure 3.6: The controller of the HTC Vive Pro with the trigger button and the touchpad button marked in yellow.

One other action has been implemented separately from the trigger button. Namely, the activation and deactivation of the panel through which most interactions with the application can be controlled. This action can be performed at any time and takes place via the touchpad button. Again, it does not matter which hand is used to perform the action. The touchpad button and the trigger button are marked yellow in Figure 3.6.

When activated, the panel is placed at face level, one meter away and facing the user. Buttons are displayed on the panel, which offer the user options that can be selected at that particular moment. This is controlled by the state, in which the application is in at any given time.

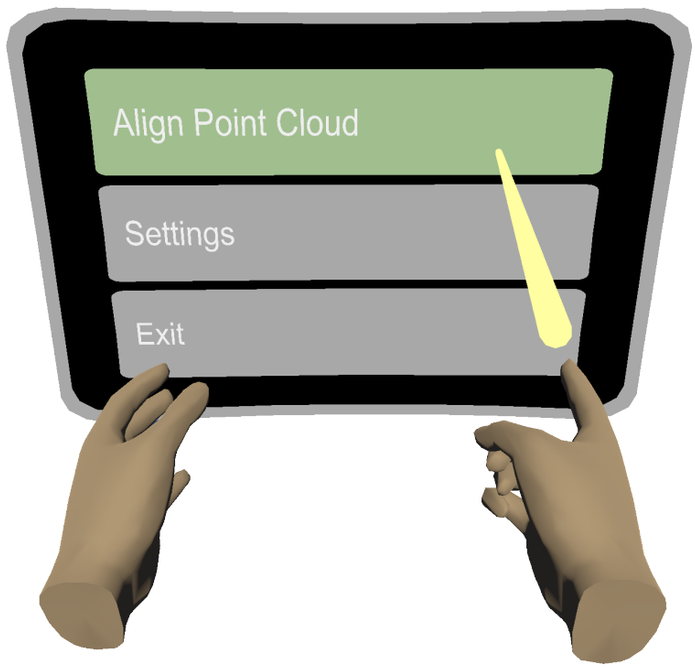

Figure 3.7: The selection of a button on the panel by pointing at it and pressing the trigger button. This is the panel the user will see in the beginning after launching the application.

As shown in Figure 3.7, a ray appears on the hand that is currently triggered. If both hands are triggered, then the beam appears only on the hand that was activated last. This gives intuitive and unambiguous control. When the ray is pointed at a button, it is indicated by color shift whether a button is selected or activated. The distance from the player to the button does not matter in this case, because a ray from the hand with the same direction as the visual ray is cast and checked for collisions with a button. The activation takes place by pressing the trigger button completely.

3.4.3 Room Alignment

In order for the scanned data to be displayed correctly in the virtual environment, the data must be aligned properly. This process ensures that users have the ability to move freely in the virtual space without running into walls or other obstacles. This means that the size, orientation and position of the point cloud in the virtual environment correspond to the dimensions of the real environment. The handling of the orientation is designed to be as simple as possible and should not be too complicated for the user to use. Therefore, when loading a new point cloud in the application, it is always necessary to realign the area. Certain operations can automatically be adjusted, contributing to more consistent alignments. Specifically, these are the adjustment of the size and the adjustment of the position on the height axis of the point cloud. Because of the intrinsic size of the scan, the dimensions already match those of the virtual environment, and I could assume that this would not need to be changed for the alignment.

The adjustment of the height position can also be neglected because the points of the scan representing the ground are set to height 0. Since the floor of the virtual environment is also known, these floor points can be set to the virtual floor and thus all point data can be positioned correctly along the height axis.

I decided that the user is free to choose whether they want to do the size and height orientation automatically or manually. However, by default they are set to automatic. By setting the height and scale manually, it is possible to scale an position a room in any size at any position. These adjustments can be made in the settings at the beginning after the application start.

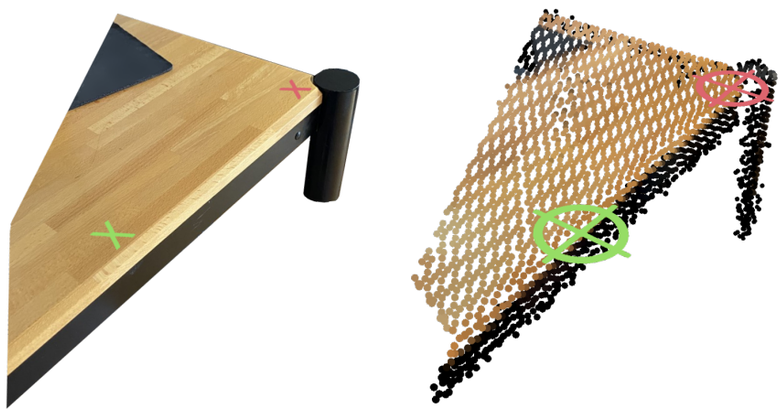

Figure 3.8: Left: The reference points are marked in the real world. Right: The points have to be adjusted in the Unity scene on the scan at the same position like the real world markings. This process is done before launching the application in order to have references to then align the point cloud at runtime.

The first thing the user sees after starting the application is a black room with a wireframe background [Lab14]. He can move around freely in it and set the aforementioned settings. It is the lobby, so to speak, in which the application can be closed, or a point cloud can be loaded. In order to load a point cloud, the user must click on the ”Align Point Cloud” button after activating the panel at his position by clicking on the touchpad button as shown in Figure 3.7. When the user selects this option, a text appears on the panel describing what to do to perform the alignment. “Drag the line from your preset from the marked starting point to the endpoint”. The application is now in pinch state and allows the user to drag a line. The hand movement changes from pointing to pinching.

To perform a proper alignment, one of the hands must now be moved to the position of the green marker in the real world. Once this position is reached, the user can trigger the controller to perform a pinch movement. This sets the green marker in the virtual environment when the trigger button is fully pressed. This process can be seen in the Figure 3.5 on the right. Then the hand should be pulled in the direction of the red marker and then release the trigger button at the red marker in the real world in order to set the red marker in the virtual environment.

The point cloud is then aligned according to the line drawn and the settings made. If the size and height positioning is set to be aligned automatically, then only the horizontal position and the horizontal orientation are adjusted.

The rotations around the two horizontal axes x and z do not need to be taken into account, since the scan is already correctly aligned. In order for the transformations to be performed on a single object, all objects associated with the point cloud are structured as children of a point cloud parent. All the transformations that are then applied on the parent, are directly transmitted on the children. Additionally the position of the point cloud parent is set onto the green reference point set in the scene before. The first transformation is the adjustment of the position of the point cloud. The green point previously set in the Unity scene is used as the reference point for the new center. The point cloud is moved, so that this point coincides with the green point set when the line was drawn.

Next, the direction of the vector between the points set in the scene is compared to the same measured direction of the vector from the points set in runtime. For the comparison, the vectors are projected onto the horizontal plane, since the height axis is not relevant for the reorientation of the point cloud. The comparison results in an angle that describes the necessary rotation needed to align the point cloud to the drawn line. In C# an angle of two vectors can be measured using Unity's built in Vector3.Angle() function. However, this angle is always positive and at most 180 degrees. This leads to problems with the reorientation of the point cloud, since there are ultimately two possible initial situations for each angle. In order to distinguish between positive and negative angles, I have made the cross product of the two vectors to find out the angle polarity. This creates a new vector that is perpendicular to the two others and whose orientation provides information about the constellation of the measured vectors. This is shown in the following code Listing 3.1.

