BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

BETTER CHAPERONE BOUNDS USING 3D SENSORS

LEPŠÍ VYMEZENÍ HERNÍHO PROSTORU PRO VR POMOCÍ 3D SENSORŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR Bc. JAN TINKA
AUTOR PRÁCE

SUPERVISOR Ing. PAVEL NAJMAN
VEDOUCÍ PRÁCE

BRNO 2018

Zadání diplomové práce/21012/2017/xtinka05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Tinka Jan, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Lepší vymezení herního prostoru pro VR pomocí 3D sensorů**
Better Chaperone Bounds Using 3D Sensors
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s zařízeními pro VR, zejména s HTC Vive a Oculus Rift a 3D senzory Kinect a ZED.
2. Prostudujte problematiku vymezení herního prostoru a zobrazení jeho hranic ve VR.
3. Navrhněte lepší způsob zobrazení virtuálních hranic s využitím dostupných sensorů.
4. Navrhnutý způsob implementujte a demonstrujte v ukázkové aplikaci.
5. Vyhodnoťte implementovaný způsob zobrazení virtuálních hranic pomocí uživatelských testů.
6. Své řešení prezentujte krátkým videem.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Najman Pavel, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

[illegible]

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstract

Room-scale tracking encourages users to move more freely and even walk. Even though there has been much research on making the limited physical workspace feel larger in the VR, these approaches have their limitations and require certain conditions to be met. This thesis proposes an alternative approach to the conventional play-area boundaries of high-end VR products such as the HTC Vive and Oculus Rift which are set by the user in a 2-D fashion as a means of enhance workspace utilization. A 3-D scanner is used to make a 3-D point-cloud model of the play area's surroundings. This model is then used to detect collisions and provide feedback to the user. Evaluation based on user tests showed that this approach can be useful, is well accepted by users and might be worth further research.

Abstrakt

V posledních letech se na spotřebním trhu začala objevovat VR zařízení, což má za následek celkový nárůst popularity VR. Špičkové produkty jako HTC Vive a Oculus Rift s vysoce přesným room-scale sledováním podporují z důvodu bezpečnosti a příhodnosti hranice virtuální herní oblasti. Room-scale VR uživatele nabádá k volnějšímu pohybu nebo i chození. A ačkoliv se problematikou zvětšení vnímané velikosti virtuální herní oblasti v rámci velikosti skutečného pracoviště zatím zabývala řada výzkumů, jejich řešení nejsou dokonalá a vyžadují splnění jistých podmínek. Kvůli tomu a faktu, že chování uživatelů lze ne vždy předvídat, jsou hranice herní oblasti stále potřeba. V běžném room-scale VR jsou ale tyto virtuální hranice uživatelem definovány v podstatě dvourozměrně, a proto jsou stěny těchto hranic ploché a nezachycují bližší detaily případných překážek.

V rámci této diplomové práce představuji alternativu k těmto hranicím, díky které by mohlo být možné lépe využít dostupný prostor například nad překážkami jakými jsou židle, stoly a postele. Navrhuji a implementuji příklad této alternativní hranice. Na základě uživatelských testů potom prezentuji vyhodnocení vypracovaného řešení.

Navrhované řešení je založené na využití 3D skeneru pro získání trojrozměrných informací o bezprostředním okolí herní oblasti. Tyto informace jsou zpracovávány v podobě mračna bodů a jsou zaznamenávány prostřednictvím stereokamery ZED připevněné k virtuálním brýlím HTC Vive disponujícím přesným optickým sledováním SteamVR, kterého je využito pro získání přesné polohy stereokamery v rámci sledovaného prostoru. Body se nejdříve ukládají do prostorově omezené 3D mřížky, která slouží také k jejich vzorkování, odstraňování ojedinělých šumových bodů a zajistí konečnou velikost výsledného mračna bodů. V následujícím kroku jsou body filtrovány pomocí filtru založeného na počtu sousedů. Zpracované body jsou následně uloženy do souboru, odkud jsou později načteny pro využití v rámci hranice herní oblasti.

Při nahrávání bodů jsou tyto rozděleny na menší shluky z důvodu odlehčení hernímu enginu Unity a možnostem vykreslování. Tyto shluky jsou umístěny do scény, kde jsou ve výchozím stavu neviditelné. Každý shluk disponuje rendererem a prostředkem pro detekci kolizí. Virtuálním brýlím i ovladačům lze přiřadit jiné druhy těchto prostředků. Tyto mají tvar koule a nebo rotačního válce s polokoulí na horní podstavě. Mezi těmito tvary a body jsou počítány kolize, na jejichž základě se rozhoduje o poskytnutí vizuální nebo hmatové zpětné vazby uživateli. Pokud se uživatel nebo některý z ovladačů přiblíží k neviditelným bodům hranice, tyto se zviditelní a v případě ovladače příslušný ovladač začne vibrovat. Body hranice se zobrazují jako malé kruhy v prostoru, které jsou viditelné i pokud se nachází za nějakou překážkou, což je důležité z hlediska bezpečnosti.

Implementované řešení bylo podrobeno uživatelským testům, kterých se zúčastnilo 12 studentů naší fakulty. Bylo testováno, zda-li je hranice implementovaného řešení založená na bodech z 3D skeneru vůbec použitelná jako hranice herního prostoru, zda-li jsou účastníci testování schopni vyhnout se překážce nacházející se uvnitř herní oblasti a jaký vliv má na účastníky testování prolínání se virtuálních a skutečných objektů v podobě bodů hranice při blízké manipulaci s objekty. Z pozorování a odpovědí na otázky k testům vyplynulo, že alternativní vylepšená hranice fungovala, jak měla, byla účastníky testování velmi dobře přijata a alespoň za některých podmínek ji většinou preferují před výchozí SteamVR hranicí Chaperone. Většina účastníků testování měla v hranici důvěru a nebála se, že by do něčeho mohla narazit. V průměru přišla účastníkům testování vylepšená hranice prostornější a méně narušovala jejich ponoření do virtuálního prostředí. Ukázalo se, že překážka uvnitř herní oblasti nepředstavuje problém.

Uživatelské testy také odhalily nebo ukázali na několik nedostatků. Malé překážky je možné přehlédnout, zvlášť pokud se uživatel nedívá jejich směrem. Účastníkům u vylepšené hranice chyběla hranice, která by je zastavila při odchodu z herní oblasti tam, kde nejsou překážky, a tudíž ani body hranice. Je špatně rozeznat, které body se nachází uvnitř, a které vně virtuálních objektů, což se projevilo u testu s překrývajícími se virtuálními a skutečnými objekty.

Keywords

virtual reality, VR, Unity, HTC Vive, Oculus Rift, Microsoft Kinect, Stereolabs ZED, The Chaperone, play space boundary, virtual boundary, point cloud, feedback, collisions

Klíčová slova

virtuální realita, VR, Unity, HTC Vive, Oculus Rift, Microsoft Kinect, Stereolabs ZED, The Chaperone, hranice herní oblasti, virtuální hranice, mračno bodů, zpětná vazba, kolize

Reference

TINKA, Jan. *Better Chaperone Bounds Using 3D Sensors.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Pavel Najman

Better Chaperone Bounds Using 3D Sensors

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Mr. Pavel Najman. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Jan Tinka
May 23, 2018

Contents

Introduction

Nowadays, virtual reality (VR) appliances are growing increasingly popular and the associated industries, most notably gaming industry, are gaining momentum.

While there are many ways of realising VR appliances, head-mounted displays (HMDs) such as Oculus Rift and HTC Vive made a breakthrough with the two being the first consumer articles of this kind adopted in such a scale. One of the main advantages of HMDs is that they offer superior immersion, partially because they block the user's vision and isolate them from their surroundings.

Development of VR experiences, games and applications is made more accessible and convenient for developers by game engines like Unity, Unreal Engine or Cry Engine. All of these three engines are widely adopted and support the aforementioned devices.

At the time of writing of this thesis, both Oculus Rift and HTC Vive provide support for the so-called room-scale VR, which by contrast to e.g. the seated VR, where the user remains more or less stationary, provides room-scale tracking enabling them to move much more freely.

Rooms in which users typically engage in VR are usually limited in space and as the user can move freely without being able to see their surroundings it is possible that they reach an end of their designated VR area and unintentionally bump into or hit objects of the real world. Real-world space limitations inherently must affect either the way users move through virtual environments, or the virtual environments themselves.

Informing the user about the real-world limitations using an in-VE synthetic barrier, such as the HTC Vive's Chaperone, is one of the approaches that don't alter the user's input. However, the walls of the Chaperone are flat only, with their placement being defined two-dimensionally by the user walking around with a controller that sends, among others, its positional data.

The main focus of this thesis is to explore a way of improving the boundary by incorporating 3-D data gathered using a stereo camera. Addition of the 3-D data could help users utilize available space more efficiently. For instance the user can't walk through a table but they might be able to freely move their hands above it.

The 3-D data can be gathered and processed either statically or dynamically. When the real-world room isn't scanned during the run of a VR application and only previously captured set 3-D data is used to calculate and somehow convey the boundary to the user, that's a static approach. Dynamic approach assumes that the data is gathered and processed on the fly.

The feasibility and usefulness of the static approach of the data gathering is to be assessed in this work, as well as to some extent the methods of conveying the 3-D boundary to the user. The assessment is to be based on measures and test-subjects' subjective reports of the boundary's aspects such as accuracy, perceived size or how intrusive they are regarding the VR experience.

All implementation and experimentation will be carried out in Unity. Apart from evaluation of how useful and feasible the approach laid out by this thesis is, a Unity plugin package will be created that will allow for easy integration of the boundary based on 3-D data into VR Unity projects.

This work is a follow-up to my term project and incorporates most of its content as it is still relevant. Most of what is left of the original work stayed the same. However, at some places phrases had been reworded and grammatical and typographical errors corrected.

Most of the introduction chapter remained the same with the exception of this final part which describes the relation between this thesis and the term project. Exclusive of minor changes to the section proposing the improved boundary (section 1.3), chapter 1 consists entirely of material that was already present in the term project. Chapter 2 was reorganized, but the hardware overview part remains almost unchanged as well. Section 2.4 was extended to reflect upon the needs for more detailed information arising from the newly written implementation chapter (chapter 4). Lastly, the first, second and last paragraphs of the conclusion that refer to the aforementioned overviews, the proposal and discuss HTC Vive Pro respectively, are all reused.

Chapter 1

Current state of VR and improvement proposal

Virtual reality (VR) or a virtual environment (VE) is an artificially created environment viewed from a first-person point of view, while the view is controlled by the user in real time. The user can interface with such virtual environment using plethora of input and output hardware devices differing in principle. For instance a visual output might be in the form of a surround-screen display, a hemispherical display (both of which display images on surfaces around the user), a head-mounted display (HMD) or a virtual retinal display.[6]

1.1 Room-scale VR

For the last several years the field of VR has marked significant increase in popularity especially thanks to the emergence of head-mounted displays (HMDs) such as the Oculus Rift, HTC Vive, Samsung GearVR or Sony Playstation VR. While there are many other kinds of VR input or output devices this thesis focuses on VR as interfaced by the current generation of HMDs and associated technology such as positional tracking. In particular it is focused on the so-called room-scale VR.

Room-scale VR enables the user to use more play space (or workspace) and to move relative freely thanks to tracking that's provided on a room scale. Travelling in a virtual environment can still be based on teleportation, throwing, air grabbing, using a controller or other locomotion techniques[6] as it was the case before but what's new is that the larger play space allows the user to travel in VR by walking and gives them the ability to explore the VR in a larger area providing them with options such as walking around a virtual object and looking at it from all directions without the need to manipulate it, the same way they could in real world.

Researchers had worked on a VR with larger-scale tracking that enabled for walking before, and there had been some very successful implementations such as The HiBall Tracking System by Greg Welch, Gary Bishop et al.[28]. However, in the consumer products market, room-scale was pioneered by the HTC's VR system Vive with tracking by Valve Corporation in 2016[21].It was followed by the Oculus Rift, which finally started fully supporting room-scale tracking in 2017[5].

Both aforementioned devices make use of invisible optical tracking based on a combination of a passive emitter and an active sensor. The tracked device is in one case covered by photosensitive elements and surrounded by synchronized but otherwise passive light emitters, and covered with LEDs while surrounded by sensors in the other.

