FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Erkundbare Halbleiter

Interaktion mit Kristallstrukturen in Virtual Reality

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Jakob Knapp
Matrikelnummer 1327386

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dipl.Ing Tobias Klein

Wien, 23. September 2017

Jakob Knapp	Tobias Klein

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Explorable Semiconductors

Interacting with Crystal Structures in Virtual Reality

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Jakob Knapp
Registration Number 1327386

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl.Ing Tobias Klein

Vienna, 23rd September, 2017

Jakob Knapp   Tobias Klein

Erklärung zur Verfassung der Arbeit

Jakob Knapp
Obere Amtshausgasse 20/30, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 23. September 2017

Jakob Knapp

Danksagung

Ich möchte mich bei Tobias Klein für seine Betreuung während meines Bachelor-Projekts bedanken. Nicht nur stand er immer für Ratschläge zur Verfügung, sondern gab mir auch die nötige kreative Freiheit, die ich zur Implementierung benötigte. Ebenso gilt mein Dank Josef Weinbub und Thomas Windbacher vom Institut für Mikroelektronik, welche durch ihre Zusammenarbeit mit dem Institut für Computergrafik und Algorithmen meine Arbeit ermöglicht haben. Es war toll, mit euch zu arbeiten! Nicht zuletzt möchte ich mich bei Meister Eduard Gröller und Ivan Viola für ihr ausführliches Feedback bedanken, ohne welches das Projekt nicht zu einem solch zufriedenstellenden Ergebnis gekommen wäre.

Ganz besonderer Dank gebührt meiner Familie, welche mich seit jeher in meinen Entscheidungen unterstützt, mich immer motiviert und mir das Studium in Wien ermöglicht, sowie meiner wundervollen Freundin Anna, ohne die ich wahrscheinlich nicht durchgehalten hätte. Danke, ihr seid die besten!

Acknowledgements

I would like to thank Tobias Klein for his support during my Bachelor project. Not only was he always available for advice, but also gave me the creative freedom I needed. I am also grateful to Josef Weinbub and Thomas Windbacher of the Institute for Microelectronics, who have made my work possible through their collaboration with the Institute of Computer Graphics and Algorithms. It was great to work with you! Last but not least, I would like to thank Meister Eduard Gröller and Ivan Viola for their detailed feedback, without which the project would not have come to such a satisfactory conclusion.

My special thanks goes out to my family, who has always supported me in my decisions, always motivated me and made my studies in Vienna possible, as well as my wonderful girlfriend Anna, without whom I would probably not have made it through. Thanks, you're the best!

Kurzfassung

Bedingt durch die immer größer werdende Popularität von Virtual Reality erschließen sich auch Anwendungsfelder für VR Applikationen abseits der Unterhaltungsbranche. Diese Arbeit beschreibt die Entwicklung einer VR Simulation im Auftrag des Instituts für Mikroelektronik an der TU Wien, welche es BenutzerInnen erlaubt, ein Elektron durch die atomare Kristallstruktur eines Halbleiters zu manövrieren. Die Applikation basiert auf dem cellVIEW framework, welches in der Unity Engine implementiert wurde, um riesige biomolekulare Datensätze zu visualisieren.

Zuerst wird dem/der LeserIn Hintergrundinformation zum Thema geboten, indem definiert wird, worum es sich bei Virtual Reality handelt, wie diese kreiert wird und welche Evolution die Technologie von Charles Wheatstones Stereoskop bis hin zum modernen HTC Vive VR System durchlaufen hat. Basierend auf diesem Wissen wird der/die LeserIn in die Probleme eingeführt, mit welchen sich die Forschung im Bereich VR momentan konfrontiert wird, besonders im Bezug auf die Vermeidung der sogenannten "VR-Krankheit". Bei der Entwicklung der Simulation mussten zwei Hauptprobleme gelöst werden: das Bewegen des Benutzers/der Benutzerin durch die virtuelle Welt, ohne dabei körperliches Unwohlbefinden auszulösen, und die korrekte Behandlung von Kollisionen zwischen dem Elektron und den tausenden Atomen der Kristallstruktur, ohne dabei Performance einzubüßen. Zur Lösung des ersten Problems werden ähnliche Arbeiten auf diesem Gebiet analysiert, um Inspiration für das Design der eigenen Applikation zu gewinnen. Um "Vektion" zu vermeiden wird die Bewegung des Benutzers/der Benutzerin in der Simulation auf die Vorwärtsrichtung beschränkt und externe Kräfte auf eine Sphäre, anstatt direkt auf das virtuelle Selbst des Benutzers/der Benutzerin, übertragen. Um die Bildrate trotz der potentiell riesigen Anzahl an Atomen hoch zu halten, wurde eine "Octree"-Datenstruktur implementiert.

Abstract

Due to Virtual Reality's recent rise in popularity, other application areas apart from the entertainment sector have shown interest to use VR applications for their purposes. In this thesis, a VR simulation, which enables the user to maneuver an electron through a virtual semiconductor's crystalline atomic structure, is devised on behalf of the Institute of Microelectronics at the TU Wien. The application is based upon the cellVIEW framework, which was developed using the Unity game engine to visualize large sets of biomolecular data in real time.

At first, background regarding the topic at hand is provided by defining what VR is, how it is achieved and how it has evolved from Charles Wheatstone's stereoscope to the HTC Vive VR system. Building upon that knowledge, the reader is introduced to the main problems that research in the field of Virtual Reality is currently facing, especially considering the avoidance of VR sickness. For the development of the simulation devised in the course of this thesis, two main obstacles had to be overcome: moving the user through the virtual world without causing discomfort and handling collisions between the electron and the thousands of atoms in the crystal structure without losing performance. To address the first issue, related work is reviewed and drawn inspiration from. To avoid "vection", movement in our simulation is limited to the forward direction and external forces are applied to a sphere representing the electron instead of influencing the user himself. To keep the frame-rate high despite the potentially large number of atoms in the virtual scene, a space-partitioning octree data-structure has been implemented.

Contents

CHAPTER 1

Introduction

Virtual Reality, commonly abbreviated as VR, has come a long way since Ivan Sutherland devised his head-mounted three dimensional display back in 1968 [Sut68]. Technological advances, especially in the field of graphic cards, have led to a steady rise in its popularity in the recent past by enabling the generation of more realistic images in real time. Companies like Sony, which focus on electronics and gaming, have contributed highly to Virtual Reality equipment becoming affordable by private users. Still, gaming and entertainment are not the only fields of application for Virtual Reality anymore. Experts in health care, education, tourism or physics have started to show interest and come up with ideas to use virtual reality in their respective fields.
In comparison to Virtual Reality, the concept of 'gamification' has originated more recently, with its first appearance dating back to 2008. According to Deterding et al., one goal of gamification is to "make other, non-game products and services more enjoyable and engaging" by incorporating certain elements, which are typically seen in video games, into other types of products [DDKN11]. To a certain degree, there are similarities to current trends in VR, where Virtual Reality is incorporated into new fields of use to enhance the user's experience.
In this thesis, Virtual Reality and gamification will be used in the context of microelectronics to show certain physical aspects of semiconductors to an audience at science to public events. While VR is utilized to make the experience immersive and to integrate the spectator into the simulation, various aspects of game design will be applied to make the simulation itself as enjoyable and entertaining as possible.

1.1 Aim of this Work

The aim of this thesis is to devise a software, which allows users to explore the crystalline structure of semiconductors in Virtual Reality. They are able to maneuver an electron through the semiconductor while interacting with the atoms, of which the structure

consists, and manipulating global parameters, such as the temperature of the scene. Despite the recent progress in the field of VR, there are still certain obstacles, like sickness induced to the user through visual input, that have to be overcome or circumvented when implementing immersive simulations. Some of those and possible solutions for them will be shown and discussed in this work.
It is important to mention, that the program that is implemented will not primarily be used by researchers in a scientific context, but rather by people without knowledge of microelectronics. The main goal is to draw attention to the field at science to public events by showing an impressive, computer-generated scene in Virtual Reality to people, who are unfamiliar with the topic at hand. While being educative, the simulation should entertain at the same time to raise interest. From this, a need arises to balance physically correct simulation with entertainment, which in turn requires certain trade-offs. Those will be addressed in this thesis as well. Inspiration considering the enjoyability of the simulation is drawn from various video games, which have specifically been designed to offer a pleasant experience when played in VR. Here, the concept of 'gamification' comes into play.
Furthermore, the sheer number of atoms in the semiconductor's structure requires the use of special algorithms for detecting collisions between said atoms and the electron controlled by the user. As the crystalline formations might contain multiple millions of atoms, checking for interaction between the electron and all of those is not feasible, as it might lead to poor performance of the application. This results in only a few images being rendered over a certain period of time, called a low frame rate, which in turn affects the users experience in a negative way by causing nausea and other symptoms of VR sickness. This thesis explains how the problem of detecting collisions in three dimensional space can be dealt with efficiently using the hierarchical octree data-structure for space-partitioning.

To sum it up, the following problems will be addressed in this thesis:

- incorporating VR into a physical simulation to make the experience more immersive
- designing interaction and locomotion in a way that avoids causing sickness to the user
- balancing physical correctness with gamification aspects
- developing a collision detection system that allows for interaction with the crystalline structure

1.2 Structure of this Thesis

Before explaining the actual simulation, which has been devised in the course of this thesis, it is necessary to provide the reader with background knowledge regarding Virtual Reality. Therefore, the second chapter defines what Virtual Reality is, gives insight into the history of Virtual Reality systems up to the present time and aims to reason why the

concept of VR is so appealing. Furthermore, a very short introduction to semiconductors is included to give the reader more information about the topic at hand.
The third chapter builds upon that knowledge to present the state of the art and discuss problems regarding VR, which are currently faced by researchers, developers and designers. While some of those are hardware-related or based on the human sensory system, some can be counteracted using intelligent software-design. Based upon this knowledge, the thesis will give insight into work that is related to the topic at hand and discuss solutions that are implemented in those related applications. In the fourth chapter, the methods that are used for creating the crystal structure simulation are explained and parallels to related work are drawn. While this section focuses mainly on the theoretical background, like allowing for interaction between the user and the virtual semiconductor and detecting collisions using an octree, the following chapter aims to give an insight into the actual implementation of the software that was built as an extension to the cellVIEW framework in the Unity game engine.
The last two chapters complement the thesis by presenting and discussing the results based on reception and performance of the application. Furthermore, a conclusion of the work that was done is offered and ways to improve both the simulation that was created and VR technology itself are mentioned.

CHAPTER 2
Background

This chapter provides a theoretical background about two central topics of this thesis, which are Virtual Reality and semiconductors. The goal is to give an overview about how VR has evolved, how it works and what makes it attractive for human users, as well as providing insight about which different modern VR systems are being used at the moment. Furthermore, this chapter aims to provide some basic knowledge about semiconductors and their structure. This allows for easier understanding of the following sections of this thesis.