If the size and height are not automatically adjusted, the size ratio between the set points is measured and thus applied on the point cloud origin. When positioning, the height is also adjusted in comparison to before and not set to 0, so that this is not necessarily set on the floor. The non-automatic height or size adjustment allows users to view the point clouds in custom scenarios.

Listing 3.1: Aligning the point cloud in Unity

```
public void AlignPointCloud()
    {
        Vector3 oldDistance = Vector3.Distance(oldGreenPosition,
            oldRedPosition);
        Vector3 newDistance = Vector3.Distance(newGreenPosition,
            newRedPosition);
        Vector3 oldDirection = Vector3.ProjectOnPlane(
            oldRedPosition - oldGreenPosition, Vector3.up);
        Vector3 newDirection = Vector3.ProjectOnPlane(
            newRedPosition - newGreenPosition, Vector3.up);

        // Adjust position (the pointCloudOrigin is set to the
            oldGreenPosition)
        pointCloudOrigin.transform.position = newGreen.transform.
            position;

        // Adjust scale
        float scaleProportion =  oldDistance / newDistance;
        pointCloudOrigin.transform.localScale = pointCloudOrigin.
            transform.localScale / scaleProportion;

        // Adjust orientation according to vertical axis
        // Obtain the directions by projecting the measured vectors
             on horizontal planes
        float angle = Vector3.Angle(oldDirection, newDirection);
        // Get cross product in order to determine angle polarity
        Vector3 cross = Vector3.Cross(oldDirection, newDirection);
        if (cross.y < 0) angle = -angle;
        pointCloudOrigin.transform.RotateAround(pointCloudOrigin.
            transform.position, Vector3.up, angle);
}
```

3.4.4 Interactive Elements

In order to actively interact with the point cloud, I tried to make use of segmentation. The interactive elements give new properties to the space, by allowing it to be perceived and experienced differently than just being there, without VR. I inspired myself with Holoft [Gug21] and tried to implement various elements in order to enhance the experience besides being able to see the room as a point cloud. In this subsection I explain how the different interaction possibilities work. The first subsection describes what measures help to allow a user to manipulate the scene. The next sections explain the interactions and the themes that were implemented.

Scene Manipulation

In order to interact, the panel can be used again. For this, however, the buttons change, since the application is now in a new state and new modes can be activated. In order to use the panel, the user can place it in a suitable location and operate it from there. All possible interactions are divided on several pages and can be accessed by the pointer. The first page shows the buttons to activate and deactivate the labeled segments. It also allows changing the colors of the different elements and it allows the user to adjust the disk size of the points. The first page of the panel can be seen in Figure 3.9.

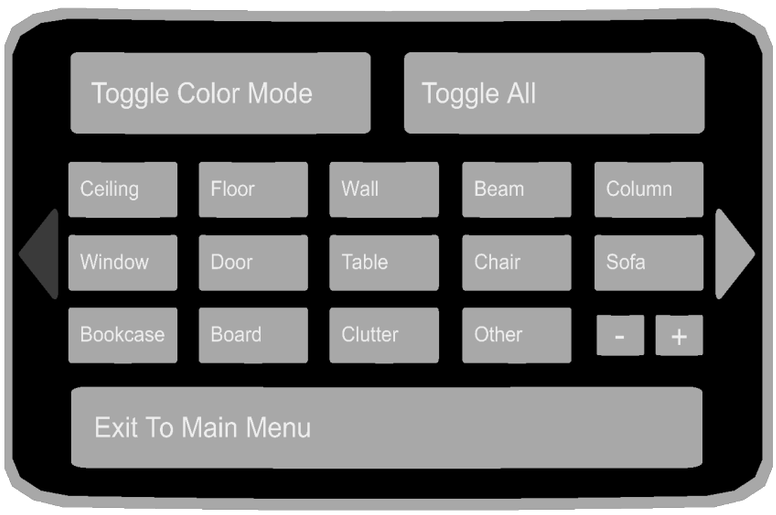

Figure 3.9: The first page of the interaction panel in order to change the appearance of the scene.

The first two interactions that I describe here are additionally used to also visualize the point cloud segmentation and do not only serve the purpose of changing the look of the space.

The simplest interaction lets you activate and deactivate the individual segments separated by labels. These labels can also be seen on the panel in the figure 3.9 and so, for example, you can hide tables and chairs in a room to see how the room would look without them. An additional ”toggle all” button helps to switch all objects on or off together, so that the original situation can be restored.

The other interaction which also helps to visualize the segmentation is the assignment of color tones for the different segments. This way, the boundaries of the segments can be seen at a glance. For this purpose, a set of 13 colors was defined as a color palette and then assigned to the segments. The user can use the “toggle color mode” button on the first page of the panel, which activates or deactivates all colors. The execution of this mode can be seen on Figure 4.1.

Another adjustment of the scene on the first panel is the adjustment of the size of the disks representing the points. This allows gaps in the scan to be filled in and makes the appearance of the scan a bit more realistic. The resizing of the dots remains during the navigation across all panels. The difference that the size of the points makes can be seen on Figure 3.10.

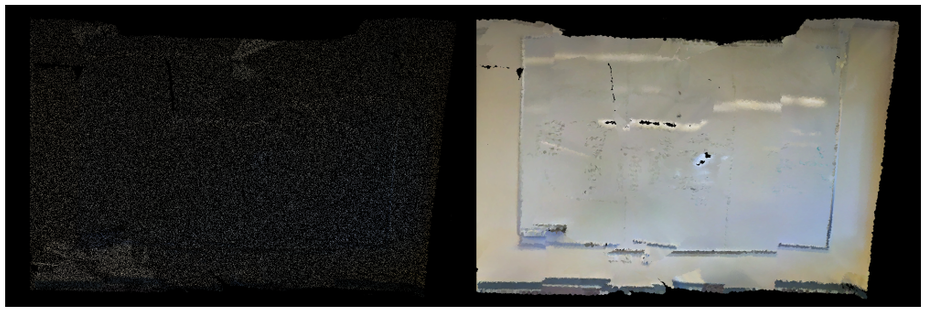

Figure 3.10: Left: A wall segment with a rather small point size. The wall looks very sparse and the color of the points is not expressive. Right: Increasing the size of the points fills up the thin look and creates a dense looking surface with color.

On the next page of the panel there are more options that allow the user to change the appearance of the scene. There are the following six options:

- Replace chairs with mesh
- Replace table with mesh
- Replace walls, floor and ceiling with mesh
- Put plants on the tables
- Put plants on the bookshelves
- Change wall textures

Similar to the method presented by Nan, Xie and Sharf [NXS12] I wanted to exchange certain segments with a mesh. With the information given by the segmentation, it was possible to identify where tables or chairs were located. I assume that there is only one table in the room and that the room is displayed in real size. The chairs and the table, which are instantiated as a prefab, have a fixed shape and size. This limits the accurate representation to the initial situation of the scenes.