Although walking in VR is possible with room-scale tracking, in normal conditions play space is limited and the user can only take so many steps before they bump into a wall or an object. They might also just walk out of the tracked area, provided the play area is situated in a spacious room or an outside setting. Extensive research has been done on the topic of making a limited play space seem larger, infinite even. There have been numerous approaches and proposed solutions to this task. For instance, Khrystyna Vasylevska and Hannes Kaufmann call the task spatial compression, of which they distinguish several types – namely, basic reorientation, sense manipulation, rendering manipulation and 3D scene manipulation. Basic reorientation is a simple method where the user is asked to reorient themselves to stay in the designated play area, which has inevitable adverse effects on immersion. Other types are less intrusive but more difficult and sometimes impossible to implement. An example of a technique that utilizes sense manipulation is „redirected walking". The technique is based on unnoticeable changes to the cameras position and rotation, usually through means of translational and/or rotational gain.[27]

There has been some research done on redirected walking in a room-scale VR setup. Indefinite walking in a straight line in a VE without the user being able to tell that they're actually following a curved path requires a physical workspace of about 45 m x 45 m. Eike Langbehn et al. [14] managed to significantly reduce the size requirements for the physical workspace by leveraging a fact that users don't usually follow a straight path in VEs. Their method utilises a rotational gain that's applied when the user walks along a curved path and makes turns.

Evan A. Suma et al. proposed a scene manipulation technique called impossible spaces. They show how to make a virtual in-door environment fully walkable even when it's larger than the play area, and achieve this by manipulating the VE so that separate rooms overlap in manner that violates the rules of an Euclidean space. Such as connecting two adjacent rooms with a corridor so short that the two rooms would be too big to fit next to each other in the real world.[23]

1.2 Play space boundary

While spatial compression can help with the enlargement of the perceived play space size it isn't perfect and the users don't always do what they are expected to do. Consequently, regardless of whether any spatial compression methods are in place or not, I think that providing the user, who's blinded by a HMD, with information about the boundary of their designated play space, if the need arises, enables them to effectively avoid crossing the boundary and potentially causing harm to themselves or their surroundings.

In a typical room-scale VR setup the user is indeed informed about the play-area boundary. The play area's boundary is set up by the user once (unless the user wishes otherwise) during the initial setup configuration and is in place from then on. It is available in any application if not disabled. The boundary is placed by the user marking it with a motion-tracked controller while walking alongside the play area's border.

The boundary is not visible not to break immersion unless it's approached by some of the user's tracked equipment. When close to the boundary a translucent pattern is visible, which is the more pronounced the closer the user gets.

Both HTC Vive and Oculust Rift have such boundary and both are set up using the same procedure. HTC's solution is called the Chaperone[10], shown in figure1.1. A boundary implementation from Oculus called the Guardian System[20] is very similar.

Thanks to the HTC Vive having a front-facing camera built in the user is also able to examine their surroundings using a pass-through video stream. As figure 1.2a demonstrates, when toggled on the stream is displayed in VR near a virtual counterpart of one of the user's hand-held controllers. This has the advantage of letting the user see everything they look at without them removing the HMD. However, in addition to negatively affecting immersion it isn't usable at all times as it requires the controller to be held ahead for the stream to be visible, and is generally more suitable for occasional use that requires precision such as untangling cables, removing an obstacle or grabbing a drink.

A more convenient alternative to that is a modification to the Chaperone nicknamed the „Tron Mode“ as seen in figure 1.2b. Whenever the user approaches the play area boundary to the point, where the Chaperone becomes visible, the video stream is processed by what seems to be an edge detecting filter and presented to the user as a translucent overlay. This approach doesn't require the user to use the controllers.

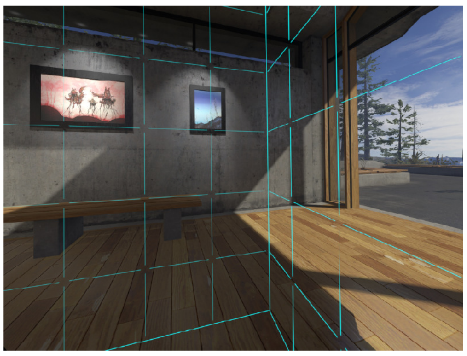
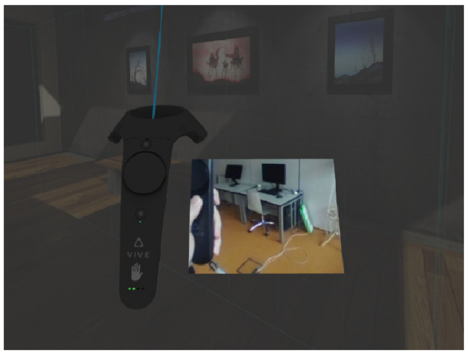
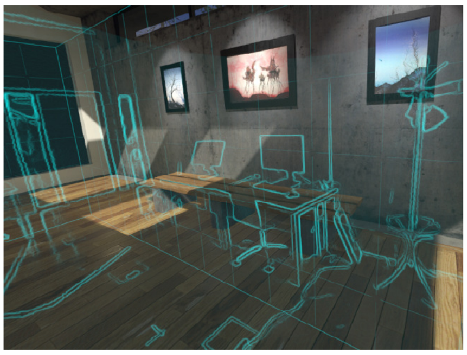


Figure 1.1: A screenshot of the play area boundary implementation of the HTC Vive – the Chaperone. The Chaperone is the blueish translucent grid.



(a) image from Vive's camera



(b) Vive's Chaperone „Trone Mode“

Figure 1.2: (1.2a) shows a screenshot of a controller with a video stream from the Vive's front-facing camera. In the same setting a screenshot of the „Trone Mode“ is shown in (1.2b).

1.2.1 Limitations

The currently used boundaries have a disadvantage of being defined in two dimensions only as they are defined by a border. The walls of the boundaries are flat and don't provide any detail. Therefore, the user usually sets up the border only where there is no obstacle from the floor to the ceiling. If being able to see details of one's surroundings is required, it is possible to use the Tron Mode. However, the Tron Mode is still toggled by the same two-dimensional border, shows everything the camera sees regardless of distance, and isn't three-dimensional as there is only one camera in the Vive.

1.3 Improved boundary

This thesis proposes an alternative approach to enlarging the play area, that could work alongside spatial compression, by creating the play area boundary based on 3-D surface data of the user's environment instead of a predefined 2-D border. The data, that is gathered using a 3-D scanning device attached to the HMD, is used for collision detection to trigger the displaying of the boundary, and as an in-VR 3-D representation of the boundary. Attaching the 3-D scanner to an already positionally tracked HMD has the advantage of a very precise pose estimation of the scanner. This makes the 3-D scanner much more usable in situations in which it wouldn't be able to properly estimate its pose based on its sensors' depth data. Examples of such situations would be poor ambient lighting conditions or interference, depending on the type of the 3-D scanner. I will be referring to this approach as „the improved boundary“ throughout the rest of this work.

Such a boundary could allow safe reaching around obstacles, tables, for instance. It could adapt to changes to the environment without much, or any, exertion of the user, and also allow them to detect people or animals inside their play area if implemented in a real-time fashion.

The goal of this thesis is to implement an example of such boundary, experiment with it and evaluate its qualities with the help of user tests. The experimentation and the evaluation should shed light on what improvements and caveats it brings, along with any unexpected side effects that might be discovered.

To provide a base for the conclusion of the experiments, a series of tests on several chosen test subjects will take place. During the tests, objective measurements are to be taken and each test subject will be asked to provide subjective evaluation of their experience.

The secondary goal this thesis sets is to provide a Unity (described on page 12) package, that would process data from the 3-D scanning device used in this thesis, or other devices if possible, and provide the 3-D boundary functionality. Hopefully, such package would then allow for easy integration into other Unity VR projects.

Chapter 2

Tools

The implementation part of this thesis makes use of a collection of hardware and software tools. Before proceeding further to the design chapter (page 15) the tools of this framework are to be introduced and rationale will be given as to why the used tools were chosen. The HMDs and 3-D scanners will be described first, followed by the software.

2.1 HMDs

As an interface to the VR during the experiments a HMD is used. The choice between two HMDs available in our lab had to be made which is to be discussed below. The two available devices are the Oculus Rift and the HTC Vive mentioned earlier. Both of them function similarly as visual-output devices. The user is presented with stereoscopic images which are displayed on a single or a dual OLED panel as side-by-side images, each side providing information for either eye. The display is seen through a pair of biconvex lenses. Both HMDs provide optical positional tracking, although they implement it differently.

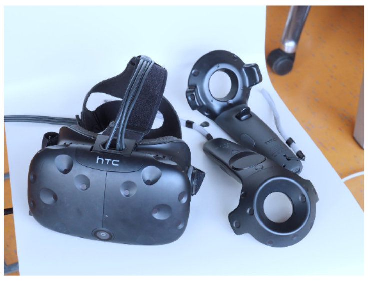
2.1.1 Oculus Rift Dk2

The version of the Oculus Rift available in our lab is the Dk2 (development kit 2). The Dk2 features a single AMOLED display with a resolution of 1920 x 1080 px and refresh rate of up to 75 Hz. Each eye views a half of the screen which accounts to 960 x 1080 px of per-eye resolution. The display is viewed through aspherical lenses[19] with alleged field of view of 100 °[1].

After much effort I wasn't able to find a definitive answer on what the actual horizontal FOV of the Dk2 is, nor does the cited source mention how was the FOV measured or whether it is horizontal, diagonal or vertical. However, [11] made a measurement suggesting horizontal FOV of about 84 °. While this can be measured and verified, I decided not to do that as the exact number isn't that important. Based on personal experience the horizontal FOV is lower in comparison with the HTC Vive.

The Dk2 has optical positional tracking and an internal positional tracking system consisting of an accelerometer, gyroscope and a magnetometer. The internal tracking provides additional precision, especially of measurements of rotation and orientation. The two main parts of the optical tracking system are infrared LEDs covering the headset, and a positional tracker. The positional tracker is an infrared sensor that is usually placed near a monitor while facing the headset. Both the tracker and the headset have a maximum range and FOV within which they need to face each other. The tracker has a 100 ° FOV and a range of two meters.[19] However, for the headset to be tracked, the tracker needs to be within 72 ° of the headset's FOV. Both devices are shown in figure 2.1a.

(a) Oculus Rift Dk2


(b) HTC Vive

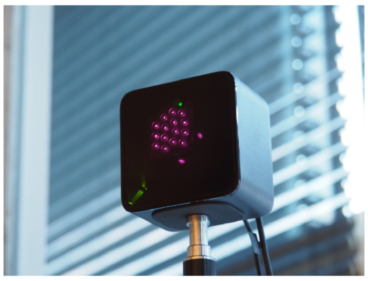

(c) HTC Vive Base Station

Figure 2.1: The HMDs and associated devices: (2.1a) shows a the Oculus Rift Dk2 and its motion tracker. Attached to the HMD is an unrelated ZED camera. (2.1b) The HTC Vive with wireless controllers. (2.1c) The SteamVR Base Station (a.k.a. the Lighthouse) for the HTC Vive on a stand.

2.1.2 HTC Vive

The Vive uses two AMOLED displays, each with resolution of 1080 x 1200 px (2160 x 1200 px in total) and a 90-Hz refresh rate. Through a pair of biconvex Fresnel lenses the user is provided with a horizontal field of view of 110 °. The Vive is supplied with two wireless controllers and tracking stations, all of them shown in figures 2.1b and 2.1c. The headset and controllers are positionally tracked by the SteamVR optical tracking system. Additional tracking precision of the headset is achieved using an inertial unit with an accelerometer, a gyroscope and a magnetometer.[10]

The SteamVR optical tracking system is what provides room-scale tracking for the HTC Vive. It uses passive infrared emitters and active sensors. Each tracked device is covered with infrared photodiodes which capture synchronizing pulses and swipes of laser beams emitted by the base stations. The signal of the voltage generated on the sensors is then used to calculate a position of the device based on timing information about the synchronizing pulses and swipes.[3] The base station system is scalable, but with usual two-base-station configuration the tracked area is about 5 x 5 meters.[10]

2.1.3 Comparison

As this thesis focuses on experimentation with and development of the play area boundary, the support for room-scale tracking and tracked controllers of the HTC Vive were considered as crucial. While the development with the use of the outdated Dk2 would still be possible, the benefit of a native room-scale tracking and tracked controllers for both the convenience of development and user tests made the Vive a clear choice. The HMD also provides higher resolution and wider field of view.

2.2 3-D data acquisition

The proposed approach makes use of 3-D positional data of the user's surroundings. To acquire such data some sort of 3-D scanning is required to take place. There are various types of scanner devices and processing methods.

Contact scanning involves a physical probe touching the scanned object and wasn't considered for the purpose of this thesis as it didn't seem suitable for either real-time usage, deployment on a consumer-product scale or scanning of an environment simply enough.

Non-contact scanning approaches don't rely on a physical contact with the scanned object or environment. Non-contact scanning can be either active or passive depending on whether or not the scanning conditions are being modified to allow for, or enhance, the scanning process, or not.

When scanning passively, only the information already present in the environment or coming from the scanned object to the sensor is used. Examples of such approach are methods based on obtaining 3-D data from image data captured by a single or a stereo camera. Active scanning on the other hand incorporates some sort of a wave or light emitting to aid the process, such as a laser line or a pattern. Active scanners have the advantage of being more accurate. Another advantage is that light-based devices don't depend on lighting conditions as much as passive scanners.[8]

2.3 3-D scanners

For the purpose of this thesis two non-contact scanner devices available in our lab were considered – one active, one passive. The Kinect[17] from Microsoft is the active scanner. The ZED[13], made by Stereolabs, is the passive one.

