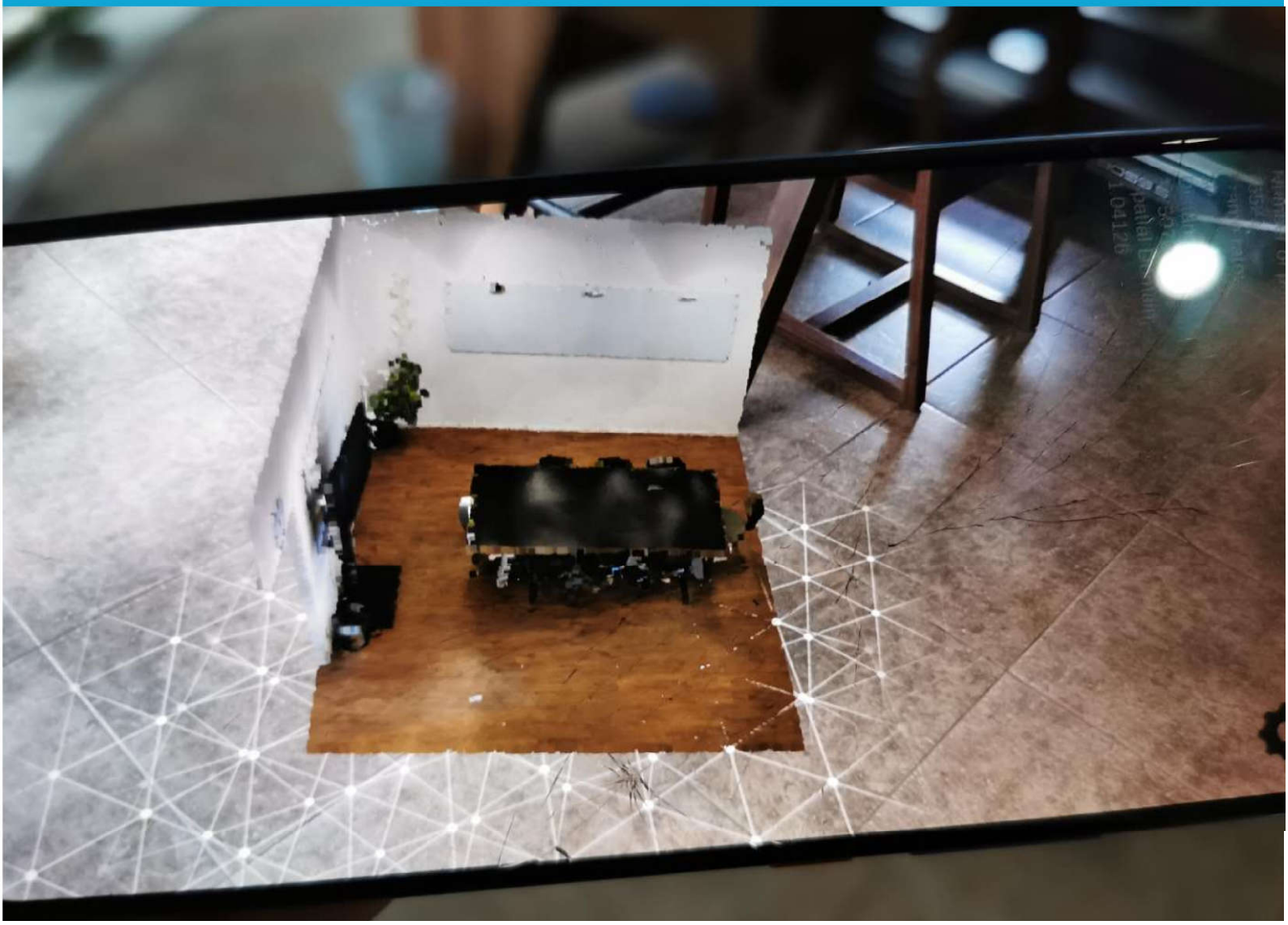
MSc thesis in Geomatics for the Built Environment

Visualization of Point Cloud Models in Mobile AR Using Continuous Level of Detail Method

Liyao Zhang

2020



MSc thesis in Geomatics

Visualization of Point Cloud Models in Mobile Augmented Reality Using Continuous Level of Detail Method

Liyao Zhang

September 2020

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

Geo-Database Management Center
Department of Architectural Engineering and Technology
Faculty of Architecture and the Built Environment
Delft University of Technology

Supervisors: Prof.dr.ir. Peter van Oosterom
Haicheng Liu, MSc
Co-reader: Paul de Ruiter

Abstract

In recent years, with the rapid development of in-situ and remote sensing technologies, people can easily get detailed digital spatial information of the real world at minimal cost and time. The resulting datasets, i.e., large scale and unstructured point clouds, have become one of the essential sources of data in multiple geospatial fields, such as the generation of city models, forest mapping, and building information modelling. However, point clouds with higher quality and quantity also bring new problems. The storage, processing, and management of these large datasets become particularly challenging.

Visualizing the point clouds is an integral part of processing the data, which enables users to explore and interact with the point clouds more intuitively. However, most of the current point cloud renderers are developed in non-immersive environments. In the last few years, some new technologies, such as Augmented Reality (AR), Virtual Reality (VR) and Mixed Reality (MR), have emerged and introduced new ways of presenting the 3D content. Among them, AR is the most commonly used technology, since AR applications can be used on mobile devices without specific equipment like helmet and handles.

There are already plenty of existed applications for mobile AR environment. These mobile AR applications play important roles in fields like architecture, industrial design, navigation, advertisement, medicine and gaming. However, the use of point clouds in mobile AR environment is still waiting to be explored. Like rendering point clouds on other platforms, the biggest issue of showing point clouds in the mobile AR environment is that point cloud datasets usually have a massive amount of data. Moreover, the mobile devices have relatively limited CPU and GPU resources, reaching ideal performance, and the visual quality requirement is quite challenging. Also, the current Continuous Level-of-Detail (cLoD) methods are developed for desktop, VR, and web-based viewers. The cLoD method has to be improved and revised in order to fit the mobile AR environment.

In this paper, an interactive visualization of point clouds using cLoD method in the mobile AR environment will be realized. The main idea of this method is to reduce the number of points to be rendered considerably. A cLoD model that has an ideal distribution over LoDs is used in the method, with which can remove unnecessary points without sudden changes in density as present in the commonly used discrete level-of-detail approaches. Camera position, orientation and distance from the camera to point cloud model are taken into consideration while filtering the points as well. In order to further improve the visual quality, an adaptive point size rendering strategy is applied. What is more, for user's convenience, in the Graphic User Interface (GUI) some setting options are provided to the rendering system to change the value of parameters required in the rendering.

The performance of the rendering system is evaluated with multiple quantity indicators and examined by different types of datasets. The result shows that our method can significantly improve rendering performance and meanwhile achieve good visual quality. The finalized rendering system is suitable for most of the indoor applications and some of the outdoor applications. What's more, a comparison between our cLoD method and the traditional mesh-based approaches will be presented to show that our cLoD method has the potential to replace mesh models in some cases.

The resource code and the installation package are available at: https://github.com/LiyaoZhang0702/AR_PointCloud.

Acknowledgements

2020 is a quite tough and special year for everyone. I went back to China for our Chinese new year in the January. However, due to the coronavirus, I could not go to the Netherlands and had to change the topic of my thesis in the halfway. It used to be quite tricky for me to continue working on my thesis, but many people provided their kind help to me. Only with their help can I finish my thesis in the end, and I want to express my gratitude to them.

First, I want to appreciate my first mentor, Peter van Oosterom, for his patience, valuable advice and selfless help. He can always get to the core of the problems and give nice suggests. As an international student, having such a kind person to be my mentor is one of the most precious memories in the Netherlands. Next, I would like to thank Haicheng Liu, who is generous to spend his time to help me. Many ideas in this thesis were devised with his advice. Special thanks to Martijn Meijers for his useful suggestions, and many thanks to Edward Verbree for his comments.

During my studies, my family always supported me behind. Thanks to my parents for their infinite love and sorry to have worried them in the past two years. Thanks to my friends for listening to my troubles and always being with me.

Thanks to everyone in my life and wish you all the best. I believe we can get through all the difficulties in 2020 in the end.

Contents

List of Figures

List of Tables

Acronyms

ADB Android Debug Bridge . . . 35
AR Augmented Reality . . . 1
ASPRS American Society for Photogrammetry and Remote Sensing . . . 18
CD Cumulative Density . . . 22
CDF Cumulative Distribution Function . . . 18
cLoD Continuous Level-of-Detail . . . v
CMD Command Prompt . . . 35
dLoD Discrete Level-of-Detail . . . 19
GUI Graphic User Interface . . . v
GPS Global Positioning System . . . 27
HLSL High-Level Shading Language . . . 25
IMU Inertial Measurement Unit . . . 27
LoD Level-of-Detail . . . 6
MP Megapixels . . . 25
MR Mixed Reality . . . 1
NIR Near-infrared . . . 27
PC Personal Computer . . . 20
RAM Random Access Memory . . . 25
SDK Software Development Kit . . . 2
SLAM Simultaneous Localization and Mapping . . . 27
UI User Interface . . . 33
VR Virtual Reality . . . 1

1 Introduction

In this chapter, a brief introduction of this thesis will be presented. More specifically, Section 1.1 is the motivation of this thesis, Section 1.2 introduces the research questions of this thesis, and Section 1.3 presents the outline of the whole thesis.

1.1 Motivation

In the past few years, in-situ and remote sensing technologies have developed rapidly. Using technologies like 3D laser scanners and LiDAR (light detection and ranging), detailed digital spatial information of the real world can be obtained at minimal cost and time. The resulting datasets, i.e., large scale and unstructured point clouds, have become one of the most important sources of data in different fields. For instance, point clouds are widely used in applications such as the generation of city models, forest mapping, and building information modelling. However, high-quality and high-volume point cloud datasets also arise new problems. It is quite challenging to get efficient storage, processing, and management of large point cloud datasets.

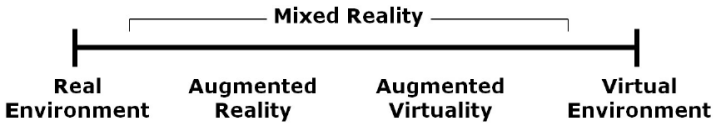
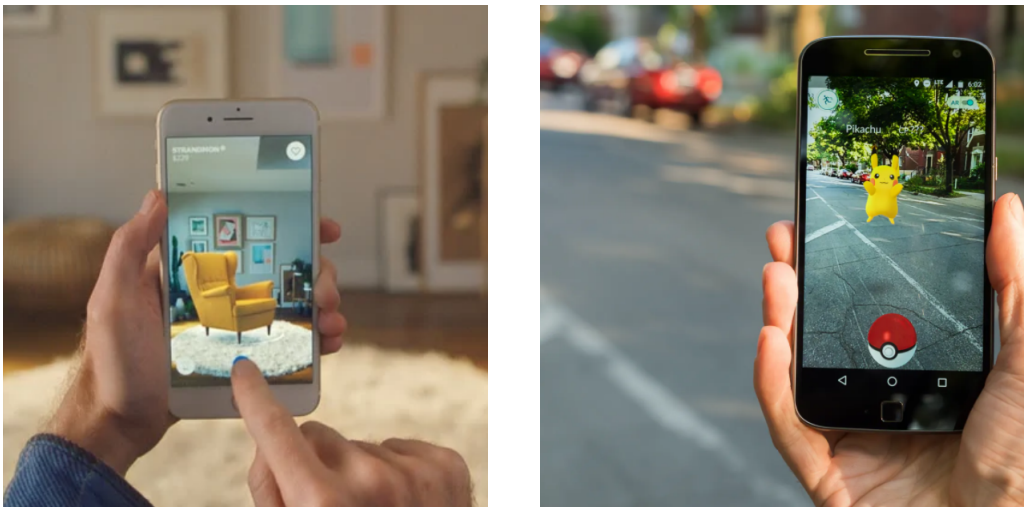


Figure 1.1: Mixed reality continuum [Milgram and Kishino, 1994]

Visualizing the point clouds is an integral part of processing the data, which enables users to explore and interact with the point clouds in a more intuitive way. However, most of the existed point cloud rendering systems are optimized for non-immersive environments [Discher et al., 2018b]. In the last few years, some new technologies, like Augmented Reality (AR), Virtual Reality (VR) and Mixed Reality (MR), have emerged and introduced new ways of showing the 3D content. Figure 1.1 shows their preferences of the real environment and virtual environment. Among them, Augmented Reality enhances the real environment by the virtual content. Augmented Reality is a very attractive technology, since AR applications can be used on mobile devices without specific equipment like helmet and handles.

Augmented Reality System is a system that combines real and virtual objects in a real environment; runs interactively and in real-time; and aligns real and virtual objects with each other [Azuma et al., 2001]. There are already plenty of existed applications for mobile AR environment. These mobile AR applications play essential roles in fields like architecture, industrial design, navigation, advertisement, medicine and gaming. The most famous mobile AR applications might be the mobile AR game, Pokémon GO, and the IKEA Place app (see Figure 1.2).

Nevertheless, few mobile AR applications use the point clouds. How to utilize point clouds in the mobile AR environment is still a problem that waited to be solved. If we can show point clouds in mobile AR with good performance and visual quality, then we can replace some of the currently used models by point cloud models. Directly getting use of the easily obtained point clouds in mobile AR environment can avoid extra effort, pre-processing steps like turning them into meshes are no longer needed. Thus, in this paper, we will try to realize an interactive visualization of point clouds in the mobile AR environment using cLoD method.



(a) IKEA Place (b) Pokémon GO

Figure 1.2: Mobile Augmented Reality applications

(a) Point cloud model of a chair (b) Point cloud model of an office

Figure 1.3: Point cloud models in the Mobile AR environment

There are three mainstream Software Development Kit (SDK) to develop mobile AR applications. ARKit is an SDK provided by Apple, focusing on the IOS platform. ARCore is provided by Google, which develops AR applications on the Android platform. AR Foundation of Unity game engine provides a layer of abstraction to ARCore and ARKit, which supports both Android and IOS platforms, but the feature update will be a few months later than ARCore and ARKit [Fung, 2019]. In this thesis, the mobile

AR point cloud rendering system is developed with Unity and the ARCore SDK. Augmented Reality is a relatively complicated technique, which involves sophisticated mathematics, physics, and computer science knowledge. Fortunately, ARCore provides plenty of built-in functions which cover most of the basic operations needed by AR applications, such as motion tracking, lighting, and feature detection. Together with the Unity game engine, we can develop a good-quality point cloud renderer in the mobile AR environment.

There are a lot of challenges of rendering point clouds in mobile AR environment. Like rendering point clouds on other platforms, the biggest issue of showing point clouds in the mobile AR environment is how to deal with the massive amount of data of point cloud datasets. Second, since the mobile devices have relatively more limited CPU, GPU resources and memory, reaching ideal performance and the visual quality requirement is very challenging. In mobile AR applications, the frame rate needs to be at least 30 fps. Third, Augmented Reality applications are sensitive to visual artefacts. For example, holes between neighbouring points, visual clutter due to the inappropriate size of points, overdraw and underdraw phenomenon, all these visual artefacts will be more noticeable when moving around the point cloud models. Finally, although there are some existed cLoD-based visualization, like the cLoD point cloud viewer by Schütz et al. [2019] that supports both desktop and VR environment, and the web-based cLoD point cloud viewer by Guan [2019], the cLoD-based visualization has not been used in mobile AR environment before. Thus in this paper, we have to revise and improve the cLoD method to make it fit the mobile AR environment.

In order to get the required visual quality and performance, the number of points to be rendered will be reduced considerably. A cLoD model that has an ideal distribution over LoDs is used in this paper, with which can remove unnecessary points without sudden changes in density as present in the commonly used discrete level-of-detail approaches. During the filter, camera position, orientation and distance from the camera to point cloud model are taken into consideration as well. In order to further improve the visual quality, an adaptive point size rendering strategy is applied. What is more, some setting options are provided in the GUI of the rendering system to change the value of parameters required in the rendering. Since the value depends on specific datasets, manually adjustment of parameters are sometimes needed in order to get better rendering results.

In respect to the above, the ultimate goal of this project is to realize an interactive visualization of point clouds in the mobile AR environment using the continuous level-of-detail method.

1.2 Research Questions

This thesis aims to realize an interactive visualization of point clouds in the mobile AR environment using the continuous level-of-detail method. In order to achieve this goal, we should get a balance between the amount of computation, the utilized memory, and the final visual quality. The final rendering system should be able to process large point cloud datasets, visualize them with required frame rate and visual quality, and realize simple interactions like rotation, and scaling smoothly.

The main research question of this thesis is:

- To what extent can an interactive point cloud visualization system in the mobile AR environment using continuous level-of-detail method be created?

In order to answer the main research question, the following sub-questions are relevant:

1. How to realize and improve the continuous level-of-detail method so that the amount of computation can fit the resources of mobile devices?
2. What are the relevant parameters that are needed to be taken into account when using the cLoD method?
3. How to organize the unstructured point clouds so that the algorithm can speed up?

1.3 Thesis Outline

The rest of this thesis document is organized as follows [1]:

- Chapter 2 reviews the related work of this thesis, starting from the point cloud visualization. The concept and development of the cLoD methods are then presented. Relative concept of Augmented Reality is included as well.
- In Chapter 3, the methodology of realizing the interactive point cloud visualization in mobile AR environment is introduced. The whole working pipeline is divided into five parts, and are elaborated with figures and formulas.
- Chapter 4 introduces the tools and datasets used in the implementation, and then shows the implementation details step by step, together with screenshots of used software and part of the intermediate results.
- Chapter 5 presents some of the rendering results and analysis based on the experiments, together with a comparison between our cLoD-based point cloud visualization and mesh-based visualization.
- Chapter 6 gives the answers to the research questions, a brief discussion about the contribution, limitation, and possible applications, and future work of this thesis.

[1] Part of this thesis has been presented at the 3D GeoInfo conference [Zhang et al., 2020]

2 Related work

This chapter aims to provide the relevant knowledge of point cloud visualization as well as the Augmented Reality technology. This chapter is organized as follow: Section 2.1 introduces some representational methods of point cloud visualization; Section 2.2 focuses on the development of cLoD methods; Finally, a brief introduction and the history of Augmented Reality technology are presented in Section 2.3.

2.1 Point Cloud Visualization

Nowadays, millions of points can be gathered within a few seconds with modern scanning devices such as 3D laser scanners and LiDAR (light detection and ranging). Visualizing these huge-amount point cloud data becomes an intractable problem. The major difficulty of rendering massive point clouds is how to deal with the conflict between the limited hardware resources of GPU and CPU, and the vast amount of point clouds that sometimes exceeds memory size. Here are some typical algorithms and solutions for showing massive point clouds. Although most of them are not based on the continuous level-of-detail algorithm and deviate from this thesis's primary method, they have reference significance for further study of this thesis.

To sum up, for rendering the massive point clouds, there are two basic ideas to improve the efficiency of loading and performance of rendering. One is reducing the amount of data to be rendered considerably so that the point clouds can be visualized in real-time; another is the so-called out-of-core algorithms that partition the datasets and only process needed subsets [Livny et al., 2009], [Nebiker et al., 2010]. There are some algorithms and solutions that combine these two ideas. Some typical methods of showing massive point clouds are introduced in the following text.

Pajarola et al. [2005] develop a LoD-based out-of-core rendering system called XSplat. XSplat organizes the points by a block-based sequential multi-resolution point hierarchy. Then it uses an LoD-block paging mechanism and dynamic mapping into video-cache to realize a high-performance level of detail visualization from out-of-core. Figure 2.1 shows how does XSplat grouping nodes of significantly different LOD into the same level. XSplat can visualize massive point cloud datasets sufficiently even when it exceeds the in-core memory by only loading the relevant blocks.

Following the idea of XSplat using sequential Point Trees, Wimmer and Scheiblauer [2006] develop an algorithm called Instant Points. Instant Points uses a combination of memory-optimized sequential point trees (MOSPT) and nested octree as an out-of-core structure, which optimally explores the hardware resources. Instant Points can render point clouds with strongly varying densities and holes.

Richter and Döllner [2010] create an out-of-core rendering system that combines specialized data structures and level of detail (LoD) concepts to realize real-time rendering. Their algorithm first uses similar pre-processing steps according to QSplat [Rusinkiewicz and Levoy, 2000] to create a spatial data structure, and then determine the point representation of a point at an arbitrary level of detail based on this structure during runtime. It also enables memory management to respectively load parts of the spatial data structure from external memory. This algorithm can automatically adapt to the available hardware resources.

Goswami et al. [2013] use multi-way kd-trees to realize a novel hierarchical level of detail organization (see Figure 2.2), which makes memory management more efficient and enables users to control the LoD-tree height. As for rendering point clouds, they develop a rendering-on-budget method that loads data asynchronously. This method uses LoD geo-morphing and deferred blending techniques to realize

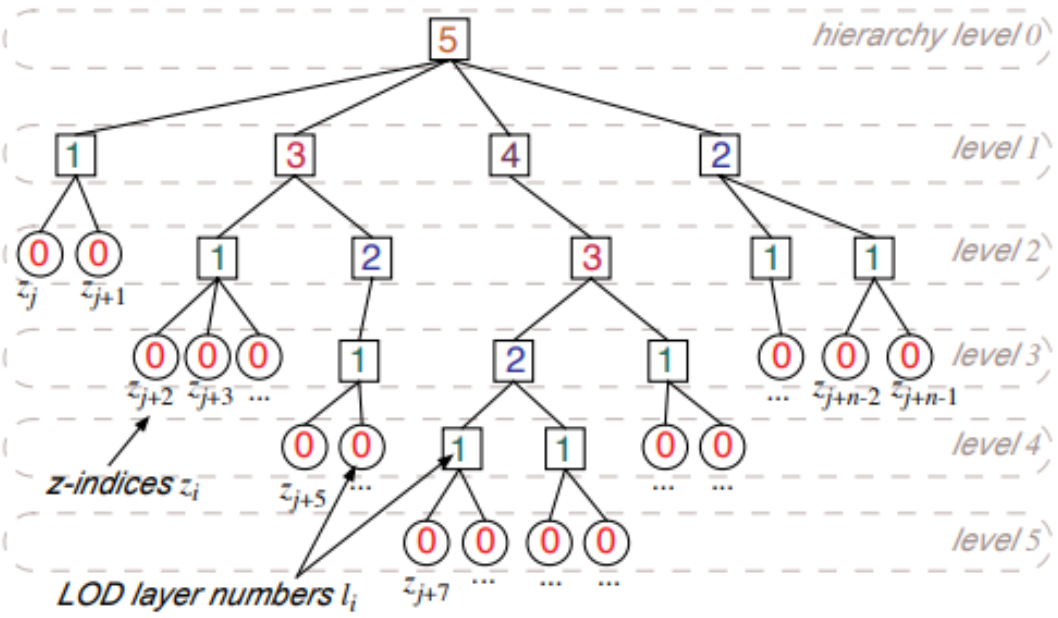


Figure 2.1: Hierarchical level- and layer-based LOD classification, and linear ordering of leaf nodes in z-index [Pajarola et al., 2005]

fully continuous, high-quality rendering. Their algorithm supports local rendering and cluster-parallel distributed rendering as well.

Martinez-Rubi et al. [2015] use Potree to reorganize the point cloud data into an multi-resolution octree data structure. Potree is a technique that reorganizes the large point cloud dataset into an multi-resolution octree data structure. The author successfully establishhed a web-based viewer of the AHN2 dataset with 640 billion points.

As the massive point cloud rendering algorithms become more and more mature, they begin to be used in many emerging areas. Tredinnick et al. [2016] realize progressive feedback point cloud rendering in the VR environment, combing with out-of-core algorithms. Discher et al. [2018b] create a web-based point cloud rendering system, which uses semantics-dependent and out-of-core rendering techniques. Discher et al. [2019] use a modular service-oriented processing pipeline together with GPU-based processing approaches and out-of-core algorithms in their geoinformation management system, which can explore, inspect and analyze massive point clouds.

2.2 cLoD Methods

In the computer graphics community, the practitioners always struggle with the conflict between the complexity of datasets, limited rendering performance, and storage capabilities. Level-of-Detail (LoD) is a discipline created to simplify the datasets and improve processing performance. Clark first rule the basic principles of the level of detail [Clark, 1976]. Level of detail techniques scale the detail of the objects according to their visual importance within the scene [Luebke et al., 2003]. The book of Luebke et al. [2003] elaborates on lots of algorithms, solutions, and questions of LoD. According to this book, the LoD methods can be mainly divided into three categories: discrete LoD, continuous LoD, and view-dependent LoD.