2.1 Defining VR

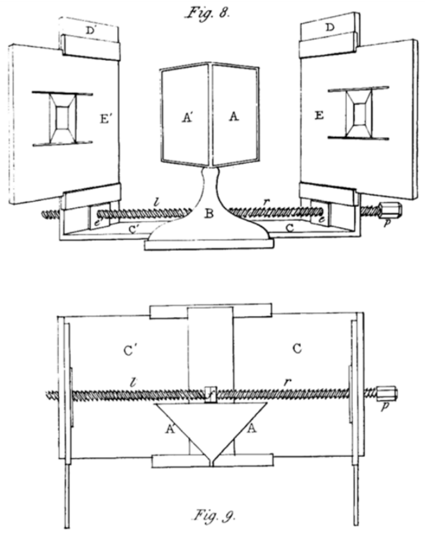
At least since Oculus' successful Kickstarter campaign from 2012, the image of Virtual Reality is inseparably connected to head-mounted displays. However, those displays only serve as means to access or visualize a virtual world. Although, according to Burdea and Coiffet, the general public associates Virtual Reality with those "goggles" due to them being "the first devices being used in simulation", it is important to note, that those pieces of hardware do not define or limit Virtual Reality in any way [BC03]. On the contrary, those can be replaced by projection screens, for example. Burdea and Coiffet simply define VR as "a simulation in which computer graphics is used to create a realistic-looking world", without mentioning any means for displaying said world to a user. Furthermore, they argue that the world being simulated is not static, but can be interacted with in real-time. This means, that the user's actions and movements being delivered to the computer directly influence the virtual world and its behavior. As part of the "Virtual Reality Triangle", consisting of "Immersion", "Interaction", and "Imagination", interaction between the user and the simulation plays a crucial role.
An alternative definition is provided by LaValle in his 2017 book "Virtual Reality" [LaV17]. He describes VR as "inducing targeted behavior in an organism by using artificial sensory stimulation, while the organism has little or no awareness of the interference". An

organism could, for example, be a human person, while the targeted behavior LaValle speaks of is more of an "experience" that the creator wants the user to have. Artificial stimulation is, most of the time, achieved by rendering a computer-generated image onto a screen while also providing auditory input to the user. In the best case, the simulation also stimulates the organism's tactile and vestibular senses, for example by moving the seat in a driving simulator. Crucial for the immersion that Burdea and Coiffet talk about is the user having "little or no awareness of the interference". This means, that the person perceiving a Virtual Reality should not be aware that their surroundings and the stimuli are of virtual nature, implying that the simulation that is generated must follow certain rules the user is accustomed to. The images that are rendered must be created in a way that complies with the "real" world, for example regarding perspective and visualization of depth. Concerning the simulation devised in the course of this thesis, it is important to design interaction in a way that does not cause interference between the user's "virtual self" and his or her body. Problems like nausea are known to arise when the brain is presented with images that suggest that the body is moving, despite it staying at the same place.

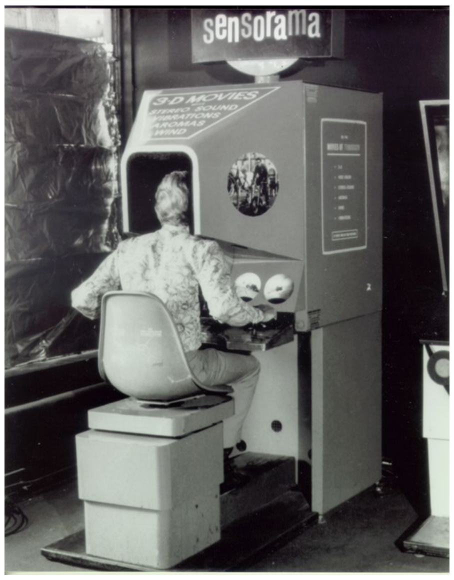
2.2 A Short History of Virtual Reality

The foundation of Virtual Reality as it is today was lain by Charles Wheatstone in the early 1830s, when he invented the stereoscope [Wad02]. Prior to this, people's experiences with "unreal" worlds were pretty much "centered on staring at a rectangle that is fixed on a wall" [LaV17]. The stereoscope, which can be seen in Figure 2.1a, was an apparatus that presented a different image to each eye using mirrors. When presented two dissimilar images, the person looking at those could perceive an appearance of depth through stereopsis, in contrast to the strictly flat images that were paintings and photographs.
Building upon that concept, Morton Heilig patented his "Sensorama" in 1961, an apparatus, that aimed "to stimulate the senses of an individual to simulate an actual experience realistically" [Hei62]. An image of the Sensorama can be seen in Figure 2.1b. Besides a system, which displays a stereoscopic movie of a motorbike-ride to the user, there are several other stimuli affecting the users experience. Heilig devised his Sensorama to direct a breeze towards the user's face to simulate movement together with vibrations used at the right time. Additionally, different odors were released into the breeze and binaural sound was utilized to encapsulate the observer as closely as possible into the experience Heilig wanted him to have.

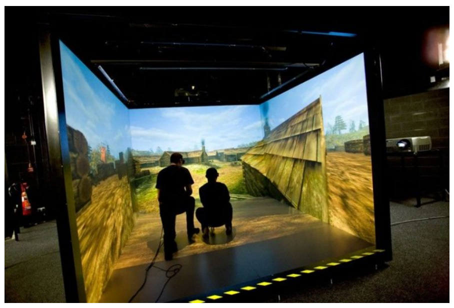
The next important step towards an immersive Virtual Reality experience, according to LaValle, was increasing the field of view, which is perceived by the user. As for the "Stereoscope" and the "Sensorama", moving the head to view previously hidden parts of the virtual world was not possible. Fred Waller's Cinerama system, which had its debut in 1952 at the Broadway Theater, counteracted this by using three projectors on a curved screen to generate a 146 degree wide panorama, combined with a directional sound system [IJs05]. Because the peripheral visual system is more sensitive to motion than the foveal system, as Ijsselsteijn noted, the observer felt more involved in the motion picture.



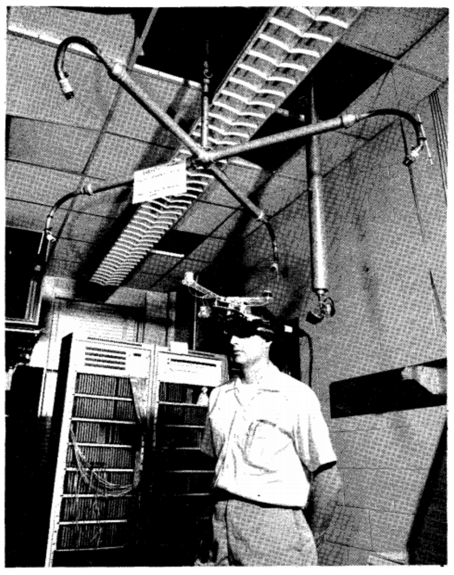
(a) The stereoscope, invented by Charles Wheatstone. Two mirrors were used to reflect two images towards the viewer's eyes to create an appearance of depth. [Wad02]



(b) The Sensorama, created by Morton Heilig, which used a stereoscopic movie, binaural sound and a fan to create an immersive VR experience. [Afn]



(c) The CAVE system, developed by Cruz-Neira et al. Polarized light is projected onto all six sides of the cuboid the user is located in. [Afn]



(d) The head-mounted three dimensional display devised by Ivan Sutherland. [Sut68]

Figure 2.1: Four different historical VR systems, which show how Virtual Reality devices have evolved since the 1830s.

Along the lines of the Cinerama, LaValle also mentions the CAVE system, created by Cruz-Neira et al. at the University of Illinois [CNSD$^+$92]. This virtual reality interface, as seen in Figure 2.1c, allows users to enter a room "in which video is projected onto several walls" [LaV17]. Together with polarized light observed through special glasses to create stereoscopic vision, head tracking is used, which, in comparison to the aforementioned systems, allows some form of interaction with the virtual world. Additionally, Ijsselsteijn also mentions the CyberSphere, "a spherical variation of CAVE-style wall-projection systems". As opposed to the CAVE, the images were projected onto a sphere, not a cube. To overcome the limited possibilities of motion in such fixed systems, the CyberSphere was constructed in to rotate as the user walks in any direction, while the images being projected are updated accordingly.
In 1968, Ivan Sutherland devised his head-mounted three dimensional display called the "Sword of Damocles", which paved the way for head-mounted displays as they are popularly used today. Sutherland's invention, which is displayed in Figure 2.1d, relied less on stereoscopic images to create an immersive experience, but rather on an effect he called "kinetic depth effect" [Sut68]. This means, that the images being rendered must behave like the observer would expect the real world to behave. Objects that are static must stay at the same place. While the observer's perspective changes, their position in space does not. LaValle calls this the "perception of stationary" and gives it as a reason that motion tracking instruments have to be integrated into VR systems. Building upon Sutherland's invention, first commercial headsets started to release in the 1980s. Neither those nor the following VR systems published by the games industry were commercially successful. LaValle explains this by the lack of comfort and immersion.

2.3 Modern HMD-Based, User-Fixed VR Systems

Due to the production of displays with higher resolution and better sensors for motion tracking, not least through innovations by the smartphone and video game industry, the aforementioned flaws from early head-mounted devices, often abbreviated as HMDs, could be improved. This in turn led to a rise in popularity and commercial interest in HMD-based, user-fixed VR systems over the recent years.
Palmer Luckey presented the first prototype of his Oculus Rift HMD in 2012 at the Electronic Entertainment Expo, where it gained massive attention [LYKA14]. Made possible by a successful crowdfunding campaign, which gathered more than two million US-Dollars, the company released two developer kits in 2013 and 2014. The second one, called Oculus Rift DK2, features two OLED screens with a resolution of 960x1080 pixel and a refresh rate of 90 Hz, both of which display an image for either the left or the right eye of the observer, thereby enabling stereoscopic vision. Like all modern HMDs, the Oculus Rift contains two lenses, through which the displays are viewed. It offers a field of view, commonly abbreviated as FOV, of 110 degrees and orientation-, as well as position-tracking. Orientation-tracking means that it observes the way that the user's head is oriented. This allows to detect rotation, pan and tilt, enabling the user to look around in Virtual Reality. Position-tracking, however, makes it possible for the user to

move around in VR, for example closing the distance between his head and a virtual object. This is realized by three different components, a gyroscope, an accelerometer and a magnetometer. While the gyroscope is used for determining the head's orientation, the accelerometer and the magnetometer work together to find its position. The images, which are rendered onto the two displays, are generated on a separate PC, to which the Oculus Rift is connected via HDMI and USB. A set of two controllers, called the Oculus Touch Controllers, were made available following the consumer edition's release in 2016. Those allow for interaction with a Virtual Reality, as the controllers are tracked too and offer six degrees of freedom, which means they can be translated and rotated along all three perpendicular axes. Each Oculus Touch Controller features buttons, a trigger and a joystick, which enables the user to give input to the VR simulation. It also allows the developer to provide tactile feedback to the user through vibration. In addition to the increased display size of 1080x1200 pixel, the newest Oculus Rift comes with an advanced tracking system called "Constellation". It consists of infrared sensors, which can track the HMD as well as the controllers. Combining more of those sensors allows for tracking of objects within a room.

Another modern VR system that offers that capability is the HTC Vive, which is

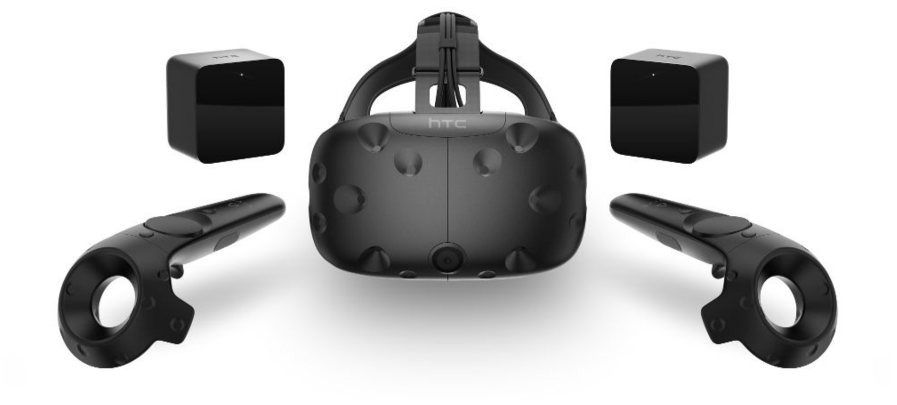


Figure 2.2: The HTC Vive headset, developed by HTC and Valve. It consists of the HMD (center), two Vive Controllers (front left and front right) and two tracking stations (back left and back right). Source: `http://www.currys.co.uk`

developed by HTC in cooperation with Valve (Figure 2.2). While most of the technical specifications, such as screen resolution, refresh rate and FOV are identical to the Oculus Rift, it comes with two base stations, called “Lighthouse”, which send out lasers to determine the users position within a room. This so called “room scale” tracking is enabled by 32 infrared sensors on the HMD and 19 on each of the two Vive Controllers and allows for exact positional tracking.
In comparison to the two aforementioned systems, Sony’s PlayStation VR is slighly more limited in its capabilities. It offers one 960x1080 OLED display per eye and comes with two controllers, called PlayStation Move. As opposed to the Oculus Rift and the HTC Vive, the PlayStation VR is not designed to be connected to a PC, but rather to Sony’s video game console PlayStation 4. This limits its field of use to entertainment purposes by design, whereas the Rift and Vive can be used for, e.g., medical purposes as well. As of now, it does not support room-scale tracking either.
Another type of HMD-based VR systems that are worth mentioning are the Gear VR, which is developed by Samsung in cooperation with Oculus, and Google’s Cardboard. Both utilize the processor and display of a smartphone to generate and display images to the user. This means that, as opposed to all the aforementioned systems, the computer that calculates the HMD’s position and orientation via built-in accelerometer and renders the image is directly integrated into the VR headset. The actual HMD only serves as case for the phone that straps it onto the user’s head and provides the lenses. The Gear VR and Cardbord come without controllers and use the phone’s touchscreen to input commands and interact with the Virtual Reality. This makes systems like those a lot cheaper, but implies limited capabilities.