Several parameters of these devices were considered and compared to assess their suitability. In the next several paragraphs the two devices will be shortly described and their parameters listed, followed by the comparison.

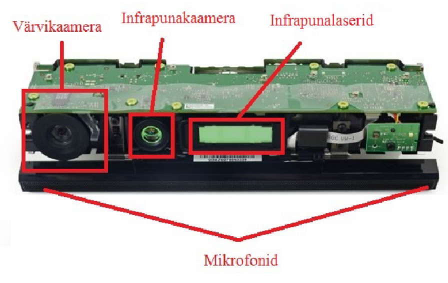
2.3.1 Microsoft Kinect V2

The device in question is the Kinect for Xbox One[17] (or a functionally identical Kinect for Windows v2). It is a now-discontinued successor to the first Xbox 360 Kinect. Hereinafter it will be referred to simply as „Kinect“ as the previous version won't be mentioned at all. Kinect is a peripheral input device sold as an accessory for the Xbox One console or a Windows PC. It can also be connected to a personal computer or other devices not running the Windows operating system and used for various purposes without Microsoft's software thanks to the existence of open source drivers[2]. It is used to track body movement and posture of one or several users as a means of providing control input for the console. For example the user might control an in-game character by moving their body in a certain manner while the in-game character mirrors their movement.

There are three integral parts of Kinect providing it with its functionality. A color camera, a depth sensor and a multi-array microphone. 3-D data can be acquired using the depth sensor. The depth sensor consists of an infra-red (IR) radiation emitter and an IR sensor, both of which can be seen in figure 2.2b.

Parameters important for the 3-D data gathering task are those of the depth sensor. The resolution the sensor provides is 512 x 424 px at 30 frames per second. It has a field of view of 70 x 60 ° (HxV) and operating range from 0.5 to 4.5 meters [18].

(a) Microsoft Kinect



(b) taken apart

Figure 2.2: Photos of Kinect for Xbox One. Figure 2.2a shows the whole device with only the color camera visible (source:[4]). In figure 2.2b (source: [15])the color camera is in the leftmost red square. The rectangular part in the middle is the IR emitter and between the aforementioned two is the IR sensor.

2.3.2 Stereolabs ZED

The ZED camera, as seen in figure 2.1a(attached to the Oculus Rift HMD), is a stereo camera made by Stereolabs[13]. The device itself doesn't provide any depth sensing features and can be handled the same way as any other USB video device class device. Processing of the stereoscopic image data is done on a USB 3.0 capable host machine (computer or other devices, such as NVIDIA Jetson TX2) by a software provided by Stereolabs as a part of their SDK. On the host machine the image/depth data processing takes place on a CUDA capable GPU making the process fast.

The depth data parameters heavily depend on the parameters of the cameras video output. There are several resolutions and frame rates the camera can be set to output and the resulting depth data (depth map) reflects them.

The resolution/frame rate combinations are as follows: 2208 x 1242 px @ 15 Hz, 1920 x 1080 px @ 30 Hz, 1280 x 720 px @ 60 Hz and 672 x 376 px @ 100 Hz. Field of view of the camera is 102.4 x 70 ° (calculated from 110 ° diagonal FOV and 16:9 aspect ratio) and operating range 0.5 - 20 meters[13].

2.3.3 Comparison

The Kinect provides superior precision thanks to its infrared time-of-flight solution, that's not dependent on ambient lighting conditions. However, the ZED seemed to be a better choice for several reasons. It has a wider FOV, which could be of use in the case of a real-time operation, as it covers more of the field of view available in the Vive HMD. Especially because the Kinect has been discontinued and the ZED seemed to be a more near-future-proof option of the two. The ZED is also less bulky.

2.4 Unity and packages

To carry out the software development a game engine called the Unity[25] was used. The Unity is widely adopted and it's well known among VR developers. It provides all the typical functionality of a game engine, such as scripting in C#, rendering, animations, scene graph and an excellent editor. Apart from that there is a great added value to the Unity in the ecosystem surrounding it. There is a comprehensive online documentation, tutorials, huge community and most importantly – packages. Unity supports so-called Unity packages which are collections of files that easily integrable into Unity projects. They can provide in-game models, additional functionality to the project or even the Unity's editor. The packages can be downloaded from a website called the Asset store, which is integrated into the Unity editor. They can also be imported into projects manually which is extensively used by developers who don't want to or are unable to supply their packages through the Asset store. For instance, a package can be used to add a support for a device as is the case with all the devices described in the sections 2.1 and 2.2. In both cases the provided packages are used as interfaces to the devices' external SDKs which are installed independently of Unity and aren't part of its ecosystem.

There are some key concepts in Unity projects that are to be referred to later in this text which will be now briefly introduced. A Unity project typically has several „scenes". A scene in unity is what can be pictured as a level in a game, it contains lights, cameras, models and so on. A main menu of a game is very likely to be its own scene as well.

Every object in a scene is an object of type `GameObject`. These are in a hierarchy and can be arbitrarily rearranged and nested. Every `GameObject` holds the so called components. There are many types of components and custom ones can be made. One type of a component that every `GameObject` has is `Transform` which defines the objects local position and rotation under its parent `GameObject` if it has one. Manipulating `Transform` of a `GameObject` will manipulate the whole hierarchy under this `GameObject` as expected. Any `GameObject` can also be saved as a file called a „prefab". Prefabs allow `GameObjects` to be stored along with their components and their state if that's possible which provides a convenient way of speeding up and packaging `GameObject` configurations and arrangements.

One other very important type of a component is „script". Scripts are just source code files containing classes that inherit from class `MonoBehaviour`. `MonoBehaviour` has many methods but the ones used most usually by scripts are `Start()`, `Update()`, `FixedUpdate()`. These are magic methods that the developer defines but which are called automatically on different occasions. `Start()` before the first frame is rendered, `Update()` every frame, `FixedUpdate()` every tick of Unity's physics system.

Unity also allows for custom shaders. These can be written either entirely in their shader language „Shaderlab" or only the top level of the shader is described in ShaderLab and the shader programs (e.g. vertex, fragment) are written in a variant of HLSL. A shader can consist of several sub-shaders which can consist of several „Passes". Passes are what is being rendered and what contains the individual shader programs.

A crucial component of any Unity VR project is the support for VR headsets (HMDs), positional/motion tracking and controllers. The support for the most popular HMDs, including the Oculus Rift and the HTC Vive, is provided by the SteamVR package[26]. The package also makes a VR application development easier thanks to settings, menus and other GUI elements that are consistent across different HMDs.

Another package is the VRTK (Virtual Reality Toolkit)[24]. It makes use of the SteamVR package to interface with the VR devices. This package implements a lot of common VR functionality such as locomotion, object manipulation, 3D controls (e.g. levers), haptic feedback and menus. Also included is the VR Simulator which enables for running and trying the VR application without the specific hardware connected. One of VRTKs features is object aliasing. Which is a VR-SDK agnostic way of assigning functionality to HMDs, controllers and others in a uniform and automatic fashion so that the developer doesn't have to assign this functionality to new objects every time they choose to use a different VR SDK. When later in chapter 4 adding components to `GameObject`s of controllers and the HMD is mentioned, this way is used instead of assigning the components directly.

2.5 ZED SDK and Point Cloud Library (PCL)

The manufacturer of the ZED camera provides a SDK and a Unity package as a way for Unity to interface with it. This SDK processes the video and provides access to the depth map and point cloud of each frame. It also provides a spatial mapping feature which captures a 3-D mesh of the scanned environment[12] that would be well suited for the task of creating the proposed boundary. All of my attempts to try this feature out in Unity have unfortunately failed with a crash of either Unity or the graphics card driver. Therefore, I decided to only work with point clouds.

As the improved boundary operates with point clouds, some kind of point-cloud processing is required. I wasn't able to find any solution that would be readily available for use in Unity. The best overall external solution seemed to be the Point Cloud Library (PCL)[22]. PCL specializes on point-cloud processing and provides a whole range of functionality, including filtering, segmentation, feature recognition, searching and surface reconstruction.

Unfortunately, the library is written in C++, while Unity heavily relies on the managed .NET C#. Wrapping the PCL functionality that would be useful for the improved boundary didn't seem realistic. Instead, a simple, less time consuming, custom solution was implemented. It provides a basic filter and limits the number of stored points by spatially sampling input points. The point-sampling makes use of a 3-D grid, which is inspired by a step in a fast Poisson disk sampling algorithm presented by Robert Bridson[7]. Both filtering and sampling are described in more detail in chapters 3 and 4.

Chapter 3

Design

This chapter sets out to explain the overall way I've designed the improved boundary, as well as giving a detailed explanation of the main ideas behind its components. Firstly, the overall approach will be summarised. Secondly, an insight will be given into the 3-D data acquisition and processing. Thirdly, the actual usage of the gathered data will be explained.

The improved boundary is a 3-D virtual boundary consisting of a point cloud, data of which are a result of a 3-D scan performed on the room around the play area. This point cloud is used for detecting collisions with any tracked devices and for providing the user with feedback based on these collisions. The structure is illustrated by a diagram in figure 3.1.

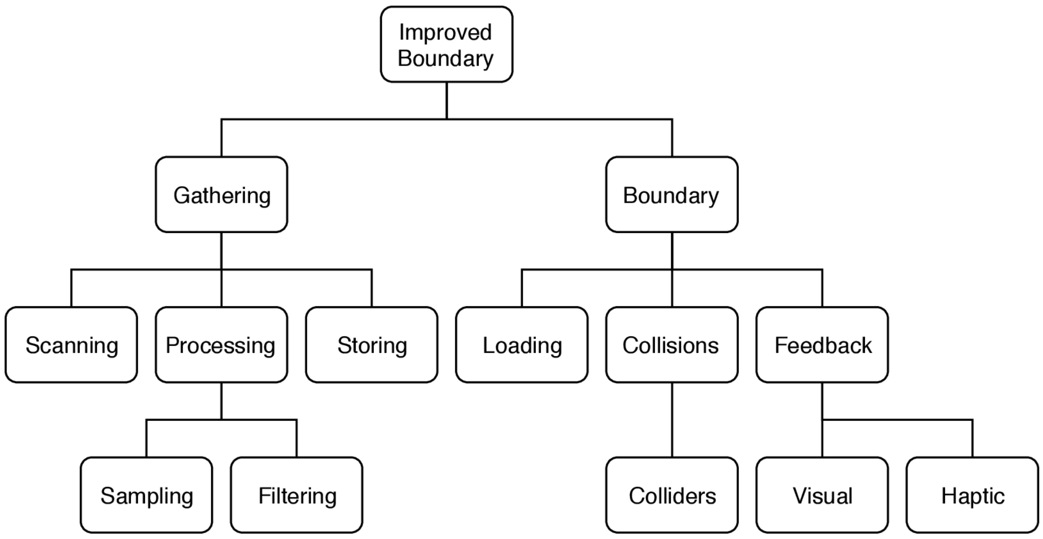


Figure 3.1: Diagram of the structural design of the improved boundary. Nodes are in chronological order (where applicable), from left to right.

3.1 Gathering

The 3-D data is first acquired using a hand-held ZED camera – a 3-D scanner. The depth maps provided by the 3-D scanner are processed into 3-D point cloud with positions in the scanner's coordinate system. These points are then transformed into the VE's world coordinates. To simplify the transformation and to enhance the precision of the scanner's pose estimation the scanner is attached to an HTC Vive HMD. Thanks to the SteamVR optical tracking the HMD's position can be measured quite precisely.

Many of the points are overlapping, many of them can be clustered inside a small area and their number is potentially infinite. To solve this issue the points are stored in a 3-D grid in a way, that samples the points with the size of elements of the grid. Each element can represent one point or none. This makes the maximum amount of gathered points depend only on the size of the elements and the grid. For instance, if an element had volume of 1 cm^3, then a grid that is 5 m^3 of volume will contain at most 5 million points.

After the scanning has finished, content of the grid is filtered, to reduce noise and remove any stray points, which would have negative effect later on, mainly on collision detection. Filtering is done using a simple algorithm based on neighbour counts. A point is filtered out if it doesn't have any neighbours that would have at least a certain number of neighbours themselves.

As the last step of the gathering process the filtered points are stored into a file.

3.2 Using the points

Once the points are processed and stored a VR application can make use of them by detecting any collisions between them and the user, and by conveying their existence and position to the user. The points are loaded from a file, usually at the start of the application.

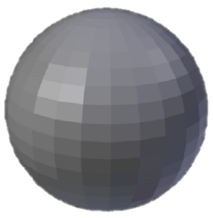
By default the boundary isn't visible unless being collided with to avoid lessening immersion. Only when a collision is detected and appropriate feedback is provided – visual and/or haptic. Collisions are calculated using points of the boundary and colliders representing tracked devices worn or manipulated by the user such as the HMD or controllers.