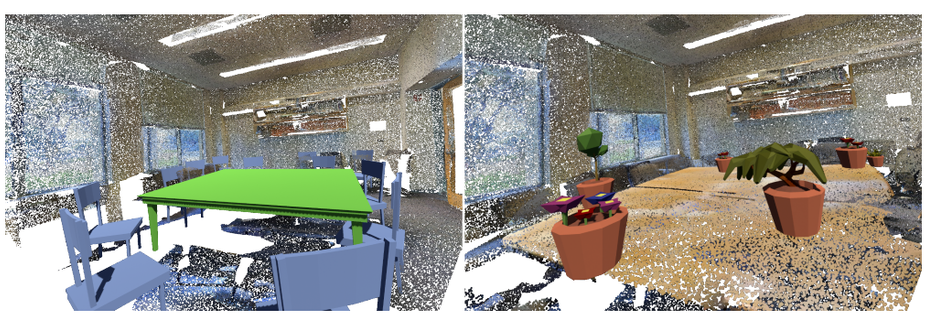

Figure 3.11: Left: The table and the chairs have been replaced with a mesh object. Right: Plants have been placed on all surfaces labeled as a table.

To get the center point for the placement of the table, rays are cast down over the whole scene from above the point cloud. The rays are arranged in a grid and close to each other. As soon as a ray collides with a searched object, this point is saved. From all the points saved after all the rays are cast, an average point is calculated to get the placement point for the table. The height of the point does not matter, because the table is placed on the ground anyway.

The same method is used for the placement of the chairs. However, instead of choosing a center point, a subset of all collisions is chosen where the points have a certain minimum distance. This prevents two chairs from being placed inside each other. In comparison to the table, the chairs are additionally rotated towards the table. This action is based on the assumption that chairs in a room are basically rotated towards tables. Thus, I am carrying out the assumption made by Nan, Xie and Sharf [NXS12] and make use of contextual information.

For the placement of the plants on the table and on the bookcases I use the same process as for the placement of the chairs. Here the collisions are chosen as placement points and random plants are set as prefab. The process is explained in the Figure 3.12. The result of the chair and table replacement as well as the positioning of the plants can be seen on Figure 3.11.

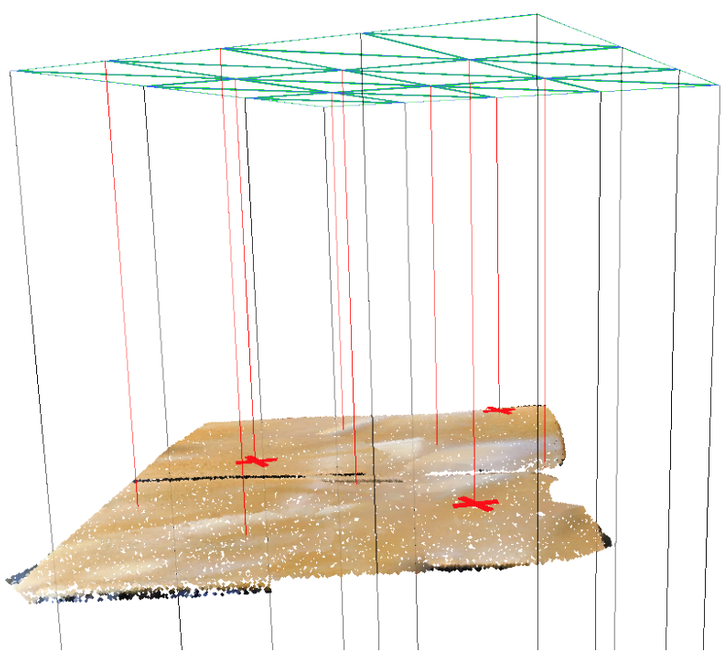

Figure 3.12: A grid (green) casts rays downwards in order to detect valid objects. From all the rays that collide with the specific label (red rays), a random subset of the ray collisions having a specific minimal distance between each other is chosen (red crosses) as possible points to place objects.

In order for the collisions to be detected correctly, the objects must be provided with a mesh collider component in Unity. Since Unity can only use these colliders in convex form for performance reasons, there is a problem with complicated and scattered objects. If the labeled segments are considered as whole objects and one mesh collider component is used per object, then concave topological properties are ignored and the colliders span the object as a whole. This is very unfavorable for certain interactions and for the exact detection of collisions with rays. The problem can be seen on the left side of the figure 3.13.

I have solved the problem by taking the separated patches as objects for the collision detection. They can still be assigned to a label and are placed at the same position as the rendered points, but are not rendered themselves. Due to their simple topology, the mesh colliders can be applied more precisely. The application of the mesh collider can be seen on the right side on figure 3.13 and shows how the collider (green bounding box) fits much more accurately around the chairs.

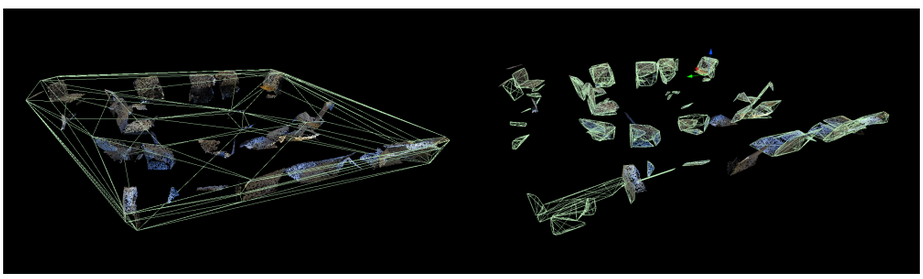

Figure 3.13: Left: The convex bounds of the mesh collider applied on all the points together labeled as chairs, Right: The convex bounds of the mesh collider applied on all the patches separately labeled as chairs.

The walls, floor and ceiling can also be replaced. The corresponding point clouds are hidden and the mesh is loaded from the surface reconstruction 3.3.3. This mesh also has the point cloud origin as parent and is therefore in the correct position. The “Change Texture” button randomly activates wall and floor textures from a set of predefined textures. In addition, when activating the mesh, the lighting must also be set from the directional light to a point light, because otherwise the room is not lit from the inside. The position is placed 1m above the table.

The Figure 3.14 shows the effects of the materials used. The parquet floor, for example, has a high smoothness value given from the used the Unity standard shader, and therefore looks shiny.

Figure 3.14: The replacement of the walls, floor and ceiling with a mesh from the surface reconstruction. The textures on the walls and on the floor are chosen at random from a selected set of textures.

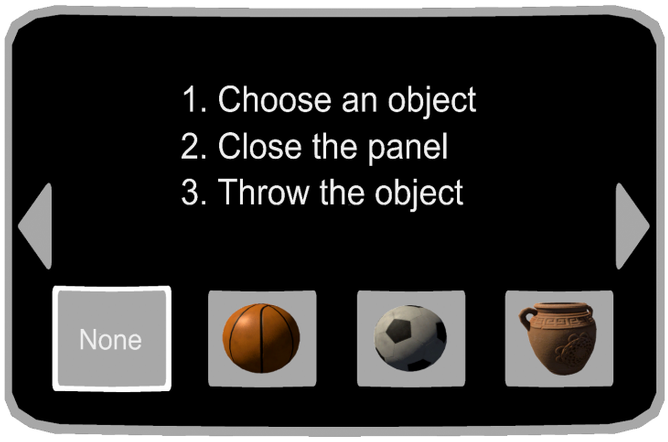

Figure 3.15: The three throwing objects, which can be chosen and thrown from both hands. Basketball: Very bouncy. Soccer ball: Less bouncy. Vase: Breakable

Scene Interactions

In order to interact directly with the scene, and to take advantage of the patch segmentation, I decided to use the throwable objects. On the third page of the panel, the user can choose between the three throwable objects as shown in the figure 3.15.