2.4 How Does VR Work and What Makes It so Popular?

LaValle describes VR as an interaction between three components: hardware, software and an organism, which perceives signals via certain senses [LaV17]. In most cases, this organism is a human person, which is wanted to have a certain experience, as noted by LaValle in his definition of VR. According to Ijsselsteijn, “virtual environments derive their strength precisely from allowing participants to jump in the driver’s seat – to interact with content in real time” [IJs05]. Alternatively, it can be said that VR is so popular because it takes the user to a different world by providing artificially generated stimuli to his senses. These stimuli differ depending on the information about the user’s motion the hardware receives, whether through button presses, head movements or other types of input. In turn, “the hardware produces stimuli that override the senses of the user” [LaV17].
The hardware is made up of three components: sensors, computers and displays. The sensors are used to gather input from the user, the computers process those inputs and generate output, while the displays are used to deliver that output to a certain sensory organ of the user. The computers are mainly used to run a software called virtual world generator (VWG), which maintains the Virtual Reality. Different kinds of Virtual Realities exist. Those may be completely synthetic, but might as well be recordings of the

physical world or any mixture of those. According to LaValle, one of the most important tasks of a VWG is matching motions between the users physical and virtual self. To a certain degree, the movements of the user's virtual self should be determined by how the person moves in the physical world. Going one step further, some functionalities in the Virtual Reality may go beyond the physical boundaries of the real world context. This is, for example, when simulating forms of locomotion, where the user's physical self does not move but his virtual self does. The VWG has to handle those movements correctly without disturbing the users immersion, which is still a problem up until now. It implies that the VWG also has to take physics into account when generating stimuli for the user. Not only does this include, how the virtual self interacts with other objects via collision. It also means that real world phenomena like the propagation of light and sound and their interaction with surfaces must be modeled in a way that makes sense for the observer.
The third component that LaValle talks about is the human physiology and perceptional system. As the human body has evolved, it has adapted to the natural world. When experiencing a virtual reality, this can lead to problems from headache to nausea because the body is not used to such environments and stimuli. The artificial VR might interfere with our natural perceptual processes and interpretations of the world surrounding us. Psychophysical theories must therefore be taken into account when designing VR systems. Stimuli are interpreted differently by different individuals. Also, some human users might adapt to the VR while some will not, which is why it has to be tested thoroughly on a number of subjects.
When those three components play together well, it creates a world which the user will ideally believe to be real, as it reacts to his inputs like the physical world he is used to. This allows to create experiences in VR, which are impossible to have in the real world. This "escape from ordinary life" [LaV17], that allows the user to "jump in the driver's seat" [IJs05] and have experiences that can not be had in the real world is what makes Virtual Reality so attractive and popular in modern times.

2.5 Semiconductors

As mentioned in chapter 1, the aim of this thesis is to use a VR system to explore a virtual semiconductor's crystalline structure, which will be used to raise interest in microelectronics. Therefore, it is important to have some knowledge of the topic at hand. Microelectronics is a sub-discipline of electronics, which specializes in the manufacturing of tiny electronic components. Most of the time, semiconductor materials are used to create these components. Additionally, they can be provided with oxide layers for various reason, which include isolation or cover to protect the material from corrosion [hal]. Semiconductors are defined by their electrical conductivity [Spe55]. While this value is lower than the value of conductors, for example metals, it is higher than the value of an insulator. To define it in concrete terms, a semiconductor is a material with an electrical conductivity value between $10^4\Omega^{-1}cm^{-1}$ and $10^{-12}\Omega^{-1}cm^{-1}$. The conductivity of semiconductors relies on the materials temperature. The higher the temperature is, the better the conductivity. While amorphous structures are also possible, classical semiconductors have a crystal structure, “a well-structured periodic placement of atoms” [SN06]. An example of such a structure is the diamond cubic crystal structure shown in Figure 2.3. The symmetries of this structure determine the properties of the crystal as a whole [Kit05]. The crystal structure is created by repeating and translating the primitive cell along a grid by so-called base vertices. The primitive cell is the smallest assembly of cells with which that can be accomplished. Its dimension is described by the lattice constant a [SN06]. Silicon, for example, which is one of the most important semiconductors, has a diamond structure with a lattice parameter of 5.43 Ångström [Kit05]. As for this thesis, it is important to keep these symmetrical properties in mind as they are to be shown in the simulation.

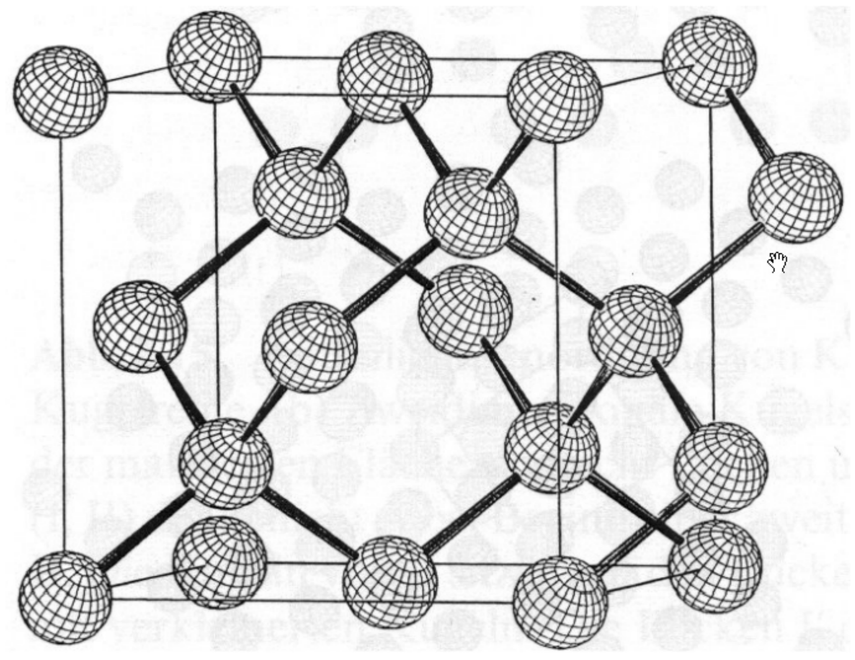


Figure 2.3: The diamond cubic crystal structure, from which silicon and other semiconductors are built. [Kit05]

CHAPTER 3

State of the Art and Related Work

This section provides an overview of the current State of the Art, primarily in terms of Virtual Reality and its applications in the video game industry. At first, the most severe problems that VR research is currently facing are introduced. Steven M. LaValle's book "Virtual Reality" from 2017 serves as the main source of information for this part, as especially chapters 5, 7, 8, 10 and 12 offer an extensive and up to date insight into the current state of things [LaV17]. Building upon that knowledge, various modern VR applications are introduced and discussed with regards to how they approach the aforementioned problems.

3.1 Current Problems in Virtual Reality

As explained in Chapter 2, Virtual Reality has come a long way and has finally reached popularity among consumers, with more than one million exemplars of PlayStation VR having been sold as of June, 2017 [Web]. The most serious issue, which has hindered the breakthrough of VR for decades and still presents a severe problem as of today, is Virtual Reality systems inducing physical discomfort in users. LaValle identifies this as "VR sickness" and lists several possible symptoms, with the most frequent ones being nausea, dizziness and fatigue [LaV17]. While users are immersed into the virtual world, they might start to experience pain in their stomach region or their head, feel sleepy or exhausted. This is due to flaws in the hard- and software of the VR system in cooperation with the human organism's sensory system. Said sensory system has the ability to adapt to its environment, making a human user able to adapt to flaws in a VR system to a certain degree. This adaption however can cause discomfort when exiting Virtual Reality. The sensation that the body is still moving can remain in the outside world. When the user's eyes have to adapt to the display of the head-mounted device, his vision can be blurry after usage. While the symptoms of VR sickness after exiting the virtual world are mostly due to adaption of the human sensory system to flaws in the visualization or hardware, the aforementioned symptoms that appear during usage are mostly due to a phenomenon called “visually induced apparent motion”, also called “vection”. Vection is what makes VR sickness closely related to motion sickness. It means that the signals, which are retrieved from the human body’s vestibular and visual systems, do not match. While the user’s eyes suggest that the body is moving (in the virtual world), the vestibular system, which provides a sense for balance and spatial orientation, suggests that the body is static (in the real world). For motion sickness, which can, for example, happen when reading a book in a moving car, the sensory inputs are vice versa, with the vestibular system indicating movement while the eyes do not. This conflict between the vestibular and visual system is very apparent in VR applications due to VR systems primarily stimulating the users visual senses. Vection can be used for entertainment purposes, such as in amusement park attractions where it creates illusions of the visitors moving around in a room while actually the room moves. However, it is unwanted in most VR systems.

To sum it up, LaValle identifies two root causes with regards to “sensory conflict theory”:

- The engineered stimuli do not match the stimuli, that are expected from the human organism’s sensory system, closely enough.
- Some sensory systems, like the vestibular organs, do not receive any artificial signals at all.

As already stated in chapter two, VR works through interaction between three main actors: hardware, software and an organism. As the first two are used to provide an experience to an organism with a certain sensory system but cause syndromes of sickness to it as a side effect, both still have to be improved. In the following sections, possible solutions or improvements regarding hardware and software that are researched at the moment are presented and discussed.