The colliders follow the positions of the tracked devices inside the VE. Both the collider positions and points of the boundary are in common world coordinates of the VE based on the same real-world tracking system. Therefore, whenever a collision in detected inside the VE it correspond to a „collision“ in real world. The improved boundary uses two shapes of colliders, to accommodate for slightly different needs of a HMD and other tracked devices. Tracked devices other than the HMD use a simple spherical collider, while the collider of the HMD is shaped like a half-capsule – it is a right circular cylinder with a hemisphere on top of it. Both the shapes can be seen in figure 3.2. These shapes have the advantage of being defined only by a position and radius.

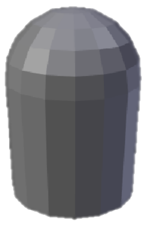
In the half-capsule collider the position defines both the centre of the hemisphere and height of the cylinder. The cylinder's bottom base always touches a virtual floor or other in-VE object the user is standing on which is level with real floor. This is to provide a whole-body collision detection mechanism in the absence of on-body trackers. A spherical collider would be insufficient for any obstacles below waist level or, if it had large enough radius to touch the ground, it would be both too wide and too tall to be useful.

Collisions are calculated between points and colliders. In this context a collision is a situation in which a point is situated inside a collider. Given the geometrical shapes of the colliders the calculations required to determine collisions are reduced to a comparison of the distance of the point to the collider's centre and the collider's radius.

If a collision is detected the points of the boundary are made visible and in case of the controllers haptic feedback is also provided by the respective controllers that triggered the collisions. Both visibility and haptic-feedback intensity are based on the proximity of the closest point participating in the collision.



(a) One type of a collider is a simple sphere.



(b) A half-capsule shape

Figure 3.2: Collider shapes, a sphere in figure 3.2a and a half-capsule shape consisting of a right circular cylinder and a hemisphere on top of it in figure 3.2b.

Chapter 4

Implementation

This chapter focuses on the resulting implementation of the improved boundary concept. Most of the chapter discusses implementation details of the improved boundary and is organised in a manner similar to chapter 3. Broad and common information that is not specific to any particular component is discussed first, followed by sections more focused on individual components. These are organised chronologically to follow a typical use case roughly corresponding to the third row of nodes in figure 3.1. At the end of the chapter a short description of a typical use case will be given.

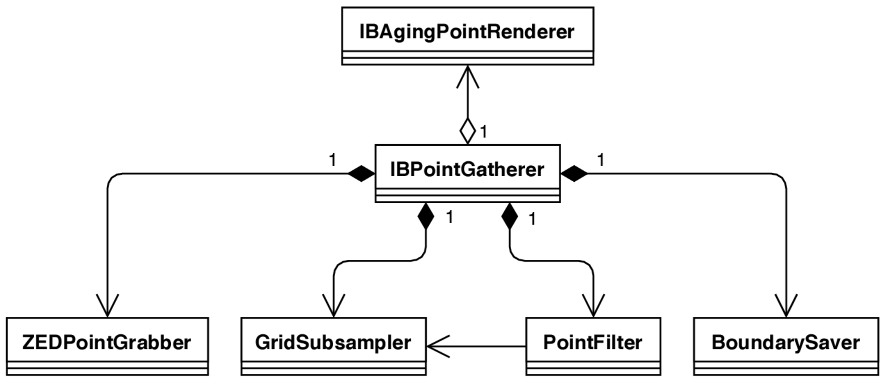
The boundary is implemented in Unity and the source code is written in C#. The whole solution can be thought of as being binarily dividable in two different ways as follows. The C# program source code files and the rest of the solution, the part concerned with point-gathering and the boundary part. The C# program source code files will be hereinafter simply called „classes“, as them being files containing the classes holds little significance.

The classes can be divided into two categories based on whether they inherit from `MonoBehaviour` and therefore can be assigned to a `GameObject` as a component. Name of all the classes that are intended as components is prefixed with „`IB`“, which stands for „improved boundary“ to make a distinction from other similarly named components when working within the Unity editor. The remainder simply relies on the use of namespaces. The classes and the way they depend on each other will be explained in the following sections while figures 4.1a and 4.1b help to illustrate this visually.

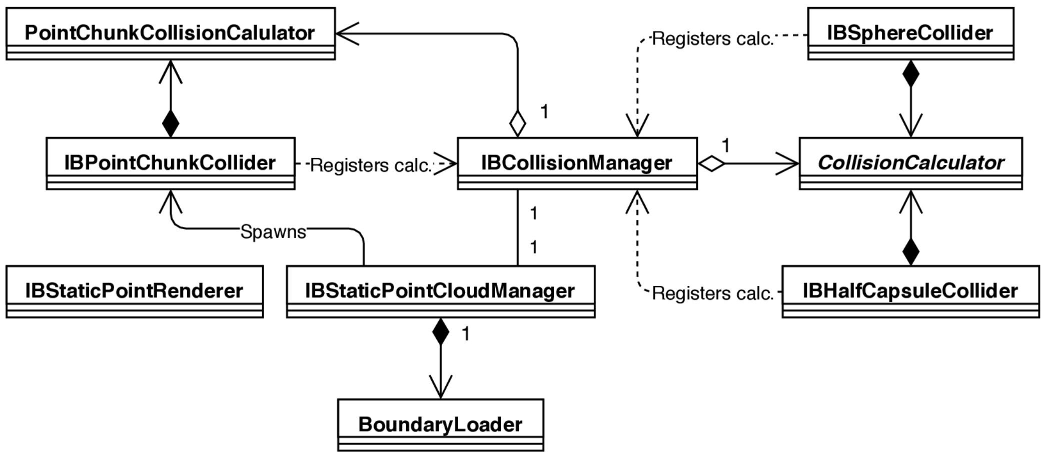
The rest of the solution consists of hardware arrangements, a shader program, scenes, `GameObject` components and other elements and systems within Unity and their interactions and connections. All of the important ones will be described in the following sections as well.

4.1 The gathering process

The process of gathering point for their later usage consists of scanning, sampling, filtering and storing the points in a file. This process is controlled by the `IBPointGatherer` class which makes use of other classes as illustrated in figure 4.1a. The process with everything required is showcased in scene `Scanning`. For `IBPointGatherer` to function properly an instance of `ZEDManager` exist somewhere within the scene which would typically be provided by the `ZED_Rig_Mono` prefab. Both of them are provided by ZED SDK Unity package.



(a) classes involved in the point-gathering process



(b) classes involved in utilising the points

Figure 4.1: Incomplete class diagrams illustrating mainly what classes interact with each other during the point-gathering process and later when the points are used as a part of the improved boundary. Classes that can be assigned to a `GameObject` as a component are prefixed with „`IB`“.

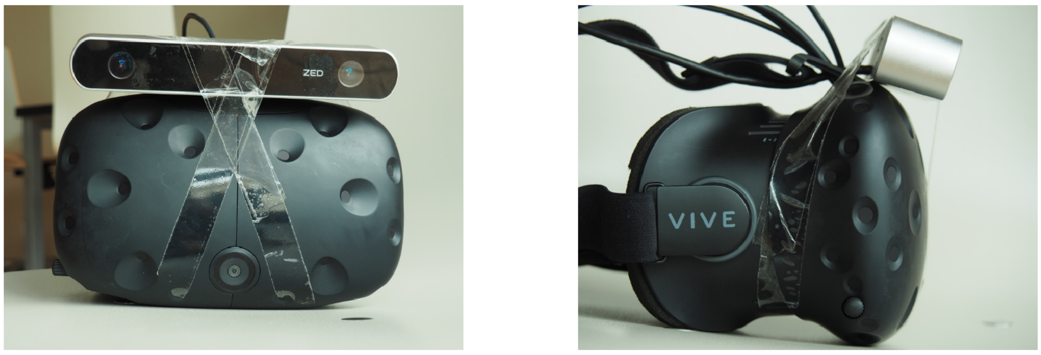
4.2 Scanning

When scanning is initiated `IBPointGatherer` first creates an instance of `ZEDPointGrabber` and then periodically requests new points each `LateUpdate`. These points are all calculated from the same frame. `ZEDPointGrabber` return points that are already in the world coordinates. These are then stored in a 3-D array for further processing as discussed in the following section. Points of each „grab“ are also sent to `IBAgingPointRenderer` which renders them in the scene, until a certain amount of time passes or the points are replaced with new ones. How points are rendered is described in more detail in section 4.7.

When `ZEDPointGrabber` is asked to return a new batch of points it first uses ZED SDK to get an array of points that are in the ZED camera's coordinates (its left camera's). Points that were properly measured and are valid with good enough confidence are transformed to world coordinates based on the pose of the HMD and the local pose of the ZED that is attached to it. Both finding the pose of the ZED camera in world coordinates and calculating the transformation were solved by creating a `GameObject` and making it a child of the

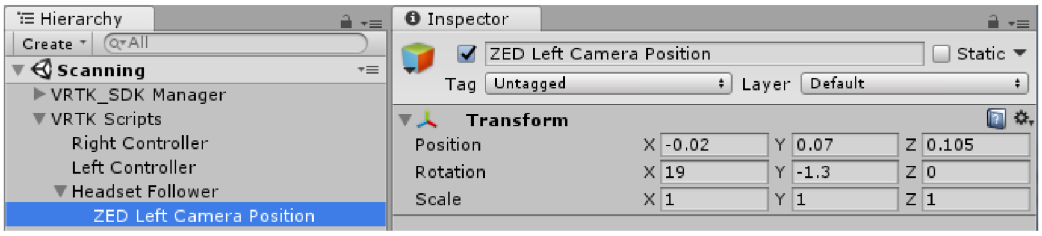
`GameObject` that represents the HMD. The former then has its local position and rotation set to that of the ZED related to the HMD. The object's `Transform` component provides a method that calculates transformations of positions from local to world coordinates which is the exact transformation that is needed for the scanned points.

The local position and rotation values for the ZED camera were approximately estimated to the values shown in figure 4.3. The left-hand side of this figure shows to parent-child relationship. Note that `ZED Left Camera Position` isn't parented directly by the HMD until the application is run and SteamVR detected and running because a work flow of VRTK was followed. The ZED camera is attached on the top of the HTC Vive HMD as shown in figure 4.2. The virtual HMD's position was experimentally found out to located in left eye's position inside the real HMD. From the side view this is about where the circular hinged joints are as can be seen in figure 4.2b.



(a) front view (b) side view

Figure 4.2: The ZED camera attached to the HTC Vive HMD. Figure 4.2a shows both from the front. Figure 4.2b better shows the position and rotation of the ZED relative to the HMD.



Hierarchy
Create
Scanning
- VRTK_SDK Manager
- VRTK Scripts
  - Right Controller
  - Left Controller
  - Headset Follower
    - ZED Left Camera Position

Inspector
ZED Left Camera Position Static
Tag Untagged Layer Default

Transform

Position	X	-0.02	Y	0.07	Z	0.105
Rotation	X	19	Y	-1.3	Z	0
Scale	X	1	Y	1	Z	1

Figure 4.3: Screenshot of the „ZED Left Camera Position“ `GameObject` in the Unity editor's inspector. On the left-hand side it is shown that the `GameObject` is parented by VRTK's headset alias the „Headset Follower“. Right side shows the object's `Transform` component and parameters of its pose relative to its parent.

4.3 Processing

As points are being scanned they are being fed to an instance of `GridSubsampler` which spatially samples the points. `GridSubsampler` contains a 3-D array which represents the 3-D space being scanned. This array functions as a grid over the space and divides it into cubes. The dimensions of this grid depend on the set precision – the size of the cubes – and volume of the 3-D space that's relevant to the boundary. Physical dimensions of the grid are set by the user before points start being gathered. Cubes with edges of 2.5cm seemed sufficiently small to allow for relatively accurate and dense point-cloud boundary, while big enough not to consume too much memory. For instance, an area of 6m x 6m 3m high would require the array to have 6912000 elements with said precision as shown in equation 4.1.

$$\frac{6}{0.025} * \frac{6}{0.025} * \frac{3}{0.025} = 6912000 \tag{4.1}$$

Each cube holds a counter and represents scanned points with positions that are inside of it. Whenever `GridSubsampler` receives new points it finds what cube each point belongs to and increases the counter. After scanning has ended the `GridSubsampler` is passed to `PointFilter`, which ignores all points whose counter didn't reach a certain threshold. This itself filters out anomalies such as stray points. The threshold depends on volume of the cubes.

All the cubes suitable for further filtering are then filtered using a simple algorithm based on neighbour counts. For a cube to pass it must have a neighbour that has at least 14 neighbours itself. When a point passes it is added to a list of passing points.