The traditional approach of LoD is referred to as discrete LoD. The principle of discrete LoD is assigning each object a fixed number of LoDs separately while pre-processing, then pick each object's LoD according to the object's distance or other criteria at runtime [Biljecki, 2013]. Discrete LoD is widely deployed in graphics applications because of its low complexity and handy manipulation. Many graphics

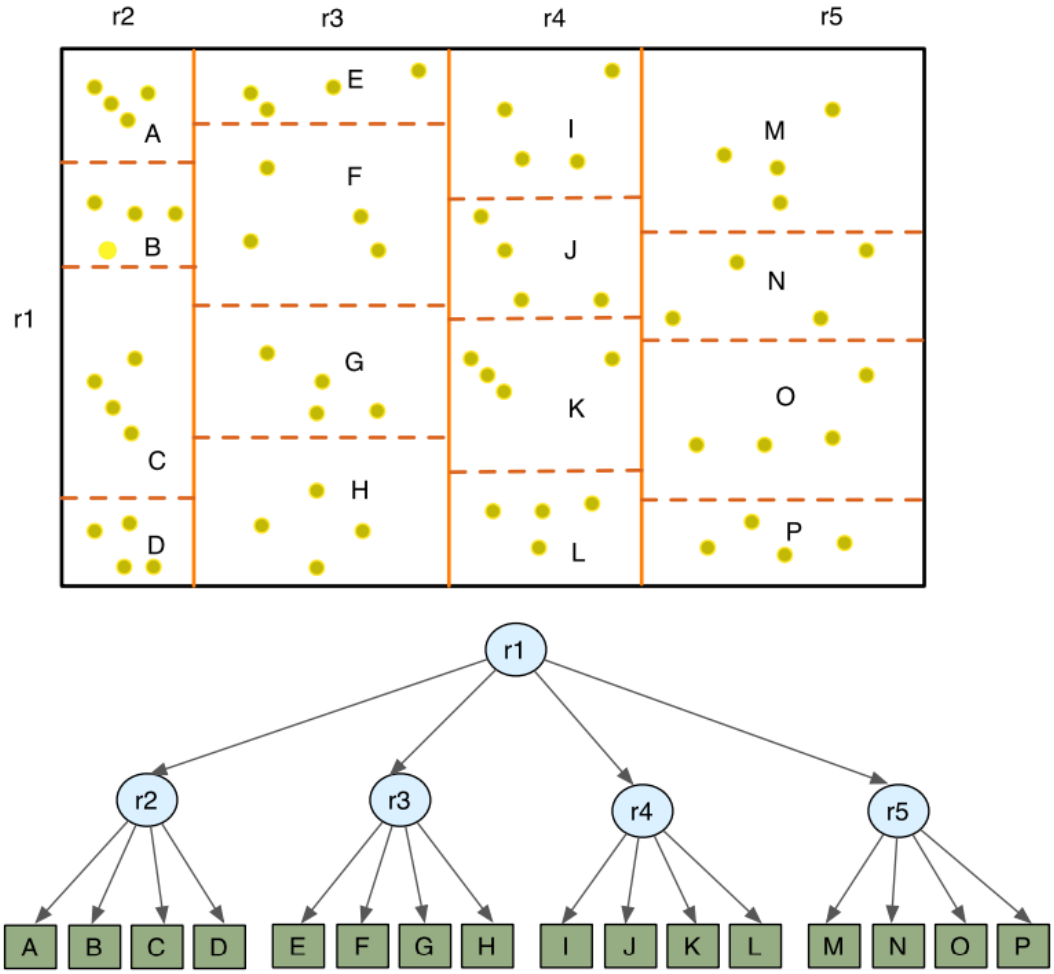
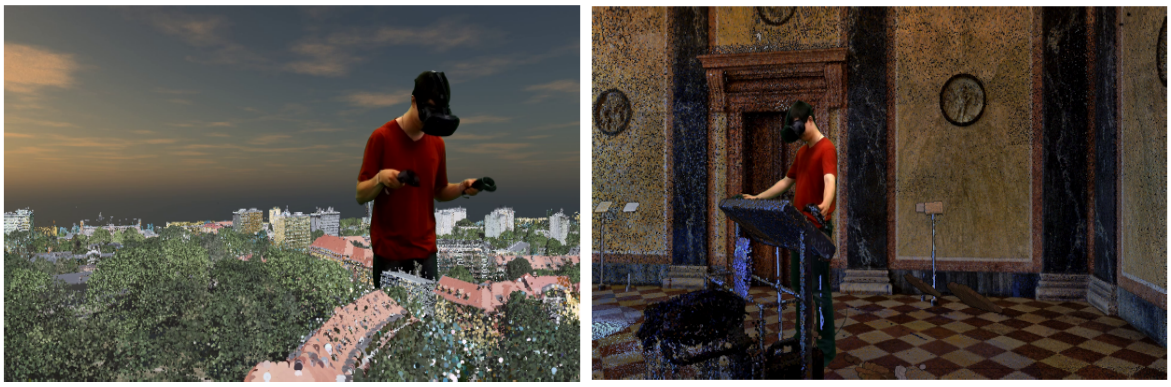


Figure 2.2: Multi-way kd-tree example for N = 4. Each of the leaf regions contains nearly the same number of points [Goswami et al., 2013]

libraries, like OSG, VRML, OpenGL, and most of the game engines, include discrete models due to their simplicity [Ripolles et al., 2009]. The major problem of discrete models is the popping, i.e. noticeable flicker, occurs when there are any changes in the pre-calculated levels-of-detail, or neighbors of different dLoD in perspective view. Although this effect can be minimized by using blending to make the transition between different LoDs smoother, this solution will still produce ghosting effects [Southern and Gain, 2003].

With the evolution of hardware, continuous LoD gradually enters our sight. Continuous LoD departs from the discrete LoD approach. Different from the discrete LoD approach that creates individual LODs at the pre-processing stage, the continuous approach creates a data structure encoding a continuous spectrum of detail. Continuous LoD approach provides better fidelity, using continuous levels of detail can hide visual artefacts that occur when switching between the discrete levels of detail and improve the efficiency of rendering [Southern and Gain, 2003].

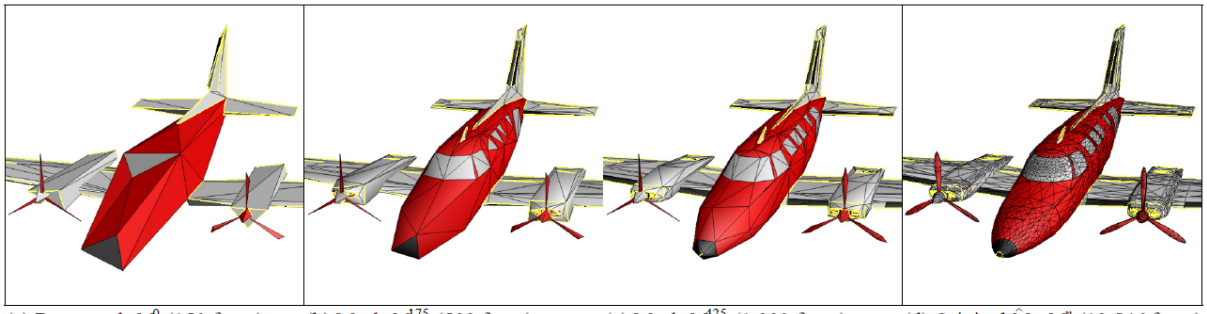
Terrain rendering is one of the most popular fields applying LoD algorithms. The first few famous continuous models are all developed for meshes. Progressive Meshes [Hoppe, 1996] is the most well-known continuous model. In the following years, Progressive Meshes has been developed in many different aspects. Sander et al. [2001] and Chen and Chuang [2006] add textures to the progressive meshes, Turchyn creates a memory-efficient algorithm named sliding window progressive meshes [Turchyn, 2007] which uses static on-GPU memory buffer to store the mesh data. As for other cLoD methods for meshes, Röttger et al. [1998] use a top-down strategy to generate cLoD for height fields dynamically. Scherzer



(a) Airborne scan of a city (b) Terrestrial indoor scan

Figure 2.3: Examples of visualizing point clouds in VR using HTC Vive as output device [Discher et al., 2018a]

Figure 2.4: An example of the LoD paradigm in computer graphics: representation of simpler versions of a complex model [Biljecki, 2013]



(a) Base mesh M^0 (150 faces) (b) Mesh M^{175} (500 faces) (c) Mesh M^{425} (1,000 faces) (d) Original $\hat{M}=M^n$ (13,546 faces)

Figure 2.5: The Progressive Mesh representation of an arbitrary mesh captures a continuous-resolution family of approximating meshes [Hoppe, 1996]

and Wimmer realize a smooth transition in image space by interpolating between adjacent discrete levels of detail [Scherzer and Wimmer, 2008].

Since the point has become one of the most promising rendering primitives, cLoD methods begin to be used to visualize points. QSplat is one of the first multi-resolution point-based rendering systems which is created by Rusinkiewicz and Levoy [2000]. QSplat first creates a point-based bounding sphere hierarchy, in which leaf nodes store the point data, and interior nodes are set to the average of the properties in the subtrees. When rendering the point clouds, QSplat traverses this hierarchy until a desired level of detail is reached. QSplat is a lightweight rendering system that can interactively and

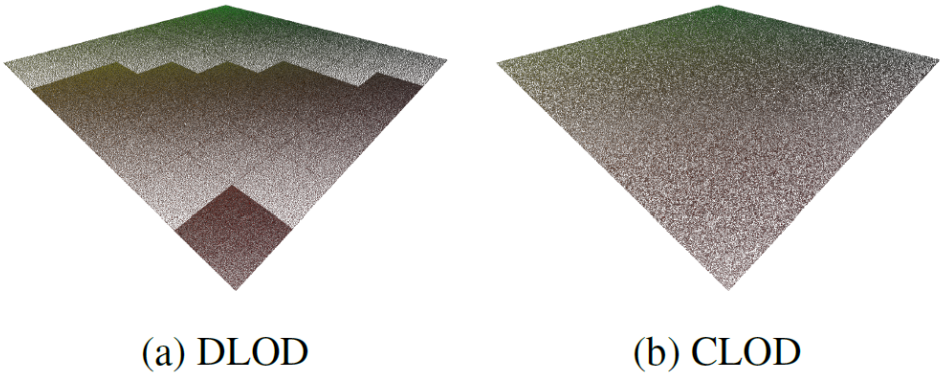


Figure 2.6: Point clouds with (a) discrete LoD and (b) continuous LoD [Schütz et al., 2019]

progressively render large point clouds with weak hardware.

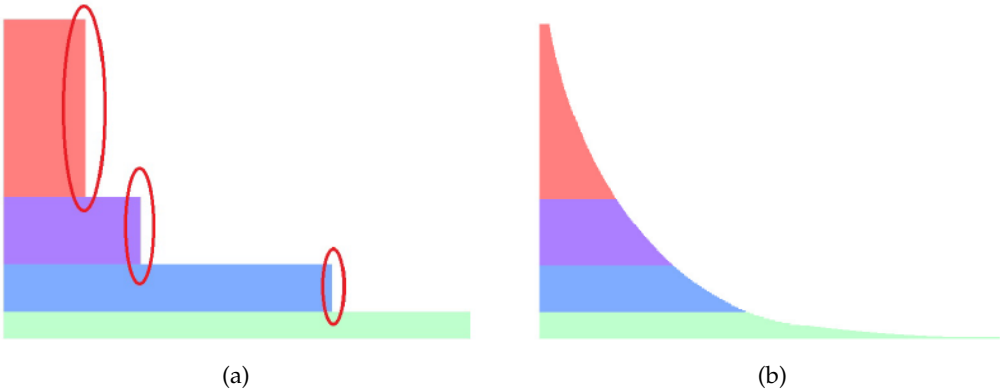


Figure 2.7: Density pyramid with (a) and without (b) density jumps [van der Maaden, 2019]

Schütz et al. [2019] develop a point rendering system using cLoD method and down-sampled vertex buffer in point-wise to speed up the rendering and improve the visual quality. They also propose a user study that shows the advantage of cLoD among discrete LoD when visualizing point clouds. van der Maaden [2019] creates a vario-scale visualization method for point clouds. This algorithm focuses on post-processing the discrete level of detail. It first trims the discrete levels and makes them continuous, Figure 2.7 shows the density pyramids with and without density jumps. Then it assigns radius to points based on the distance to the camera. Finally, 70% to 90% of the points are removed by only selecting points with an empty circle/sphere. Thus, the efficiency is improved by reducing the number of points considerably.

Apart from the related domains, Le Muzic et al. develop a tool name CellView [Le Muzic et al., 2015] to visualize large biomolecular datasets on the atomic level. They propose a level-of-detail scheme to accelerate the rendering and reduce the visual clutter. CellView enables a smooth transition between different LoDs of a molecule and the zoom-in operator from a molecule shape to its atoms.

2.3 Augmented Reality

2.3.1 Introduction of Augmented Reality

The virtual objects in AR system display information that the user cannot directly detect with his senses and the information conveyed by the virtual objects can help a user perform real-world tasks [Azuma,

1997]. That's why Augmented Reality comes into public view and becomes an interesting topic: it is what Brooks Jr [1996] calls intelligence amplification (IA), which is using the computer as a tool to make a task more accessible for a human to perform.

As Azuma et al. [2001] defines, an Augmented Reality system is a system that has the following properties: combines real and virtual objects in a real environment; runs interactively, and in real-time; aligns real and virtual objects with each other. These properties also reveal the technical requirements of an AR system. An AR system should have a display that can show the combination of the real and virtual images, a computer system that can generate interactive graphics responding to user's input in real-time, and a tracking system that can figure out the position of the users' viewpoint and make sure the virtual images are visualized at a fixed position in the real world [Billinghurst et al., 2015].

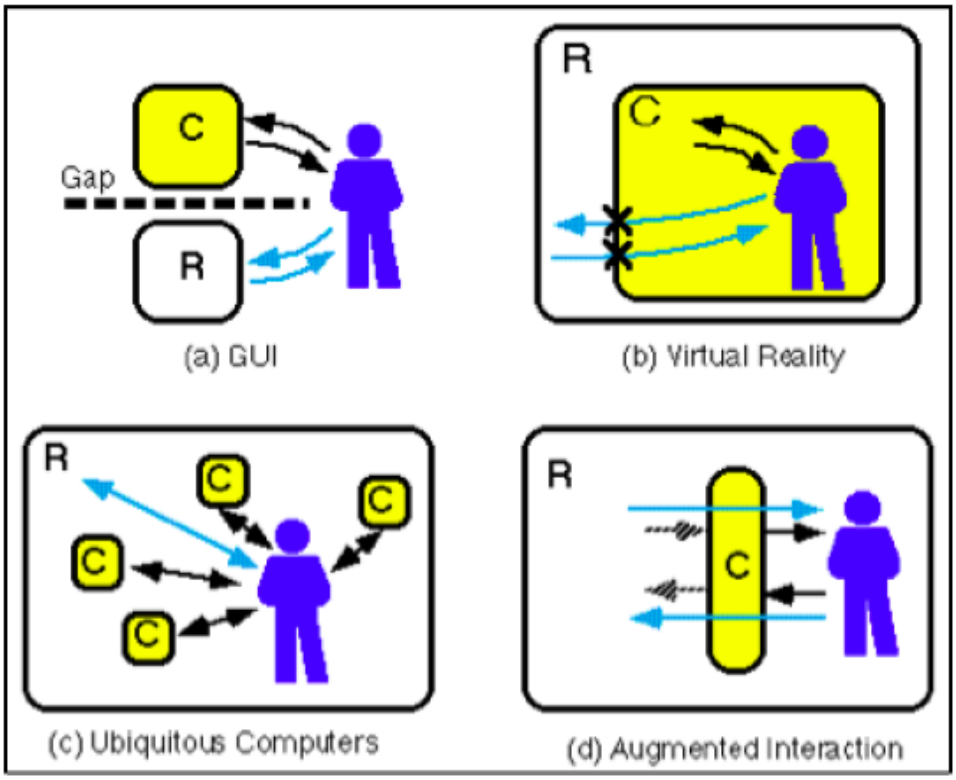


Figure 2.8: Rekimoto's comparison of Human–Computer interaction (HCI) styles (R = real world, C = computer) [Rekimoto and Nagao, 1995]

Figure 2.8 shows the difference in Human-Computer Interaction styles of traditional desktop GUI, Virtual Reality and Augmented Reality. As we can see in Figure 2.8, AR interfaces enhances the interactions with the real world. The goal of an Augmented Reality system is to enhance reality with virtual objects in a non-immersive way [Billinghurst et al., 2015]. This means that the display of an AR system can have smaller field-of-view (FOV) and use minimal displays, such as a tablet on a smart phone. Also, AR systems don't need realistic images. Minimal rendering is enough for Augmented Reality. AR displays can be divided into different categories based on what technology they use to combine virtual objects with the real world: video-based displays, optical see-through displays, projection-based displays, and eye multiplexed AR displays.

There are various input methods of AR systems. For example, the traditional input methods such as the keyboard, mouse, touch on the screen, and user interfaces (UI), and newly developed 3D and multimodal interfaces like handle, speech and gesture operations. With different input methods, AR systems can realize different types of interactions, ranging from simply viewing virtual information and interacting with virtual content, to more complex interaction like motion capture [Piumsomboon et al., 2014] and user tracking [Lee et al., 2013].

Although the requirement of display in AR system is not that restricted, the tracking system has to be accurate in order to create the illusion that the virtual objects stay at a fixed position in the real world. The human eye can detect the millimeters' mismatch between the real world and virtual content. In order to align virtual content to the real world, the relationship between the position and orientation of the user and "anchors" in the real world should be determined. The "anchors" can be divided into two different types: one is physical objects such as magnetic tracker and paper image marker, another is defined locations that are determined using GPS or inertial tracking. With the anchors, the AR system can be registered in 3D. First, the pose of the viewer with respect to the real world anchor is determined, then the system will update the pose of the viewer that is relative to a previously known pose [Billinghurst et al., 2015].

2.3.2 History of Augmented Reality

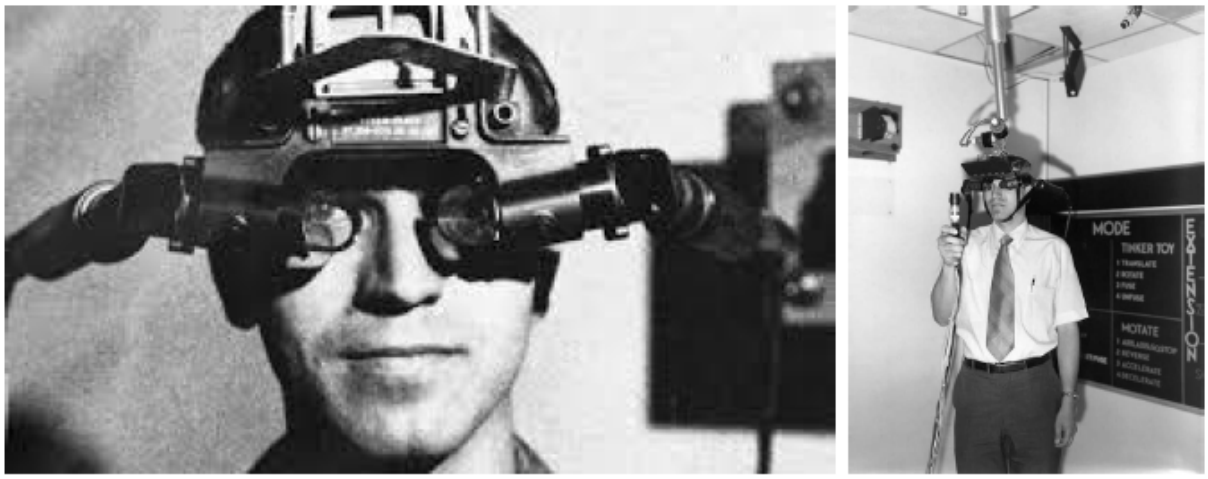


Figure 2.9: First prototype Augmented Reality system [Sutherland, 1968]

Augmented Reality has become a popular technology in recent years, but its origin can be traced back to 1960s. In Harvard University, the first prototype AR system was developed by Sutherland [1968], which combined a CRT-based optical see-through head-mounted display, a ceiling-mounted mechanical tracking system, and custom graphics hardware, as shown in Figure 2.9. At the same time, the US Air Force began to work on the Super-Cockpit program [Furness, 1969]. This program was focusing on developing new ways to present flight details for pilots in case of overloaded information (see Figure 2.10).

Later on, the National Aeronautics and Space Agency (NASA) built their HMD (head-mounted display), which was called the Virtual Visual Environment Display (VIVED). VIVED, together with a magnetic tracking system and graphics computer, became the Virtual Interface Environment Workstation (VIEW), Which can create a totally immersive environment [Fisher, 1986]. The crucial part of the VIEW was the glove interface developed by a company called VPL Research [Kalawsky, 1993]. By the late 1980s, VPL Research was able to produce the DataGlove glove and Eyephone HMD. This meant that the technologies to conduct Augmented Reality in academics and industrial researches were available by the end of 1980s.

By the end of the 20th century, some of the key technologies for an AR system were still on the way of developing, such as tracking technologies, displays, and interactions. Although the technologies were not mature at that stage, various interesting AR applications were emerged and laid the foundation for the subsequent researches. Some applications focused on enhancing collaboration and social contact, for instance, Rekimoto [1996] and Billinghurst [1999] developed an AR system that enables users at the same position to share and interact with the same virtual content. Some researchers worked on AR systems for medical use, Fuchs et al. [1998] enabled visualization of laparoscopic surgery, [Leventon,

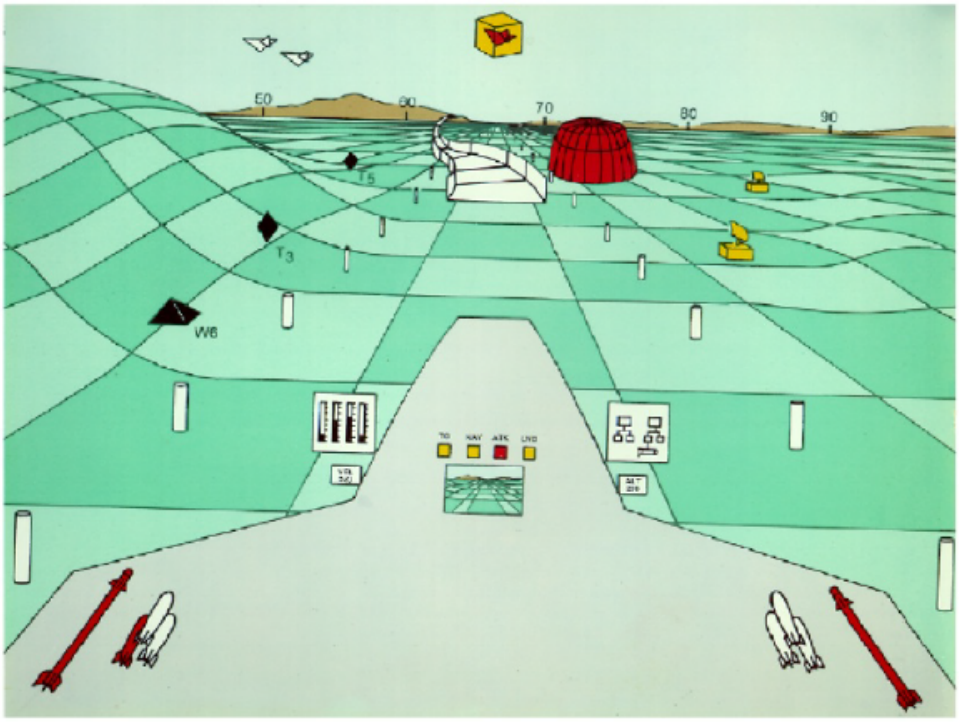


Figure 2.10: Artist's drawing of the view seen through head-mounted display of Super Cockpit system [Furness, 1969]

1997] developed image guided surgery to assist doctors, and Navab et al. [1999] enhanced X-ray over the patient's body (see Figure 2.11). Wearable computers ware emerged at the same time, similar to mobile phones, which were both natural platforms for AR systems.

After key technologies of an AR system became mature, some widely used AR systems emerged. At that time, most AR systems were developed for the research laboratory, museums, and amusement parks. For example, the ARCO project by Wojciechowski et al. [2004] focused on showing visual artefacts in the museums using AR technology (see Figure 2.12), and the Guangdong Science and Technology Center of China provided experiences of an AR adventure game [Huang et al., 2009].

One of the first AR consumer experiences that were widely available was the game on the Sony PlayStation 3 platform, *The Eye of Judgement* that was released in October 2017 [1], which was able to show AR characters on physical playing cards. After that, Augmented Reality technologies, which were only used in the research laboratory and museums before, began to come into public view.