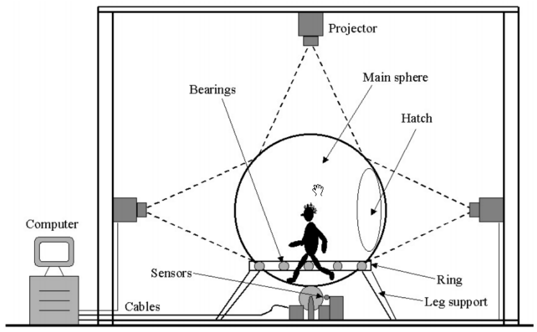
3.2 Improving VR through Hardware

Flaws in the hardware of VR systems can have effects that differ in severity. While some lead to lack of immersion or visible artifacts in the virtual world, which are mainly aesthetic problems that make the experience as a whole less believable, some cause physical discomfort to the user. However, both have to be addressed.
The display of a head-mounted device serves as window into the virtual world. As stated in the previous section, most VR systems aim to stimulate the user’s visual sensory system. Therefore, the display is an essential part of modern VR systems. LaValle identified three crucial factors concerning displays: spatial resolution, intensity resolution/range and temporal resolution [LaV17]. Spatial resolution means how many pixels are present on a certain area on the display, while intensity resolution and range refers to how many different intensity values can be produced and what the minimum and maximum

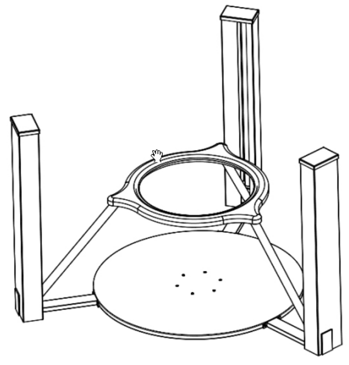
values are. In opposition to those two factors, which essentially define the quality of the resulting image that is displayed, temporal resolution means how fast these images can be displayed, or rather how fast the pixels can be swapped.
Considering spatial resolution first, it is important to determine the best pixel-per-inch (PPI) value for human sight depending on the distance between the eye and the display. As for Virtual Reality systems, the display is usually right in front of the eye, from which it is separated by lenses. Those lenses bring the display even closer towards the observer's eyes. In this case, a high PPI value is essential to avoid artifacts. One of these is the so called screen-door effect, where black lines in between the pixels are noticeable, despite them being thinner than one pixel. If the pixels themselves are too big in size, they can be perceivable as red, green and blue squares respectively when very close to the eye. In 2010, Apple coined the term "retina display", which means a display with such a high PPI that single pixels are not visible for the human eye at a normal viewing distance. While Apple claimed that 326 PPI is enough back then, LaValle argues that it is not, especially considering the lenses in between the eyes and the display. To support that, he uses the unit "cycles per degree", which describes "the number of stripes that can be seen as separate along a viewing arc" [LaV17]. 60 to 77 cycles are the maximum that can be distinguished by the human eye. In the US, perfect visual acuity is defined by the value 20/20, which means that a person standing 20 feet away from a chart, for example, can see what a human eye should "normally" see at that distance. For comparison, if a person has 20/40 vision, it means that the person standing 20 feet away from a chart sees what a "normal" person sees when standing 40 feet away, making his or her vision worse than average. To achieve a perfect 60 cycles for a human person with 20/20 vision, keeping in mind the lenses that are about 1.5 inch away from the display, a spatial resolution of about 2292 PPI must be reached in order to render single pixels invisible. For people with vision that is even above what is "normal" for humans, which is rare but has to be considered nevertheless, the PPI value of a display must be 4583 at least. A way to circumvent this would be to apply "foveated rendering". This would exploit the fact that there are more receptor cells towards the center of the retina than on the outside. One way to achieve this would be to create a tiny screen that moves around to always be exactly where the eye is looking. Another way would be to focus rendering on the spot that is being looked at while keeping the rendering outside of that spot way simpler. Both these approaches require exact tracking of the eyeball and a latency that is nearly zero, which is close to impossible and very expensive at the moment. The State of the Art for displays that are in use at the moment is the Super AMOLED screen made by Samsung, which is used by the Gear VR system. It has only 577 PPI, which indicates that a lot of research is still to be put in display technology to create displays for VR systems, which do not produce any noticeable artifacts for the human eye. The same applies for intensity resolution. According to LaValle, the human eye's photo-receptor cells "can span seven orders of magnitude of light intensity", while a display can only produce 256 distinct intensity values. As for temporal resolution, the perfect display would not produce any latency at all and instantly swap all pixels at once. While some LCD displays could take up to 20ms to swap all pixels, OLED screens can do it in under 0.1ms. Further

improvements in this area can mainly be achieved through software solutions, which are discussed in the next section.
Another thing to consider is the Field of View (FOV) of human vision, which is about 270 degrees including eye movements. With a flat screen, less than 180 can be achieved, while bringing the display closer to the user by using different lenses would in turn require an even higher PPI value. Curved displays could help, but are still not optimized for VR systems at the moment. However, a study conducted by Lin et al. suggests a correlation between increased FOV and VR sickness [LDP$^+$02]. They found out that an increased FOV in turn increases immersion into the simulation. This means that users felt more present in the virtual world. But, despite this higher immersion, participants were more susceptible to symptoms induced by VR sickness, in turn experiencing less entertainment. Due to this, it remains to be seen if investing into producing displays that cover a higher FOV is worth it until other issues that cause VR sickness, especially software-wise, are resolved.

While those improvements that are being researched at the moment focus more on the first point mentioned in the last section, where the generated stimulus does not match the expected, natural one because it is not accurate enough, there are also hardware-based approaches that focus on the second point, which is generating input for sensory organs other than the visual system. As already mentioned, vection is a big problem for VR simulations, as it causes mismatches between the vestibular and the visual system. Most of the time, these mismatches arise from locomotion. Locomotion means the ability to move the user's virtual self in a way that is not possible and not reflected in the real world. For example, a simulation might make the user fly or walk across an area, that is



(a) The CyberSphere input system. It features a giant ball, which rotates as the user within walks around. These generates signals, which are used to move the viewer through the Virtual Reality that is projected onto the ball. [IJs05]



(b) A prototype of the Virtualizer developed by Cyberith. It features a low friction floor plate and a ring, in which the user is fixed. The images are displayed using an HMD. [CH14]

Figure 3.1: Two different systems, which allow users to move through a Virtual Reality by walking.

bigger than the room that the user's physical self resides in. If the space is big enough, the yaw of the virtual camera can be adjusted continuously to make the user walk in circles without noticing. If this is not the case, developers have to come up with methods to move the player's virtual self through the virtual world. While most of them, like using a controller to maneuver, induce vection due to the body not moving, there is research that focuses on hardware-based solutions where the user can actually move around physically. An early example of such an apparatus is the "CyberSphere" (Figure 3.1a), that is mentioned by Ijsselsteijn [IJs05]. A modern variant is the "VirtuSphere", on which Skopp et al. conducted a study in 2013 [SSMA$^+$14]. The VirtuSphere is essentially a huge ball placed on a podium, which allows it to freely rotate along all three axes. The fact that the study did not find any significant difference in presence and regarding VR sickness between the VirtuSphere and a common game controller implies that there is still room for improvement in this field. An alternative to the VirtuSphere is the "Virtualizer" (Figure 3.1b), which is being developed by the Austrian company Cyberith [CH14]. It is based on a floor plate with very low friction, which makes the users feet slide over the ground. The user is fixed in a ring, which can be rotated by 360 degrees and freely moved vertically. This allows the user to walk and run without actually moving from the start point. Due to the low friction of the floor, however, the human sensory system detects that the body is moving forward. Sensors in the ring, the pillars holding the ring and the floor plate produce signals that are interpreted to move the user's virtual self in VR.

3.3 Software Solutions for Improving Performance

As discussed in the previous section, the enjoyability of a VR simulation depends heavily on the quality of the output being presented to the user. But hardware cannot solve all issues. Moreover, some aspects of hardware-based improvements depend on software. An example is the aforementioned temporal resolution of a display, which indicates, how fast pixels can be swapped. However, being able to swap pixels in under 0.1ms is useless if the software cannot generate the images at least as fast. Another big problem of state of the art VR systems that has not yet been discussed is latency. Latency describes the time that passes between the input from a user and the display of the image that results from that input. In an ideal system, which can never be achieved due to physical limitations, latency would be zero. In reality, the goal is to keep it as low as possible. Therefore, hardware-based research has to take care of making sensors that detect motions faster, cables that transmit signals faster, graphic cards that can simultaneously render multiple images faster or displays, that swap pixels faster, for example. Software-based research, however, focuses on designing algorithms that reduce latency. LaValle describes several software-based approaches to lower latency, the first of which is simplifying the virtual world [LaV17]. Especially when working with data taken from the real world, for example through 3D-scanning, the models used in the virtual world are often way more complex than necessary. State of the art research, e.g. from Jonathan David Cohen [COM98], focuses on developing algorithms that decrease the complexity of these models, as they often contain more details than necessary. These details cannot be observed by the

human eye and are therefore slowing down the rendering pipeline for no apparent reason. Some algorithms that are popularly used in computer graphics to simplify the rendering process, such as texture mapping, which is used to put a texture on a model, or normal mapping, which can make flat surfaces look bumpy by altering normal vertices, do not work in VR due to different images being generated for each eye. While VR makes those algorithms less usable, it also offers room for totally new approaches, one of which is multi-resolution shading (MRS). When using an HMD in a VR system, the user's eyes do not stare at a rectangular display directly, but through lenses. These lenses distort the image that is rendered on the screen. Currently, this is accounted for by warping the image before displaying it, which makes the image appear correctly when viewed through lenses. When the image is warped, information, which has been computed by the GPU, is lost at the corners, which means that work is being done unnecessarily. MRS is similar to foveated rendering as it renders more details in the middle of the image than at the edges, which are warped anyways. By rendering the edges with much less detail, the rendering speed can be improved a lot. Brad Chacos wrote in an article for PCWorld that a reduction in pixel-work of about 30 percent led to no noticeable difference in the output image when viewed in VR at all [Cha]. However, much less pixels have to be generated.
Further optimizations can be made when rendering the image to the screen. Currently, lines on the display are generated by reading out images from video memory. However, when the video memory is being written to at the same time, the images being rendered are not correct. One way this is solved at the moment is by using vertical sync (VSYNC). When the flag is activated, the video memory can only be written to while it is not read, therefore limiting performance. Another way to circumvent this is buffering, where the image is written to one or more buffers (double- and triple-buffering) and then copied into the video memory. It would be ideal if all pixels were swapped at the same time, which would allow for further improvements in this regard.
Finally, one of the biggest improvements that are being researched at the time is prediction. If it is known when the pixels are switched and it can be predicted how the user's head will be orientated at that certain time, an image that shows the virtual world at that time from the users exact position and orientation can be generated, in turn leading to almost no latency at all. LaValle argues, that "you have no free will in the scale of 20ms", meaning that "momentum dominates" and the head's position can be very accurately predicted 20 milliseconds beforehand [LaV17]. If this fails, however, there is still the possibility to do a post-rendering image warp before displaying it on screen. This requires that the image being rendered is larger than the screen. If the position of the head is slightly different than predicted, for example translated by a bit along the x-axis, then the image can just be translated to fit the actual position of the head without notable difference. This works as long as the movement is not too big or the image too small, as this would result in black regions at the image borders because the required part has not been rendered. This method can also be used to artificially increase performance by generating warped images in between "real" images, which is aptly called "inbetweening" or "tweening".