It seemed reasonable to filter out about 10 % of points as keeping less than 90 % didn't noticeably improve the result and keeping much less than that was counterproductive as it removed details such as corner points. Second-degree neighbour counts were chosen because first-degree neighbour counts were distributed somewhat evenly throughout the points and therefore didn't provide much information. This is shown in figure 4.4a. Distribution of the second-degree neighbour counts suggested much more correlation between the neighbour counts and points being erroneous as shown in figure 4.4b. The figure also shows that 90 % of all points have a neighbour with 14 or more neighbours. Multiple measurements were taken and while the first-degree neighbour count distribution has occasionally varied, second-degree neighbour count distribution was significantly more stable and overall didn't change.

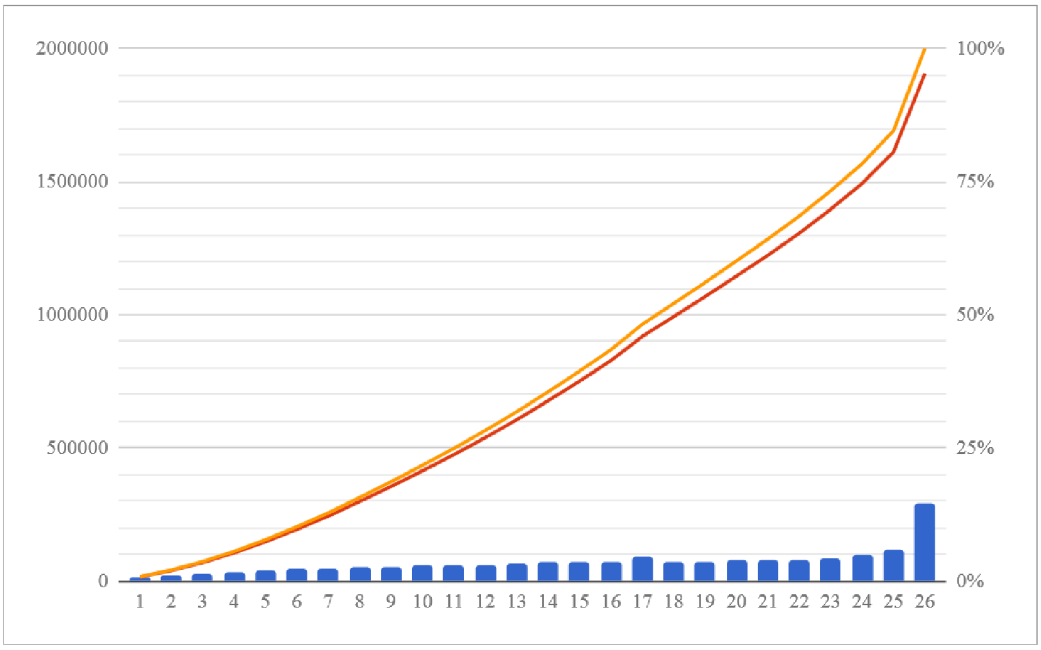
4.4 Storage

After the filtering has ended the resulting point list is passed by `IBPointGatherer` to `BoundarySaver` which writes it to a plain-text file. By default this plain-text file is named `boundaryVertices.off`. Each point is written down on a separate line as a triplet of floating-point numbers separated by spaces. These numbers represent x, y and z coordinates respectively.

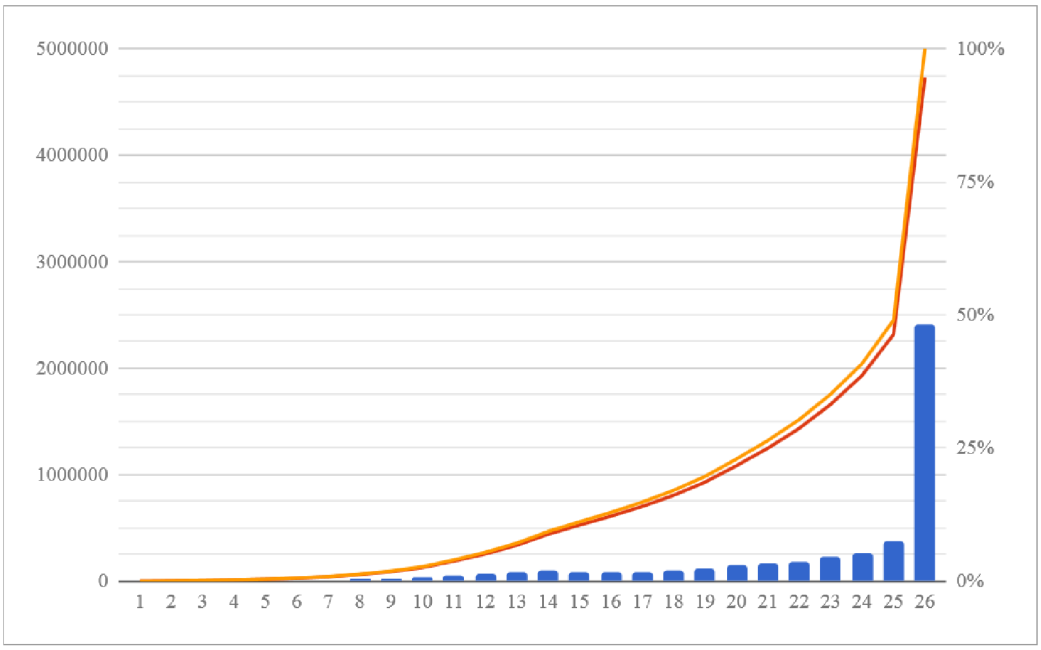
By storing the processed points the gathering process ends. When the points are needed again they are loaded using `BoundaryLoader` which just reverses what `BoundarySaver` does.

4.5 The improved boundary

After the gathering process has finished the points can be used in the improved boundary. The boundary is controlled by `IBStaticPointCloudManager` which initialises it and is later used by `IBCollisionManager` to set visibility of the boundary. When the boundary is initialised `IBCollisionManager` periodically calculates collisions as explained in the following section. The main classes and their dependencies involved in functioning of the boundary are illustrated in figure 4.1b.



(a) neighbours



(b) maximum neighbours of neighbours

Figure 4.4: Neighbour counts on the horizontal axis and number of points on the vertical axis. Blue bars show the point numbers, orange coloured are cumulative point numbers and yellow is also cumulative but is described in percentage. Figure 4.4a is based on the number of neighbours, figure 4.4b is based on maximum neighbour counts of point's neighbours.

The initialisation begins with the points being loaded using `BoundaryLoader`. They are loaded by chunks. For each chunk `IBStaticPointCloudManager` creates a new `GameObject` to which `IBPointChunkCollider` and `IBStaticPointRenderer` components are added. These are given the point chunk. After that the `IBPointChunkCollider` component registers its `PointChunkCollisionCalculator` in `IBCollisionManager`. Both of these will be explained in the following section.

4.6 Collisions

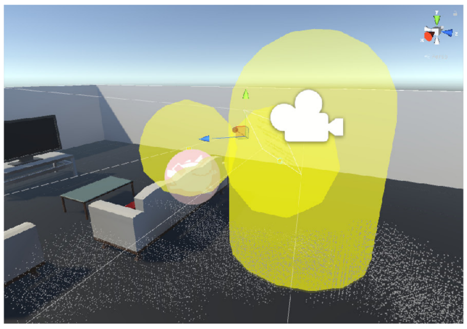
Collisions are managed by `IBCollisionManager`, which periodically calculates collisions using collisions calculator classes. It does so during `FixedUpdate`. At startup, collision calculators are added to `IBCollisionManager` by their respective colliders that created them and own them. There are three types of colliders and each uses a different type of a collision calculator.

One type of a collider is `IBPointChunkCollider`. When a point chunk is created `IBPointChunkCollider` creates an instance of `PointChunkCollisionCalculator` that it then passes to `IBCollisionManager` which adds it to a list.

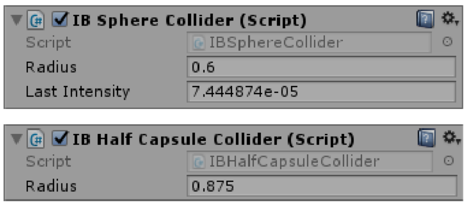
Both the second and third type of colliders use collision calculator classes that inherit from abstract `CollisionCalculator`. The second type is `IBSphereCollider` which also passes its collision calculator of type `SphereCollisionCalculator` to `IBCollisionManager` which then again adds it to a different list. The third one is `IBHalfCapsuleCollider` with `HalfCapsuleCollisionCalculator` that is also passed to `IBCollisionManager`. These are kept in the same list. For the collisions to be calculated properly even when the user moves inside the VE without moving in the real world, such as by teleporting, the collision calculators are given a reference to the VR rig `GameObject`'s `Transform` component by `IBCollisionManager`. When calculating collisions as explained on page 23, positions of the colliders that are used are relative to the `Transform`. The proper `Transform` is assigned to the `IBCollisionManager` manually in the inspector of Unity editor.

`IBHalfCapsuleCollider` and `IBSphereCollider` are manually added as components to their respective `GameObjects` in Unity editor. The former is added to the one representing the HMD while the latter type is added to the controllers or possibly other tracked devices. Example of this can be found in scene `Test1-Boundaries`. Both of them are shown in figure 4.5a. The colliders have configurable radius as shown in figure 4.5b. Radii of 0.6 m for the spherical colliders and 0.87 m for the half-capsule one ended up being adequately large to pro provide feedback soon enough not to bump into obstacles yet not forcing the boundary to be constantly visible etc.

Collisions are calculated in the following manner. First, `IBCollisionManager` passes the `CollisionCalculator` list to each `PointChunkCollisionCalculator`. These then iterate over all of their points and ask all of the `CollisionCalculators` they received in the list if the points collide with them. `SphereCollisionCalculator` calculates that by comparing the distance of the point to its radius. `HalfCapsuleCollisionCalculator` does the same for any point that is above the centre of its hemisphere and for any points below that it calculates the distance using only two dimensions – x and z. Each `CollisionCalculator` keeps statistics of its collisions each time collisions are calculated. Most importantly it remembers distance of the point of its closest collision along with a relative distance that is divided by the collider's radius. The distance is also used in calculating collision intensity of individual `IBSphereColliders` which is then used for haptic feedback as described in the following section.



(a) colliders



IB Sphere Collider (Script)

Script	IBSphereCollider
Radius	0.6
Last Intensity	7.444874e-05

IB Half Capsule Collider (Script)

Script	IBHalfCapsuleCollider
Radius	0.875

(b) collider components

Figure 4.5: Figure 4.5a shows collider gizmos as being drawn in the scene view of Unity editor. There is one half-capsule collider and two sphere colliders. One of them is further away as the hand holding the controller it is attached to is extended. Figure 4.5b shows the collider components with configurable radii.

Based on the closest relative distance among all `CollisionCalculators`, visibility of the boundary is calculated by `IBCollisionManager` which then sets visibility of a material that every `IBStaticPointRenderer` uses for rendering, as will be described in the following section.

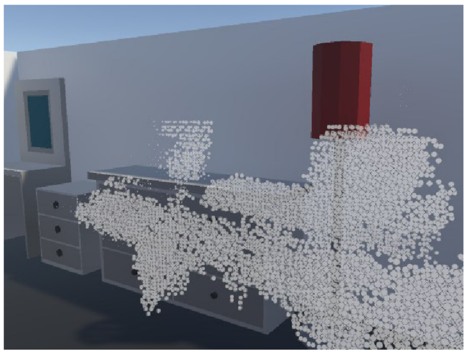
4.7 Feedback

The improved boundary provides feedback for the user in two ways. First, there is visual feedback – the points of the boundary become visible if any collision occurs – and second, haptic feedback is provided – the controllers vibrate if their collider measured a collision.

The visual feedback is driven by `IBCollisionManager` which, based on relative distance of the closest collision among all colliders, uses `IBStaticPointCloudManager` to set `Visibility` parameter of material `StaticPointOverlayMaterial`. Visibility is set to 1 when the minimal relative distance is below 0.25. From 0.25 to 1, which is the maximum, it's scaled linearly down to 0, as shown in figure 4.9a.

The aforementioned material uses a custom shader with several configurable properties, one which is `Visibility`. While it isn't used to set an alpha value it acts in a similar manner and controls the points' opacity.

The custom shader is `GSBillboardOverlayShader`, it renders semitransparent points in the following way. The points are rendered as disks that are always facing the camera as shown in 4.6a. When the points are not in a direct line of sight of the camera, such as when they are behind or inside a virtual object, whatever is blocking the line of sight is seemingly made semitransparent so the points can be seen. Figure 4.6b shows this. Points are being rendered only up until a certain distance from the camera. When points simply appear and disappear in front of the user's eyes it draws their attention, to avoid this the points at the edge of the rendering distance shrink out as their distance increases and the other way around. The closest points cover points that are behind them.