The objects have four different properties that are noticeable when thrown. The basketball has a so-called physics material with a high bouncyness and bounces off all objects. The football, on the other hand, has no bouncyness. The vase breaks and shatters into many pieces in a collision with another object, if the speed at the impact was high enough. A small cloud of dust is created to enhance the effect. These objects

are assets from the Unity Asset Store [Ism15], [JN319]. An example of this feature in action can be seen on Figure 3.16, where jars are shattered by throwing them at other objects.

Figure 3.16: Demonstration of the throwable objects thrown from both hands. On the ground many basketballs soccer balls and fragments from the jar are visible

Scene Themes

Figure 3.17: Left: Under water theme. Right: Nature theme.

I was inspired by the work presented in section 2.1 and came up with two possible themes that could work well in this VR application. The user is immersed in a new world and experiences the space in a completely new environment.

For the underwater theme, all elements except the ceiling, floor, chairs, table and columns are removed. This opens the space enormously and allows the fish to swim through the space. Fish look for random points within a defined bounding box and then check if a possible aiming point is behind an obstacle. If this is the case, they look for new points. The fish have an animation and play it in a loop. There is also a humpback whale [Jan12], but it does not get too close to the user. The two sharks [Jan18] will choose random points like the fish, but they are very calm and won't attack at any time. The caustics [Ait14], which can be seen on the ground, were projected onto the ground plane with a projector component and an additive shader. The water surface is another shader [Ves21] from the Asset Store. A screenshot from the under water scene can be seen in figure 3.18. An enhanced underwater effect is also underlined with a matching soundscape and supported with a dense ambient fog as well as with a specific visual post processing profile.

The nature theme uses a completely different post processing profile, and I try to emphasize the atmosphere a bit with a bloom effect. In order to have a vivid sky, I added a dynamic skybox with a sun disk provided by Unity to the scene. Also an adjusted soundscape and specific lighting conditions give the scene a believable look [War15] and [Pol15]. Two deer [Jan18] target random points around the user and then go there to eat for between 5 and 15 seconds before targeting new points. They use the Unity navigation mesh in order to navigate on the terrain mesh. This way they are avoiding the objects like the point clouds or the rocks [K418] in the distance. If the player gets too close and within a certain radius to the deer, they look for a new point in the approximate opposite direction and a certain distance and run towards it until they return to their normal activity. This effect as well as the clouds [Wor17] add depth to the scene and put the player in this world. Figure 3.19 shows how the nature theme looks in combination with the point cloud.

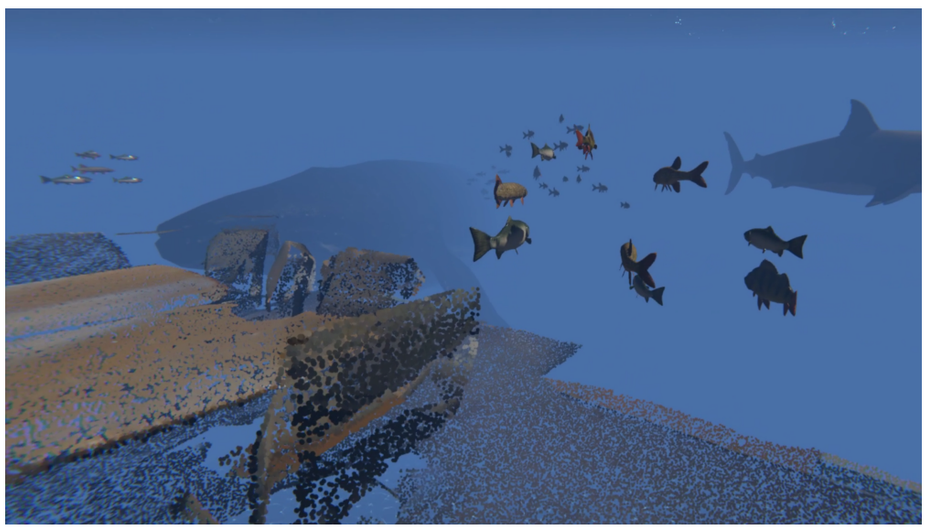

Figure 3.18: A glimpse of the underwater world with a humpback whale and sharks in the background.

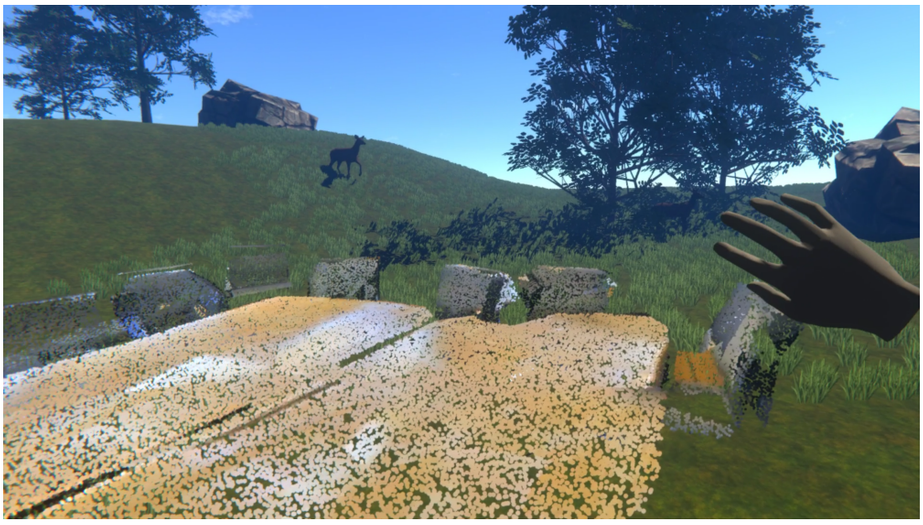

Figure 3.19: A frame of the nature scene with the deer in the background and the isolated table and chairs in the foreground.

3.5 Implementation

Although the previous sections deal with various implementation processes, I would like to take a general approach to the workflow and methods in the following section. I show how I proceeded with the implementation and what problems I encountered. The section covers the code structure and the methods used to test the system. The events are explained here in chronological order.

3.5.1 Project Setup

At the very beginning of the project Prof. Dr. Thomas Fritz, Prof. Dr. Renato Pajarola, Lizeth F. Perez and I set the goal to create a VR application that displays a room scan in real dimensions and in which a user has the ability to interact in various ways with the environment. Additionally the goal was to automate the process of taking a scan and using it in the mentioned interactive application. First, based on this goal, a rough outline was laid out and methods were discussed to achieve this project. The HTC Vive Pro was then provided by the University of Zurich and from there on regular meetings were held with the leading professors and the assistants at 2-week intervals to discuss the state of progress and immediate goals. The code of the different parts was shared within a repository over GitLab.