The emergence of the Augmented Reality used by general people was mainly because of the appearance of Flash-based AR [Amin and Govilkar, 2015] and smartphone-based AR. Flash-based AR arose at the end of 2008 since Adobe's popular Flash platform added camera supports. The ARToolKit library was integrated into the Flash platform, and thus the AR library for Flash called FLARToolKit was created [Koyama, 2009]. With FLARToolKit, people first had AR experience on their web browsers, which lead to the emergence of web-based AR systems. Figure 2.14 shows a demo of Flash-based AR [2] developed by FLARToolKit. As for the smartphone-based AR, although mobile AR had appeared several years ago, the smartphones were much easier to develop and provided better platforms for the AR developers. A detailed introduction of mobile Augmented Reality will be provided in the next section.

[1] http://www.playstation.com/en-us/games/the-eye-of-judgment-ps3
[2] http://3dvrm.com/flar/

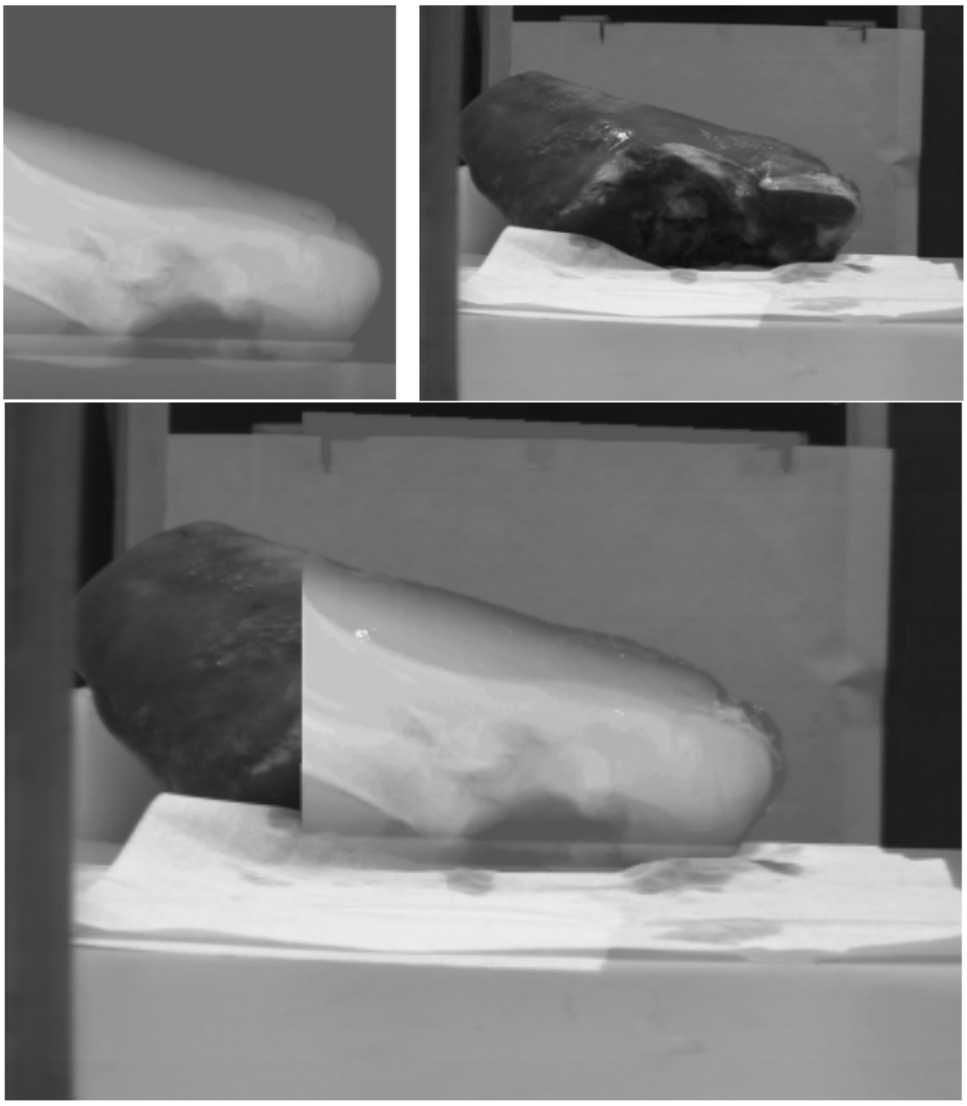


Figure 2.11: X-ray image (top left) and video image (top right) of a part of a pigs leg, overlayed optical and X-ray image of a part of a pigs leg (bottom) [Navab et al., 1999]

2.3.3 Mobile Augmented Reality

In the past few decades, the computers had been developed rapidly: their sizes become smaller and smaller, the power increases, and the endurance is getting powerful. With these developments, new mobile and wearable devices occurred. As mentioned in the last section, the wearable devices are the natural platform for AR applications. At the end of the 1990s, some mobile AR systems already appeared as the wearable devices arose.

As mentioned before, there are some basic technical requirements for an AR system: it should have a display that is able to show the combination of the real and virtual images, a computer system that can generate interactive graphics responding to user's input in real-time, and a tracking system that can figure out the position and orientation of the users' viewpoint and make sure the virtual images are visualized at a fixed position in the real world [Billinghurst et al., 2015]. For a mobile AR system, the requirements are similar.

However, there are some additional requirements for a mobile AR system: it needs efficient data storage and access technology [Höllerer and Feiner, 2004]. In order to interact and explore the surrounding

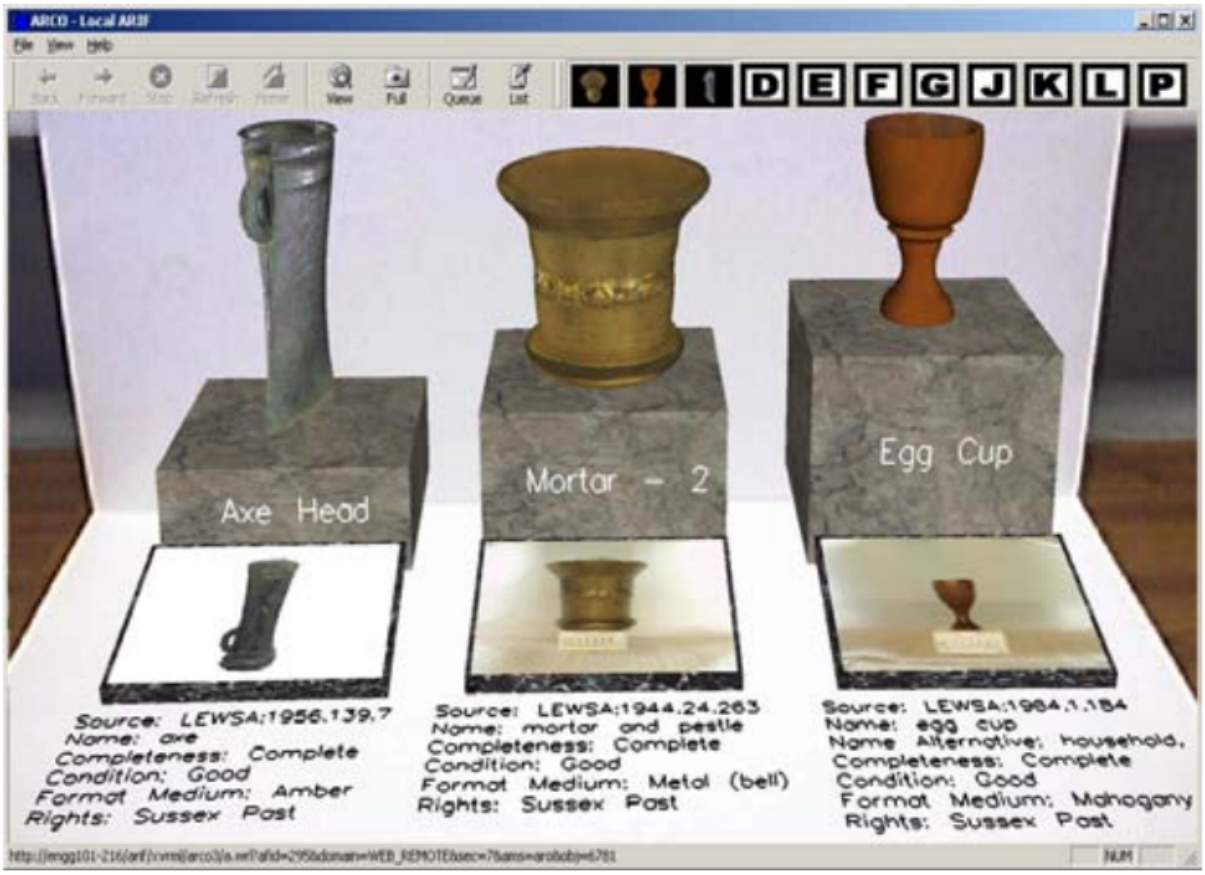


Figure 2.12: Simple exhibition built in an AR environment [Wojciechowski et al., 2004]

environment, the mobile AR system needs to detect and store information about the environment. Since the resources of the smartphones are limited, efficient data management techniques are required, in order to fetch the most relative information and avoid the information overload at the minimum cost.

Some other technical requirements of a mobile AR system are needed in specific cases. For instance, some mobile AR applications need to use web servers for the hosting of data, and such applications need to be equipped on smart devices. As for location-based mobile AR applications, they need to figure out the user's location and direction, which is realized by the gyroscope, IMU, the GPS system, or image processing. Moreover, all the mobile AR applications should be run at mobile devices with fast CPU and GPU, enough RAM capability and clear camera. The new generation of mobile devices, smartphones, are the devices that are able to meet these requirements.

In 2007, the launch of the iPhone announced the era of smartphones arrived. The smartphones are much easier to be developed than precious mobile phones. Their fast processors and powerful 3D graphics enable real-time computer vision tracking and high-quality image rendering. What's more, smartphones can access online resources as they want. The flexibility and robustness of smartphones enable them to exploit the surrounding environment and make more mobile AR applications come true.

Various mobile AR applications emerged in different fields, such as gaming and advertisement. The HIT Lab NZ created one of the first mobile AR campaigns in 2007 for advertisement [Billinghurst et al., 2015]. In late 2008, the Wikitude AR browser for Android devices was released [4], which showed virtual tags over the live video of the real world. The Esquire magazine used mobile AR to enhance their pages by virtual content. In 2009, Huizenga et al. [2009] integrated the mobile AR games called Frequency 1550.

[4] http://www.wikitude.com/

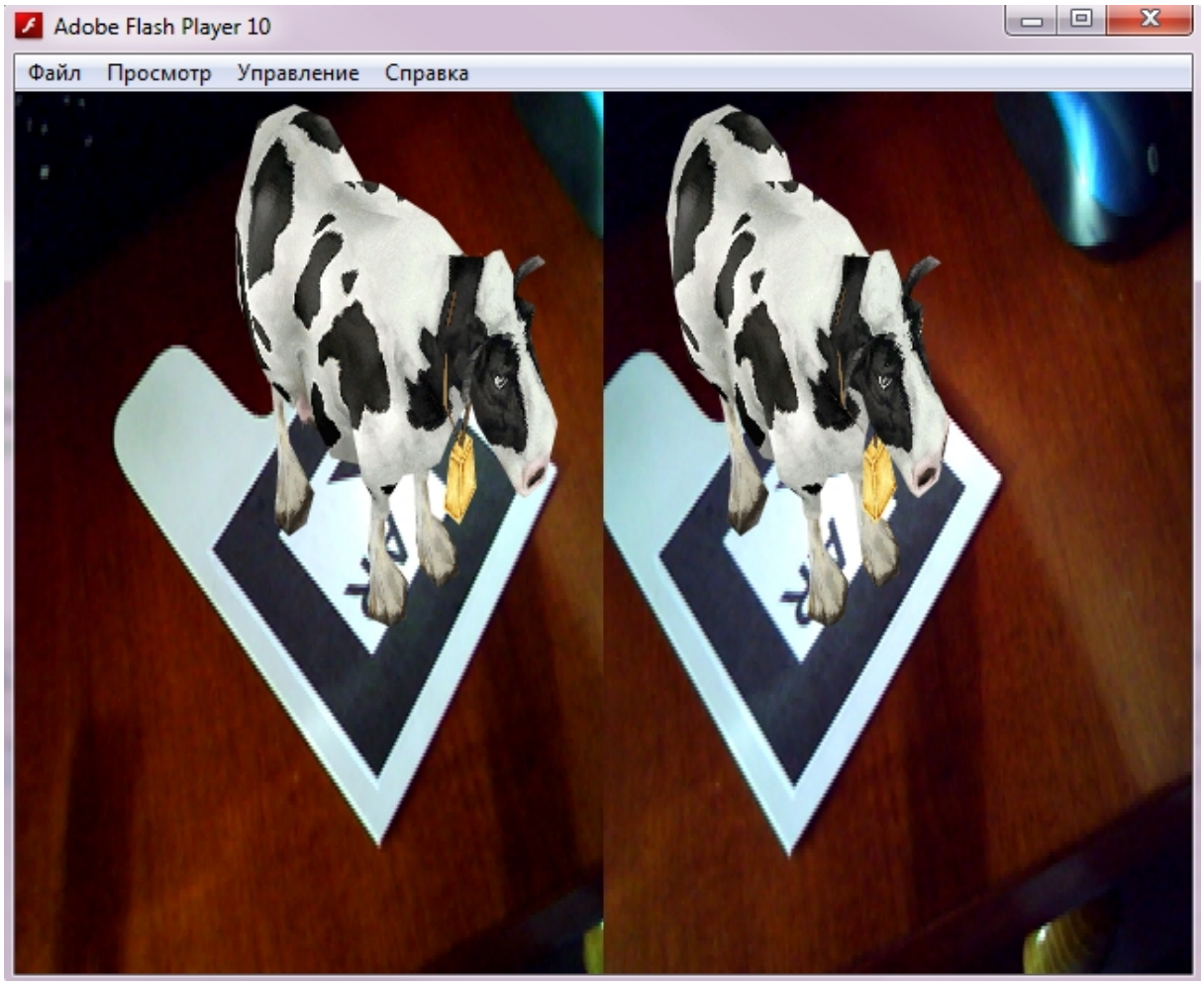


Figure 2.13: A demo of Flash AR developed by FLARToolKit (left and right images for 3D stereo)

Figure 2.14: The AR browser - Wikitude [3]

Nowadays, the mobile AR industry is still growing strongly. Mobile AR applications are widely available in multiple fields such as gaming, medicine, mobile and marketing. There are many excellent AR SDK for development on mobile devices, such as ARKit, ARCore, Vuforia, Wikitude, and Maxst. With these SDKs, developing mobile AR applications become much more feasible than before. In this thesis, we will use the ARCore SDK to develop a point cloud renderer for Android devices.

3 Methodology

This chapter will give an in-depth introduction to the research methodology that is developed to address the research questions (see Section 1.2). As shown in Figure 3.1, the methodology can be divided into six steps:

1. Point clouds input
2. Continuous level-of-detail calculation
3. Data organization
4. Selective Query
5. Adaptive point size strategy rendering
6. Evaluation

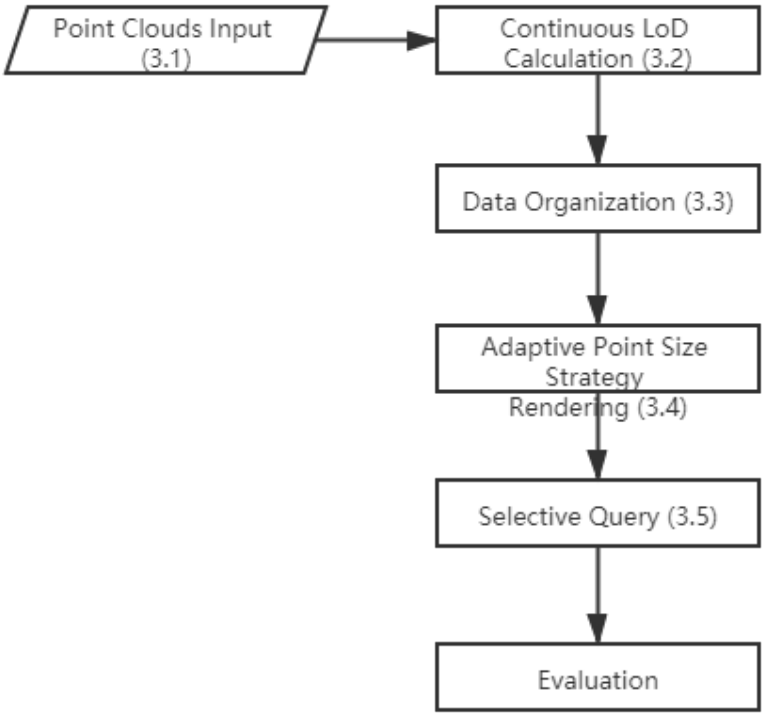


Figure 3.1: Overview of methodology

The implementation details of these steps are presented in Chapter 4. After loading point clouds from the file, the continuous level-of-detail of each point will be calculated and stored as an additional attribute of the point cloud. Then the points will be organized using a B+ tree on cLoD value in order to speed up the selective query step. All the points that are selected in the selection are stored in the vertex buffer and are finally rendered in the scene using the adaptive point size strategy. The methodology aims to efficiently and considerably reduce the number of points to be rendered in the scene so that

an interactive visualization with both nice visual quality and good performance can be realized in the end.

3.1 Point Cloud Input

Point cloud files formats can be divided into non-binary files and binary files. The traditional non-binary files, like files with '.xyz' and '.txt' extensions, store points as separated lines that list the attributes of points. Although non-binary files are flexible and easy to understand, they will cause troubles when storing millions of points: parsing the data and querying within the file are insufficient for the non-binary files.

Compared to non-binary files, binary files have some advantages while processing: they are more compact than the non-binary files; they can carry more standardized information; reading binary files are much quicker than reading non-binary files. Considering these concerns, and the purpose of facilitating the exchange of LiDAR data between different data vendors, users, and software packages, the American Society for Photogrammetry and Remote Sensing (ASPRS) created a simple binary exchange format - the LAS format. Although LAS files are developed mainly for the exchange of LiDAR data, they also support the exchange of any 3-dimensional data. As time goes by, the LAS format has become one of the standard formats for the point clouds.

In our point cloud rendering system, we take the losslessly compressed variant of LAS, the LAZ files as input. In comparison to the LAS files, the LAZ files are more efficient in file management and data transmission and have lower file system I/O. In our tests, compared to reading text files, the speed of reading LAZ files is at least two times quicker than reading ASCII files, and the size of LAZ file is much smaller than the ASCII file.

After loading point cloud from the file, we first apply some simple transformation to the point cloud model. First, the point cloud model is translated according to the coordinates of the centre of the bounding box. After that, the referenced origin is located at the centre of the point cloud. Second, the point cloud is scaled based on the size of its bounding box. The scaled and transformed point cloud will be stored as a list in the memory and wait for the following operations.

3.2 Continuous Level-of-Detail Calculation

In this thesis, a continuous level-of-detail model that is created by Van Oosterom [2019] will be applied. This cLoD model is developed based on the idea of refining ideal discrete level-of-detail. Such as that in Figure 3.2, as the number of sub-levels (blue bars) increases, their distribution will be closer and closer to the continuous function (red curves). Thus the Cumulative Distribution Function (CDF) of the ideal continuous distribution function over levels, together with a random generator is used to generate continuous levels for each point. Formulas used to calculate the continuous level for each point are presented below.

Let L be the max level, l be a level between 0 and L+1, and n be the number of dimensions. For nD point clouds, there is an ideal continuous distribution function over levels, that is:

$$f(l,n) = \frac{2^{(n-1)l}(n-1)\ln 2}{2^{(n-1)(L+1)} - 1} \tag{3.1}$$

The ideal continuous distribution function (Equation 3.1) has a Cumulative Distribution Function (CDF):

$$F(l,n) = \frac{2^{(n-1)l} - 1}{2^{(n-1)(L+1)} - 1} \tag{3.2}$$

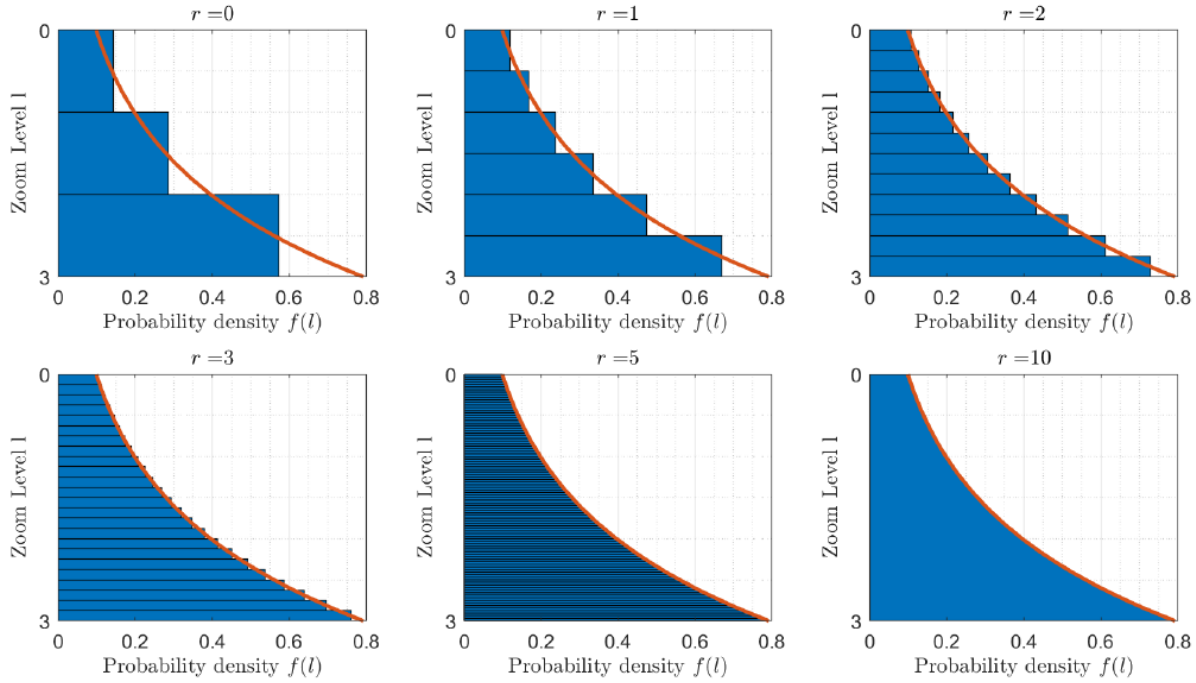


Figure 3.2: blue bars: refined discrete level-of-detail, red curve: continuous function [Van Oosterom, 2019]

In this cLoD model, a random generator U (uniform between 0 and 1) is added to assign a cLoD dimension l (value between 0 and L+1) for the next point in nD space:

$$l = F^{-1}(U) = \frac{\ln\left(2^{(n-1)(L+1)} - 1\right)U + 1}{(n-1)\ln 2} \tag{3.3}$$

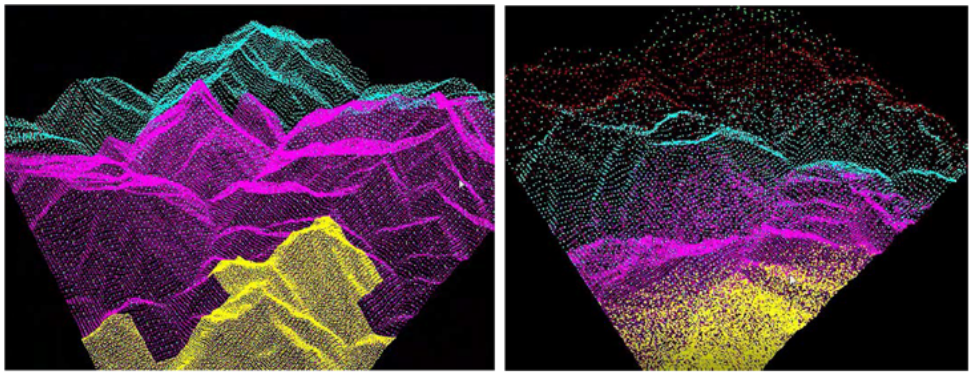


Figure 3.3: DLoD model(left) and cLoD model (right) [Guan, 2019]

In a word, Equation 3.3 is the formula we use to generate continuous level-of-detail for each point. Figure 3.3 shows what the traditional Discrete Level-of-Detail (dLoD) model and cLoD model look like. As we can see from the figure, the biggest issue of the conventional dLoD model is that there will be a sudden change of density at the splice of different levels, which will attract the attention of the user. In contrast, the transition between different levels is more smooth and gradual in the cLoD model.