(a) boundary points

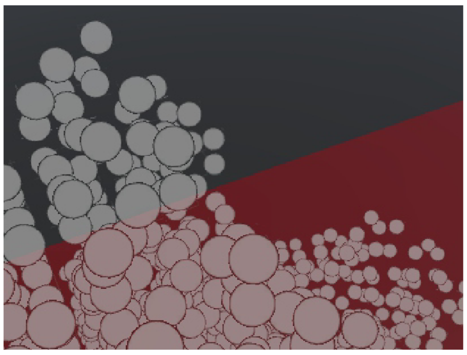
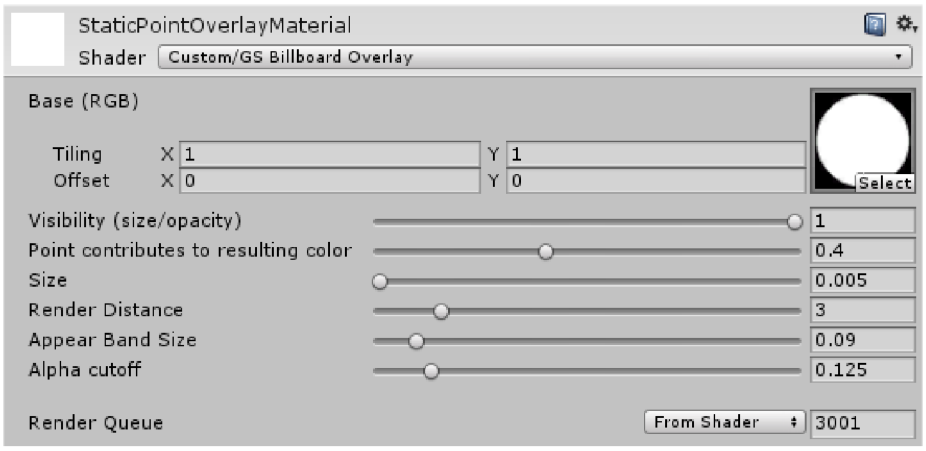

(b) points up close

Figure 4.6: Figure 4.6a shows how points of the improved boundary are rendered. Figure 4.6b shows a close-up screenshot of boundary points at the edge of a virtual bed[1] which is red while ground is grey. Small points in the bottom right are actually below the bed.

The shader is configurable and has properties that can be changed during run time. However, the only one that is changed is `Visibility` property that is indirectly modified by `IBCollisionManager`. All the properties are shown in figure 4.7.


StaticPointOverlayMaterial
Shader Custom/GS Billboard Overlay

Base (RGB)

Tiling X 1 Y 1
Offset X 0 Y 0

Select

Visibility (size/opacity)	1
Point contributes to resulting color	0.4
Size	0.005
Render Distance	3
Appear Band Size	0.09
Alpha cutoff	0.125

Render Queue From Shader 3001

Figure 4.7: Screenshot of the shader's inspector in Unity editor shows all the options it has.

Property `Render Distance` is used to set rendering distance. Limited rendering distance is implemented in vertex shader, which calculates distances of vertices to a line that connects the camera to the floor and discards any that are further away than the set rendering distance. One end of the line is the camera position with the other end being the same position with the y(height) coordinate set to 0. This is calculated analogously to collisions in `HalfCapsuleCollisionCalculator`, meaning 3-D distance is calculated above the camera and 2-D distance below it. In effect this only renders points inside a half-capsule shape such as the one shown in figure 3.2b. This shape seemed more reasonable than a simple distance from the camera as it allows the user to see boundary points down to the floor level where their legs are. Property `Appear Band Size` controls what part of the rendering distance is used for fading, or rather gradual increase in size of points as they get closer.

[1] Everything that is shown in these screenshots that's a part of the VE comes from a free „Simple Home Stuff" asset by https://twitter.com/mohelm97 available from the Asset Store.

In this distance band the size increases linearly from 0 at the very outer edge to full size at the inner edge. This is implemented by changing distances between vertices generated in a geometry shader.

The geometry shader expands each input vertex into four vertices making it a square made of two triangles in a triangle stream. This square is then used to render each point by applying a disk texture. Thanks to the geometry shader the vertices passed to the shader can be the points themselves without any further preprocessing.

The most challenging proved to be implementing the effect shown in figure 4.6b. The difficulty is in that with this many points it is inevitable that some are rendered in front of the others, thus the points need to be correctly z-ordered. However, the points need to render even when inside or behind a virtual object, which at the same time some requires some kind of blending to take place to allow for rendering of the points behind the object in a semitransparent manner. This is important because rendering opaque points inside other objects makes it look like a graphical glitch, especially when the user is presented with a stereoscopic image and can look at them from different angles.

This was solved by having three passes inside the shader. First pass stores current content of the frame buffer to a texture. In the second pass Z-testing is turned off, all the points are rendered and each of their fragments sets their depth-buffer pixel to a value that means „as far as possible“. During the final pass the points are rendered with z-testing turned on which makes them order correctly between each other. Each fragment in the final pass then uses its colour and blends it with the corresponding colour stored in the texture from the first pass. Currently the blend function is a simple addition of the point's weighted colour from the one of the stored texture. The weight which is the extent to which the point's fragment colour contributes to the output fragment colour is controlled by property `Point contributes to resulting color`. However, as the points are mainly white they don't actually change the hue of resulting colour most of the time. This poses a problem when the original colour is already bright. Therefore, if any component of the original colour reaches a threshold of 0.4 subtraction is used instead of addition. That produces the effect shown in figure 4.8. There is room for improvement but it guaranties high-enough contrast of the points in the VE. If the points were of different colour that would impact hue more profoundly there would still be some complications if two similar colours were being blended.

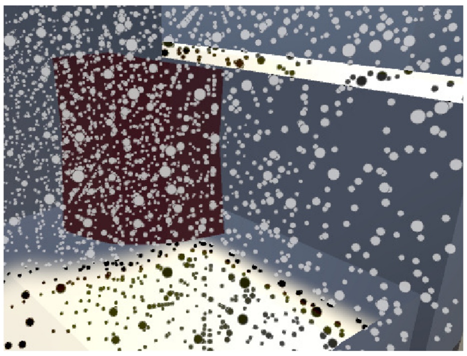
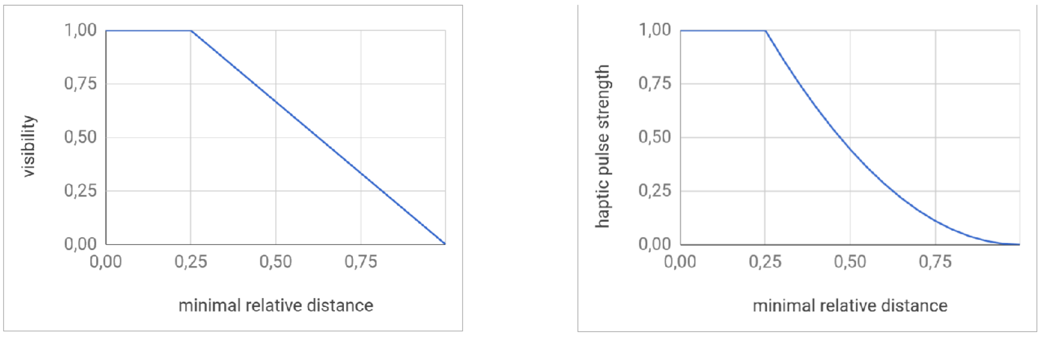


Figure 4.8: Points render differently depending on the lightness of the colour they blend with to ensure high enough contrast. Figure shows a detail of points and a sofa with dark and brightly lit areas.

Haptic feedback is controlled by `IBSphereColliderControllerHaptics` which uses VRTK to interface with SteamVR to send haptic-pulse requests to the controllers. This component requires a `GameObject` to have a `IBSphereCollider`. When it's added to a controller's `GameObject`, it asks `IBSphereCollider` for the intensity of its last collision. It uses the intensity to control strength of haptic pulses, which rises quadratically down to a relative distance of 0.25 where it reaches its maximum as is shown in figure 4.9b. The shape of the curve is quadratic as the feedback felt more natural than when the increase was linear.



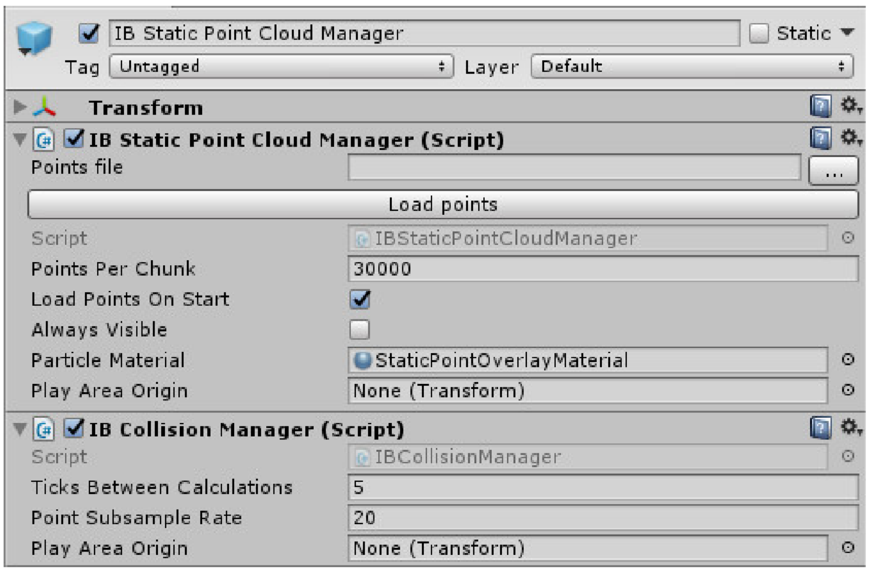
(a) visual response (b) haptic response

Figure 4.9: Response curves of the visual and haptic feedback. Figure 4.9a shows how visibility of the boundary depends on the relative distance of the closest collision over all colliders. Figure 4.9b shows dependence of intensity of haptic feedback on the minimal relative collision distance of the collider it is related to.

4.8 Usage

From the viewpoint of a user the current implementation of the improved boundary is used as follows. The users runs an application that will be used for point gathering. Then they take a controller and walk to the centre of their desired play area and set the maximum gathering area by raising their hand, pulling the trigger of the controller, walking and reaching to the part of the desired play area that's furtherest from the centre and releasing the trigger. The user then needs to take the HMD with the ZED attached, activate gathering by pressing the touchpad on the controller and slowly turn and walk around to scan their surroundings. When they've finished the touchpad needs to be pressed again. With that the scanning process ends. The user just needs to wait for the processing to finish. After that they can use the improved boundary in another application, such as a game, in which they are provided with visual and haptic feedback based on their proximity to the boundary.

From a developer's point of view using the improved boundary only requires the developer to import a package, drag the `IB Static Point Cloud Manager` prefab into their scene, set two `Transform` references to that of the VR rig and assign colliders to controllers and the HMD. The prefab is shown in figure 4.10. If haptic feedback is needed it can be added by adding `IBSphereColliderControllerHaptics` component.



IB Static Point Cloud Manager   Static
Tag: Untagged   Layer: Default

Transform

IB Static Point Cloud Manager (Script)

Points file	
Script	IBStaticPointCloudManager
Points Per Chunk	30000
Load Points On Start	✓
Always Visible	
Particle Material	StaticPointOverlayMaterial
Play Area Origin	None (Transform)

IB Collision Manager (Script)

Script	IBCollisionManager
Ticks Between Calculations	5
Point Subsample Rate	20
Play Area Origin	None (Transform)

Figure 4.10: `IB Static Point Cloud Manager` prefab in Unity editor's inspector. It has the two main `IB` components. Both have a `Play Area Origin` field where one is used to set a reference for rendering the other for collision calculations.

Chapter 5

Evaluation

The implemented solution was evaluated based on a series of three test scenarios. These scenarios focused on different aspects of the boundary. The first tested if the boundary in fact works at all and at the same time serves can be compared to the Chaperone in the setting. The second scenario tested if users are able to avoid obstacles that are inside the play area as well as being part of the boundary around it. The third test was trying to find out how confident users are manipulating with or near a virtual object that is intersected by a real object.

Twelve people participated in the tests, all of them students of our faculty within the age of 19 to 25. Two of them were women. After participating in the test scenes the students were asked several questions. Some of these were binary, some were a choice of a number from within a range from 1 to 5 and some of them were open questions. There was also some objective evaluation made but it is limited to a single one as it proved to be quite difficult to design any kind of an objective metric for this kind of testing. The participants were also asked about their experience with VR in general. Subjective assessment of the tests comprises of answers to the following questions:

1. What is wrong with the improved boundary?
2. What is good about it?
3. How would you improve it, if you would?
4. How much were you concerned that you'd hit a real-world obstacle?
5. Quantify how large the play area felt with the improved boundary compared to the Chaperone.
6. Quantify how immersive the improved boundary felt compared to the Chaperone.
7. Which one of the two do you prefer?