3.4 Designing Software That Does Not Cause VR Sickness

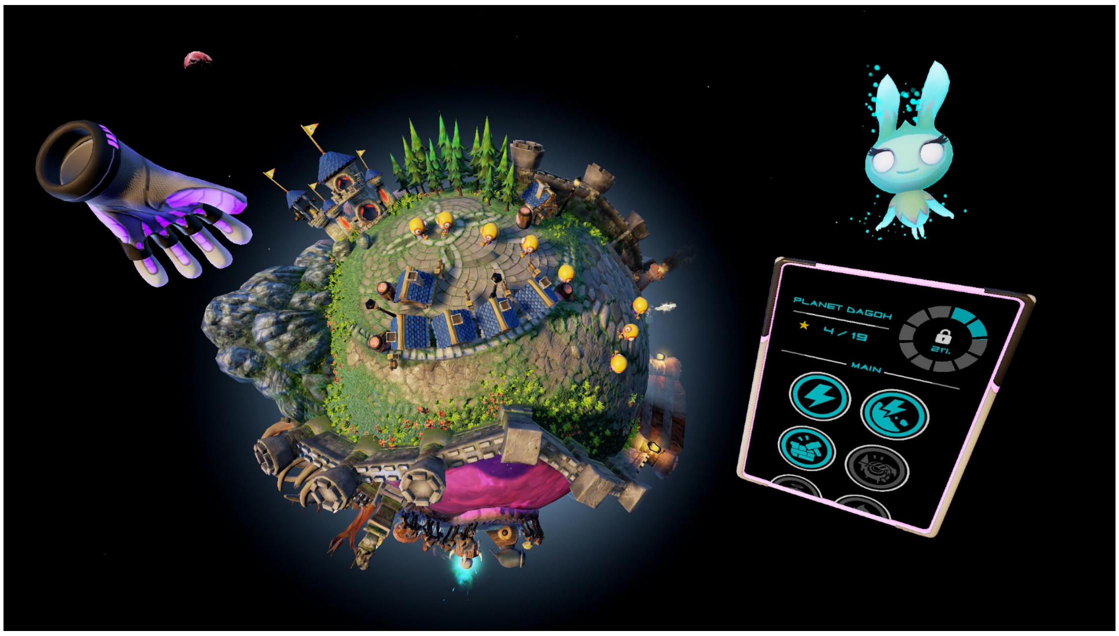
Although the aforementioned improvements lead to more immersion into the Virtual World and to a more realistic feeling overall, improvements in performance and hardware cannot solve all problems that VR research is facing at the moment. Moreover, creating better VR systems yields new challenges for developers of applications, which run on those systems. As LaValle puts it:

"If a headset is better in terms of spatial resolution, frame rate, tracking accuracy, field of view, and latency, then the potential is higher for making people sick through vection and other mismatched cues." [LaV17]

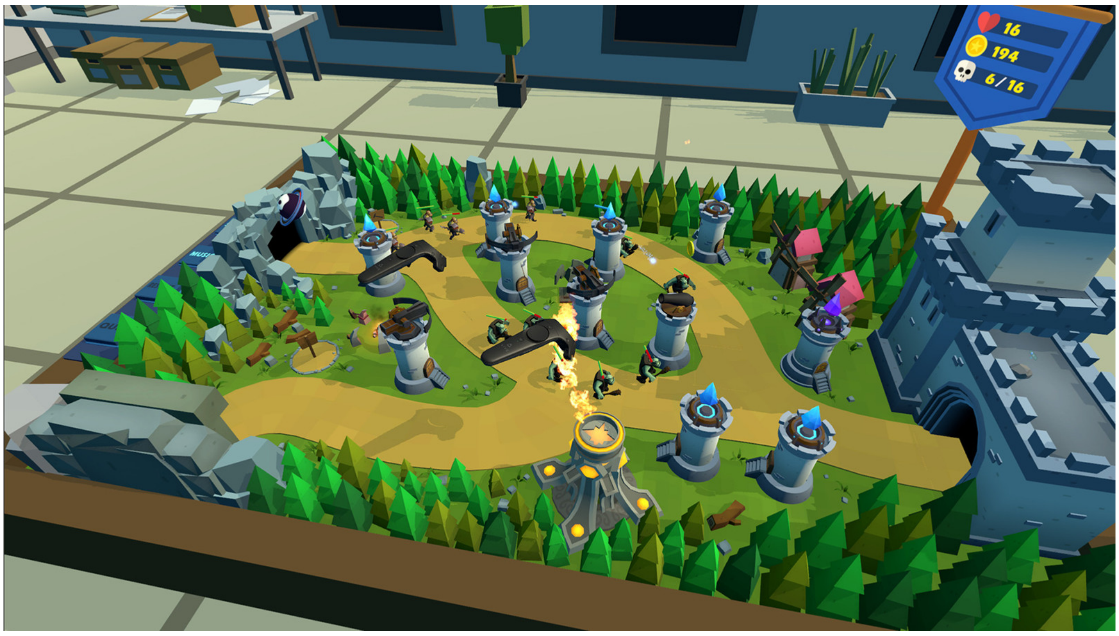
This means that the better VR systems become and the higher and more realistic the quality of the output is, the more present the user feels in the virtual world. If this world does not behave physically correct or at least as expected or predicted, then it has even more potential to make you feel sick. This is very important knowledge that has to be kept in mind for the simulation devised in the course of this thesis. The goal is not to develop any new hardware or mess with the rendering pipeline of the underlying CellView-framework, but to design the simulation in such a way that it does not cause sickness to users.
An essential role in this plays vection, which was already introduced in the beginning of this chapter. LaValle even claims that vection is "the largest culprit on VR experiences being made today" [LaV17]. In this section, various modern VR applications and their approaches to avoiding VR sickness caused by vection are discussed.
One popular approach is to avoid locomotion by design. This is, of course, not applicable to all software that is to be designed for VR. Some applications, like the one being developed in the course of this thesis, aim to move the user through a virtual world. However, games or simulations that have the user's virtual self stay at the same position for the whole duration cause no vection at all. Examples of this would be board-games, like chess, or games, like "O! My Genesis VR" for PlayStation VR. In these cases, the user remains stationary and interacts with objects, that are either stationary as well, like a chess board on a table, or move around in virtual space. In "O! My Genesis VR",which is shown in Figure 3.2a, the player has to care for a planed inhabited by different creatures. Therefore, he enters a position, where the virtual planet is only as big as a football. This planet is placed right in front of the user, who can rotate it with his hands and pick creatures up as he pleases. This causes no mismatch between visual and vestibular system when done right, as the observer's virtual self behaves exactly like his real self. If the tracked space is big enough for the user to freely walk around, VR simulations can also enable users to walk around a virtual room of the same size. This does not cause any vection either due to the same reason. In "Castle must be mine", a tower-defense game for the HTC Vive shown in Figure 3.2b, the user can walk around a virtual table as he pleases to get a better view of the battle, which takes place on top of the table.

While such simulations may cause close to no vection at all, they have apparent limitations. The user cannot leave the predefined play area whatsoever, as his virtual self can only move as far as the real room the VR system is located in allows him to. One approach to counteract this would be to use one of the solutions mentioned in the previous hardware section, such as the “Cyberith Virtualizer”. Because this might be expensive and still limits the user to basically move on a plane, VR games and application have developed various software based solutions. One would be to place the user in some sort of virtual “cage”, an environment, which is perceived as static in comparison to the users virtual self. Examples for this would be cars in racing simulations, planes or spaceships. In “EVE: Valkyrie”, which is available for HTC Vive, PlayStation VR and the Oculus Rift, the player takes on the role of a spaceship pilot. While the spaceship, which is steered via controller, moves through the virtual universe as seen in Figure 3.3a, the players position within the ship stays the same at all times, offering an easy orientation within the cockpit to avoid vection. “The Body VR”, a screenshot of which is seen in Figure 3.3b, follows a similar approach. In this simulation, the user travels through a hugely magnified human body. The user’s virtual self is sitting or standing in a capsule, which serves as static environment around the player. In comparison to the spaceship in “EVE: Valkyrie”, the capsule travels along a predefined path, not allowing for any interaction between the player and the path the capsule it taking. However, this enables the user to walk around in the capsule, look outside to observe the surroundings on each side or interact with the panel that is placed in the center. Although this is a good approach for showing guided tours, it limits the user’s possibilities for interacting with the virtual world he is passing through.

While these approaches can potentially counteract VR sickness caused by locomotion, it is not always possible to integrate such a “cage” in a meaningful way. In these cases, other supportive measures have to be implemented. One such would be to limit movement to some particular directions in relation to the user’s orientation. Two games that apply such techniques are “Drift VR” (Figure 3.4a) for Samsung’s Gear VR system and “Eagle Flight” (Figure 3.4b) for Windows and PlayStation4. In “Drift VR” the player acts as a virtual bullet flying through a building. The bullet and therefore the player only travels forward in direction of the users view, who can thereby control the bullet’s flight by rotating and panning his head. The goal of the game is to maneuver the bullet to its target without hitting any obstacles, such as walls, on its way. To fly around sharp corners, time can be slowed down for a few moments, which gives the user more time to readjust his head. Even without an environment in which the user is static, the limitation to only travel forward implies very easy control of the game and counteracts vection. In comparison, “Eagle Flight” lets the player fly through a post-apocalyptic version of Paris as an eagle. Although the basic principle is the same, the user always travels forward in his viewing direction, the game includes a more sophisticated way for the player to control his virtual “animalic self”. By tilting his head to the left or right, the player initiates a turn. Depending on the angle that the head is tilted, the turn is more or less sharp. Visual cues, like black artifacts on the side of the users field of view, towards which the curve is directed, are included to indicate that the limit has been reached, meaning that

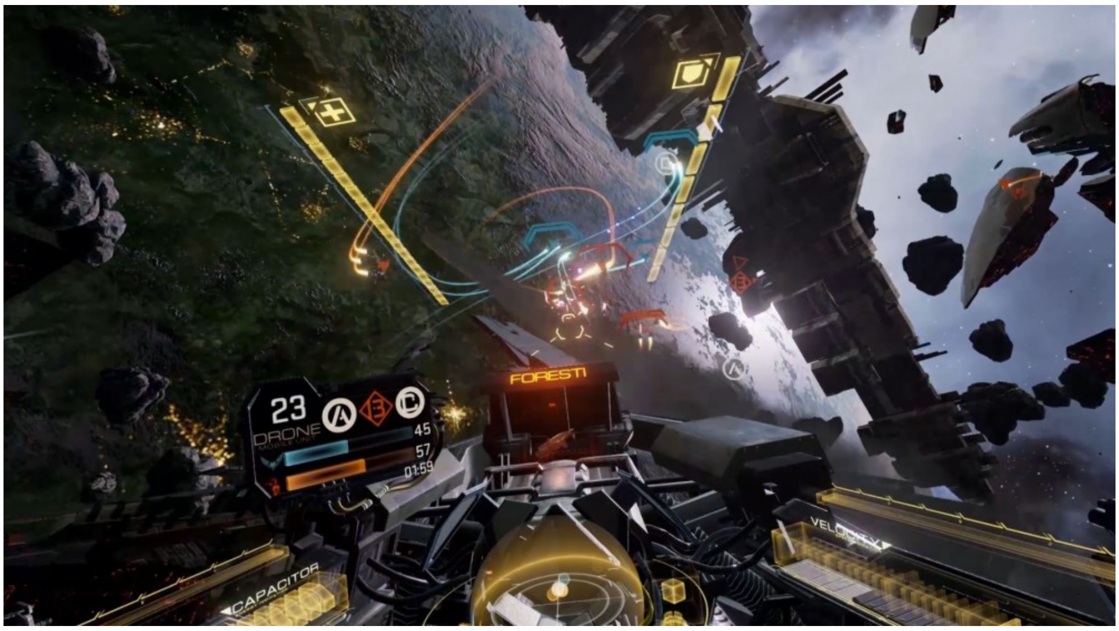


(a) Screenshot of the VR game "O! My Genesis VR", in which the user can interact with a virtual planet in VR. While the planet can be rotated and the inhabitants can be picked up, the user remains stationary. Source: `https://store.playstation.com`

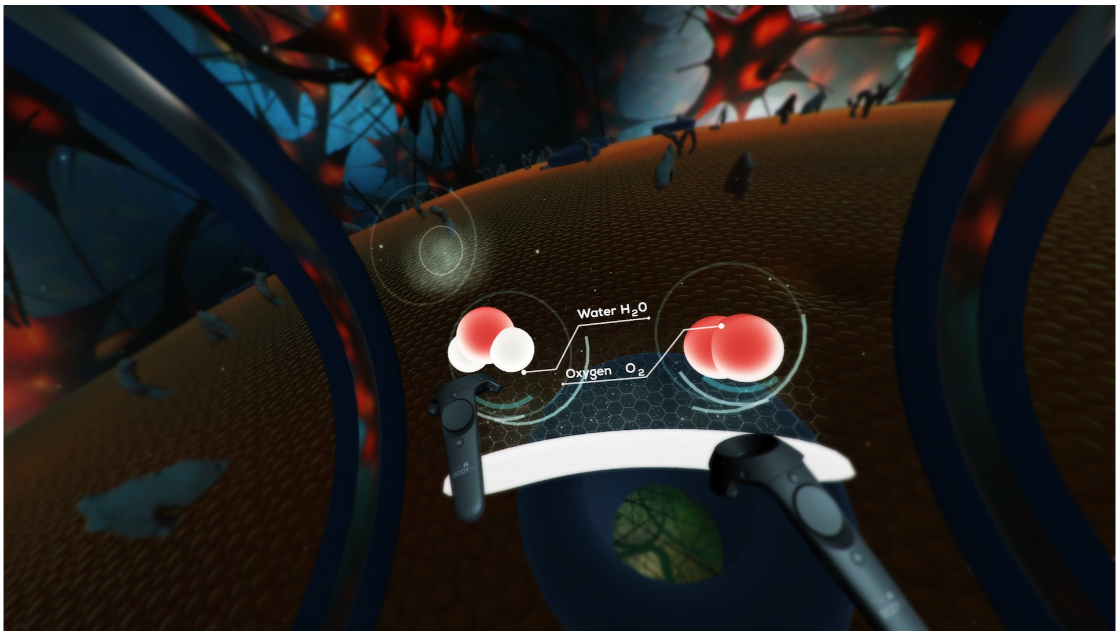


(b) Screenshot of "Castle must be mine", a tower defense game for the HTC Vive. The user can stand up and move around within the virtual room by walking in the real world. Source: `https://vrgamecritic.com`

Figure 3.2: Two examples of VR games, in which the user either remains stationary or can walk around in a restricted environment. No locomotion is involved.