3.5.2 Procedure

The source code for the PGCNet was provided by Prof. Dr. Pajarola and studied with the paper [Sun+20] released in the visual computer journal. For this purpose, the S3DIS dataset [Arm+16] was downloaded and with some consultation with Dr. Yuliang Sun, Prof. Dr. Renato Pajarola and Lizeth F. Perez we were able to configure the code in such a way that it was possible to read the data correctly but not to train it yet. This process proved to be particularly difficult, as some components in the code had to be changed or added in order to work. Additionally, the code for the surface reconstruction [Nan21] was also brought in and studied, with the runtime environment set up using cmake and Visual Studio 2019. For this purpose, an example cube object was read in with the help of Lizeth F. Perez and set aside after successful processing.

In a next step, the code for patch processing was obtained to meet the requirements to train the network. Due to certain characteristics, the patch processing code [Mat+14] could not be compiled in Visual Studio 2019 because the automatic reformatting would have changed essential parts of the code in such a way, that the application would no longer work correctly. Therefore, Visual Studio 2010 had to be installed, with which I managed to compile the code correctly. The patch processing code was then adapted directly as it was needed for the project. This means that the application, which was ultimately responsible for these processes, carried out further calculations after the patch processing and saving the files needed. These other processes were not isolated, as they did not have a large impact on the performance of the calculation and were therefore not considered disruptive.

For the pre- and postprocessing steps of the data I first tried to integrate these processes into the given code too, but this turned out to be rather complex and difficult. Therefore I wrote specific python scripts for the individual data processing steps, which processed the data accordingly. This meant that some steps in the pipeline, especially between the calculations, had to be done manually from the beginning to the end. This included transferring data and running scripts by hand, which took some more time, but allowed for more transparency during development.

After training the network with the generated patches, the test accuracy was in an inaccurate range below 50% at first. Further discussions with Dr. Yuliang Sun have revealed that the data of the patches need to be normalized to obtain more consistent results. In addition, various parameters such as the batch size or the choice of CPU or GPU were adjusted until the results were sufficiently good.

I tried to generate meshes from the point clouds, however there was no quality improvement of the appearance of the data. I then decided to keep the point clouds as they are and with the ability of changing the size of the points the appearance could have been enhanced enough to look convincing in Unity. This effect is demonstrated in Figure 3.10.

In Unity, I started looking for frameworks and plugins that are suitable for this project and first came across the SteamVR plugin under the XR plugin management, which was standard before Unity version 2020.2. However, this was partially replaced by the OpenXR plugin at the beginning of 2021 and so I decided to build the Unity application with the new plugin at a later date.

In addition to the main project, 2 other Unity related projects were created, in which the concepts mentioned were used as a mockup and for testing.

In order to test the proposed ideas and features and to show them well, an environment was created in the testing project, in which the same mechanisms could be viewed traditionally on a screen without VR. Testing and debugging the project in VR was rather straight forward because the application did not need to be built and could be run and inspected directly within editor with the HMD. For the final product, the VR application was then built and made available as an .exe for Windows.

4 Results

In this chapter I present the challenges with the corresponding justified solutions that occurred during the development of this project. The consequent main contributions are answering the two research questions that were presented in the introduction first:

- What are the challenges to realistically recreate a physical environment from point cloud data in a virtual setting with interactive elements?
- To what extent and how can the implementation of a VR application be automated?

The following two sections 4.1 and 4.2 will discuss the corresponding research questions in detail.

4.1 RQ1: What are the challenges to realistically recreate a physical environment from point cloud data in a virtual setting with interactive elements?

The question of how the pipeline for this project should look like had to be asked at the very beginning of this work. A big challenge in using point cloud data for a VR application like I did in this project, is that the data does not have the same structure like mesh data. This means primarily that the data consists only of vertices and not out of planes, edges and vertices as it would be the case in a mesh. I addressed this problem by using the game engine Unity and a custom renderer, which is able to display point clouds. Unity provides many tools that help to implement interactive environments in various settings.

In addition, the unstructured nature also means that it is not possible to differentiate between objects. The separation of the different elements is an essential basis for the interactions to be implemented. This is because in most cases the interactions are related to specific objects or areas of a room. This enriches the types of interactions and thus brings depth to the application.

It can be concluded that the use of point cloud data requires a special set of methods to transform the data into a VR application. Therefore, an essential processing part of the data is a semantic segmentation before the integration into a virtual environment. The specific challenges regarding the segmentation are described in section 4.1.1. The challenges of the handling of the data must also be addressed in the integration in Unity, which follows after the segmentation and is described in section 4.1.2.

Figure 4.1: The visualisation of the semantic segmentation in VR. The labeled segments of a room are colored after toggling the color mode.

4.1.1 Segmentation Challenges

The semantic segmentation of data is a very complex task. Although segmentation can be carried out very precisely by hand, it is a very tedious task and, in the case of large data volumes, it is not always feasible. For larger amounts of data, neural networks are suitable, which, once trained, can evaluate or label large amounts of data in a relatively short amount of time. In this project, the PGCNet [Sun+20] was used, which is specially optimized for indoor environments. It uses planar input patches, that represent planar segments of the raw data, which I also generate [Mat+14]. These methods allow efficient fine-grained segmentation and semantic segmentation of the data. On the one hand, this has advantages because large amounts of data can be processed in a short amount of time. On the other hand, however, there are also challenges caused by the use of these methods. Nevertheless, when extracting patches, care must be taken to ensure that the raw data does not contain details that are too finely granulated, as these could not be captured as planes and would then be lost.

Patch Segmentation

The segmentation of patches, which is based on a seeded region growing method, extracts planar planes from a scan. This means that the raw data should preferably contain many flat areas so that as many points as possible can be assigned to a patch. Such requirements can be well met by indoor scans, especially from offices, as they fulfill these characteristics. On average, about 90% of the points could be assigned to specific patches from the data used in this project. This is a small loss of information and mainly concerns data that could be considered noise or insignificant anyway.

Semantic Segmentation

The next segmentation, the semantic segmentation, is then done using a convolutional neural network called PGCNet. The challenges in relation with the network in this project consisted on the one hand in the correct adaptation of the code, so that the patches could be read in correctly. On the other hand, however, the composition of

the data with which the network has to be trained and also tested with, has a major impact on the quality of the segmentation. The network was trained with batchsize 4 using Areas 2 to 6 from the S3DIS dataset within 56 minutes and was able to test Area 1 under one second. The test accuracy of epoch 200 amounted to 70.35%. However the quality of the segmentation was not as good with tests from a scan of a University of Zurich office. The UZH scan contains slightly different colors and different types of furniture, which makes the recognition of the trained labels more difficult. It is important to note that semantic segmentation should be based on a relatively similar data set with a ground truth to be trained on as the set to be tested.

4.1.2 Unity Challenges

Because point cloud data really differs from the usually used mesh data in VR applications, it comes with some specific advantages and disadvantages. An advantage that I was able to use for this project for example, is the correct proportions of the scan. This made it easy to scale the scene in Unity later on, as it could be taken directly from the original data. A mesh does not necessarily have the correct proportions when it is created by an artist for example.