Questions 4-6 were answered with a number between 1 and 5. For question 4 number 1 had the meaning of „not at all“ while 5 meant „absolutely certain“. In questions 5 and 6 the values 1-5 had the same meaning: „much worse“, „worse“, „the same“, „better“ and „much better“ than the Chaperone. Question 4 was asked for every test.

All of the questions were asked and explained to the test participants carefully until it was certain they understood the questions while care was taken not to influence them with my own views and expectations.

The only objective assessment had a straightforward metric, which was whether or not the test subjects bumped into or hit an obstacle while performing their task.

5.1 Test scenarios

All the tests took place in the same room. The play area was surrounded by desks, chairs, a coat hanger and a robot. The setup of the room slightly changed between each test to accommodate for the different scenarios. In every test the test participants were given a task. The first two tests made the participants walk, the third was stationary and required no walking. In the walking test scenarios the participants could only move using their own body and rotate themselves by 90 deg inside the VE using a controller. There was no teleport or gamepad functionality provided. This way of movement required the test participants to reorient themselves in the real world every time they rotated their camera for them to be facing the same way in the VE as before. In the following several paragraphs each test scenario will be described followed by the answers and measurements. After that a summary of the results will be provided and some conclusions made.

The first test scenario served mainly to find out if the improved boundary is sufficient and in fact usable and to provide a direct comparison between SteamVR's Chaperone and the point-based improved boundary. In this test the task was to locate a virtual object and move it to a target location. The room, in which all the tests took place, was set up in such a way that obstacles were only around the play area. Figure 5.1a shows the setup during the third test, in the first test the chair in the middle and the one to the right of it weren't present. The test participants were first asked to perform the task with the Chaperone turned on, and after completing it they were asked to perform it again but this time the Chaperone was replaced by the improved boundary.

The second test scenario consisted of a slightly modified task present in the first test. This time a chair was added to the room setup. In figure 5.1a it is the chair in the middle, the one to its right still wasn't present in the second test scenario. Figure 5.1b shows the improved boundary in this test from a top-down orthographic perspective. The test participants performed this task only once, with the improved boundary, as attempting it with the Chaperone which couldn't have been aware of the chair in the middle of the play area had an easily predictable result and would pose an unnecessary health hazard. The purpose of this test scenario was to find out how well the test participants are able to avoid an obstacle that's inside the play area and to find out whether they walk around it and use the free space as usual or if they avoid passing it and stay in one part of the play area, etc.

The third test scenario was focused on close-distance object manipulation near virtual objects intersected by real objects represented by the boundary. In this scenario a table and a chair were present in the VE. On the table there were 6 cubes and a piece of cheese. The test participants were asked to build a pyramid on the chair using the 6 cubes and the piece of cheese on the table. There were real objects inside the virtual table and chair – the chairs in the middle of figure 5.1a. In the VE this situation looks as shown in figure 5.2.

5.2 Results

Of the 12 test participants 5 had no experience with VR, 2 have tried it before and 5 of them had spent more than an hour in VR. This didn't seem to have too much influence on the rest of the results but some of the test participants with no experience needed a few

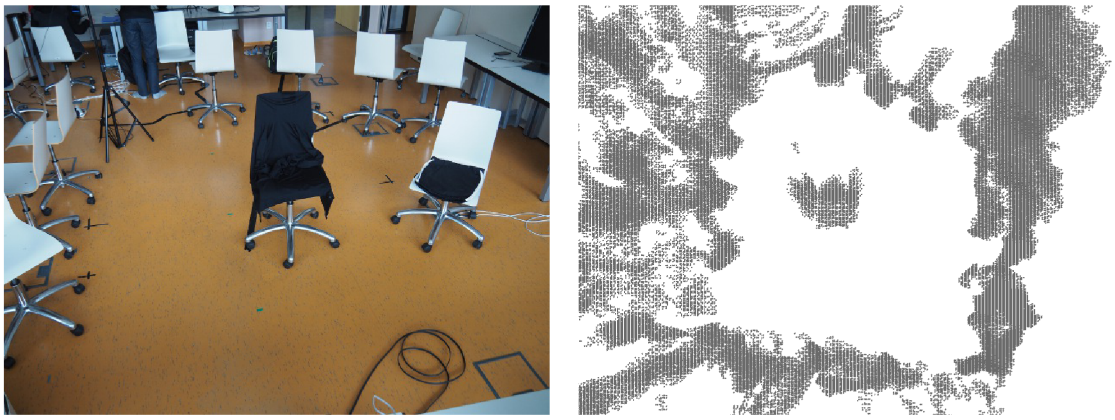

(a) room setup of the third test (b) the second test – points from above

Figure 5.1: Setup of the room the tests took place in. Figure 5.1a shows a photo of the room setup of the third test. In figure 5.1b points of the boundary in the second test are viewed from above using orthographic projection.

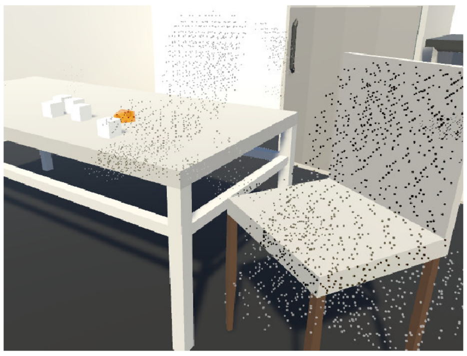


Figure 5.2: The table, chair and small objects involved in the task of the third test with the points of the boundary going through both pieces of furniture.

minutes to get accustomed to VR and the improved boundary. There seemed to be some slight correlation between lack of experience, carelessness and some of the negative results. Enough to be noted but not enough to draw any conclusions.

The rest of the questions will now be discussed in the same order as in the list above. To the question about what negatives the improved boundary had, 5 test participants didn't say anything. Four said that sometimes it was difficult to recognise the edge of the play area because there were holes between the obstacles the improved boundary is based upon and that they would appreciate a mechanism to tell them they shouldn't go any further such as a wall. Two test participants found the boundary to sometimes not be accurate and noisy. One didn't enjoy the appearance of the boundary and said it didn't fit well in the VE and one test participant thought the improved boundary requires more attention and effort to evaluate when inside VR.

When asked about the positives 8 of the test participants said they enjoyed that they had a good idea about where the obstacles were, that they didn't fear hitting anything and knew what they could afford. Three of them felt safer in that regard. Three thought the improved boundary felt less intrusive or more natural than the grid of the Chaperone, at least one of them (I'm note sure how many as I forgot to take notes of this) said they were able to almost completely ignore the boundary while still using it unconsciously. Two test participants commented that the boundary allowed them to utilise the space near the boundary, especially above it. Haptic feedback was also appreciated by two.

In response to the request for suggestions for improvements 4 test participants suggested that the precision of the boundary be increased to better reflect the actual shapes of the obstacles. Three test participants didn't think of any suggestions. Two proposed a real-time obstacle detection especially to avoid hurting pets. Two suggested that it might be worth experimenting with a boundary based on meshed obstacles instead of a point cloud. Two recommended increasing contrast of the boundary's points or otherwise making them stand out more. Finally, one test participant suggested making a clear visual distinction between points inside and outside of virtual objects to avoid confusion.

Overall the test participant were concerned very little that they'd bump into or hit something. This is demonstrated on a chart in figure 5.3. This shows a high degree of their confidence in the improved boundary. The averages over all three tests were: 1.25, 1.4 and 1.36 for a total average of 1.33 which is a third of the way between „not concerned at all“ and „concerned a little“. One person couldn't attend the second and third tests and one answer for the second test was unfortunately lost, so there are only 10 and 11 answers respectively for these tests. Three test participants stated that during the third test – close-distance object manipulation – the improved boundary was confusing as it was difficult for them to tell whether the points were inside or outside a virtual object. Three test subjects felt safer compared to the Chaperone. Two said that haptic feedback was a significant beneficial factor that gave them further confidence.

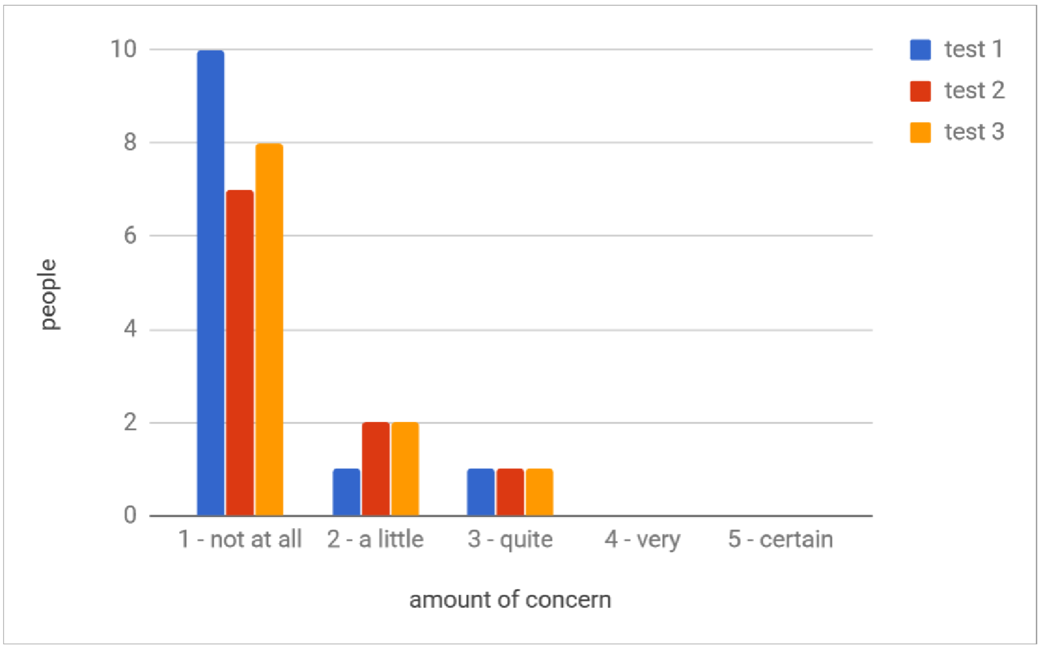
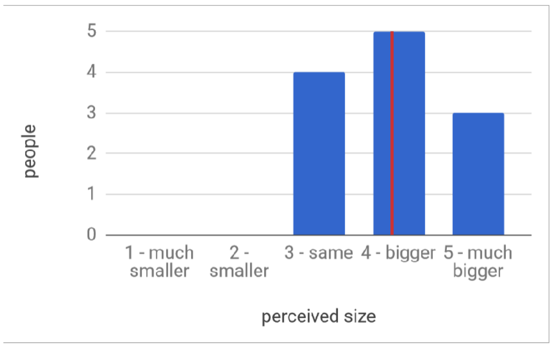


Figure 5.3: A chart showing quantified amount of concern of hitting or bumping into an obstacle expressed by the test participants. The values for all three tests are provided.

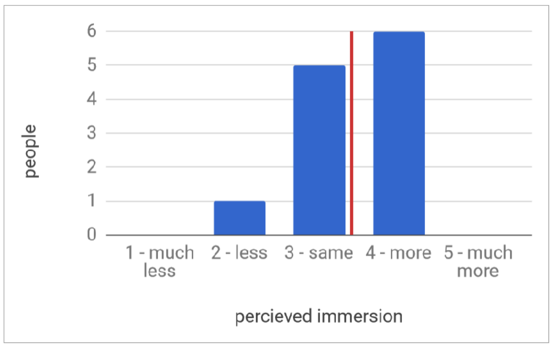
The test participants compared perceived size of improved boundary to the Chaperone as illustrated on a chart in figure 5.4a. This comparison yielded an average of 3.92 which means that on average the test participants thought the improved boundary felt bigger.

One test participant noted that they felt that they found the boundary to be larger thanks to the usable space above the obstacles.

When asked to compare how immersive the improved boundary felt in contrast to the Chaperone the answers averaged 3.42, as shown in figure 5.4b. This is slightly lower than the average of the size comparison but it's still making the average perceived immersion of the improved boundary slightly better than the Chaperone's. Three test participants commented that they found the improved boundary less intrusive. Three said that by being more natural while giving them more information the improved boundary didn't require them to actively think about their surroundings so they could better engage in the VR experience.



(a) perceived size



(b) perceived immersion

Figure 5.4: Charts showing a quantified comparison of the improved boundary to the Chaperone as answered by the test participants. Figure 5.4a shows a comparison of perceived size with an average of 3.92 while comparison of perceived immersion is illustrated in figure 5.4b and has an average value of 3.42. In both figures the average is signified by the red vertical line.

Of the 12 test participants 2 weren't asked about their preference. The rest answered in the following manner. Four preferred the improved boundary over the chaperone in the tests. Five preferred the improved boundary but stated that it might differ on a case-to-case basis, one example being an empty room without any obstacles where the improved boundary would be pointless. One test participant preferred the Chaperone as they found it less intrusive.