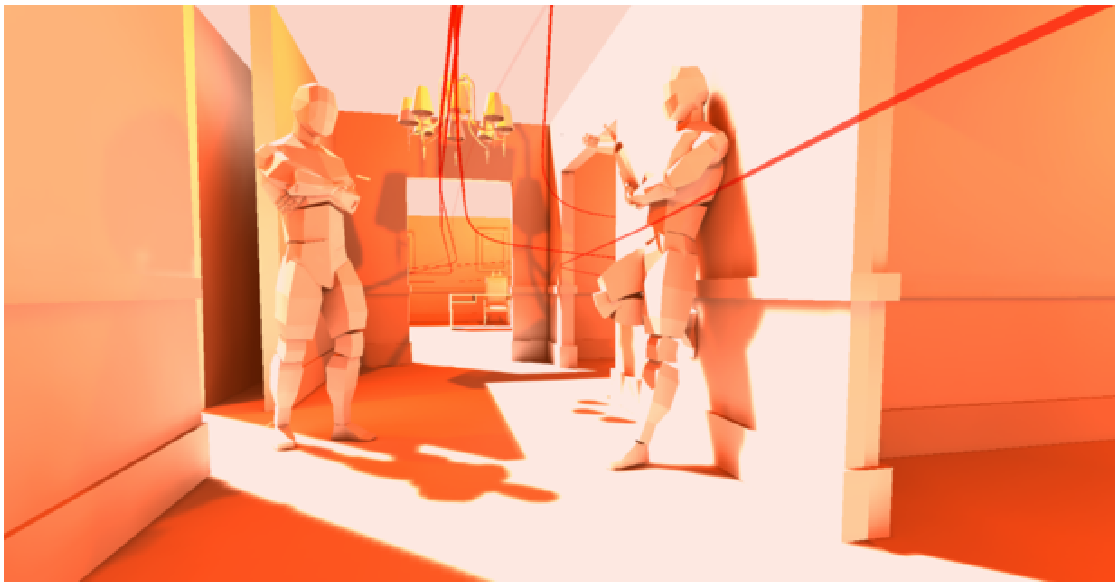


(a) In “EVE: Valkyrie”, the user fights in a space ship, which is maneuvered using a controller. While the ship moves through space, the user always retains his position relative to the ship within the cockpit, which serves as a point of reference in virtual space. [Jag]

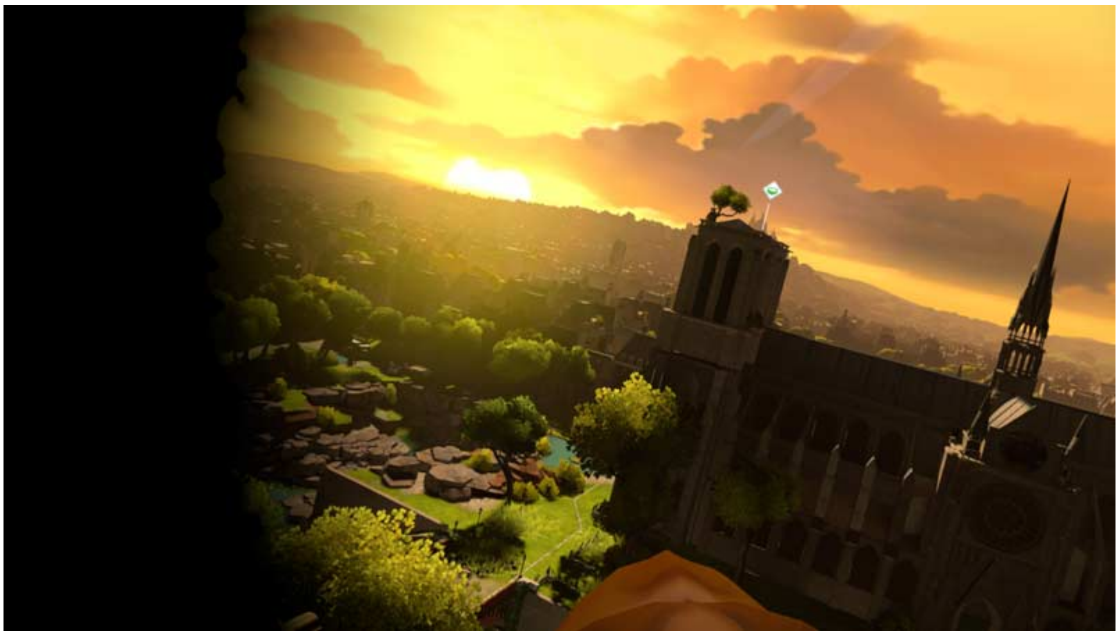


(b) “The Body VR” offers the user a guided tour through the human body. While the capsule the user resides in moves through the virtual world, the user himself can walk around in the capsule. [The]

Figure 3.3: Two examples of VR games, in which the user moves through virtual space in enclosed environments, which serve as a point of reference.



(a) In "Drift VR", the user steers a bullet through a virtual building to hit a target. This is done in first-person view by re-orienting the head. The paths that have been taken in previous attempts are visualized using red lines. Source: `https://vrjam.devpost.com/submissions/36336-drift`



(b) "Eagle Flight" is a first-person VR game that lets the user embody a virtual eagle. The eagle always moves forward and can be turned by tilting the head. The farther it is tilted, the steeper the turn. Additional cues like a beak in the center, decreased FOV and lines emerging from the center are provided to avoid VR sickness. [Tom]

Figure 3.4: Two first-person VR games utilizing locomotion to move the player around a virtual space much larger that the real space the user resides in. The user's flight is controlled via head movements in both games.

the bird cannot turn any faster. Other cues, like lines emerging from the center, indicate how fast the eagle is flying at the moment. Additionally, the user can see the beak of the bird where his nose would be located in the real world, which provides some kind of reference point. LaValle mentiones another effect, why those simulations are working without inducing heavy vection in most cases. As the user moves his head to orient it differently, which in turn results in a changed flight path, the vestibular senses are stimulated as well [LaV17]. If those signals received by the vestibular organs correspond with what the users sees, then the effects of vection can be reduced or mitigated.
The last popular approach discussed in this section is teleportation. Teleportation means that the user's virtual self instantly changes position, omitting acceleration and movement at a constant speed. In most cases, the user selects a point in space where he wants to go upon the press of a button. This is usually done by using ray casting. A ray, which can also be visualized, is sent into the scene and the point where it first intersects with an object serves as destination for the teleport. Those rays can either be adjusted by using a controller, which gives the impression of using a laser pointer, or by looking in a certain direction, in which case the ray is sent along the users viewing direction. In most cases, the user changes position instantly, therefore not causing any vection. However, it is important to keep in mind that changing locations quickly can result in loss of orientation for the user. An example of a game using teleportation is "Budged Cuts" (Figure 3.5), which is currently in development for HTC Vive. It utilizes portals, which can be fired from a gun using the Vive Controller to aim. The goal is to move through buildings as a spy without getting caught by the robotic security guards. As it is very fast paced, showing the user exactly where he will end up after teleporting is essential to not lose orientation.

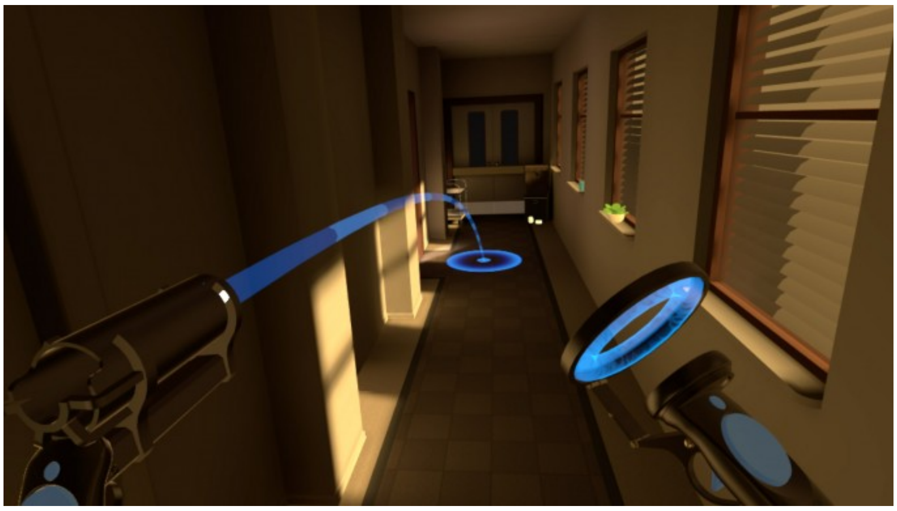


Figure 3.5: Teleportation is used in "Budged Cuts" to move the player around the virtual building. The user can aim the gun, which produces a portal to go through, using a Vive Controller. Source: `https://www.roadtovr.com`

CHAPTER 4

Methodology

Based on the problems of VR and related projects mentioned in the last chapter, this section aims to provide a theoretical background on the solutions applied in this thesis. As already explained in the introduction, the goal of this thesis is to implement a simulation in VR, visualizing the atomic crystal structure of a semiconductor while also allowing the user to explore it by guiding an electron through the crystal. Considering the aforementioned flaws and obstacles when developing applications, which allow the user to immerse into a virtual world, there are two that have to be kept in mind specifically for this thesis: locomotion and performance.

4.1 Solution Idea

To solve the issues of locomotion and performance, common techniques of related applications, some of which have been introduced in the last chapter, will be applied. To enable the player to move through the crystal structure composed of a large number of atoms, some form of locomotion has to be implemented because it is impossible to track a room as large as the virtual structure. Furthermore, walking around in a room only allows for movement in two dimensions, jumping and crouching aside. For this simulation, moving upwards and downwards is important as well, which is why hardware solutions like the "VirtuSphere" are not a good alternative either. The electron, that will fly through the grid, will interact with other objects, like the atoms in the structure and the oxide layer. If the electron comes near an atom, it will be repelled. As moving the player unexpectedly in Virtual Reality causes VR sickness very frequently, it is not a good idea to let the player's virtual self embody the electron directly. Therefore, the electron is represented by a separate object, a yellow sphere that is placed in front of the user. This allows to apply the forces and effects of collisions to that sphere and not to the player's virtual self, who acts as an observer instead. In the next section, it will be discussed how the player could move the electron and which approach fits the application

the best.
Aside from locomotion, bad performance is the second big factor that presents an issue when developing the simulation. The crystalline structure can potentially contain millions of atoms, many of which have to be rendered each frame. Therefore, the rendering capabilities of CellView are utilized [LMAPV15]. CellView is a framework for rendering huge amounts of spheres using billboards to gain performance. As it is based on Unity, a popular game engine, it is pretty straightforward to visualize scenes created with CellView in VR using the "SteamVR" plugin for Unity. As CellView, Unity and SteamVR take care of high-performance rendering, the factor that could potentially influence the performance of the simulation in the most negative way is the interaction between the electron and the atomic structure. As collisions between the electron and potentially millions of atoms have to be calculated and handled, the need for an effective collision detection system arises. In the second part of this chapter, it will be discussed how collisions are mathematically detected, which methods are needed for this simulation and high-performance collision detection in three-dimensional space can be achieved using an octree data-structure.