To conclude, compared to the state-of-art dLoD model, this cLoD model has the following advantages: first, it has an ideal distribution over LoDs; second, it can realize smooth transition in density, and avoid density shocks as present in traditional dLoD approaches especially for perspective views; and it can keep the desired relative point density as much as possible.

3.3 Data Organization

Due to the limitation of size, power supply, and heat dissipation capacity, mobile devices always have more limited memory, CPU, and GPU resources than the desktop Personal Computer (PC) of the same era. Although there are already some existed cLoD based visualizations that are developed for PC, such as a desktop rendering system developed by Guan [2019] and a VR rendering system developed by Schütz et al. [2019]. However, such PC-based cLoD visualizations require computation in point-wise, like calculating the distance from each point to the camera or calculating the spacing between points. This kind of computation is too expensive for mobile devices and will cause an extremely low frame rate or even software crashes. In order to get a balance between the performance and the visual quality, a threshold of the cLoD is used to filter the points in each selective query. As mentioned in Section 3.5, the value of the threshold is determined by the ideal density and the distance from the center of point cloud model to the camera, which means the value of threshold will changing during the run-time as the device moves or the value of parameter changes.

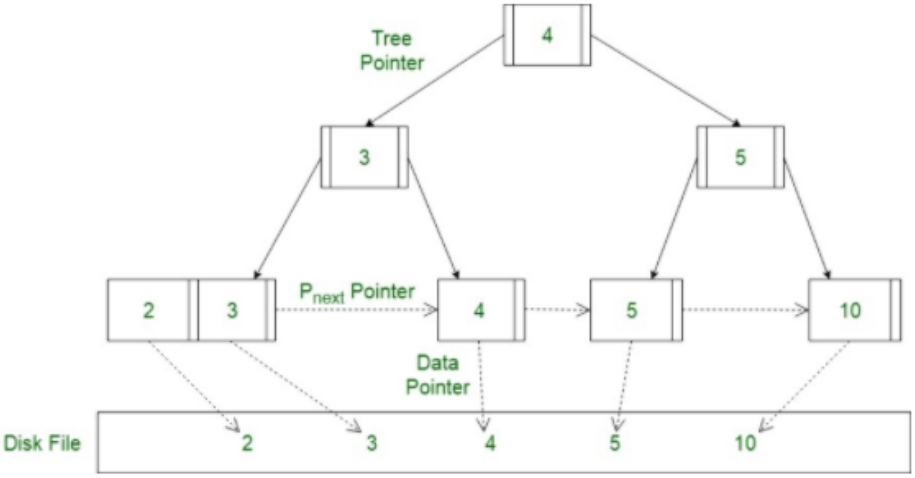


Figure 3.4: An example of B+ tree [GeeksforGeeks, 2020]

Since the selective query only focuses on the cLoD attribute, space partitioning is not needed in this case. Therefore, instead of the most common data structure that is used for nD point cloud query such as K-D Tree and Octree, B+ Tree is used to organize the points in this thesis. The B+ tree is established with the cLoD value as the key.

A B+ tree is an n-ary tree with n variable, whose value of n is often a large number (> 100). B+ Tree is a variation of the B-Tree data structure, which is mostly used for implementing dynamic multilevel indexing [Elmasri, 2008]. Similar to B-Tree, a B+ tree consists of a root, internal nodes and leaves. The root of B+ Tree may be either a leaf or a node with two or more children. Figure 3.4 shows an simple example of a B+ Tree.

The most significant difference between B-Tree and B+ Tree is that B-Tree can store the keys and records as internal as well as leaf nodes, while B+ tree only stores the records as leaf nodes and stores the keys as internal nodes. In a B+ Tree, the records are linked to each other as a linked list, and the internal nodes are called index nodes, which means a B+ Tree has two orders: one for internal nodes and the other for leaf nodes, which makes the searches with B+ Tree more accurate and faster than that with B-Tree. The average case time complexity of search operation with B+ Tree is $O(logN)$.

Using B+ Tree to organize the point data significantly improves the efficiency of query on the cLoD attribute. Before organizing data with B+ Tree, the frame rate is extremely low (<5 fps) when only processing 1 million points. After data organization, the rendering system can process at most 10 million points and visualize them at 30 fps after selection.

3.4 Adaptive Point Size

Typically, when showing point clouds, the points will be shown at the same size. However, there are some issues with this traditional rendering strategy.

Figure 3.5: Points with adaptive size(left) and with the same size(right)

First, if the size of the points is too small, then there will be apparent holes between points, which will be more noticeable when zooming in. Second, if the size of the points is too big, the neighbouring points will overlap a lot and cause a loss of information. As shown in Figure 3.5, points rendered as adaptive sizes have much better visual quality than point rendered as the same size. Therefore, in order to get better visual quality, the point size of each point is set as different values in this thesis.

Based on the perspective projection matrix, we derive a formula to calculate ideal point sizes at different depth in the viewing z-axis direction. The ideal point sizes are determined by the shape and size of the view frustum, the height of the screen, and the distance from a point to the viewpoint in viewing z-axis direction.

As shown in Figure 3.6, r in the Equation 3.4 is the right coordinate of the near clipping plane, and n is the distance from the viewpoint to the centre of the near clipping plane. The coefficient s is used to scale the points, and the predefined value of s is 5. Although visualizing points with s as five are considered to be the ideal situation, the value of s can be changed when the rendering results are not satisfying. For instance, when visualizing sparse point clouds, the value of s can be larger to make up visual artefacts like holes between points. To conclude, with this formula, points close to the camera will be bigger,

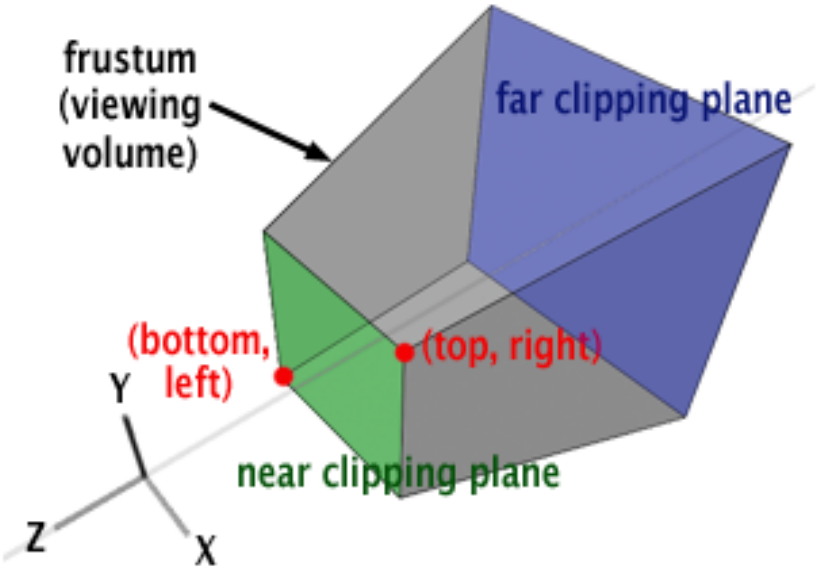


Figure 3.6: Principle of perspective projection [Scratchapixel, 2014]

while points far away from the camera will be smaller.

$$size = \frac{s * n * r * screenHeight}{z_{eye} * tan(0.5 * fov)} \tag{3.4}$$

where r = right coordinate of near clipping plane
s = coefficient to scale the points
n = near clipping plane distance
fov = field of view
$screenHeight$ = height of device screen
z_{eye} = point depth in local space

These ideal point sizes are not only used while rendering the points, but they will also be used in the next chapter to find the proper value for the parameter (Ideal Density) of the formula used in the selective query step.

3.5 Selective Query

In this thesis, the main idea of filtering the points is to reach an ideal point cloud density. Because of the limited CPU resources of mobile devices, the computation in point-wise, like calculating the distance between each point and the camera or calculating spacing between points, should be avoided. Otherwise, an extremely low frame rate or even software crashes will occur. Thus we choose to use a threshold over cLoD to filter the points.

First, the Cumulative Density (CD) at a certain level is needed. The Cumulative Density at a particular continuous level l for nD case can be computed from the Cumulative Distribution Function, using the following Equation 3.5:

$$CD(l,n) = \frac{F(l,n)N}{E^n} = \frac{(2^{(n-1)l} - 1)N}{(2^{(n-1)(L+1)} - 1)E^n} \tag{3.5}$$

where l = continuous level
N = total number of points in the dataset
n = number of dimensions
E^n = size of spatial domain in nD case

Then the Cumulative Density is wanted to be the ideal density at a certain distance, as shown in the Equation 3.6. The value of ideal density is chosen based on the ideal point sizes we computed before. After experiments, we get the recommended value of the ideal density D, which is 100,000 to 200,000 points/m^2. The value of the ideal density for different datasets differs a lot. This is because our method performs well when working with evenly distributed point clouds. However, when working with point clouds that are not evenly distributed, the value of ideal density needs to be smaller so that we can sufficiently reduce the number of points in the scene. What's more, size of the scene also affects the value of the ideal density.

Furthermore, considering the effect of distance and the logarithmic distribution of the cLoD model, a logarithm of distance from the centre of the point cloud model to the camera is set as the denominator. So that we can get higher density when the model is nearby, and lower density when the model is far away. Meanwhile, the logarithmic-distributed cLoD model can keep its distribution.

$$CD(l,n) = \frac{D}{\ln\sqrt{((x-u)^2+(y-v)^2+(z-w)^2)+1}} \tag{3.6}$$

where l = continuous level
D = ideal density
x, y, z = coordinates of point cloud center in world space
u, v, w = camera coordinates in world space

With the Equation 3.6, the particular level l with which we can reach the ideal density can be derived. By only visualizing all the points whose continuous level are less than l, we can finally reach the ideal density required by the user. All these selected points will be stored in the vertex buffer and wait to be rendered.

4 Implementation

This chapter will introduce the implementation details of the interactive visualization of the point cloud in mobile AR environment. The theory has been introduced in Chapter 3.

The chapter is organized as follows: Section 4.1 introduces the tools used to implement the methods, including the used software, language, hardware, and datasets used in the experiments. After that, Section 4.2 gives the details of each step in the implementation.

The source code and the installation package of the rendering system are available at: `https://github.com/LiyaoZhang0702/AR_PointCloud`.

4.1 Tools and Datasets Used

The rendering system has functions shown in the Table 4.1. In this section, an in-depth introduction of how the rendering system realizes the functions will be given.

#Number	Functionality
1	Loading Points
2	Rendering Points
3	Transformation of Models
4	Showing Point Count and Frame Rate
5	UI to specify settings (point size, ideal density, etc)

Table 4.1: Functionality of the rendering system

4.1.1 Software and Language

The software used to implement the proposed methodology is:

ARCore ARCore is a platform developed by Google for establishing Augmented Reality experiences on mobile devices. ARCore provides different APIs that enables mobile phones to sense the environment, understand the world and interact with information. ARCore has three core capabilities to enhance the real world seen through the camera by the virtual content, which are used in the rendering system: Motion tracking enables the mobile phone to understand and find the relationship between its position and the real world; Environmental understanding provides the mobile device with the functions of detecting the size and location of all type of surfaces in the nearby environment; and Light estimation allows the phone to estimate and simulate the lighting conditions in the current environment.

Unity Game Engine Unity is a cross-platform game engine developed by Unity Technologies. Till now, Unity has been extended to support more than 25 platforms. Among them, developing with ARCore is available on Unity as well. Developing with ARCore can be activated by using the Unity's ARCore XR Plugin package, in order to get ARCore support via Unity's multi-platform XR API. In this thesis, Unity is used to implement the framework of the rendering system.

The language and library used to write the scripts are:

C# In Unity, *GameObject*s are the fundamental objects that represent characters, props and scenery. They don't accomplish much in themselves, but they act as containers for Components, which implement the real functionality [TeamUnity, 2019]. Each *GameObject* in the scene must be attached to a script so that it can be called by Unity. Scripts are written in a particular language that Unity can understand and the user can work with. The language used in Unity is C#, an object-oriented scripting language. Thus, most of the scripts in this thesis are written in C#.

LAZ file system In this thesis, storage of the raw dataset is done with the LAZ format. A C# library called LASzip that provide compression and decompression functionality into other software is used, which is available at: `https://github.com/shintadono/laszip.net`.

High-Level Shading Language (HLSL) Shader is a type of computer program that is used for shading 3D content in computer graphic's theory. In Unity, all shaders are written in a declarative language called ShaderLab. Shaders describe properties that the Material Inspector displays contain multiple shader implementations for different graphics hardware, and configure fixed-function hardware states [TeamUnity, 2019]. In practice, Unity's shaders can be classified as programmable shaders like vertex shaders, fragment shaders, and surface shaders, these shader programs are written in HLSL language. In this thesis, HLSL is used to set and transfer vertex information, index, colour, and point size of each point in the shaders.

4.1.2 Hardware

The tests and benchmarks are carried on a Redmi K20 Pro model that has the following details: Qualcomm Snapdragon 855 processor at 2.84 GHz, 8 GB of Random Access Memory (RAM), 128 GB of SSD memory, 2340 x 1080 pixels resolution, and a triple-camera setup of a primary camera with 48 Megapixels (MP), a tele camera with eight MP, and an Ultra-wide camera with 13 MP.

4.1.3 Datasets

The test datasets used in the experiments can be mainly divided into three categories:

- **Furniture Point Clouds** Scanned point clouds of furniture, such as chair, table, and sofa.
- **Architecture Point Clouds** Point clouds of an underground garage, obtained by the NAVVIS M6 indoor mobile mapping system. Available at: `https://www.navvis.com/m6-pointclouds`
- **Terrain Point Clouds** NEON AOP Discrete Return Light Detection and Ranging (LiDAR) Point Cloud, which is an American Society for Photogrammetry and Remote Sensing (ASPRS) LASer format data product in UTM map projection. Available at: `https://data.neonscience.org/data-products/DP1.30003.001`; and the AHN2 (Actueel Hoogtebestand Nederland) dataset, which is the digital elevation map for the whole Netherlands. Available for download via the PDOK (Publieke Dienstverlening Op de Kaart).

Table 4.2 shows attributes of these test datasets.

Category	Number of Points	RGB colorized	Format
Furniture	10,000 - 50,000	Yes	TXT
Architecture	500,000 - 10,000,000	Yes	PLY
Terrain (NEON)	1,000,000 - 5,000,000	Yes	LAZ
Terrain (AHN1)	600,000 - 4,500,000	No	LAZ

Table 4.2: Attributes of test datasets

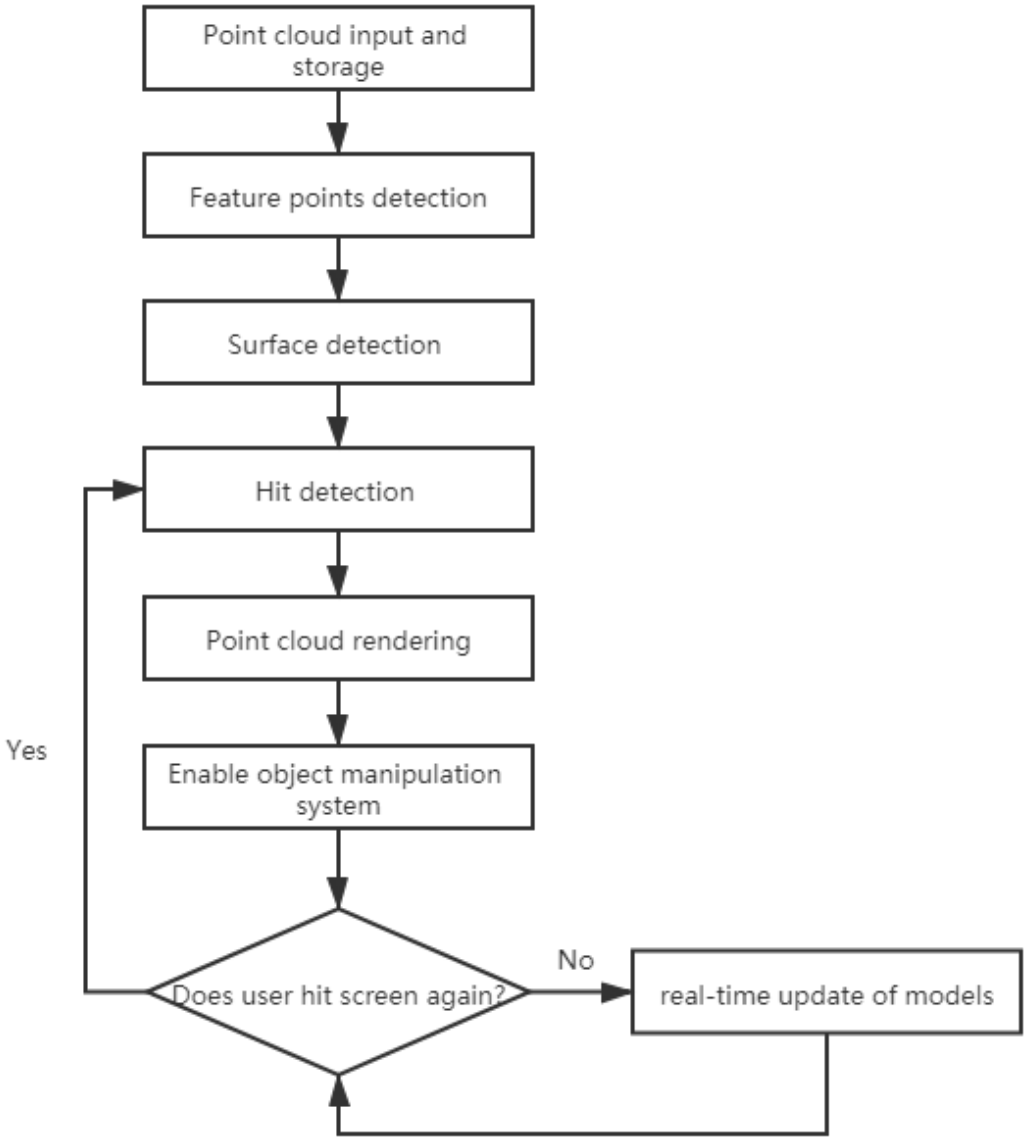


Figure 4.1: Overview of implementation workflow

4.2 Implementation

Figure 4.1 shows the overall workflow of the implementation. As shown in the figure, the workflow can be described as follow: First, the point cloud information is loaded from the files and stored as a list in the memory. Then, use ARCore's built-in functions to detect the feature points in the neighbouring environment, and detect surfaces based on the detected feature points. Next, the rendering system will see if the user hits the detected planes. If the user hits the detected planes on the screen, the hit pose (transformation matrix and rotation angle) will be recorded. The point cloud model will be put into the scene based on the hit pose. Meanwhile, ARCore's object manipulation system is loaded to enable scaling and rotation of the point cloud model. Afterwards, the rendering system will keep detecting the hit on the screen, if the user hits on the detected planes again, then the system will repeat the previous steps and put another point cloud model to the scene. If not, then the current point cloud model in the scene will keep being updated in real-time.

In the following sections, some of the implementation steps will be introduced more specifically.

4.2.1 Point Cloud Input and Storage

As mentioned in Section 3.1, the point clouds are stored as LAZ files in order to reduce the file size and speed up the loading. In the implementation, a C# library called LASzip, which enables the integration of compression and decompression functionality into other software, is used to reading the LAZ files.

The greatest strength of the LAS format is that it stores the coordinates as scaled and offset integers, which needs the data producers to consider what is the actual precision of the scanned data and to choose proper increments for storing the coordinates [Isenburg, 2013]. This can remove the unnecessary float, such as double-precision floats or 20 digit ASCII representations in the traditional non-binary formats since the 15 of the 20 digits are just scanner noise. Removing this kind of incompressible noise makes the LAS files efficient in compressing with an lossless manner.

Each LAS file has a header at the beginning of the file before the actual point data, which can be followed by any number of point records. The header includes basic metadata that can identify the LAS file and retrieve the data. A header describes the following information of the LAS file: the number of variable-length records, the offset to the start of the points, the type and size of each point, the number of points, the offsets and scale factors for the integer point coordinates, and a bounding box that describes the extends in x, y, and z of all points in the file [Isenburg, 2013]. The header is followed by the actual point data, which records each point's x, y, z coordinates, RGB colour and some other specific attributes like classification, Global Positioning System (GPS) time, and Near-infrared (NIR) channel.

The LASzip used in this thesis is a library that supports compression, decompression, and streaming of the LAS/LAZ formats. LAZ files are the losslessly compressed variants of LAS files and are used as input file format in the rendering system. Compared to the large LAS files, the compact LAZ files are more efficient in data management and data transmission and have lower file system I/O. After experiments, loading point clouds from LAZ file is at least two times faster than loading point clouds from ASCII files.

In the source code, the minimum and maximum x, y, z coordinates of the point cloud are first read from the header of the LAZ file and waiting to be used in the transformation. Then the coordinates and colour of each point are loaded. Meanwhile, the continuous level of each point is calculated in the same iteration. The coordinates, colour and cLoD of each point are stored as one list, and information of the whole point cloud is stored as a list of the list.

After loading point cloud from the file, a simple transformation is applied to the point cloud model. First, the point cloud model is translated according to the coordinates of the centre of the bounding box (see Figure 4.2). Thus the referenced origin is located at the centre of the point cloud. Second, the point cloud is scaled based on the size of its bounding box. The list that stores the transformed points is then organized using a B+ tree.

4.2.2 Feature Points & Surface Detection

Feature points in mobile AR applications can be used for motion tracking. The means of ARCore figuring out the relationship between the mobile phone and the nearby environment is call Simultaneous Localization and Mapping (SLAM). When the mobile phone moves around, ARCore will detect the visually distinct features in the images captured by the camera. These features are called feature points and are used to compute how the position of device changes. Using feature points together with the inertial measurements from the mobile phone's Inertial Measurement Unit (IMU), ARCore can estimate the position and orientation of the camera relative to the real world at all times.

Feature points are also used by ARCore to improve its understanding of the real-world environment. Besides constantly detecting feature points, ARCore looks for clusters of feature points as well. Clusters appear to lie on horizontal or vertical surfaces, like tables, ground, and walls will be recognized as detected surfaces. ARCore will make these surfaces available to the mobile AR app as detected planes.

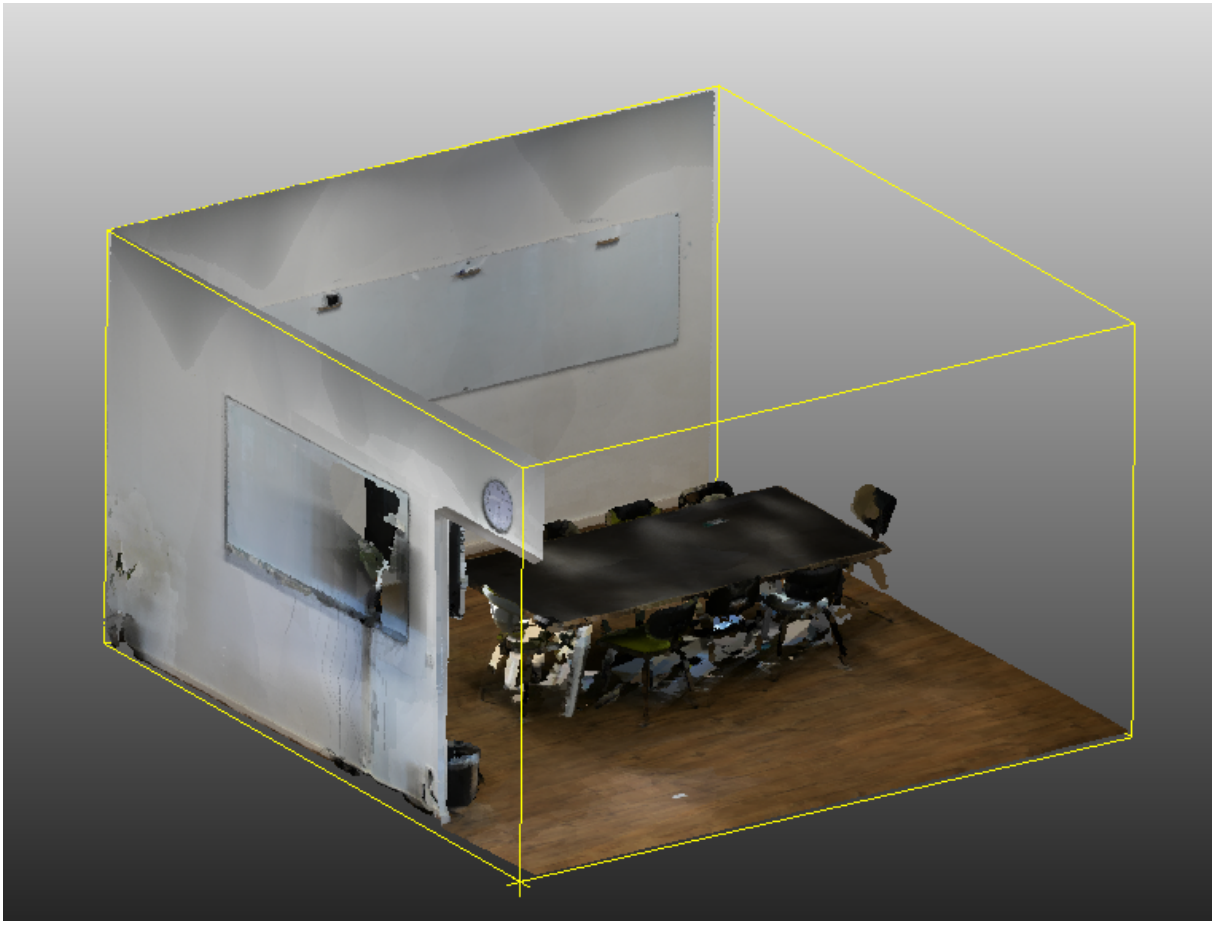


Figure 4.2: Bounding box of a point cloud

At the same time, the boundary of the detected plane is detected. The boundary information can be used to put virtual objects on the surfaces.