The last, objective, assessment of the usability of the improved boundary was whether the test participants bumped into or hit any obstacles. During the first test 9 participants avoided any collisions while 3 participants hit an obstacle at one points. Two of the test participants didn't notice an object. One of them didn't notice an object that was about $40 - 50\ cm$ high. While this participant was more than $190\ cm$ tall this shows that not noticing low obstacles below the cameras view frustum or in areas of the user's vision that are too far away from the areas of the focus of their attention is something to take into consideration in any future designs of the boundary. The third participant that did hit an obstacle probably did so as a result of continuing to walk while turning their camera with their controller. In the second test everybody successfully avoided all obstacles. One test participant didn't participate in either of two last two tests. Only one participant hit a chair in the last test which was due to the point not being aligned with the chair accurately enough.

5.3 Summary and conclusion

In summary the tests showed that the improved boundary was very well accepted among the test participants, helped to point out its shortcomings and find improvements. Of the 12 test participants only one clearly preferred the SteamVR Chaperone over the improved boundary. It was trusted almost completely by the participants who on average also thought it felt bigger and slightly more immersive. Some have explicitly reported they enjoyed the option to and the space available from reaching around obstacles, mostly above them, which is one of the goals of the proposed approach. The same goes for haptic feedback which wasn't disliked by anyone. Many enjoyed having a good idea about their surroundings, some said it requires them to think less about them while one said the opposite.

Some shortcomings were found and improvements suggested. The most serious shortcoming is that some of the smaller obstacles can be overlooked or not be seen at all if the user isn't looking in their direction. Another problem was the lack of a mechanism that would stop the user from leaving the play area by walking in between obstacles. The test participants also pointed out there should be a visual distinction between points that are in direct line of sight and points that are behind or inside a virtual object. Lastly, the improved boundary was found to be noisy and inaccurate in places, which had the most impact on close-distance object manipulation near the boundary.

The test participants also made two suggestions for improvement or exploration which were both considered in this thesis but weren't implemented for reasons not related to their expected usability – real-time object detection or dynamic boundary, and meshed boundary.

In conclusion I think this shows the concept of the improved boundary is viable, useful and preferable to the 2-D Chaperone in many cases and that it might be worth further research and development.

Conclusion

This document provided an overview of the current state of virtual reality with the primary focus on an in-VR play-space and its boundary present in the current generations of high-end room-scale-capable VR devices on the consumer product market. Some already-existing methods for enlarging the perceived play space size were named and introduced. After that, the importance of the boundaries was shortly illustrated, followed by an identification of some of their shortcomings, such as vertical walls that are flat only and provide no detail.

An alternative approach to the boundaries was proposed. The proposed approach suggests using a 3-D scanner attached to a position-tracked HMD of choice, to capture the detail of the user's environment and build its in-VR 3-D representation for display and collision detection.

The following chapter provides an overview of the hardware and software tools available. The hardware devices are briefly described and then compared to determine the best choice of device combination of a HMD and a 3-D scanner for the purpose of this thesis.

Next, the abstracted design and details of my implementation were described. The implementation was carried out in Unity and does the following. ZED stereo camera is used to scan 3-D points. The camera is attached to a Vive HMD, whose optical tracking is used to provide accurate position of the camera in the VE. During scanning the points are stored in a finite 3-D grid, by which they are also sampled. Then they are filtered using a neighbour-count-based algorithm and stored in a file. When to be used the points are loaded into the scene and are made invisible. Colliders of two different shapes are assigned to the HMD and controllers. Collision between the points and the colliders are calculated periodically. If a collision is detected points become visible depending on the the collision strength, which is also the case for haptic feedback.

Finally, the implementation was evaluated based on the results of a series of user tests performed by 12 test participants. The results have shown that the improved boundary is well accepted, works well as a boundary and is preferable to the Chaperone in at least some scenarios. The test participants trusted it and on average felt it was bigger and slightly more immersive. Some of the participants stated that the usable space around obstacles is an advantage. The test have also shown that some small obstacles can be overlooked and that the boundary lacks some kind of a mechanism that would stop the user from leaving the play area by walking between obstacles.

Based on the evaluation I think that the proposed approach achieves its goal and that it might be worth further research. The improved boundary might benefit form real-time scanning and obstacle detection. More advanced point processing might also be beneficial. This could be helped by wrapping the PCL in a managed dynamically-linked library. Experimenting with a mesh-based boundary might reveal even better solution than a boundary based on point clouds. Nevertheless, this approach would still likely require further processing of the scanned point cloud. Notifying the user about obstacles the might overlook, implementing a wall that automatically fills out gaps in the improved boundary, and improving the visual aspect of the boundary to avoid confusion would also likely be helpful while arguably being less difficult to realize. Currently collisions are calculated on the CPU. If more performance is needed this computation can be much accelerated using compute shaders on the GPU. Lastly, haptic feedback and other kinds of feedback that would utilize the 3-D nature of the boundary, such as positional auditory feedback, could be explored more. For instance haptic feedback that would change precision and range depending on the task at hand.

As it often happens in the fast-progressing field of information technology, a newer and upgraded version of the HTC Vive, called HTC Vive Pro, has been announced earlier this year(8th January 2018)[16]. Along with other improvements, it introduces a build-in stereo camera. The product page of the HTC Vive Pro also mentions that „with Chaperone, the technology re-creates a virtual outline of your environment, enabling the user to see and feel the current surroundings without the need of removing the HMD“[9], which might mean that by the time this thesis is finished, it might have been rendered obsolete. One of the initial ideas of this thesis was to suggest waiting for a VR head set that would incorporate a stereo camera or other depth-sensing device in order to apply any gained knowledge. This is becoming reality sooner than I expected.

Bibliography

[1] *Oculus Rift Specs – DK1 vs DK2 comparison*. RiftInfo. [Online; visited 16.01.2018]. Retrieved from: https://riftinfo.com/oculus-rift-specs-dk1-vs-dk2-comparison

[2] *OpenKinect/libfreenect2*. doi:10.5281/zenodo.50641. [Online; visited 15.01.2018]. Retrieved from: https://github.com/OpenKinect/libfreenect2

[3] Elliot Williams: *Alan Yates: Why Valve's Lighthouse Can't Work*. Hackaday. [Online; visited 16.01.2018]. Retrieved from: https://hackaday.com/2016/12/21/alan-yates-why-valves-lighthouse-cant-work/

[4] Amos, E.: *Xbox-One-Kinect.jpg*. [Online; visited 15.01.2018]. Retrieved from: https://commons.wikimedia.org/wiki/File:Xbox-One-Kinect.jpg

[5] Arguinbaev, M.: *Oculus Rift Owners Can Finally Enjoy Full Room-scale Tracking*. [Online; visited 15.01.2018]. Retrieved from: https://thevrbase.com/oculus-rift-owners-can-finally-enjoy-full-room-scale-tracking/

[6] Bowman, D.: *3D user interfaces : theory and practice*. Boston: Addison-Wesley. 2005. ISBN 978-0-201-75867-2.

[7] Bridson, R.: Fast Poisson Disk Sampling in Arbitrary Dimensions. In *ACM SIGGRAPH 2007 Sketches*. SIGGRAPH '07. New York, NY, USA: ACM. 2007. ISBN 978-1-4503-4726-6. doi:10.1145/1278780.1278807. Retrieved from: http://doi.acm.org/10.1145/1278780.1278807

[8] Curless, B.: From Range Scans to 3D Models. *SIGGRAPH Comput. Graph.*. vol. 33, no. 4. November 1999: pp. 38–41. ISSN 0097-8930. doi:10.1145/345370.345399. Retrieved from: http://doi.acm.org.ezproxy.lib.vutbr.cz/10.1145/345370.345399

[9] HTC Corporation: *VIVE Pro | The professional-grade VR headset*. [Online; visited 16.01.2018]. Retrieved from: https://www.vive.com/us/product/vive-pro/

[10] HTC Corporation: *Vive Virtual Reality System*. [Online; visited 15.01.2018]. Retrieved from: https://www.vive.com/us/product/vive-virtual-reality-system/

[11] IMFROMSPACEMAN: *DK2 vs DK1 FOV comparison (camera inside rift)*. reddit. [Online; visited 16.01.2018].
Retrieved from: https://www.reddit.com/r/oculus/comments/2c2ifv/dk2_vs_dk1_fov_comparison_camera_inside_rift/

[12] Inc., S.: *Spatial Mapping*. [Online; visited 14.01.2018].
Retrieved from:
https://docs.stereolabs.com/overview/spatial-mapping/introduction/

[13] Inc., S.: *ZED Stereo Camera*. [Online; visited 14.01.2018].
Retrieved from: https://www.stereolabs.com/zed/

[14] Langbehn, E.; Lubos, P.; Bruder, G.; et al.: Application of redirected walking in room-scale VR. In *2017 IEEE Virtual Reality (VR)*. March 2017. pp. 449–450. doi:10.1109/VR.2017.7892373.

[15] Lvalgma: *Kinect for xbox one ee.jpg*. Wikimedia. [Online; visited 15.01.2018].
Retrieved from:
https://et.wikipedia.org/wiki/Fail:Kinect_for_xbox_one_ee.jpg

[16] Matthew Gepp: *HTC VIVE Raises The Bar For Premium VR With New VIVE PRO Upgrade And Wireless VIVE Adaptor*. [Online; visited 16.01.2018].
Retrieved from: https://blog.vive.com/us/2018/01/08/htc-vive-raises-bar-premium-vr-new-vive-pro-upgrade-wireless-vive-adaptor/

[17] Microsoft Corporation: *Kinect for Xbox One*. [Online; visited 14.01.2018].
Retrieved from: https://www.xbox.com/en-US/xbox-one/accessories/kinect

[18] Microsoft Corporation: *Kinect hardware*. [Online; visited 14.01.2018].
Retrieved from:
https://developer.microsoft.com/en-us/windows/kinect/hardware

[19] Nirav Patel and Nate Mitchell and Kam Chin and Ryan Brown: *DK2 Specification*. GitHub. [Online; visited 16.01.2018].
Retrieved from: https://github.com/facebookarchive/RiftDK2/blob/master/Documentation/DK2Specification.pdf

[20] Oculus VR, LLC.: *Oculus Guardian System*. [Online; visited 15.01.2018].
Retrieved from: https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-guardian-system

[21] Rousset, D.: *2016 is the year of VR and room scale VR with HTC Vive is the ultimate experience*. [Online; visited 14.01.2018].
Retrieved from: https://www.davrous.com/2016/07/21/2016-is-the-year-of-vr-and-room-scale-vr-with-htc-vive-is-the-ultimate-experience/

[22] Rusu, R. B.; Cousins, S.: 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China. May 9-13 2011.

[23] Suma, E. A.; Lipps, Z.; Finkelstein, S.; et al.: Impossible Spaces: Maximizing Natural Walking in Virtual Environments with Self-Overlapping Architecture. *IEEE*

Transactions on Visualization and Computer Graphics. vol. 18, no. 4. April 2012: pp. 555–564. ISSN 1077-2626. doi:10.1109/TVCG.2012.47.

[24] Sysdia Solutions Ltd: *VRTK - Virtual Reality Toolkit - [VR Toolkit]*. Unity Asset Store. [Online; visited 16.01.2018].
Retrieved from: https://assetstore.unity.com/packages/tools/vrtk-virtual-reality-toolkit-vr-toolkit-64131

[25] Unity Technologies: *Unity*. Unity Asset Store. [Online; visited 16.01.2018].
Retrieved from: https://unity3d.com/

[26] Valve Corporation: *SteamVR Plugin*. Unity Asset Store. [Online; visited 16.01.2018].
Retrieved from: https://assetstore.unity.com/packages/templates/systems/steamvr-plugin-32647

[27] Vasylevska, K.; Kaufmann, H.: Compressing VR: Fitting Large Virtual Environments within Limited Physical Space. *IEEE Computer Graphics and Applications*. vol. 37, no. 5. 2017: pp. 85–91. doi:10.1109/MCG.2017.3621226. invited.

[28] Welch, G.; Bishop, G.; Vicci, L.; et al.: High-Performance Wide-Area Optical Tracking: The HiBall Tracking System. *Presence: Teleoper. Virtual Environ.*. vol. 10, no. 1. February 2001: pp. 1–21. ISSN 1054-7460. doi:10.1162/105474601750182289.
Retrieved from: http://dx.doi.org/10.1162/105474601750182289

Appendix A

Content of the included DVD

builds Contains two executable files - „scanning.exe“ and „example.exe“ - and their dependencies.

project_files Contains my whole Unity project including dependencies and test scenes the binaries were built from.

src Contains the source files, other original content, „LICENCE.txt“ and „README.txt“ contaning list of dependecies and instructions.

tex_files Directory containing the TeX source files for the thesis.

videos Contains a video presentation in several formats.

ImprovedBoundary.unitypackage Unity package with the same content „src“ has.

README.txt A README file.

xtinka05-masters_thesis.pdf This thesis in PDF format.