4.2 Locomotion and Interaction in VR

In the section about related work, several different methods of locomotion that are used in VR games and simulations have been introduced. Due to the nature of the simulation that is to be designed in the course of this thesis, some are more appropriate than others. Because allowing the player to move his virtual self using a controller's analog sticks is known to cause the most severe forms of vection [LaV17], the focus lies on approaches that move the player along his viewing ray. The simulation should furthermore allow the user to interactively explore the structure by taking control of an electron. Approaches like a guided tour within a static environment, like the capsule in "The Body VR" (Figure 3.3b), are therefore inappropriate. The electron, that is being controlled, is always placed in front of the user's face by using the following equation:

$$\vec{e} = \vec{p} + d * \vec{v} + \vec{c} \tag{4.1}$$

where $\vec{e}$, $\vec{p}$, $\vec{v}$ and $\vec{c}$ are three-dimensional vectors and d is a constant float value. $\vec{e}$ is the electron's new position that is calculated, $\vec{p}$ the player's position, $\vec{v}$ the direction the player is facing, d the desired distance between the player and the electron and $\vec{c}$ the collision vector, which will be explained later. As the electron is essentially controlled by the player's view, it does not make sense to integrate some sort of "cage" the user resides in, like a spaceship or a capsule. This would not fit the the theme of the simulation and lead to less immersion. Furthermore, the electron already serves as a point of reference for the player, making the creation of another one unnecessary.
The first approach for moving players in VR, which is popularly used in many applications, is teleportation. In this case, the user's position in the virtual world is instantly changed upon receiving an input, such as the press on a controller's button. Most of the applications utilizing this technique take place in some kind of limited setting, like a virtual room. This

allows the use of ray-casting, using the controller as pointer for where the player wants to go. In case of the simulation that is created in this thesis, there is no environment, such as walls or a floor, that would enable such functionality by serving as a point of reference. The only way teleportation could be implemented in this case is by moving the player a certain distance in his viewing direction. Mathematically speaking, the user's new position could be determined by the following equation:

$$\vec{n} = \vec{p} + d * \vec{v} \tag{4.2}$$

where $\vec{n}$, $\vec{p}$ and $\vec{v}$ are three-dimensional vectors representing the players new position ($\vec{n}$), his current position ($\vec{p}$) and the direction he is facing ($\vec{v}$). d is a parameter for the distance that the user's virtual self will be moved. It could either be fixed or calculated based on, for example, how long a button is kept down before being released. The main problems with this approach, which make it unusable in this simulation, are disorientation and physical incorrectness. As mentioned earlier, the user needs to know exactly where he is located after the teleport is finished. Due to the periodicity of the crystalline structure, it is possible that the user lands at a spot that looks almost identical to his start location, giving the impression that he has not moved at all. If the environment changes too much, the user cannot comprehend where he has moved. Additionally, the simulation should visualize the movement of an electron through a semiconductor. The electron is supposed to move at a certain speed and not change positions instantly and then stop for an undefined amount of time.
The approach that fits the simulation the best is constantly moving the player along his viewing direction at a certain speed, like it is implemented in “Drift VR” (Figure 3.4a), for example. Thereby, he remains aware of his position at all times while also allowing for easy control. The position of his self within the virtual world can be calculated in the following way:

$$\vec{n} = \vec{p} + s * \vec{v} * \Delta t \tag{4.3}$$

where $\vec{n}$, $\vec{p}$ and $\vec{v}$ are three-dimensional vectors, while s and Δt are scalar values. $\vec{n}$ represents the players new position after moving in direction $\vec{v}$ from current position $\vec{p}$. The parameter s determines the speed at which the player moves through the structure. Δt is the time that has passed since the last frame, which was when the last position update occurred. It is needed to assert that the movement is independent from the frame-rate. If it was not included, the player would move further when the frame-rate is high as the position is updated more often. The direction the player moves along is solely determined by the orientation of his head in the real world, which is tracked by the VR system's head-mounted device, thereby defining the viewing direction v. In this case, the player always moves forward. This limits the vection to only one dimension, which is forward, while also being very intuitive. The only disadvantage for the simulation devised in this thesis is that the player has to rotate his head by 180 degrees in order to turn around. While this can be helped by providing a chair that can spin around by 360 degrees, this can potentially be solved by implementing a way of automatically turning in VR. One application that solved this issue in a meaningful way is “Eagle Flight” Figure 3.4b. As mentioned earlier, the game allows the player, who embodies an eagle, to turn around by tilting his head. This feels very natural, as is requires only little movement, while also stimulating the vestibular system, reducing vection. However, it requires additional cues to feel as natural as it should. In “Eagle Flight”, the user moves over and through a model of post-apocalyptic Paris, providing points of reference at all times. Furthermore, it includes lines emerging from the center of the screen to indicate the speed the eagle is flying at, as well as a beak in the middle of the screen for reference. In our case, no such cues are given, making orientation harder as the head is tilted. The main reason it does not make sense in our simulation, however, is that the player does not primarily control his virtual avatar, but should focus on moving the electron through the structure. When testing this approach, it felt very unnatural and was therefore omitted in favor of the simpler approach mentioned above. The speed value s, which is needed in order to move, is controller by the user by pressing the trigger buttons on the controllers. Holding down one trigger accelerates the user while holding down the other one lowers the velocity. If none of the buttons is held down, s stays the same and the player moves forward at a constant speed.

4.3 Detecting Collisions Without Losing Performance

In the simulation devised in this thesis, it is essential for the electron to interact with the atoms of the semiconductor’s crystal structure and the oxide layer. Usually, collision is checked between objects that can be described mathematically. Complex models, like the body of a humanoid protagonist in a three-dimensional video game, are often surrounded by so-called bounding volumes, like boxes, spheres and capsules, to simplify detection of collisions. In our case, the electron and the atoms of the crystal structure are both rendered as spheres by the underlying cellVIEW framework, which makes spheres the most important bounding volumes in this simulation.
In a three-dimensional cartesian coordinate system, a collision between two spheres, which are defined by the coordinates of their centers and their radii, can be detected by comparing the distance between their centers and the sum of their radii. The euclidean distance between two spheres A and B can be calculated using the following equation:

$$d = \sqrt{(B.x - A.x)^2 + (B.y - A.y)^2 + (B.z - A.z)^2} \tag{4.4}$$

If the resulting value is smaller than the sum of the radii of A and B, the spheres intersect. If the values are equal, they touch. If the distance between the centers is bigger than the sum of the spheres’ radii, they neither intersect nor touch. In our simulation, we do not want the electron to intersect with any atom. So in case the electron intersects with an atom, the electron has to be moved away just far enough that it merely touches the atom. The collision vector, by which the electron has to be moved away from the atom, can be calculated using the following equation:

$$\vec{c} = \frac{(\vec{B} - \vec{A})}{|(\vec{B} - \vec{A})|} * ((r_b + r_a) - |(\vec{B} - \vec{A})|) \tag{4.5}$$

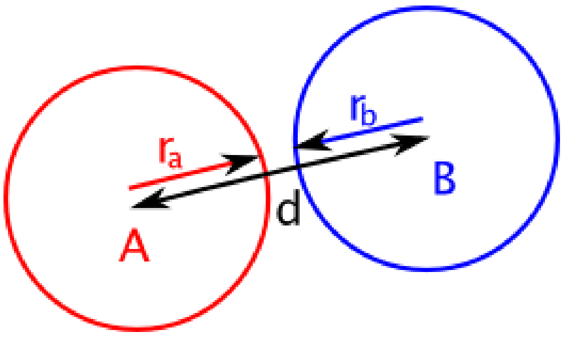


Figure 4.1: A collision between two circles A and B can be calculated by comparing the sum of their radii r_a and r_b to the distance d between their centers. This works the same way for spheres. Source: `https://wildbunny.co.uk`

where $\vec{B}$ is the position of the electron's center, $\vec{A}$ the position of the intersecting atom's center, r_b the radius of the electron and r_a the radius of the atom. The resulting vector $\vec{c}$ can then be added to the position of the electron's center to move it away from the atom, as it is done in equation 4.1.

Although calculating collisions between two spheres seems easily feasible when it comes to performance, problems can arise as the number of spheres in the simulation grows larger and larger. In our case, there might be more than a million atoms in the crystal structure, which are all mathematically represented by bounding spheres. Checking for collisions between the electron and all of those spheres every frame might lead to an underwhelming performance of the simulation, in turn resulting in an unpleasant experience at best and serious VR sickness at worst. Therefore, an algorithm that makes checking collisions in three-dimensional space faster has to be applied. The approach that is used in this thesis is based on spacial sub-division, which is "a rather ideal algorithm" for environments with uniform objects, according to Lin and Gottschalk [LG98]. As the crystal structure is cuboid and of fixed size, it can be easily divided into eight cuboid sub-spaces. These sub-spaces can recursively be divided further, which results in a hierarchical, tree-like data structure called "octree" by Meagher [Mea82].

Before this approach is explained, it is important to introduce a second bounding-volume that will be used besides spheres, called axis-aligned bounding-boxes or AABBs. An AABB is a box, who's edges are all aligned to either the x-, y- or z-axis of the underlying certesian coordinate system. Such a box can be defined by just two vertices, the minimum and maximum vertex. The minimum vertex equals to the point with the smallest x-, y- and z-coordinates that is still within the box. The maximum vertex, however, is equal to the point with the largest coordinates that is still within the box. Using this definition, it is possible to easily detect collisions between two AABBs or between one AABB and a sphere, which is the case that will be more important for the simulation. To check for an intersection between an AABB and a sphere, two steps are done. The first one is to find the point within the AABB that is closest to the sphere's center. Therefore, a technique called clamping is applied. For each axis, the sphere's center position is clamped to the box' minimum and maximum, limiting it to the interval between the minimum and maximum value of the box on each axis. In C#, this can be done by using the following code for each of the three axes:

Listing 4.1: C#-Code for clamping the center of the sphere to the minimum and maximum values of a box. The three resulting values represent the coordinates of the point within the box that is closest to the sphere's center.

```
var x = Mathf.Max(boxMin.x, Mathf.Min(sphere.x, boxMax.x));
var y = Mathf.Max(boxMin.y, Mathf.Min(sphere.y, boxMax.y));
var z = Mathf.Max(boxMin.z, Mathf.Min(sphere.z, boxMax.z));
```

The three resulting values make up the position of the point within the AABB that is closest to the sphere's center. For the second step, it is checked if this point is either inside the sphere, on its surface of outside the sphere. This is done by calculating the euclidean distance as it was shown in equation 4.4. This value is then compared to the sphere's radius. If it is smaller than or equal to the radius of the sphere, then the point is within the sphere, else it is outside.

This concept of AABBs is essential to the octree implementation in this thesis. As the crystal structure is only created at the start of the simulation, its dimensions stay the same during run-time, enabling us to represent the volume as an AABB. Therefore, the nodes of the tree are essentially axis-aligned bounding-boxes, which contain all atoms, mathematically represented by spheres, that intersect with the bounding box of the respective node. The generation of the octree that will be used is done in the following way: At first, the size of the space, which is contained within the tree, is determined.