Unity enables a simple implementation of interactive applications in VR. Nevertheless, implementing interactions with point clouds yields some additional challenges. Point clouds do not directly represent a mesh as Unity would use it, because it just represents a collection of points. This also means that the point cloud basically cannot be rendered and that Unity cannot add colliding bounds as it would with traditional meshes. The collider components, which are provided by the Unity engine, are components that can be put on a mesh object. Whenever this is done, Unity creates an enclosing mesh that represents the colliding borders of an object for the physics system of Unity.

However, because the point cloud does not represent a real mesh, a specific importer and renderer [Tak21] was used as a plugin to allow Unity to use the point clouds as meshes. With this plugin it was possible to treat the data as a normal mesh and so also so-called colliders could be added, which can then be used in the Unity physics engine and form the basis of many interactive elements. A problem that arises here, however, is the generation of convex collider bounding meshes for the corresponding point clouds. This would mean that for large and complex point clouds many details would be lost. The problem was solved by applying these bounding meshes to the separate individual patches from the patch extraction. This process is illustrated in Figure 3.13 and explained in section 3.4.4.

4.2 RQ2: To what extent and how can the implementation of a VR application be automated?

Another goal of this work is to automate the process, which starts from a 3D scan and ends with an interactive VR application. In practice, it would mean that a user can convert a scan into an interactive environment with little manual effort. Manual modeling, manual alignment or manual segmentation should be eliminated or minimized. In general, seamless data processing contributes to the automation of this process, which in this project takes place with the help of python scripts in the preprocessing and postprocessing phases. Although some steps in this project still need to be done manually, they could be optimized for future projects.

The main approach to address the question of automation can basically be answered with a method to segment the data like it is used in this project. However, since the results of the segmentation are then used to implement interactive elements directly, the question can be taken further and should be answered with the interaction possibilities that arise from the segmentation. In the following section 4.2.1, I am discussing the challenges that I have encountered during the implementation of different interactive elements.

4.2.1 Automation Challenges

While iterating over the interactions that relied on the segmentation I am addressing the challenges and the findings that I encountered during the implementation. In order to answer this question I assume that the measures of RQ1 have been realized, so that further implementations can be built upon the mentioned fundamental solutions.

Visualizing the Segmentation

The first interaction that is described in the section 3.4.4 is concerning the activation and deactivation of different labeled elements as well as the coloring of the corresponding labeled objects. This is the most basic interaction that I came up with, that is based on segmentation. It demonstrates how the segmentation actually labeled the points and could act as a simple visualization of the segmentation as well. The challenge in regards to this interaction are the logical references in Unity to the corresponding labeled segments. This problem is solved by referencing the separate files, of which each file represents a single segment, by their filenames. The filenames contain the information about the label and they are created during the reconstruction phase in the postprocessing 3.3.2 after being tested in PGCNet. Therefore, if files can be referenced automatically, they can be easily be altered by changing the color or deactivating their rendering component.

Replacement of Point Clouds with Meshes

The next panel is providing interactions that utilize the segmentation to alter the appearance of the scene in a more complex way. The option that replaces certain labeled elements with meshes could be very useful for a wide variety of applications. A scene based on meshes uses less memory, is easier to render in different lighting conditions and ideally has no noisy appearances. These are only a few advantages, however the

usability of such scenes is very dependent on the quality of the replacement. The approach that is used in this project is described in section 3.4.4, however the challenges that I had were to find the precise boundaries of an object and to orient an object correctly. Due to the methods I used to place and orient the objects, the reconstruction is not flawless in some situations. All the chairs will be facing the midpoint of the table that is detected, and therefore they might be oriented the wrong way 4.2. Nan,

Figure 4.2: The rotation of the chairs is only guided by the position of the table. This leads to chairs being rotated in the wrong direction as can be seen on the right border.

Xie and Sharf [NXS12] solved the problem of fitting objects by starting out with a highly accurate detection of different objects and boundaries in a scene. From there on a fitting mesh object is chosen from a corresponding set of objects representing the detected label. The template objects are then fitted via deformation methods and structure-preserving optimizations over several iterations. This approach yields good results as it can be seen on Figure 2.5, but it was out of the scope of this project.

Placement of Objects

The next challenge that I encountered is concerning the placement of the plants on different surfaces of labeled segments. This process is described in Figure 3.12 and illustrates a solution that searches for surfaces. Another way to approach this problem would have been to search the upper bounds of the corresponding segments directly, but then contextual information could not have been included directly. This means that for example a placement of a plant across the floor would not have considered tables or chairs that stood in the way of the plant.

Replacement of Walls, Floor and Ceiling

The process that enables the user to swap the walls with a mesh model and then change the textures was automated in such a way, that those walls are generated with the help of a surface reconstruction algorithm. This process makes use of the segmentation, because only the walls, floor and ceiling can be taken as an input. This would replace the manual labor of modelling a room confining box, but yields some further challenges like the correct UV unwrapping and application of the correct textures. This step is done manually in this project, because it causes very little effort. However this could be improved in further automated applications.

Throwable Objects

In order to implement throwable objects in Unity, the used components should be able to work with an underlying physics system. Point clouds do not resemble meshes as Unity uses them by default, which yields further challenges to implement an interactive system that works with the physics system. However those challenges are addressed and explained in RQ1.

Themed Environments

An apparent challenge regarding the themes was to activate or deactivate corresponding elements in such a way that the scene seems to fit in with the background. This means that I chose to leave the ceiling and floor on for the water scene, but i chose to leave everything out except table and chairs for the nature scene. This helps to embed the point cloud elements into their surroundings in a seamless way.

Another challenge was the adjustment of the size of the theme. In the nature theme the size of the bounding box of the room could have been taken into account in order to adjust the shape of the terrain, so that the terrain fits to the room. The water theme was a bit more open for such problems, because there were no apparent boundaries that should have been moved dynamically. For the scope of this project this problem was solved by setting the size of the base of the terrain to a standard sized square, which should fit most of the rooms in normal size.

An additional problem would have been the disabled walls or other objects, that are not perceived anymore by a user whenever they are deactivated. This could mean that a user walks into something, because he does not see the real boundaries of his walking perimeter anymore. This problem is partially solved by the underlying Vive engine that provides a user with a grid and warning whenever a border of the preset perimeter is reached. The VR application could have implemented further warn signals that show the user when she gets too close to a certain object that is not rendered at a given moment. However this was not implemented due to the scope of the project and also because the user will be warned by the underlying system provided by Vive.

5 Discussion and Conclusion

A virtual reality application has been implemented using semantic segmentation of point cloud data. I specifically mapped an office space to a virtual environment and was able to automate various integration processes by segmenting the data. The transformation of a room into an interactive virtual environment opens up many new application possibilities.

In order to implement various interactive elements, I used the segmentation in different ways. The segmentation allowed me to define precise physical bounds, replace certain elements, or reconfigure the space. The implemented interactions, as shown in chapter 3 and 4, would require significantly more manual effort without the segmentation.