The drawback of ARCore detecting surfaces is that feature points are required for surface detection, so, some smooth and flat surfaces that lack texture, like white walls, may not be able to be detected correctly. Figure 4.3 shows feature points and planes detected by ARCore.

4.2.3 Hit Detection

In the rendering system, point cloud models are put in the scene by hitting on the detected planes.

In order to place 3D virtual objects on 2D planes, ARCore performs a raycast against detected planes. A raycast is a ray that gets sent out from a position in 3D or 2D space and moves in a specific direction. Figure 4.4 shows a simple example of a raycast shoot from a cube to another. When detecting the hit on the screen, the direction of the raycast is determined by the touch position on the screen and the camera position. ARCore will return all of the planes and feature points intersected by this ray together with their intersection position.

There is some restriction of the raycast test. First, the type of the hit object should be detected plane. Second, where the ray hits the detected plane should be inside the boundary of the plane. Finally, if the ray hits the back of the detected plane, then this hit is invalid. If the ray hits on the detected plane and satisfies the conditions above, then the position where the ray hits the detected plane will be recorded. The point cloud model waited to be rendered will be transformed accords to the hit pose (position and orientation).

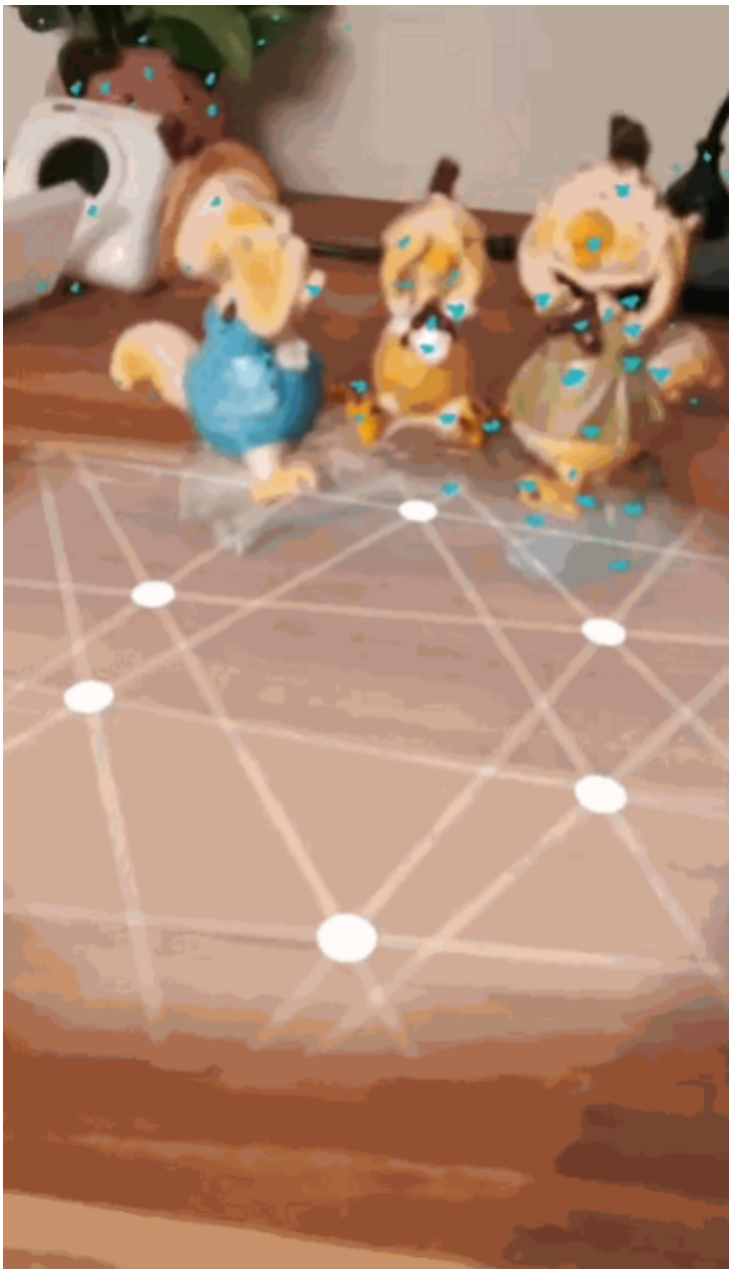


Figure 4.3: Detected feature points and planes

4.2.4 Point Cloud Rendering

As mentioned in the last section, the rendering system will put the point cloud model into the scene when the user hits the detected planes on the screen. Before rendering the point cloud model in the scene, there are few things to do:

First, all the points will be filtered accords to the continuous level of each point, using the method described in Section 3.5.

Second, the information of the selected points will be stored as a "Mesh". The Mesh class in Unity is the script interface to an object's mesh geometry. A mesh consists of arrays that contain information of a mesh, such as vertex positions, colours, normals and texture coordinates. Mesh also has some other useful properties and functions, which can assist mesh generation. Mesh can be used to present point clouds by setting the *MeshTopology* parameter of the Mesh as *MeshTopology.points*. Thus, in this case,

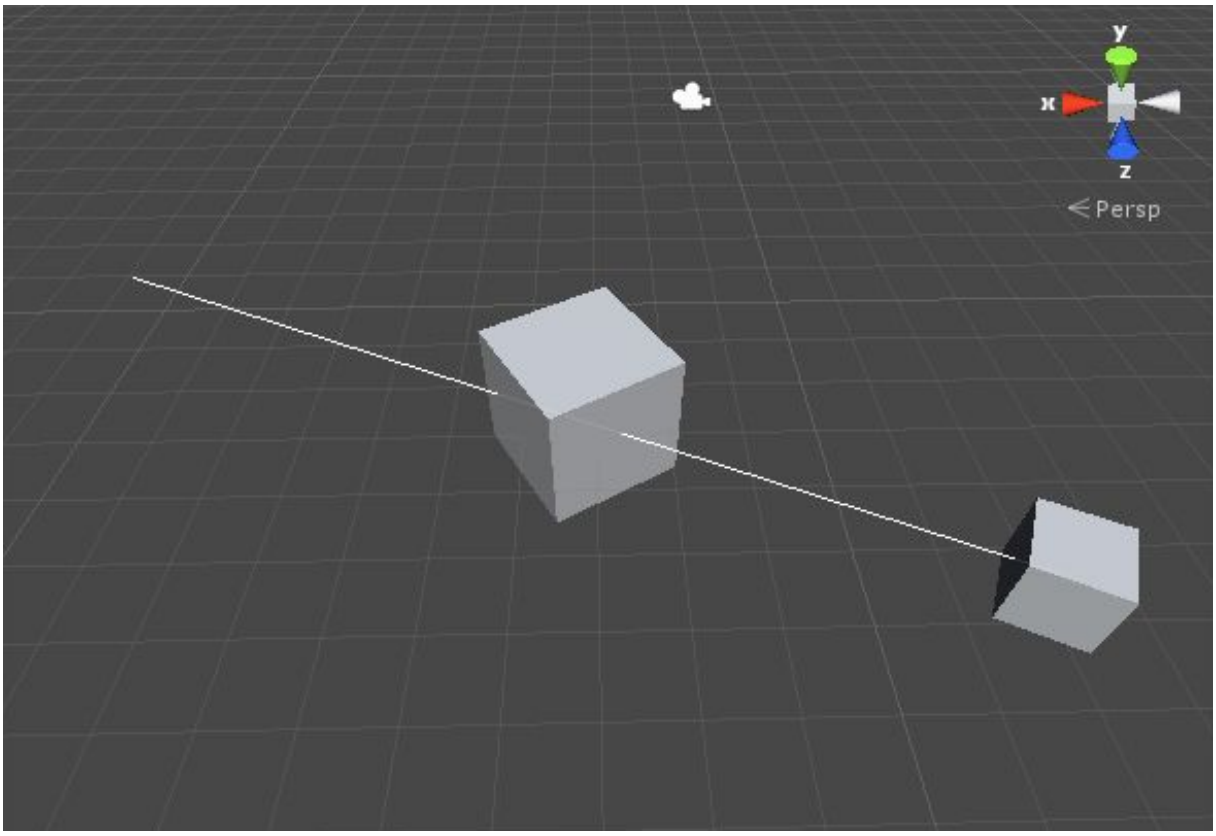


Figure 4.4: A simple example of raycast

Meshes are used as a container to store the information of point clouds, using Meshes doesn't mean that the point clouds are transformed into meshes. Using Mesh has become the mainstream method of present point clouds in Unity since it can handle a large number of points, and easily change the lighting, shadow, and the shader used to render point clouds.

Third, a new *GameObject* will be created. In Unity, *GameObject*s are the fundamental objects that act as containers for Components, which implement the real functionality. So, if we want to place a point cloud model in the scene, Mesh that store the point information and its correlative components, will be assigned as components of this *GameObject*. The correlative components of a Mesh are *MeshFilter* and *MeshRenderer*. The *MeshFilter* takes a mesh from the assets and passes it to the *MeshRenderer* for rendering the Mesh on the screen, and the *MeshRenderer* takes the geometry passed from the *MeshFilter* and renders it at the position defined by the object's Transform component [TeamUnity, 2019]. In this case, the Transform component of this *GameObject* is set accords to the hit pose detected before.

Meanwhile, anchors are created based on the hit pose. When ARCore keeps detecting feature points and planes and uses them to update their environmental understanding in real-time, the virtual objects will drift away from their original place. Thus, anchors are used to making the virtual objects appear to stay at the same position no matter how the device moves, make the users get the illusion of virtual objects placed in the real world. As shown in Figure 4.5, the point cloud model appears to stay in the same position when the viewpoint changes.

Finally, vertex positions, colours, indices are delivered to the shaders, and point sizes are computed in the shader program using the formula described in Section 3.4. All the selected points are rendered in the scene with the adaptive point size.

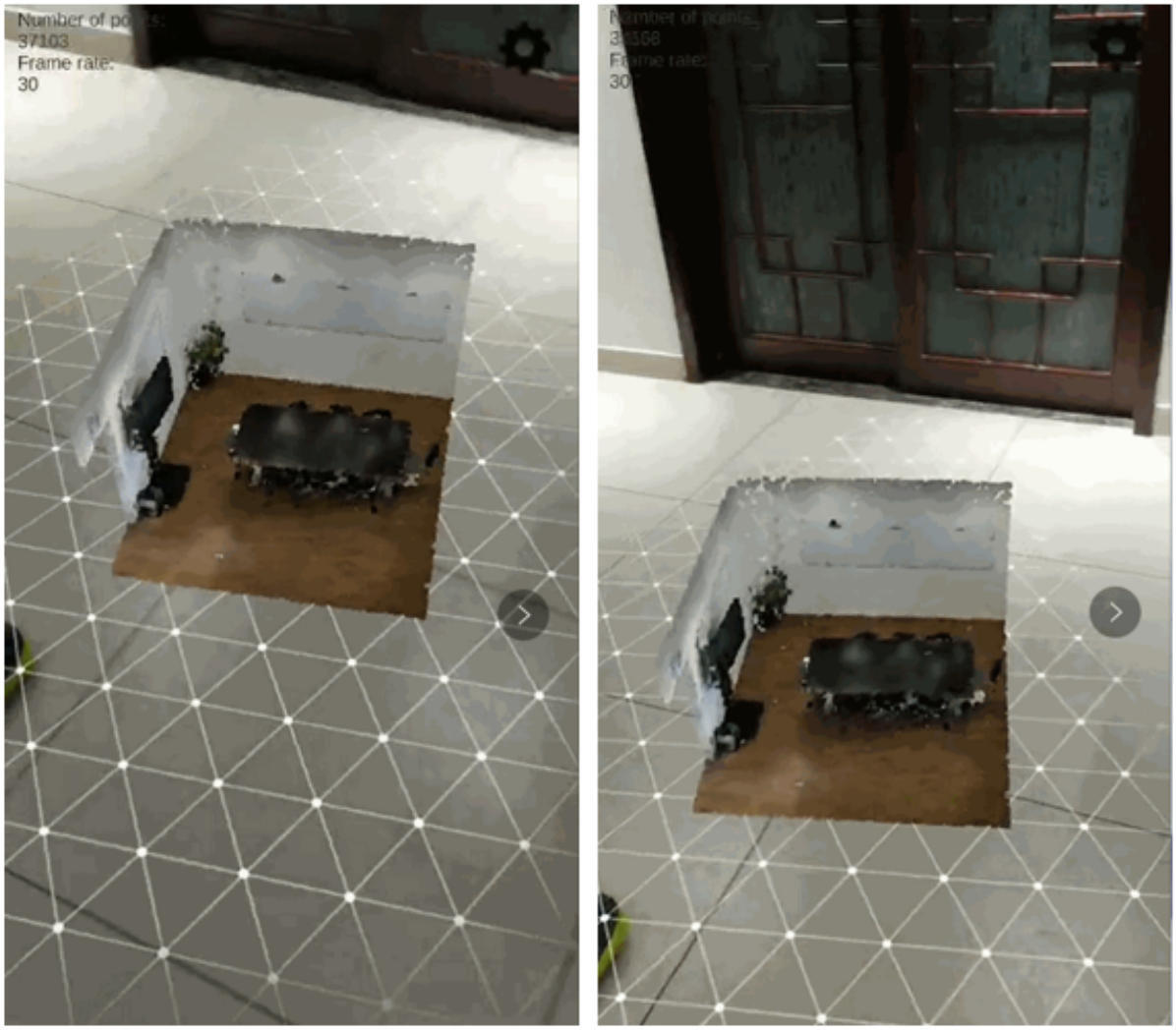


Figure 4.5: An example of how anchors work

4.2.5 Object Manipulation System

In order to scale and rotate the point cloud models, ARCore's object manipulation system is loaded. First, the *ManipulationSystem* prefab should be loaded into the scene. The *ManipulationSystem* is a prefab that performs the gesture detection and notifies all manipulators in the scene about them. This is the singleton of the ARCore's Object Manipulation system. Then the *Manipulator* prefab, which implements the required interaction corresponds to gestures of the user, is attached to the *GameObject* used to present the point cloud model. The *Manipulator* prefab is linked to the anchors as well.

Then the scripts will be revised a little bit. First, in the script of Selection Manipulator, the function *CanStartManipulationForGesture*(*TapGesturegesture*) is revised to avoid putting a new object into the scene when the user hits on an existing object. Then, the script of Scaling Manipulator is revised. Since in Section 3.5, the formula used for the selective query is affected by the area of spatial domain of the point cloud model. So, the script is changed so that more points will be rendered when zooming in, and fewer points will be rendered when zooming out. As you can see in Figure 4.6, when operating on the same model from the same distance, there are about 700,000 points when zooming in, and 150,000 points when zooming out.

Figure 4.6: An example of scaling: zooming out (left) and zooming in (right)

4.2.6 Real-time Update of Models

After putting point cloud models in the scene, the point cloud model will be updated every X frames accords to new cLoD based selection result. In other words, the points to be rendered will be updated by repeating the selective query step, and the point size will be recalculated based on the latest position of the camera in the shader program as well. X is a parameter called $UpdateFrequency$, when visualizing large point clouds, X can be set as a higher value in order to stay at sufficient frame rate.

4.2.7 User Interface

As mentioned in Section 3.5, in this thesis, the main idea of filtering the points is to reach an ideal point cloud density. Therefore, although the automatic estimation of this method performs well when working with evenly distributed point clouds, some trouble will occur when visualizing unevenly distributed point clouds. When working with point clouds that are not evenly distributed, the value of ideal density needs to be smaller so that we can sufficiently reduce the number of points in the scene. This is the major drawback of this method: the automatic estimation doesn't work well with unevenly distributed point clouds, manual operations might be required to find proper value of parameters.

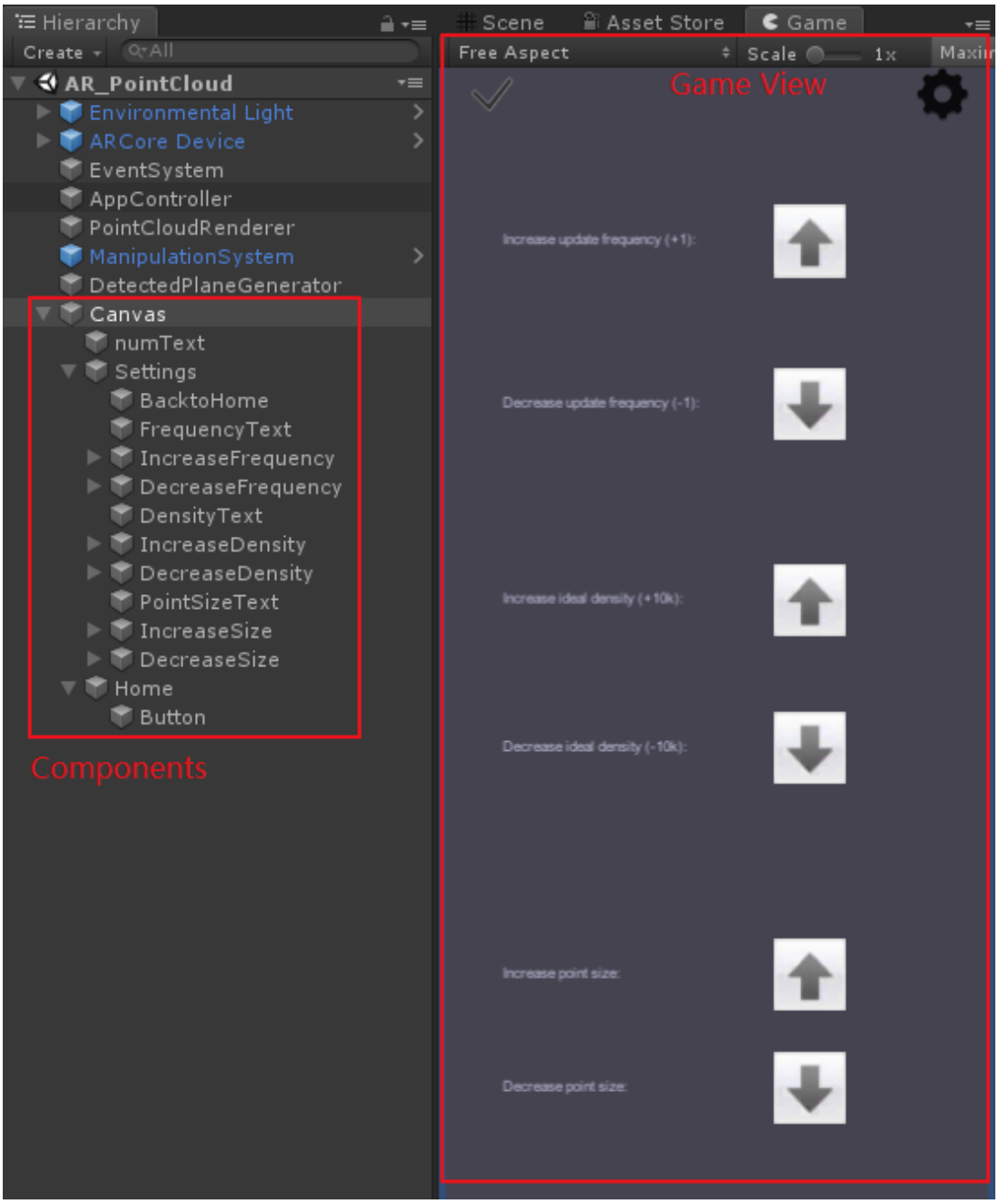


Figure 4.7: Components of UI and the Game View in Unity

So, in order to make the user change the value of parameters more conveniently, user interfaces are added to the rendering system. In Unity, UI is a UI toolkit used to develop user interfaces for games or applications, which is a *GameObject*-based UI system. To create setting options User Interface (UI), the Canvas should first be created, which is the area where all UI elements should be inside. Canvas is a *GameObject* with a Canvas component on it, and all UI elements inside the canvas are its children. After putting the Canvas into the scene, UI components can be created. As shown in Figure 4.8, in this rendering system, the UI elements can be divided into three categories: image, text, and button.

Figure 4.7 shows the components of the UI and the Game View in Unity, which are used to arrange, position and style the user interfaces. All the components are attached to a script to define their behaviour and content. For instance, the script of a text component will define the text content, and the script of a button will revise it *OnClick*() function and define what it will do when clicked.

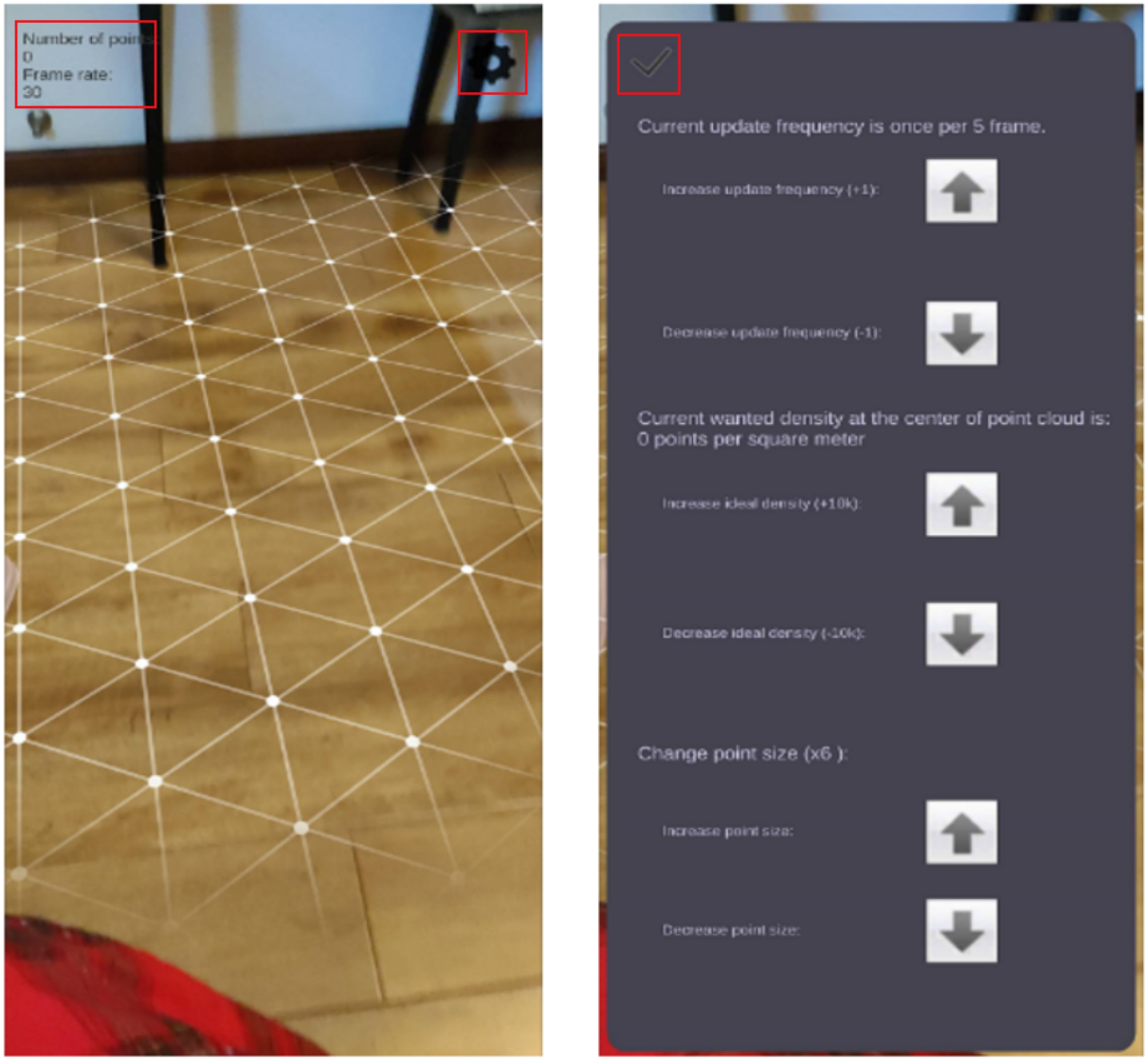


Figure 4.8: User interface of the rendering system: main page (left) and setting page (right)

Figure 4.8 shows the main page and setting page of the User Interface. In the main page, the number of points in the scene and the current frame rate are presented. The setting page can be opened by clicking the button at the upper right corner of the screen. In the setting page, we can click on the buttons and change the value of some parameters. The value of Update Frequency, Wanted Density, and the ratio of scaling point size can be changed. And user can get back to the main page by clicking the button on the upper left corner.