These were inspired by various approaches mentioned in related work from chapter 2. For example, one idea of swapping point clouds with meshes comes from Nan, Xie and Sharf [NXS12], who show how the quality of patchy scans can be greatly improved. The interactive elements, which include throwing objects, were inspired by J. Gugler's work [Gug21], among others. They bring a certain gamification to the space and have an entertaining effect.The implementation of the themes was inspired by the work of Ruvimova et al. [Ruv+20] and Soyka et al. [Soy+16] and uses segmentation by allowing walls or other obstacles to be selectively hidden, to open up the space and allow objects to fit into the environment in a more or less seamingless manner.

The following section explains an outlook on how the work can be taken further and where it could be used. I thus showed that many concepts previously shown in other work can be well implemented with the process presented in this work. The following sections address limitations of this work and explain possible future extensions.

5.1 Limitations

The quality of the VR application depends strongly on the accuracy of the segmentation. This means that if the segmentation is poor, it is difficult to benefit from the labeled segments used in this work.I also found that it was difficult to perform a good segmentation on scans that were not part of the trained dataset. It is difficult to place tables and chairs at the correct positions, if the point cloud is inaccurately labeled. In any case, the walls, ceiling, and floor were more consistently classified. Therefore, the surface reconstruction, for example, is less likely to fail.

For the point cloud alignment, it would have been useful if the built-in camera of the HTC Vive Pro could have been addressed. However, implementing an interface that would show the user a live feed of the camera proved to be particularly complex and, in this case, beyond the scope of this work. In order to correctly align the room, users will most likely have to peak under the HMD to correctly align the marked points.

A technical limitation of VR is the computing power required to display the virtual environment with sufficient frames per second. The representation of several hundred thousand points in a scene, as implemented in this project, is not possible for mobile systems. This means that only HMDs with powerful computing resources can be used for this purpose.

A requirement for using the application in a custom environment would mean that a 3D scanner would be needed to scan a room in the first place. In any case, smartphones are becoming more powerful and even have built-in LIDAR scanners that would overcome this hurdle. Therefore, in transition, I would lay out the projections for future work of this kind.

5.2 Future work

Due to the increasing computational and hardware requirements in mobile devices, especially in smart phones, fundamental new possibilities for mobile applications are constantly opening up. The ability to scan rooms and then also view them would specifically enable projects of this type for mobile devices in the future. It would therefore be possible for anyone, who owns such a device in combination with a VR system, to transform their own space into an interactive environment using the methods described here. This would be a continuation of the work that has addressed the topic of the work oasis [Ruv+20], [Soy+16], [Gug21].

An important point that could be improved is the accurate exchange of point cloud objects with meshes. The version in this work is still very dependent on the characteristics of the rooms and not so consistent. With a good implementation of this process, interactions could then be performed with individual objects rather than individual groups of objects. Implementations of gamification elements would then be evident and better realizable due to the higher granularity of the interactive components.

An interesting continuation of this work would be to make an evaluative study of the results of this project. This would mean, for example, comparing the advantages of a work oasis scanned from a real environment with an artificially built work oasis.

All in all virtual reality applications are on the rise, and with 3D scans rapidly becoming easier to handle, I am very firmly convinced that applications of the kind suggested in this thesis will have an upswing in the gaming industry. This also includes applications in connection with enhanced office environments and visualization possibilities for various new data types.

6 Acknowledgement

I would like to thank all the people who supported me with technical and moral help during this work. A big thanks goes to Dr. Prof. Thomas Fritz, Dr. Prof. Renato Pajarola, Lizeth F. Perez and Jan Gugler, who advised me at any time, supported me in case of problems and searched for solutions together. The freedom I was given to design this work made it an exciting project. I would also like to thank Dr. Yuliang Sun, who was very helpful with technical questions regarding the code of the network. Finally, I am also very grateful to my family, who supported me throughout the process of the whole thesis.

Bibliography

[Ado21] Adobe. *Mixamo Brian*. Copyright © 2021 Adobe. All rights reserved. 2021. URL: https://www.mixamo.com/. (Accessed on 05/28/2021).

[Ait14] Alastair Aitchison. *Caustics shader*. 2014. URL: https://alastaira.wordpress.com/2014/10/07/underwater-effects/. (Accessed on 05/03/2021).

[Ana+11] Abhishek Anand, Hema Swetha Koppula, Thorsten Joachims, and Ashutosh Saxena. "Contextually guided semantic labeling and search for 3d point clouds". In: *arXiv preprint arXiv:1111.5358* (2011).

[Arm+16] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. 2016.

[AYE20] Evangelos Alexiou, Nanyang Yang, and Touradj Ebrahimi. "PointXR: A toolbox for visualization and subjective evaluation of point clouds in virtual reality". In: *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*. 2020, pp. 1–6.

[Ber19] Borbála Berki. "Desktop VR as a virtual workspace: a cognitive aspect". In: *Acta Polytechnica Hungarica* 16.2 (2019), pp. 219–231.

[Bon+16] Daniele Bonatto, Ségolène Rogge, Arnaud Schenkel, Rudy Ercek, and Gauthier Lafruit. "Explorations for real-time point cloud rendering of natural scenes in virtual reality". In: *2016 International Conference on 3D Imaging (IC3D)*. 2016, pp. 1–7.

[Bry96] Steve Bryson. "Virtual reality in scientific visualization". In: *Communications of the ACM* 39.5 (1996), pp. 62–71.

[BSN14] Gerd Bruder, Frank Steinicke, and Andreas Nüchter. "Poster: Immersive point cloud virtual environments". In: *2014 IEEE Symposium on 3D User Interfaces (3DUI)*. 2014, pp. 161–162.

[CC08] Jie Chen and Baoquan Chen. "Architectural modeling from sparsely scanned range data". In: *International Journal of Computer Vision* 78.2-3 (2008), pp. 223–236.

[CPP17] Rémi Cura, Julien Perret, and Nicolas Paparoditis. "A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 127 (2017), pp. 39–56.

[DAD19] David Deibe, Margarita Amor, and Ramón Doallo. "Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures". In: *International Journal of Geographical Information Science* 33.3 (2019), pp. 593–617.

[Dai+17] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5828–5839.

[EVH20] Sami El-Mahgary, Juho-Pekka Virtanen, and Hannu Hyyppä. "A Simple Semantic-Based Data Storage Layout for Querying Point Clouds". In: *ISPRS International Journal of Geo-Information* 9.2 (2020), p. 72.

[FP06] Sagi Filin and Norbert Pfeifer. "Segmentation of airborne laser scanning data using a slope adaptive neighborhood". In: *ISPRS journal of Photogrammetry and Remote Sensing* 60.2 (2006), pp. 71–80.

[Gug21] Jan Gugler. *Augmenting working and living spaces into virtual work oases*. 2021.

[HK16] Thomas Hilfert and Markus König. "Low-cost virtual reality environment for engineering and construction". In: *Visualization in Engineering* 4.1 (2016), pp. 1–18.

[Hop+92] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. "Surface reconstruction from unorganized points". In: *Proceedings of the 19th annual conference on computer graphics and interactive techniques*. 1992, pp. 71–78.

[Ism15] Asgar Ismayilov. *Balls Free low-poly 3D model*. 2015. URL: https://www.cgtrader.com/free-3d-models/sports/game/balls-7e4f2c46-0a99-4e68-a2a4-73e5f5e93a0b. (Accessed on 04/29/2021).