The Update Frequency is described in Section 4.2.6. As mentioned before, when visualizing large point cloud datasets, the update frequency can be set as a higher value, which means the times of implementing selective query is decreased in order to stay at required frame rate. Also, when visualizing small point cloud datasets, the update frequency can be smaller, thus get more fluent rendering results.

The use of Wanted Density is mentioned in Section 3.5. If the user is not satisfied with the visual quality, the value of wanted density can be larger to place more points in the scene. Otherwise, if the user thinks the number of points is too high and the performance is not that good, then the value of wanted density can be smaller in order to reduce the number of points rendered in the scene efficiently.

As mentioned in Section 3.4, the point size is set as the ideal point size at each depth in the viewing z-axis direction, i.e. the value of s is set as 5. However, when visualizing sparse point clouds, like some

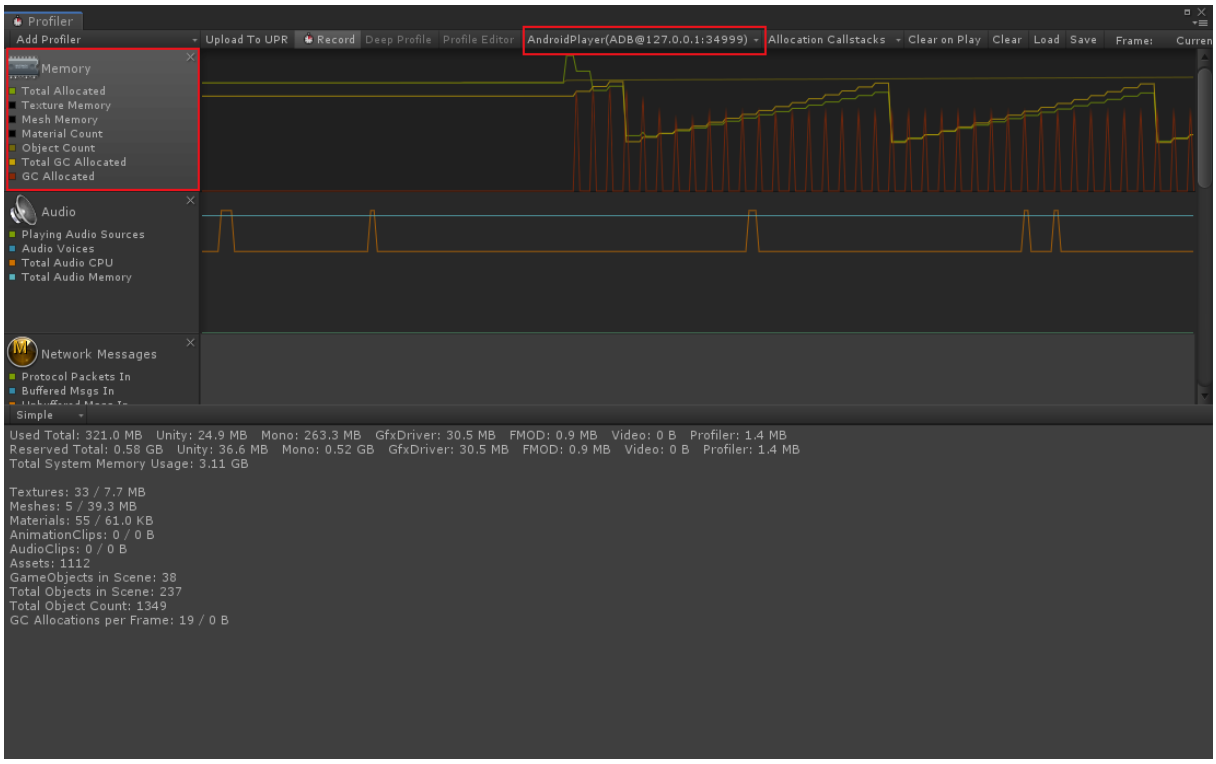


Figure 4.9: Unity profiler of the rendering app

of the airborne LiDAR point clouds, the points are better to be rendered as larger point size, so that some visual artefacts can be made up. This means the value of s should be larger. As shown in Figure 4.8, "Change point size (x6)" means that the value of s is set as 6, which means the current point size is a little bit larger than the predefined point size.

4.2.8 Evaluation

During the evaluation, several indicators are used to assess the rendering systems. Indicators that can be calculated within the scripts will be shown as text in the upper left corner in the main page of the app, such as original density and spatial domain. However, additional tools are needed when measuring the utilized memory of the app.

Unity can profile the application on its target release platform. For Android devices, there are two methods of remote profiling: via WiFi or through Android Debug Bridge (ADB). In this thesis, the test device is connected through ADB, which is realized by a series of command in the Command Prompt (CMD). After connection, enable the Development Build in the Build Settings of Unity, and then choose Build Run to launch the application on the device. The Profiler window will be opened automatically in the Unity Editor. In the drop-down menu, set Editor as AndroidProfiler(ADB@127.0.0.1:34999) and start record the profiler. The utilized memory bandwidth will be presented in the diagram below (see Figure 4.9).

5 Results & Analysis

In this chapter, some rendering results and a analysis based on the results are provided. In Section 5.1, numerous rendering results of the interactive point cloud visualization in mobile AR environment are showed. Different indicators, like frame rate, the proportion of removed points, and utilized memory, are used to assess the performance of the rendering system. Section 5.2 presents the analysis of parameters and performance.

5.1 Results

In order to examine the rendering system's capability of visualizing different types of point clouds, as mentioned in Section 4.1.3, various datasets are tested in the experiments. The test datasets can be mainly divided into three categories: furniture point clouds, architecture point clouds, and terrain point clouds. In the following sections, some of the rendering results will be presented and be evaluated with indicators.

In order to assess the results under the same criterion, all of the results are of the following condition: the distance from the centre of the point cloud model to the camera is around 1 meter, the spatial domain of the point cloud model is around 1 square meter, and the rendering results are considered to have the nice visual quality from the tester's perspective.

5.1.1 Furniture Point Clouds

Table 5.1 shows the attributes of the furniture point clouds and the value of parameters used to visualize them.

Table 5.1: Attributes of the furniture point clouds and value of parameters used to visualize them

Model	**Points**	**Original Density(pts/m^2)**	**Wanted Density(pts/m^2)**	**Update Frequency**	**Point Size**
Table	10722	10442.77	14226.93	once per 1 frame	x12
Chair	11525	11257.52	14223.77	once per 1 frame	x14
Sofa	53055	51592.20	27548.31	once per 1 frame	x5

Table 5.2 uses three indicators, which are the proportion of removed points, utilized memory, and frame rate, to assess the performance of rendering these furniture point clouds.

Table 5.2: Performance of rendering furniture point clouds

Model	**Proportion of Removed Points (%)**	**Utilized Memory (MB)**	**Frame Rate (fps)**
Table	0	19.5	30
Chair	0	20.2	30
Sofa	46.11	19.9	30

Figure 5.1 shows the rendering results of furniture point clouds at a distance of 1 meter. Figure 5.2 shows the whole look of the rendering results of furniture point clouds at a little distant place.

Figure 5.1: Rendering results of furniture point clouds at distance of one meter

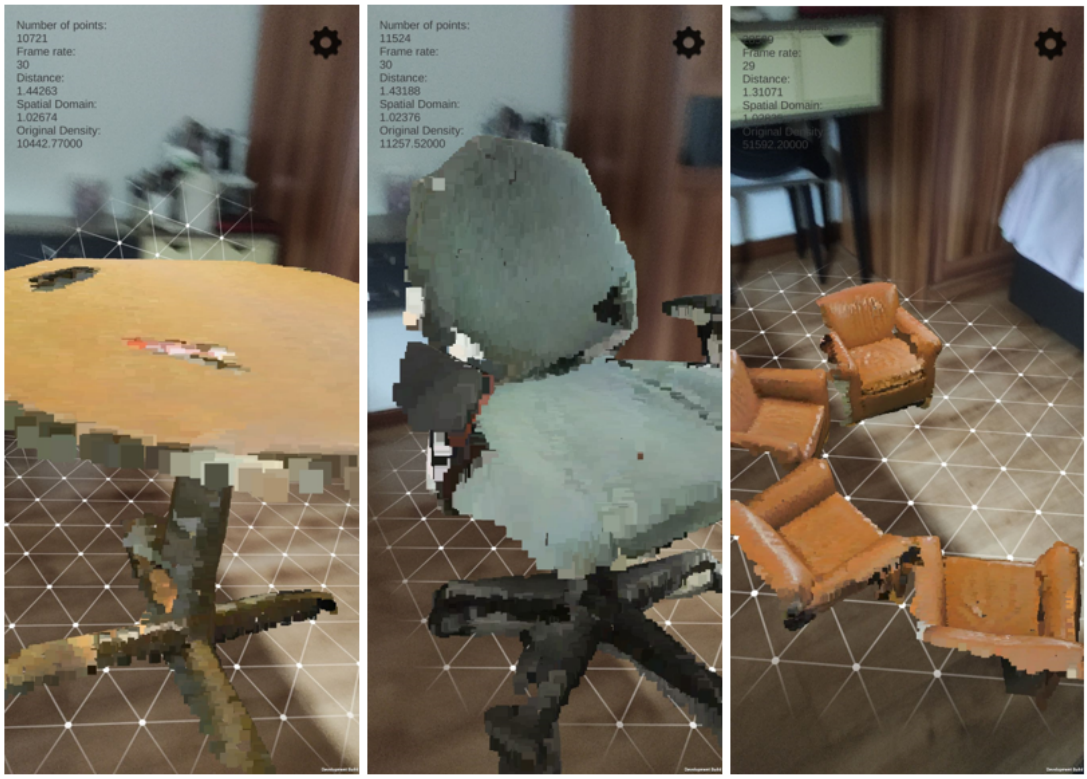


Figure 5.2: The whole look of the rendering results of furniture point clouds

5.1.2 Architecture Point Clouds

Table 5.3 shows the attributes of the architecture point clouds and the value of parameters used to visualize them.

Table 5.3: Attributes of the architecture point clouds and value of parameters used to visualize them

Model	Points	Original Density(pts/m^2)	Wanted Density(pts/m^2)	Update Frequency	Point Size
Office1	1498092	1451341.0	171499.9	once per 2 frame	x5
Garage	1002399	975602.6	146434.8	once per 2 frame	x5
Office2	8893706	8515189.3	241319.1	once per 5 frame	x5

Table 5.4 uses three indicators, which are the proportion of removed points, utilized memory, and frame rate, to assess the performance of rendering these architecture point clouds.

Table 5.4: Performance of rendering architecture point clouds

Model	Proportions of Removed Points (%)	Utilized Memory (MB)	Frame Rate (fps)
Office1	76.33	147.6	30
Garage	71.69	132.8	30
Office2	96.27	503.4	30

Figure 5.3 shows the rendering results of architecture point clouds at a distance of 1 meter.

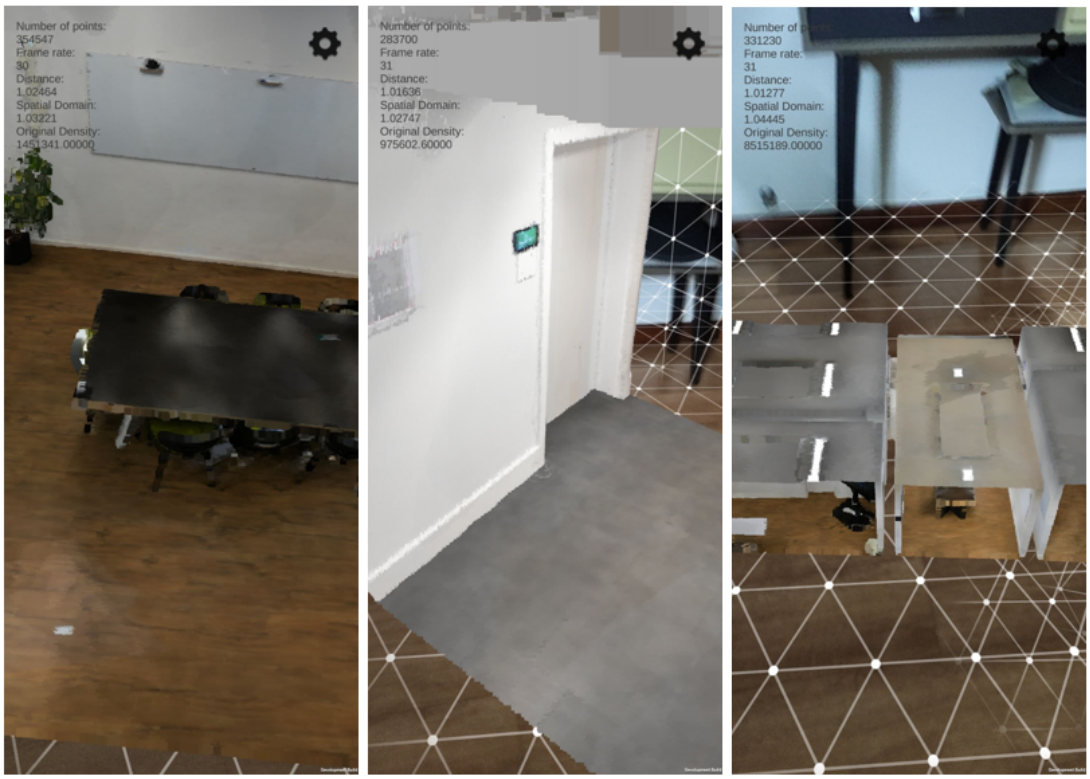


Figure 5.3: Rendering results of architecture point clouds at distance of one meter (left: Office1, center: Garage, right: Office2)

Figure 5.4 shows the whole look of the rendering results of architecture point clouds at a little distant place.

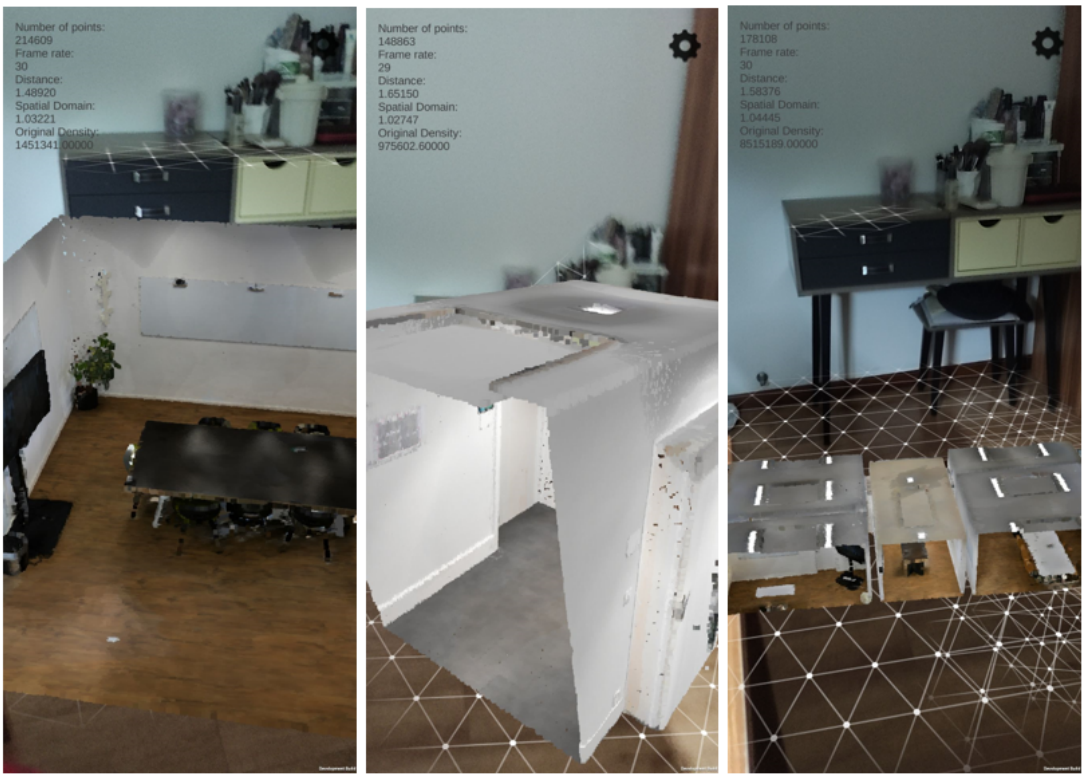


Figure 5.4: The whole look of the rendering results of architecture point clouds (left: Office1, center: Garage, right: Office2)

5.1.3 Terrain Point Clouds

Table 5.5 shows the attributes of the terrain point clouds and the value of parameters used to visualize them. Table 5.6 uses three indicators, which are proportion of removed points, utilized memory, and frame rate, to assess the performance of rendering these terrain point clouds.

Table 5.5: Attributes of the terrain point clouds and value of parameters used to visualize them

Model	Points	Original Density(pts/m^2)	Wanted Density(pts/m^2)	Update Frequency	Point Size
Terrain1	891221	865406.4	137631.7	once per 2 frame	x5
Terrain2	4309803	4167968.0	209948.6	once per 3 frame	x5
Terrain3	620503	580504.9	117289.3	once per 3 frame	x5

Table 5.6 uses three indicators, which are the proportion of removed points, utilized memory, and frame rate, to assess the performance of rendering these terrain point clouds.

Table 5.6: Performance of rendering terrain point clouds

Model	Proportions of Removed Points (%)	Utilized Memory (MB)	Frame Rate (fps)
Terrain1	71.42	132.9	30
Terrain2	95.06	315.7	30
Terrain3	47.32	127.5	30

Figure 5.5 shows the rendering results of terrain point clouds at a distance of 1 meter.

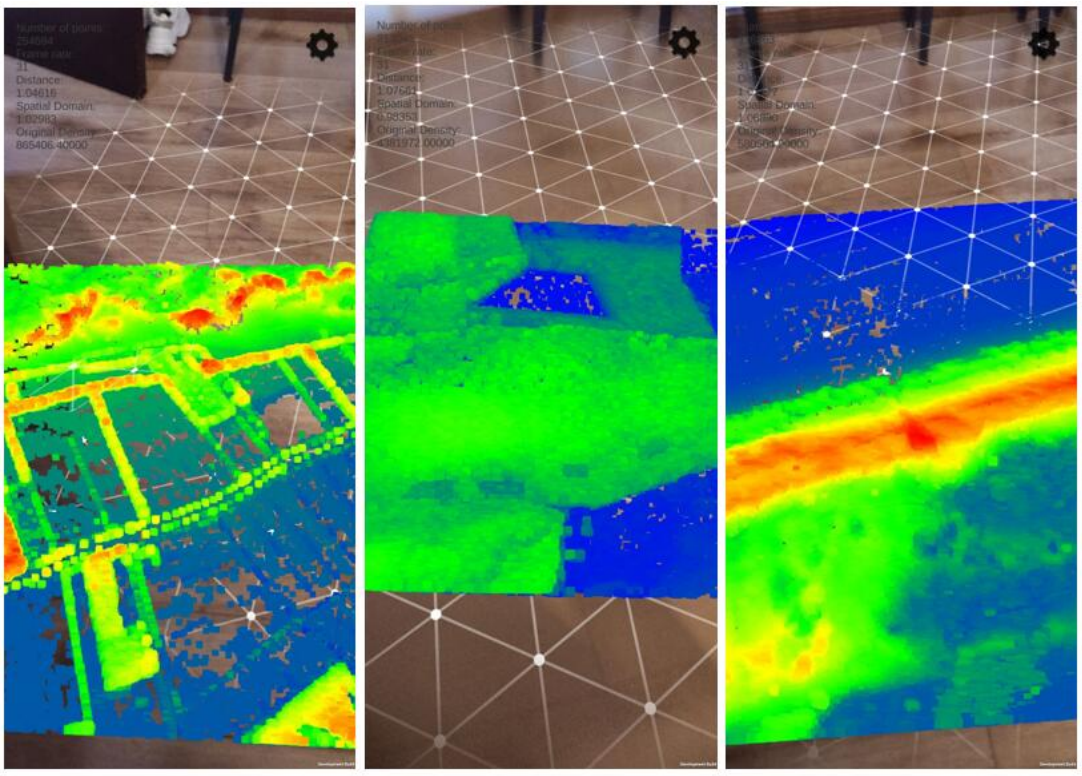


Figure 5.5: Rendering results of terrain point clouds at distance of one meter

Figure 5.5 shows the whole look of the rendering results of terrain point clouds at a little distant place.

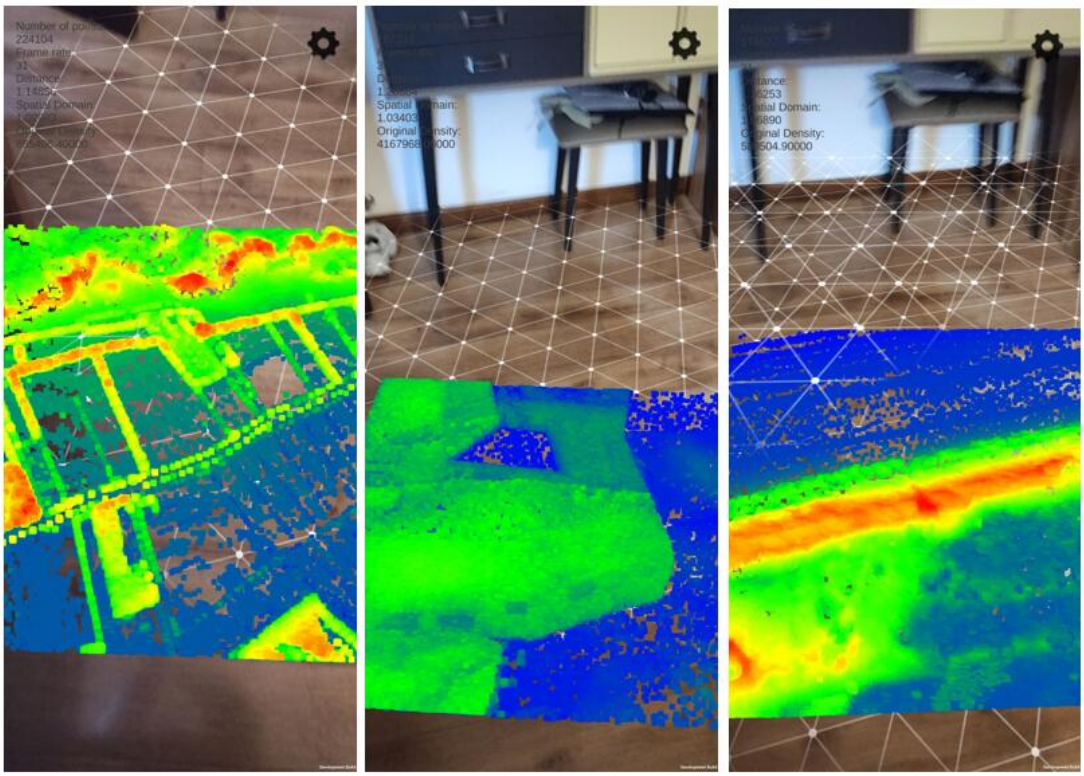


Figure 5.6: The whole look of the rendering results of terrain point clouds

5.2 Analysis

5.2.1 Parameters

As discussed in Chapter 3, various parameters are used to control the process of rendering point clouds in mobile Augmented Reality environment, such as wanted density, update frequency, and point size. In this section, the discussion about how different parameters influence the final rendering results is provided.

Wanted Density

First look at the value of wanted density as shown in Table 5.1,Table 5.3, and Table 5.5. We can see that despite the point cloud models are visualized from the same distance (about one meter) and are scaled to almost the same spatial domain (about one m^2), the value of proper wanted density still differs a lot. For example, when visualizing the sofa point cloud, the value of wanted density is 27548.31 points/m^2, and when visualizing the garage point cloud, the value of wanted density is 146434 points/m^2. What's more, when visualizing larger point clouds, the value of wanted density doesn't increase as the number of points increases.