[Jan12] Janpec. *Humpback Whale*. 2012. URL: https://assetstore.unity.com/packages/3d/characters/animals/fish/humpback-whale-3547. (Accessed on 05/02/2021).

[Jan18] Janpec. *Animal pack deluxe*. 2018. URL: https://assetstore.unity.com/packages/3d/characters/animals/animal-pack-deluxe-99702. (Accessed on 05/02/2021).

[JN319] JN3D. *Medieval destructible props*. 2019. URL: https://assetstore.unity.com/packages/3d/props/medieval-destructible-props-141203. (Accessed on 04/29/2021).

[Jun05] Yongtae Jun. "A piecewise hole filling algorithm in reverse engineering". In: *Computer-aided design* 37.2 (2005), pp. 263–270.

[K418] Manufactura K4. *Rock and Boulders 2*. 2018. URL: https://assetstore.unity.com/packages/3d/props/exterior/rock-and-boulders-2-6947. (Accessed on 05/05/2021).

[Kim+12] Young Min Kim, Niloy J Mitra, Dong-Ming Yan, and Leonidas Guibas. "Acquiring 3d indoor environments with variability and repetition". In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), pp. 1–11.

[Lab14] UCLA Game Lab. *UCLA Wireframe Shader*. 2014. URL: https://assetstore.unity.com/packages/vfx/shaders/directx-11/ucla-wireframe-shader-21897. (Accessed on 04/22/2021).

[mar17] maromero. *HTC Vive Headset Free low-poly 3D model*. 2017. URL: https://www.cgtrader.com/free-3d-models/electronics/other/htc-vive-headset-a6d928d6-a3a9-4c3e-8217-b275afc8585e. (Accessed on 05/28/2021).

[Mat+14] Oliver Mattausch, Daniele Panozzo, Claudio Mura, Olga Sorkine-Hornung, and Renato Pajarola. "Object Detection and Classification from Large-Scale Cluttered Indoor Scans". In: *Computer Graphics Forum* 33.2 (2014), pp. 11–21.

[MS15] Daniel Maturana and Sebastian Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 922–928.

[Nan21] Liangliang Nan. "Polygonal Surface Reconstruction". In: *CGAL User and Reference Manual*. 5.2.1. 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html.

[NL13] Anh Nguyen and Bac Le. "3D point cloud segmentation: A survey". In: *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*. 2013, pp. 225–230. DOI: 10.1109/RAM.2013.6758588.

[NXS12] Liangliang Nan, Ke Xie, and Andrei Sharf. "A search-classify approach for cluttered indoor scene understanding". In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), pp. 1–10.

[Pol15] PolyFix. *Realistic Tree Pack Vol.1*. 2015. URL: https://assetstore.unity.com/packages/3d/vegetation/trees/realistic-tree-pack-vol-1-50418. (Accessed on 05/05/2021).

[Qi+17a] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[Qi+17b] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. "Pointnet++: Deep hierarchical feature learning on point sets in a metric space". In: *arXiv preprint arXiv:1706.02413* (2017).

[Qui+15] Yann Quinsat et al. "Filling holes in digitized point cloud using a morphing-based approach to preserve volume characteristics". In: *The International Journal of Advanced Manufacturing Technology* 81.1 (2015), pp. 411–421.

[Ruv+20] Anastasia Ruvimova, Junhyeok Kim, Thomas Fritz, Mark Hancock, and David C Shepherd. "Transport Me Away: Fostering Flow in Open Offices through Virtual Reality". In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–14.

[Sha06] Ariel Shamir. "Segmentation and Shape Extraction of 3D Boundary Meshes." In: *Eurographics (STARs)*. 2006, pp. 137–149.

[Shi+16] Yifei Shi, Pinxin Long, Kai Xu, Hui Huang, and Yueshan Xiong. "Data-driven contextual modeling for 3d scene understanding". In: *Computers & Graphics* 55 (2016), pp. 55–67.

[Soy+16] Florian Soyka, Markus Leyrer, Joe Smallwood, Chris Ferguson, Bernhard E Riecke, and Betty J Mohler. "Enhancing stress management techniques using virtual reality". In: *Proceedings of the ACM symposium on applied perception*. 2016, pp. 85–88.

[Su+15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. "Multi-view convolutional neural networks for 3d shape recognition". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 945–953.

[Sun+20] Yuliang Sun, Yongwei Miao, Jiazhou Chen, and Renato Pajarola. "PGC-Net: patch graph convolutional network for point cloud segmentation of indoor scenes". eng. In: *The Visual computer* 36.10-12 (2020), pp. 2407–2418. ISSN: 0178-2789.

[Tak21] Keijiro Takahashi. *Pcx - Point Cloud Importer/Renderer for Unity*. 2021. URL: `https://github.com/keijiro/Pcx`. (Accessed on 02/22/2021).

[Tec17] Facebook Technologies. *Oculus Hand Models 1.0*. Copyright © Facebook Technologies, LLC and its affiliates. All rights reserved. 2017. URL: `https://developer.oculus.com/downloads/package/oculus-hand-models/`. (Accessed on 05/01/2021).

[TH20] Yuk Ming Tang and Ho Lun Ho. "3D Modeling and Computer Graphics in Virtual Reality". In: *Mixed Reality and Three-Dimensional Computer Graphics*. 2020.

[TP05] Daniel Tóvári and Norbert Pfeifer. "Segmentation based robust interpolation-a new approach to laser data filtering". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36.3/19 (2005), pp. 79–84.

[Ves21] Nicholas Veselov. *NVJOB Water Shaders V2.x*. 2021. URL: `https://assetstore.unity.com/packages/vfx/shaders/nvjob-water-shaders-v2-x-149916`. (Accessed on 05/03/2021).

[Vir+20] Juho-Pekka Virtanen, Sylvie Daniel, Tuomas Turppa, Lingli Zhu, Arttu Julin, Hannu Hyyppä, and Juha Hyyppä. "Interactive dense point clouds in a game engine". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 163 (2020), pp. 375–389.

[Wan+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. "Dynamic graph cnn for learning on point clouds". In: *Acm Transactions On Graphics (tog)* 38.5 (2019), pp. 1–12.

[War15] Scott Ward. *Grass Png Image Green*. 2015. URL: `https://freepngimg.com/png/4791-grass-png-image-green-grass-png-picture`. (Accessed on 05/05/2021).

[Wir+19] Florian Wirth, Jannik Quehl, Jeffrey Ota, and Christoph Stiller. "Pointatme: efficient 3d point cloud labeling in virtual reality". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 1693–1698.

[WO07] Jianning Wang and Manuel M Oliveira. "Filling holes on locally smooth surfaces reconstructed from point clouds". In: *Image and Vision Computing* 25.1 (2007), pp. 103–113.

[Wor17] Butterfly World. *BFW Simple Dynamic Clouds*. 2017. URL: https://assetstore.unity.com/packages/tools/particles-effects/bfw-simple-dynamic-clouds-85665. (Accessed on 05/05/2021).

[Zha+19] Jiaying Zhang, Xiaoli Zhao, Zheng Chen, and Zhejun Lu. "A review of deep learning-based semantic segmentation for point cloud". In: *IEEE Access* 7 (2019), pp. 179118–179133.