This is because the selection of the wanted density is affected not only by the original density of the point cloud but also affected by the distribution of the point cloud and the spatial domain of the rendering result. Thus, finding the relationship between the wanted density and the attributes of the point cloud is quite challenging. Although there is a recommended value of the wanted density that is from 100,000 points/m^2 to 200,000 points/m^2, which fits most of the datasets, manual adjustment is still needed when visualizing sparse point clouds or unevenly distributed point clouds.

Update Frequency

Then let us look at the update frequency. During the experiments, even when visualizing point clouds that are close to the boundary of the rendering system (i.e. the number of points is about 10 million),

the update can be implemented once per 5 frames, and the frame rate stays at 30 fps at the same time. When visualizing smaller point cloud datasets, the update can be implemented more frequently. When the update frequency is once per 5 frames or higher, the user can hardly notice the pauses. To conclude, the rendering system can visualize point clouds that are under the boundary of the rendering system at sufficient frame rate (i.e. 30 fps), and the pauses are almost in-noticeable.

Point Size

As for the point size, we can see that the ideal point size (s as 5) performs well when visualizing the dense point clouds, like architecture point clouds and terrain point clouds(see Table 5.3 and Table 5.5). However, when visualizing the sparse furniture point clouds, the adjustment of point size is required. As shown in Table 5.1, when visualizing furniture point clouds, the s used for calculating point size is 5, 12, 14. The reason why the point size is bigger when visualizing such sparse point clouds is that the holes between points are quite noticeable in the mobile Augmented Reality environment, and visualizing them in large size can make up the visual artefacts to a certain extent.

5.2.2 Performance

Boundary

Due to the limited CPU resources, GPU resources, and memory, there is a boundary of processing and visualizing points. After experiments, the boundary of the rendering system is found: the maximum number of processing points is about 10 million points in memory / CPU, and the capability of containing points in the scene is around 5 million points in GPU. The limitation of processing points is due to the limited CPU resources, steps like loading points, hit detection, and selective query are all finished with CPU. As for the restriction of rendering points, it is because of the limited GPU resource. The calculation of point size is done by GPU.

To conclude, after using the cLoD method, the large point clouds that could not be visualized before can be rendered in the mobile Augmented Reality environment. What is more, multiple point cloud models can be put into the scene only if the total number of points finally rendered is less than 5 million points.

Frame Rate

As shown in Table 5.2, Table 5.4, and Table 5.6, we can see that no matter how many points are processed by the rendering system, the frame rate stays at 30 fps stably, which is the required frame rate of mobile Augmented Reality Applications. As mentioned in Section 4.2.6, this can be realized by changing the update frequency.

Proportion of Reduced Points

Based on the results presented in Section 5.1 and results of additional experiments, a figure of relationship between the number of points and the proportion of reduced points is created (see Figure 5.7).

Since the point clouds are scaled to relatively small area (one m^2), the proportion of reduced points of point clouds with more than 3 million points is relatively high, which is more than 90%. As for the smaller point clouds (with points more than 1 million), the proportion of reduced points is beyond 70% as well.

The proportion of reduced points is determined by various factors, such as the spatial domain, distribution, and the original density of the point cloud. Thus we cannot find a precise regulation of the proportion of reduced points. Despite that, based on the current results we can still say that, for most of the indoor applications that usually have small field of vision, the number of points can be reduced significantly when visualizing large point clouds.

Utilized Memory

Based on the results presented in Section 5.1 and the results of additional experiments, a figure of relationship between the number of points and the utilized memory is created.

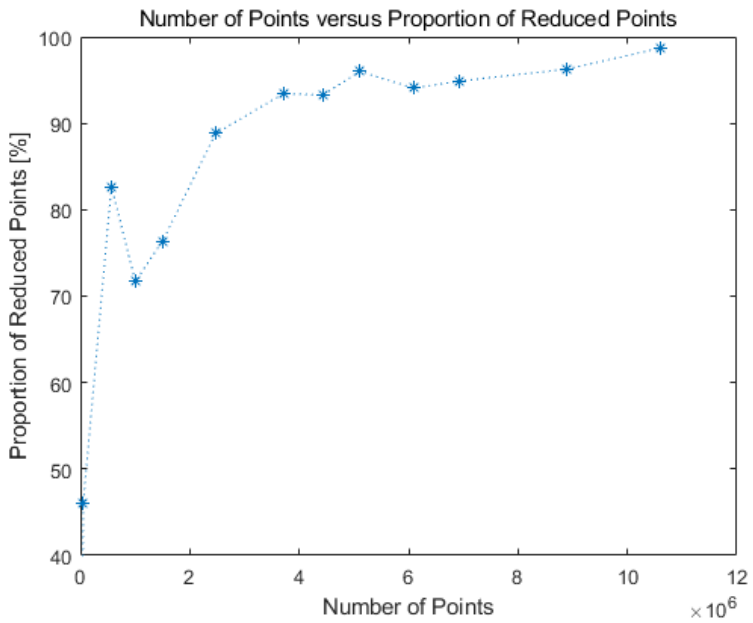


Figure 5.7: Number of points versus proportion of reduced points

As shown in Figure 5.8, we can see that the utilized memory increases when the number of points to be processed increases. When visualizing large point clouds (with more than 5 million points), the consumption of memory is quite high, which is more than 400 MB. When visualizing the large point clouds (with 10617415 points) that is close to the boundary of the rendering system, the utilized memory comes to more than 1200 MB. The consumption of memory is too high for a simple point cloud renderer.

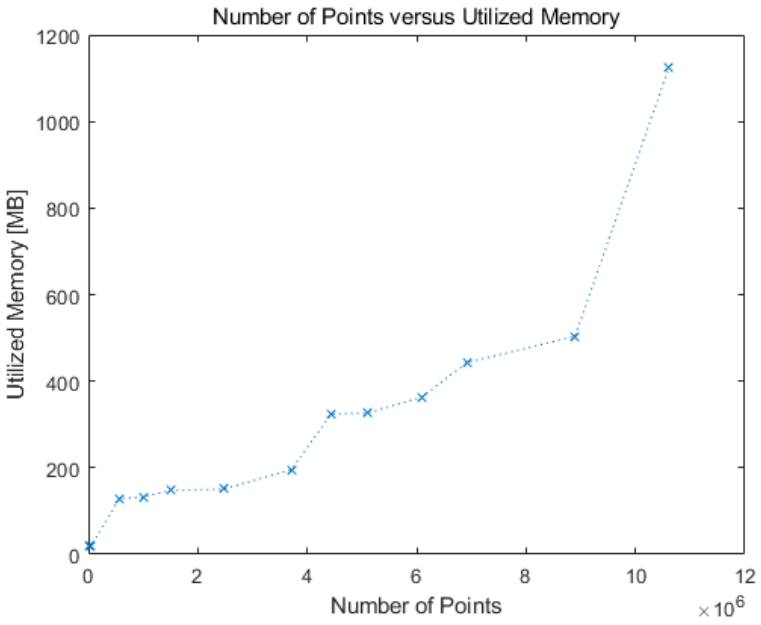


Figure 5.8: Number of points versus utilized memory

5.3 Comparison

The greatest strength of our method is that we can directly get use of the easily-obtained point clouds without complicated pre-processing steps. Therefore, we think that our method has the potential of replacing part of the mesh-based models in some applications. In order to prove this thought, we make a comparison between our cLoD-based point cloud visualization and the mesh-based visualization. The reconstructed mesh models are created by *MeshLab*, using one of the most commonly used reconstruction algorithm, the Screened Poisson Reconstruction Algorithm [Kazhdan and Hoppe, 2013].

5.3.1 Pre-processing Steps

In this section, we'll compare the pre-processing steps of our cLoD-based point cloud visualization and that of the mesh-based visualization. We'll first see the workflow of pre-processing steps of each method, and then assess them by the resource consumption during pre-processing.

Workflow of Pre-processing

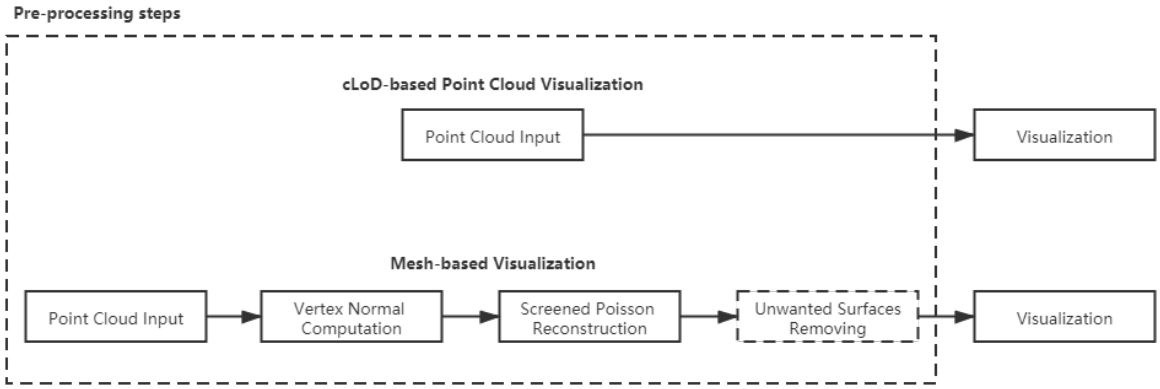
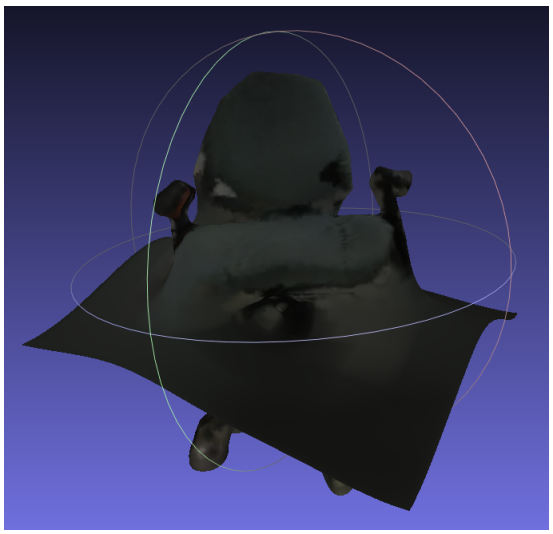
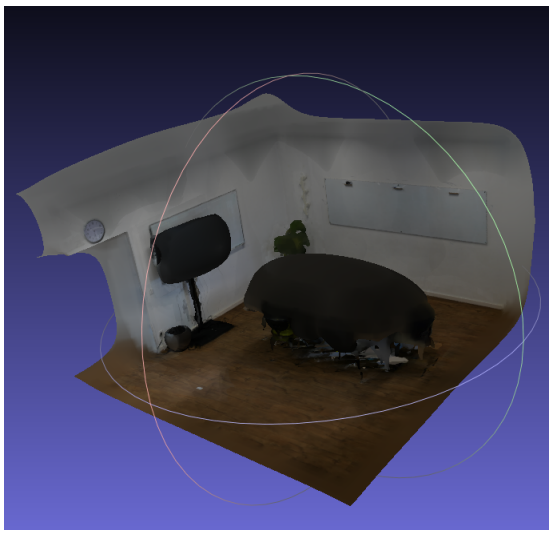


Figure 5.9: Pre-processing steps of cLoD-based point cloud visualization and mesh-based visualization



(a) Reconstructed mesh model of chair point cloud



(b) Reconstructed mesh model of office point cloud

Figure 5.10: Strange reconstruction results

Figure 5.9 shows the major pre-processing steps of the cLoD-based point cloud visualization and the mesh-based visualization. As we can see from this figure, the only needed pre-processing step of our method is storing the file to the mobile phone, and reading the point cloud from the file.

In comparison, building reconstructed mesh models needs to implement the following pre-processing steps on the computer: inputting the point cloud from the file, computing normal of each point, and implementing the Screened Poisson Reconstruction algorithm. What's more, since the Screen Poisson Reconstruction algorithm creates watertight surfaces in the end, some strange results will appear when dealing with point clouds of unenclosed objects (see Figure 5.10). Thus sometimes we need to remove the unwanted surfaces manually.

To conclude, the overall workflow of pre-processing steps of mesh-based visualization is way more complex than that of our cLoD-based visualization.

Processing Time

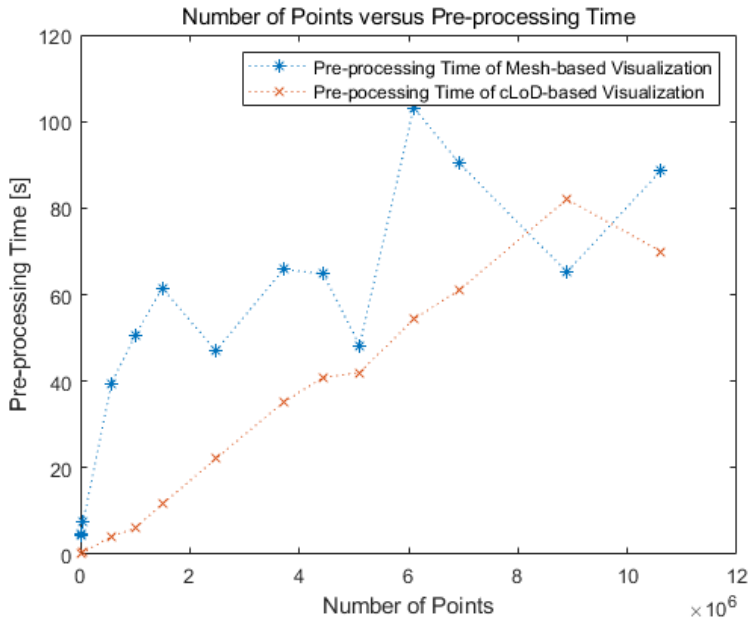


Figure 5.11: Number of points versus pre-processing time of two methods

Figure 5.11 shows the relationship between the number of points and the pre-processing time of the two methods. The pre-processing time of mesh-based visualization is the sum of the implementing time of the pre-processing steps finished by the computer, including the time to import point cloud, time to calculate the normals, and the time to implement the Screened Poisson algorithm. As we can see from this figure, although the pre-processing steps of the mesh-based visualization are implemented on the computer that has much better performance than mobile phones, for most cases the pre-processing time of our cLoD-based visualization is still less than the pre-processing time of mesh-based visualization.

Utilized Memory

Figure 5.12 shows the relationship between the number of points and the utilized memory of pre-processing steps of the two methods. As we can see from this figure, the utilized memory of our cLoD-based visualization pre-processing is from 0 to 1200 MB, which is much less than the utilized memory of mesh-based visualization pre-processing (from 1600 to 4600 MB).

5.3.2 Visualization

In this section, some rending results of both methods will be presented. By comparing the final visual quality and utilized memory, we will figure out if our cLoD-based point cloud visualization has the

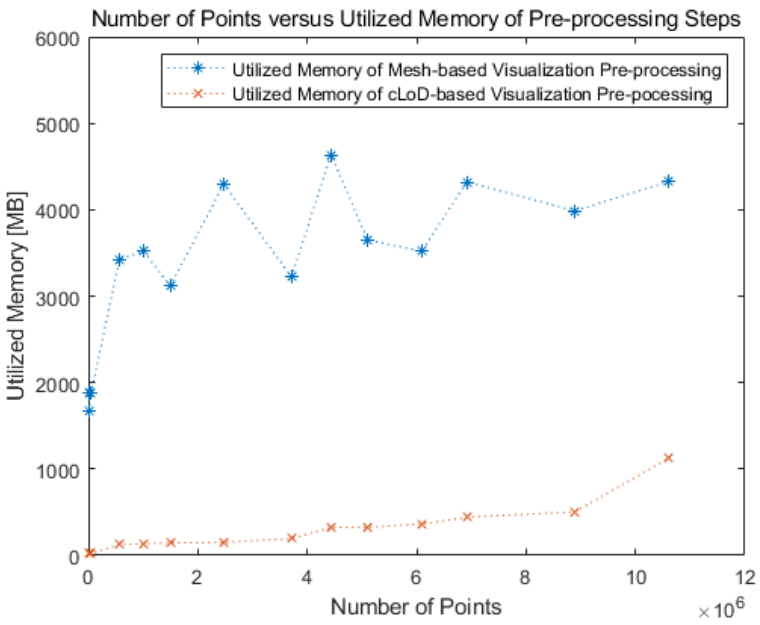


Figure 5.12: Number of points versus utilized memory of pre-processing steps

capability to replace the mesh models in some cases.

Figure 5.13: Rendering results of our cLoD-based point cloud visualization (top) and the mesh-based visualization (bottom)

Visual Quality

Figure 5.13 shows the final rendering results of our cLoD-based point cloud visualization and the mesh-based visualization. As we can see in this figure, the visual quality of these two methods doesn't have much difference. Both of them can deal with holes between the points and visualize models nicely. In some cases, our cLoD-based point cloud visualization even has better visual quality. Since some details might be removed or merged in the mesh models during reconstruction.

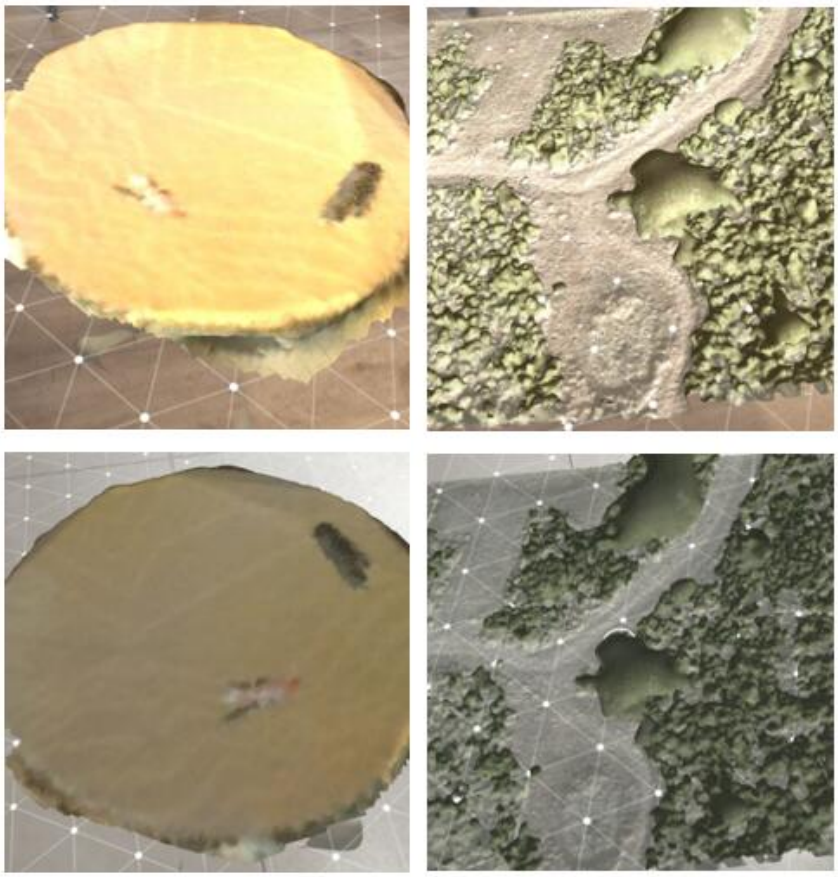


Figure 5.14: Different rendering results of the same mesh model due to different environment

However, the major drawback of our cLoD-based point cloud visualization is that the point cloud models don't have complete geometry and topology. Thus some more complex behaviours like shading and adding shadows are not feasible for point cloud models, which can performs well on the mesh models. Therefore, the mesh-based visualization can give the users more realistic feelings. Figure 5.14 shows how different lighting condition affects the shading and shadows in the rendering results of mesh models.

Utilized Memory

Figure 5.15 shows the relationship between the number of points and the run-time utilized memory of the two method. Since the mesh models contains more information, such as uv texture coordinates, texture, and normals, the run-time utilized memory of mesh-based visualization is always more than that of our cLoD-based point cloud visualization.

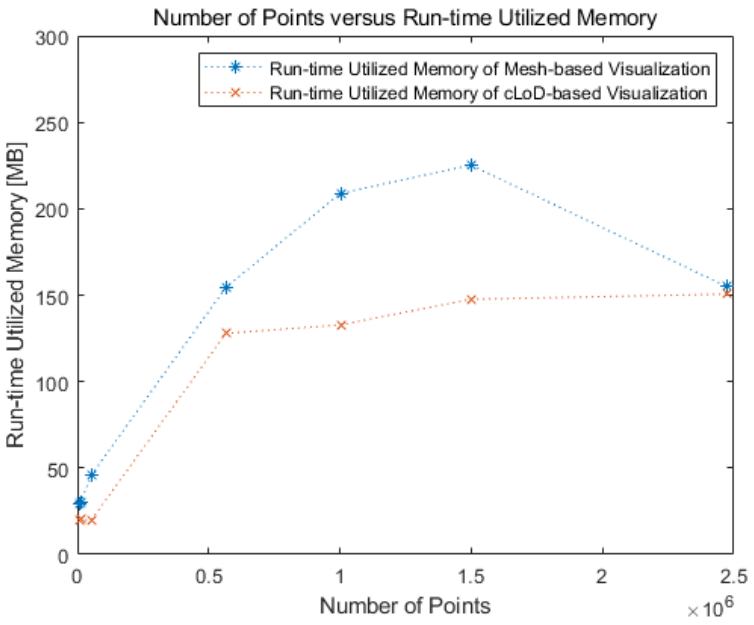


Figure 5.15: Number of points versus run-time utilized memory

6 Conclusions

In the final chapter, first, the research questions are reviewed, and the degree of how the research questions are finalized is assessed in Section 6.1. Then a discussion about this thesis, which addresses the contributions, limitations, and applications, is provided in Section 6.2. In the end, some future work is recommended with respect to the limitations in Section 6.3.

6.1 Research Overview

In this section, answers to research questions presented in Chapter 1 are provided, together with credible evidence. In order to better answer the main research question, sub-questions are answered first, since the sub-questions contains part of the answer to the main research question.

SUB-QUESTION 1

How to realize and improve the continuous level-of-detail method so that the amount of computation can fit the resources of mobile devices?

There are some existed cLoD-based visualization, like a point cloud renderer for both desktop and VR environment by Schütz et al. [2019], and a web-based point rendering system developed by Guan [2019]. However, in these rendering systems, besides the cLoD calculation, other calculation in point-wise is required. For instance, the rendering system developed by Schütz et al. [2019] needs to calculate the spacing between points, and Guan [2019] also calculate the distance from each point to the camera. Due to the limited CPU and GPU resources of mobile devices, this kind of calculation is too heavy for a mobile Augmented Reality application. During the research, the distance from each point to the camera was included in the selective query step at first. However, with the calculation of the distance, the frame rate started to decrease when only visualizing 500,000 points.

Thus, in order to reduce the amount of calculation, we decide to use a uniform threshold of the continuous level, which is determined by the wanted density and the distance from the model to the camera, to filter the points. What is more, the threshold is founded based on the idea of reaching the ideal density at a certain distance. This makes the revised cLoD method more flexible and suitable for most of the datasets.

SUB-QUESTION 2

What are the relevant parameters that are needed to be taken into account when using the cLoD method?

As describe in Chapter 4, there are three parameters that are mainly taken into consideration when using the cLoD method: update frequency, wanted density, and coefficient to scale the point size.

- **Update Frequency** In the rendering system, after putting point cloud models in the scene, the point cloud model will be updated every X frames accords to new cLoD based selection result. X is the parameter called $UpdateFrequency$. This parameter is taken into account is because the amount of calculation in each second can be reduced by setting the update frequency as a higher value. For example, if the update frequency is 5, then the selection and calculation of point size need to be done once per 5 frames, and when the update frequency is 1, then the computation is finished once per every frame. Thus, when visualizing large point clouds, the update frequency can be set as a higher value in order to stay at sufficient frame rate, i.e. 30 fps.

- **Wanted Density** The concept of wanted density is described in Section 3.5. The major problem of this concept is that the value of wanted density is affected by multiple factors, like spatial domain, original density, and distribution of the point clouds. This makes the automatic estimation be a quite challenging task, and manual adjustment is sometimes needed while visualizing. However, although additional operations of users are required, the rendering system can visualize most of the point cloud datasets with good performance and visual quality after adjustment.
- **Coefficient to Scale the Point Size** The coefficient to scale the point size is mentioned in Section 3.4, i.e. the s in the formula. Although when s is 5, the corresponding point size is considered to be the ideal point size, the value of s can be changed when the rendering results are not satisfying. For instance, when visualizing sparse point clouds, the value of s can be larger to make up visual artefacts like holes between points. As the results are shown in Section 5.1.1, the value of s is set as larger value like 12 and 14, in order to make up the holes that appear in the sparse furniture point clouds.

Sub-question 3

How to organize the unstructured point clouds so that the algorithm can be speed up?

As mentioned in Section 3.3, the unstructured point clouds are organized using the B+ tree data structure. Since the selective query only based on the cLoD attribute, which means positions are not considered in this case. Therefore, instead of the data structure that is efficient in space partitioning for nD point clouds, such as K-D Tree and Octree, B+ Tree is used to organize the points in this thesis.

Using B+ Tree to organize the point data significantly improves the efficiency of querying on the cLoD attribute. The average case time complexity of search operation with B+ Tree is $O(logN)$. Before organizing data with B+ Tree, the frame rate is extremely low (<5 fps) when only processing 1 million points. After data organization, the rendering system can process at most 10 million points and visualize them at 30 fps after selection.

Main Research Question

To what extent can an interactive point cloud visualization system in the mobile AR environment using continuous level-of-detail method be created?

In this thesis, an interactive point cloud visualization system in the mobile AR environment using continuous level-of-detail method is realized. The rendering system can be assessed from three aspects: realization of interaction, performance, and visual quality. The resource code and the installation package are available at: `https://github.com/LiyaoZhang0702/AR_PointCloud`.

- **Realization of Interactions** Basic interactions are enabled in the rendering system, like rotation and scaling. The scripts of the manipulation system are revised in order to fit the characteristics of the cLoD method. For instance, the script of scaling is revised, as mentioned in Section 4.2.5, the rendering system will render more points when zooming in and fewer points when zooming out. What is more, user interfaces are added to the rendering system. If the users are not satisfied with the automatic estimation, they can change the value of parameters, such as update frequency, wanted density, and point size, until they find the proper value of parameters to render their point clouds.
- **Performance** The rendering system can visualize large point clouds, which is not able to be rendered without the cLoD method, in the mobile AR environment. The cLoD method filters the points based on the idea of reaching the ideal density at a certain distance. Thus the rendering system can reduce the number of points sufficiently and still maintain nice visual quality. As mentioned in Section 5.2.2, we can see from the experiment results that when visualizing large point clouds in the indoor environment, 70% to 95% of the points can be removed considerably, thus significantly improve the performance. After adjustment of parameters, even point clouds that are close to the boundary of the rendering system can be visualized fluently at 30 fps with smooth interactions.

- **Visual Quality** As for the visual quality of the rendering results, we can see in the Section 5.1 that despite most of the points are removed, the rendering results still have pretty nice visual quality. This is not only because the cLoD method takes the characteristics of the point clouds into consideration but also because that adaptive point size rendering strategy is applied to the system. With the adaptive point size rendering strategy, holes between points and overlap due to the oversized points can be avoided, which improves the visual quality of the rendering results to a certain extent.

6.2 Discussion

6.2.1 Contributions

In this thesis, an interactive mobile AR point cloud rendering system is realized using cLoD method, and there are two significant contributions: improving the cLoD method and the implementation of the rendering system.

Improving cLoD Method

The first significant contribution of this thesis is improving the cLoD method so that it is able to fit the mobile AR environment. In fact, besides mobile AR applications, the cLoD method can also be used in other applications that need to render point clouds on mobile devices. Although there are already some existed cLoD-based rendering systems and the concepts presented in this thesis are not that creative, we can still say that our methods contribute a little bit to the current state-of-art. Since rendering point clouds in mobile AR environment is a new application that is waited to be explored, the cLoD method in this thesis gets a good start of utilizing point clouds in mobile AR environment.

Implementation of Rendering System

Besides the cLoD method, implementation of the rendering system is another major contribution of this thesis. Basic operations of an AR application are realized, interactions like rotation and scaling are available, friendly user interfaces are added, and proper rendering strategy is applied in the rendering system.

The rendering system can handle point cloud models with at most 10M points, contain 5M points in the scene and visualize them at 30 fps (i.e. required frame rate for mobile AR applications). Visualizing such large point clouds is not possible without the cLoD method, and we can even visualize multiple large point cloud models only if there are less than 5M points in the scene after selection.

What is more, the rendering system does not need many pre-processing steps. As discussed in Section 5.3, compared to the complicated pre-processing steps of generating mesh models, the only needed pre-processing steps of our method are storing the file to the mobile phones, and then loading point clouds from it. Once the file is loaded into the system, the point cloud models can be visualized without any delay.

Finally, the rendering system has pretty nice usability. The ideal density concept works well when visualizing evenly distributed and dense point clouds, however, when visualizing unevenly distributed models or sparse point clouds, manual operations might be required to find proper value of parameters. Although manual behaviour is sometimes required, most of the point clouds can be visualized with good performance and nice visual quality after adjustment. No matter point clouds are unevenly distributed or are very sparse, the users can find proper value of parameters to improve the rendering results in the end.

6.2.2 Limitations

The major limitation of the rendering system is that the quality of the automatic estimation. The rendering system works fine when the point clouds are evenly distributed and dense, however, when visualizing unevenly distributed and sparse models, the results of the automatic estimation are not satisfying. Thus, manual operations are sometimes required to find the proper value of parameters.

The second limitation is that the usability of the rendering system in the outdoor environment is under question. As mentioned in Section 5.2.2, the boundary of processing points is around 10 million points, and the capability of containing points in the scene is around 5 million points. This means that for some outdoor applications that will have more space to render the points and visualize more massive point clouds, the rendering system is not competent. However, for most of the indoor applications that have a smaller field of vision and visualize smaller point clouds, the rendering system can perfectly handle most of the rendering tasks. When visualizing point clouds with less than 10 million points in the indoor environment, 70% to 95% of the points can be removed considerably, thus significantly improve the rendering performance and meanwhile preserve excellent visual quality.

The third limitation is because of the invalid geometry and topology of point cloud models. As mentioned in Section 5.3, some more complicated behaviours like shading and adding shadows are not available for point cloud models, which means the rendering results are less realistic than mesh-based visualization.

Final limitation is the overused memory amount. As shown in Section 5.2.2, the utilized memory is too much for a simple mobile point cloud renderer. What is more, the phone will be overheating when running the rendering system for more than 20 minutes. This means that the consumption of CPU and GPU is quite high as well. Optimization of the app is an urgent task.

6.2.3 Applications

As discussed in Section 5.3, the greatest strength of this method is that we can directly get use of the easily obtained point clouds in the mobile AR environment without complicated pre-processing steps. What is more, the materials and shaders used to render the point cloud models can be easily changed. Thus we can replace some of the existed models by the point clouds in many applications, especially for applications that needs to simulate and visualize existed objects. There are some potential applications of this efficient interactive visualization of point clouds.

For indoor applications, first, we can put some dense point cloud models of smaller objects, such as tables and chairs, which will be useful for home renovation, just like the famous IKEA Place app. Second, large point clouds are useful as well. For instance, putting the scanned point clouds of an entire room to simulate different environments, or putting scaled point clouds of an entire building for real estate sales. Figure 6.1 shows examples of visualizing furniture point clouds and architecture point clouds in the indoor environment.

Although the usability of the rendering system in the outdoor environment is still waited to be examined, there are some potential outdoor applications. For example, our method can be widely used in architecture and industrial design fields. Users can put point cloud models of roads, bridges, or other giant objects from other location into the real world with our method before construction to simulate the final result without spending a lot of time and energy establishing 3D models. What is more, materials and shaders used to render the point cloud models can be easily changed so that the look of objects with different materials can be simulated. Figure 6.2 shows how the architects will use the AR applications. Change detection can also be added to our method so that we can highlight the changes between two point clouds in the AR environment.

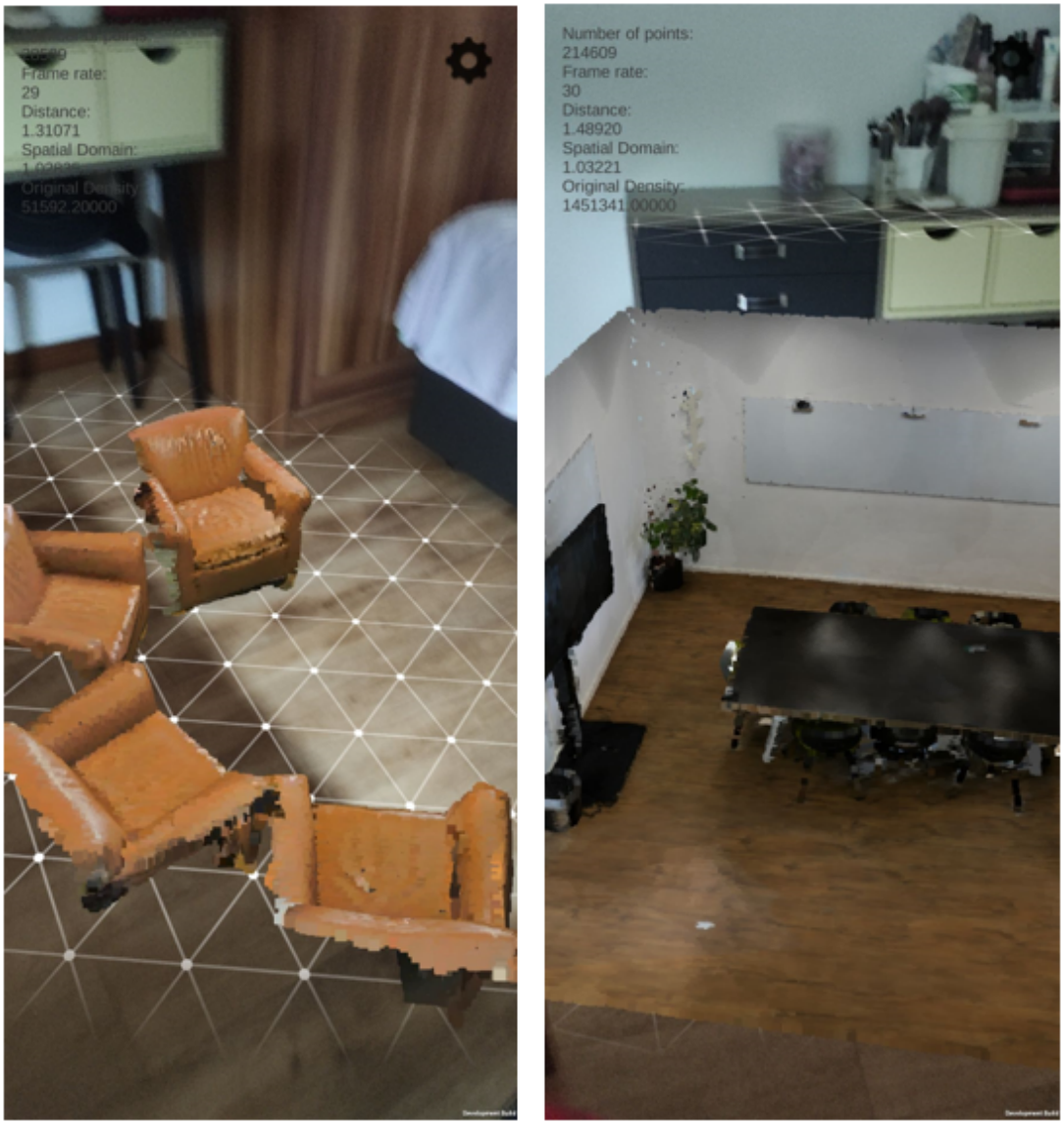


Figure 6.1: Examples of visualizing furniture point clouds and architecture point clouds in mobile AR environment

6.3 Future Work

In this section, some future work that has not been finished due to the time limitation is presented. In the future, we would like to continue in the following directions:

- The most urgent work to be finished is to improve the quality of the automatic estimation. We will use technologies, like machine learning and deep learning, to let the rendering system automatically find proper value of parameters to visualize point clouds.
- Optimization of the app is required as well. The overheating problem of the device and the overuse of the memory resources are waited to be solved.
- At the current stage, the rendering system is only tested with one device and a limited number of test datasets. In the future, we will test our method with more datasets and different devices to see its applicability.

Figure 6.2: Point clouds used by architects in AR environment [Archicgi, 2020]

- The method is only tested in the indoor environment. In the future, we will examine its capability in the outdoor environment and explore more outdoor applications.
- In order to explore the potential of the rendering system to visualize larger point clouds like the city or nationwide point clouds, an additional mechanism like databases, server and caching strategy, and information transfer between SSD and CPU will be added to the rendering system.
- If we can put part of the computation to the cloud or the hardware of mobile phones upgrades again, then we'll implement the cLoD-based selection on distance from the camera to individual points. And using nD data structure to improve the performance.
- More interactions will be applied to the rendering system, such as measuring the distance, changing colours of points, and interactions with part of the point cloud model.

A Reproducibility self-assessment

A.1 Marks for each of the criteria

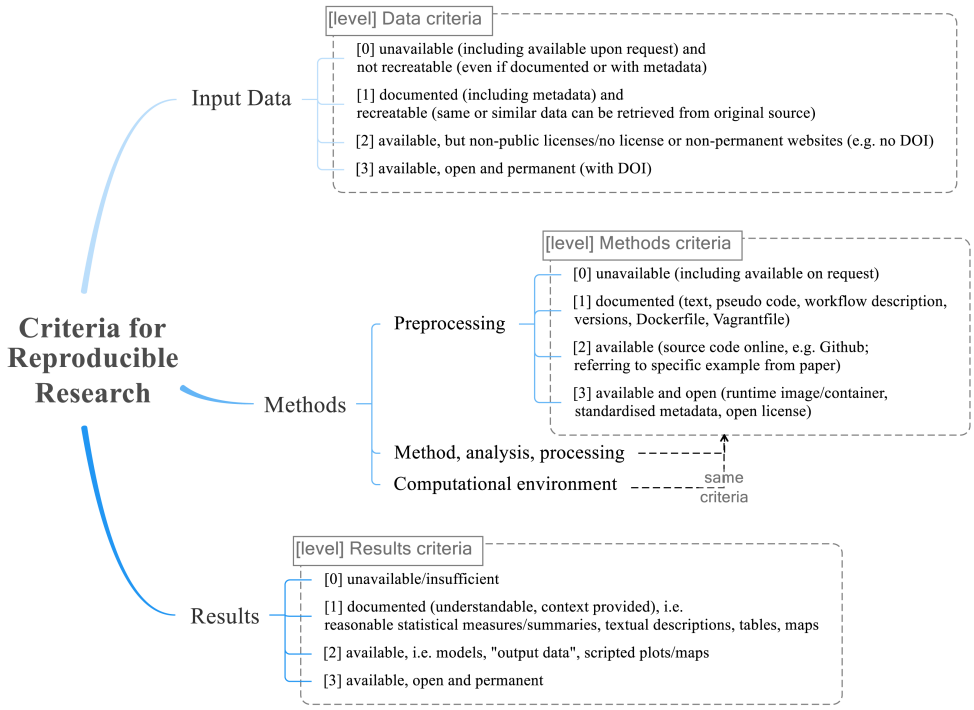


Figure A.1: Reproducibility criteria to be assessed.

The resource code and the installation package are available at: `https://github.com/LiyaoZhang0702/AR_PointCloud`. Based on the reproducibility criteria presented in Figure A.1, the grade of each criteria is showed in Table A.1.

Table A.1: Evaluation of the reproducibility criteria

Criterion	Grade
Input Data	3
Preprocessing	3
Methods	2
Computational Environment	2
Results	3

A.2 Self-reflection

The reason why most of the grades are pretty high is because the source code and APK are uploaded to GitHub with a brief instruction, which are available at `https://github.com/LiyaoZhang0702/AR_`

PointCloud. The reproducibility can be realized by simply downloading the project and APK, and then installing them on your own devices.

Although most of the steps can be reproduced easily, some additional operations are needed in some steps. For example, when inputting data, users need to change the name of the point cloud file as 'Test.laz' and put it to a specific path. And configuration of computational environment is required since the project is developed with Unity together with ARCore. All in all, most of the operations needed for reproducibility are pretty simple.

Bibliography

Amin, D. and Govilkar, S. (2015). Comparative study of augmented reality sdks. *International Journal on Computational Science & Applications*, 5(1):11–26.

Archicgi (2020). Point cloud modeling: Why it's crucial for the architectural field. https://drawings.archicgi.com/point-cloud-modeling-for-architecture/.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). Recent advances in augmented reality. *IEEE computer graphics and applications*, 21(6):34–47.

Azuma, R. T. (1997). A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385.

Biljecki, F. (2013). The concept of level of detail in 3D city models. *PhD proposal*.

Billinghurst, M. (1999). Shared space; collaborative augmented reality. In *ACM SIGGRAPH99 Conf. Abstracts and Application*.

Billinghurst, M., Clark, A., and Lee, G. (2015). A survey of augmented reality.

Brooks Jr, F. P. (1996). The computer scientist as toolsmith ii. *Communications of the ACM*, 39(3):61–68.

Chen, C.-C. and Chuang, J.-H. (2006). Texture adaptation for progressive meshes. In *Computer Graphics Forum*, volume 25, pages 343–350. Wiley Online Library.

Clark, J. H. (1976). Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554.

Discher, S., Masopust, L., and Schulz, S. (2018a). A point-based and image-based multi-pass rendering technique for visualizing massive 3D point clouds in vr environments.

Discher, S., Richter, R., and Döllner, J. (2018b). A scalable webgl-based approach for visualizing massive 3D point clouds using semantics-dependent rendering techniques. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology*, page 19. ACM.

Discher, S., Richter, R., Trapp, M., and Döllner, J. (2019). Service-oriented processing and analysis of massive point clouds in geoinformation management. In *Service-Oriented Mapping*, pages 43–61. Springer.

Elmasri, R. (2008). *Fundamentals of database systems*. Pearson Education India.

Fisher, S. S. (1986). Virtual interface environment.

Fuchs, H., Livingston, M. A., Raskar, R., Keller, K., Crawford, J. R., Rademacher, P., Drake, S. H., Meyer, A. A., et al. (1998). Augmented reality visualization for laparoscopic surgery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 934–943. Springer.

Fung, C. (2019). The future of mobile AR: A comparison between arcore, arkit, ar foundation, and vuforia. https://www.credera.com/insights/the-future-of-mobile-ar-a-comparison-between-arcore-arkit-ar-foundation-and-vuforia/.

Furness, L. (1969). The application of head-mounted displays to airborne reconnaissance and weapon delivery. *Wright-Patterson Air Force Base, Ohio, USA*.

GeeksforGeeks (2020). Introduction of B-tree. https://www.geeksforgeeks.org/introduction-of-b-tree-2/?ref=lbp.

Goswami, P., Erol, F., Mukhi, R., Pajarola, R., and Gobbetti, E. (2013). An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer*, 29(1):69–83.

Guan, X. (2019). Technical report of a 3D point cloud visualization system. Wuhan University Paper.

Höllerer, T. and Feiner, S. (2004). Mobile augmented reality. *Telegeoinformatics: Location-based computing and services*, 21.

Hoppe, H. (1996). Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM.

Huang, Y., Weng, D., Liu, Y., and Wang, Y. (2009). Key issues of wide-area tracking system for multi-user augmented reality adventure game. In *2009 Fifth International Conference on Image and Graphics*, pages 646–651. IEEE.

Huizenga, J., Admiraal, W., Akkerman, S., and Dam, G. t. (2009). Mobile game-based learning in secondary education: engagement, motivation and learning in a mobile city game. *Journal of Computer Assisted Learning*, 25(4):332–344.

Isenburg, M. (2013). Laszip: lossless compression of lidar data. *Photogrammetric Engineering and Remote Sensing*, 79(2):209–217.

Kalawsky, R. (1993). The science of virtual reality and virtual environments.

Kazhdan, M. and Hoppe, H. (2013). Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3).

Koyama, T. (2009). Introduction to flartoolkit. *Adobe System Incorporated*.

Le Muzic, M., Autin, L., Parulek, J., and Viola, I. (2015). cellview: a tool for illustrative and multi-scale rendering of large biomolecular datasets. In *Eurographics Workshop on Visual Computing for Biomedicine*, volume 2015, page 61. NIH Public Access.

Lee, M., Billinghurst, M., Baek, W., Green, R., and Woo, W. (2013). A usability study of multimodal input in an augmented reality environment. *Virtual Reality*, 17(4):293–305.

Leventon, M. E. (1997). *A registration, tracking, and visualization system for image-guided surgery*. PhD thesis, Massachusetts Institute of Technology.

Livny, Y., Kogan, Z., and El-Sana, J. (2009). Seamless patches for gpu-based terrain rendering. *The Visual Computer*, 25(3):197–208.

Luebke, D., Reddy, M., Cohen, J. D., Varshney, A., Watson, B., and Huebner, R. (2003). *Level of detail for 3D graphics*. Morgan Kaufmann.

Martinez-Rubi, O., Verhoeven, S., Van Meersbergen, M., Van Oosterom, P., GonÁalves, R., Tijssen, T., et al. (2015). Taming the beast: Free and open-source massive point cloud web visualization. In *Capturing Reality Forum 2015, 23-25 November 2015, Salzburg, Austria*. The Servey Association.

Milgram, P. and Kishino, F. (1994). A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329.

Navab, N., Bani-Kashemi, A., and Mitschke, M. (1999). Merging visible and invisible: Two camera-augmented mobile c-arm (camc) applications. In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pages 134–141. IEEE.

Nebiker, S., Bleisch, S., and Christen, M. (2010). Rich point clouds in virtual globes–a new paradigm in city modeling? *Computers, Environment and Urban Systems*, 34(6):508–517.

Pajarola, R., Sainz, M., and Lario, R. (2005). Xsplat: External memory multiresolution point visualization. In *Proceedings IASTED Invernational Conference on Visualization, Imaging and Image Processing*, pages 628–633.

Piumsomboon, T., Altimira, D., Kim, H., Clark, A., Lee, G., and Billinghurst, M. (2014). Grasp-shell vs gesture-speech: A comparison of direct and indirect natural interaction techniques in augmented reality. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 73–82. IEEE.

Rekimoto, J. (1996). Transvision: A hand-held augmented reality system for collaborative design. In *Proceeding of Virtual Systems and Multimedia*, volume 96, pages 18–20.

Rekimoto, J. and Nagao, K. (1995). The world through the computer: Computer augmented interaction with real world environments. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 29–36.

Richter, R. and Döllner, J. (2010). Out-of-core real-time visualization of massive 3D point clouds. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 121–128. ACM.

Ripolles, O., Chover, M., Gumbau, J., Ramos, F., and Puig-Centelles, A. (2009). Rendering continuous level-of-detail meshes by masking strips. *Graphical Models*, 71(5):184–195.

Röttger, S., Heidrich, W., Slusallek, P., and Seidel, H.-P. (1998). Real-time generation of continuous levels of detail for height fields.

Rusinkiewicz, S. and Levoy, M. (2000). Qsplat, a multiresolution point rendering system for large meshes. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH 00*.

Sander, P. V., Snyder, J., Gortler, S. J., and Hoppe, H. (2001). Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM.

Scherzer, D. and Wimmer, M. (2008). Frame sequential interpolation for discrete level-of-detail rendering. In *Computer Graphics Forum*, volume 27, pages 1175–1181. Wiley Online Library.

Schütz, M., Krösl, K., and Wimmer, M. (2019). Real-time continuous level of detail rendering of point clouds. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 103–110. IEEE.

Scratchapixel (2014). The perspective and orthographic projection matrix. https://www.scratchapixel.com/lessons/3d-basic-rendering/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix.

Southern, R. and Gain, J. (2003). Creation and control of real-time continuous level of detail on programmable graphics hardware. In *Computer Graphics Forum*, volume 22, pages 35–48. Wiley Online Library.

Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764.

TeamUnity (2019). Unity user manual (2019.4 lts). *GameObject*.

Tredinnick, R., Broecker, M., and Ponto, K. (2016). Progressive feedback point cloud rendering for virtual reality display. In *2016 IEEE Virtual Reality (VR)*, pages 301–302. IEEE.

Turchyn, P. (2007). Memory-efficient sliding window progressive meshes.

van der Maaden, J. (2019). Vario-scale visualization of the ahn2 point cloud.

Van Oosterom, P. (2019). From discrete to continuous levels of detail for managing nd-pointclouds. Keynote presentation at ISPRS Geospatial Week.

Wimmer, M. and Scheiblauer, C. (2006). Instant points: Fast rendering of unprocessed point clouds. In *SPBG*, pages 129–136. Citeseer.

Wojciechowski, R., Walczak, K., White, M., and Cellary, W. (2004). Building virtual and augmented reality museum exhibitions. In *Proceedings of the ninth international conference on 3D Web technology*, pages 135–144.

Zhang, L., van Oosterom, P., and Liu, H. (2020). Visualization of point cloud models in mobile augmented reality using continuous level of detail method. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 44:167–170.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.