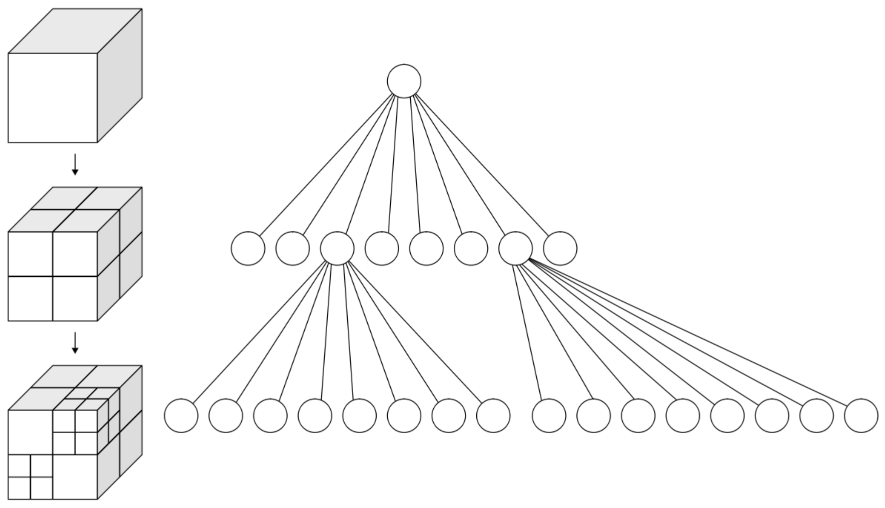


Figure 4.2: Schematic representation of an octree. While the root node covers the whole space contained in the octree, each of its children contains an eighth of that space. Some, but not all of them, are divided further. [Whi]

Therefore, the minimum and maximum points of the crystal structure are calculated and used to define the bounding volume of the whole tree. Next, the atoms are inserted into the tree. To achieve this, each atom is checked for intersection with the root first, which will always occur for atoms within the structure. Before the atom is stored in the octree data-structure, it is important to distinguish if the root or whatever node the sphere is inserted into has already been split or not. If it has not been split, then the sphere is stored in the octree-node. However, the performance gain of using an octree is based on its space-partitioning capabilities. If the octree only consists of one node, the root, then all atoms this node contains have to be checked for collisions against the electron, which is not the desired behavior as it does not increase performance compared to just using a list of all atoms. Therefore, a maximum number of atoms an octree-node can store is defined. If the node contains the maximum number of atoms and another one is added, then the node is split into eight children. Each child is a node as well, defined by a certain bounding-box, which covers exactly one eighth of the volume contained in its parent. If a node is split, all atoms it contains are distributed among its children, depending on which children's bounding boxes the atom's bounding-sphere intersects with. It is important to note that a sphere can very well intersect with more children, in which case it is added to each child-notes it intersects with. In particular, three cases can be distinguished when inserting an atom into an octree-node. In the easiest case, the octree-node has no children and contains less than the maximum number of atoms, which results in the atom being stored within the octree-node. If the octree-node has no children but already contains the maximum number of atoms, its space is split into eight sub-spaces and the atoms it contained are stored within each child-node they now intersect with, leaving the parent empty. When the octree-node has already been split, the sphere it is recursively inserted into each child-node it intersects with.
While the generation of the octree can take some time at the start of the simulation, it offers a performance gain when collisions have to be detected between the electron and the atoms of the crystal structure during run-time. At first, collision is checked for between the octree's root-node's bounding box and the electron's bounding sphere, to determine if the electron is within the crystal structure. If the electron is not within the structure, no collisions can occur and any further calculation is skipped. However, if the electron is within the structure, the octree is recursively traversed. There are different cases that can be distinguished when checking for collisions between the electron and the atoms contained in the sub-spaces of the tree. If the octree-node storing a certain subspace has no children, all atoms it contains are checked for collision with the electron. As soon as an intersection is detected, the collision vector is calculated and returned. Further iteration is not needed, as the electron can only intersect with one atom at a time due to the distance between the atoms in the structure and the electron's size in the simulation. It is also possible that sub-spaces remain empty. However, if the octree-node has been split into children, each subspace the electron intersects with is recursively checked.

4.4 Additional Considerations for Detecting Collisions in This Simulation

Because of the periodicity of the crystal structure and the way that cellVIEW works with proteins and atomic structures, which will be described in the next section, further improvements to collision detection can be made. Instead of saving the coordinates of each atom in the semiconductor, only one crystal is created. These crystals are then duplicated and placed next to each other, which means that each crystal shares the exact same structure. Furthermore, this means that it is unnecessary to create an octree containing all atoms of the semiconductor. Rather, the knowledge that the semiconductor is made of crystals with the same structure can be used to increase performance further. Instead of creating an octree for the whole semiconductor, we only create an octree for one crystal, as this crystal is cloned anyway. Next, the position of the electron within the semiconductor is calculated. Given that value and the fact that the crystals are placed next to each other along all three axis, we can find out which crystal the electron is located in. We can then calculate the position of the electron relative to this crystal's minimum position and use this value to iterate through the octree.
The collision between the electron and the oxide layers, which has not yet been mentioned, can be detected even more easily without using an octree. Therefore, the layers are abstracted by using one AABB to cover each layer, as those are axis-aligned as well. This means that only one calculation per layer has to be done, which is not computationally intensive.

CHAPTER 5

Implementation

While the previous chapters focused primarily on providing a theoretical background about Virtual Reality and its effects on the thesis at hand, this chapter aims to describe how the crystal structure simulation is implemented. Therefore, it is important to provide some information about the underlying Unity-based cellVIEW framework and the hardware that was used. Afterwards, the implementation will be explained in detail.

5.1 Prerequisites

As the purpose of this simulation is to enable users to interactively explore a virtual crystal structure, it is essential to allow interaction in real time. As those structures might consist of several millions of atoms, it is mandatory to have a solid visualization framework to build upon, which can render huge amounts of atoms in real time. cellVIEW was originally designed to "interactively visualize large biomolecular datasets on the atomic level" [LMAPV15]. To achieve that interactivity, it uses so-called "billboards", textured rectangles that face the camera at all times. Instead of processing a spherical mesh with a large number of vertices, billboards only consist of two triangles. An image of a sphere is then drawn onto that surface instead of using a three-dimensional model, which increases the performance drastically. It also offers an advanced level-of-detail implementation, which will, however, not be utilized by our simulation.
cellVIEW was developed using the Unity game engine, which allows to "prototype and deploy quickly and to leverage performance via advanced GPU programming" [LMAPV15]. Although cellVIEW does not make use of Unity's WYSIWYG-capabilities ("What You See Is What You Get"), it still allows for very easy testing and building of our code. Coding in Unity is mostly done using the object oriented programming language C#, which has been used for developing all extensions we needed to implement for this thesis. While offering an easily usable scene tree, which allows for adding three-dimensional objects into a scene and managing that scene, it also offers an extensive API, a drag-

and-drop system for public variables and references, as well as the opportunity to change public parameters interactively as the simulation is running. In Unity, all objects that are present in the scene are called “GameObjects”. A GameObject has a position and an orientation within the virtual world’s space and can also have meshes, scripts, colliders and other components attached to it. C# classes, which are attached to a GameObject and act as scripts, are automatically derived from the class “MonoBehaviour”, which provides methods that are called on startup or each frame, making it easy to implement specific behavior without having to manage a game loop. Not all objects created in C# code have to be attached to a GameObject though. Unity offers its own asset store, which allows for easy use of extensions developed by others. One particularly important extension for our simulation is the “SteamVR Plugin”, which allows for interaction between SteamVR and the Unity engine. SteamVR is an application for Microsoft’s Windows operating system that serves as an interface between the computer and various VR systems, such as the HTC Vive, which was used in this thesis. The SteamVR plugin, which can be downloaded from Unity’s asset store, offers GameObjects for tracking the camera rig and user input. By dragging the objects that are needed into the Unity scene tree, the user can navigate through the virtual world created in Unity by moving around in the real world, look around by orienting his head and give additional input by using the Vive Controllers and their buttons. Vice versa, the virtual camera rig can be moved using C# code, which allows us to move the user through the crystal structure. As discussed earlier, this locomotion could lead to vection and VR sickness in turn, so it has to be handled with care. Integrating the SteamVR plugin in cellVIEW was also done without much difficulty.
For developing a simulation like ours, special hardware is needed. The Windows PC that this project was developed and tested on contained an Intel i7-920 processor, 12 GB RAM and an NVIDIA Geforce GTX 1070 graphics card. The most important piece of hardware, however, is the virtual reality system. For this project, we used the HTC Vive, which includes an HMD, two controllers and two so-called “lighthouses”, which serve as tracking stations that monitor the users position and orientation by observing the HMD and controllers with lasers. These inputs are transferred from the HMD to the PC via USB, while the images that are generated on the GPU are sent to the HMD via HDMI, where they are then displayed on their respective screens. The controllers offer two kinds of buttons, which we used for providing input to the simulation: the trigger and the grip button. The trigger is placed at the backside of the controller and is usually pressed with the index finger. The grip buttons are placed on the side and while they appear to be distinct, pressing either one of those two, usually with either the thumb or the middle finger, leads to the same input.

5.2 Generating and Rendering the Atomic Structure

Because generating the crystal structure and visualizing it is not the main topic of this thesis, it will only be explained in short here. The atoms are characterized by three properties: position, radius and color. The position is a three-dimensional vector, which

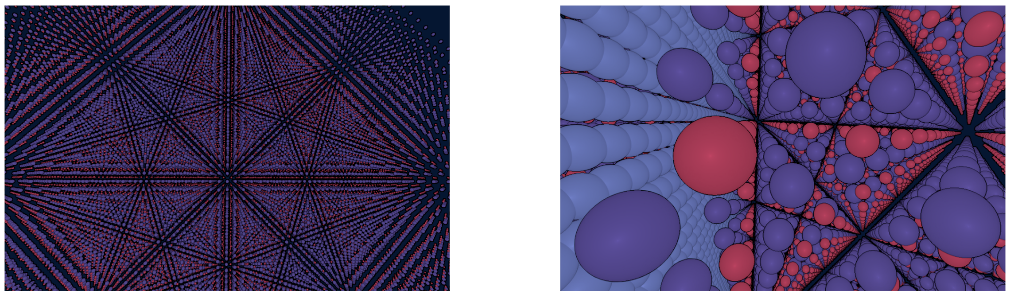

(a) (b)

Figure 5.1: Two screen-shots of the crystal structure visualized by the simulation. In both, the symmetrical properties of the structure can be recognized. In Figure (b), the oxide layer, which is represented by light-blue spheres, can be seen.

is stored as a "Vector3"-object in C#. The CrystalStructureGenerator generates the positions of all atoms within a crystal depending on the input parameters, such as primitive vectors and the lattice-parameter. As already mentioned, cellVIEW specializes in visualizing biological structures, but can still be adjusted to visualize crystalline structures. Usually, it renders proteins, which possess a certain atomic structure. If we replace the positions of those atoms with the positions of the crystal grid generated by the CrystalStructureGenerator, we get a protein with a crystalline atomic structure as a result. These artificially generated "proteins", which are crystals essentially, can then be placed next to each other to generate the large structure of a semiconductor. As soon as we provide cellVIEW with the necessary positions, radii and colors, it takes care of the rendering. This is done by using the methods CPUBuffers.Get.ProteinAtoms.Add(), CPUBuffers.Get.AtomColors.Add() and CPUBuffers.Get.ProteinInstancePositions.Add(). The atoms' positions and radii are stored as a four-dimensional vector in a buffer on the CPU, which is once per frame written to a to the GPU. As the atoms are vibrating depending on the temperature of the scene, this is necessary as the positions might have changed slightly. However, the updated positions must still be available on the CPU as the collision detection is implemented on CPU side.

5.3 Player Interaction

In the previous chapters, various causes for VR sickness were discussed, one of the most severe being vection. As the purpose of our simulation is to move the player through a huge structure of atoms, it is important to keep in mind that the movement must feel natural in order to create an immersive experience. Therefore, we introduced different approaches to locomotion in the related work section, while discussing their applicability for our simulation in the methodology chapter. Through discussion and testing we came to the conclusion that moving the player along his viewing ray at a certain speed leads to the best experience. This functionality is implemented in the CameraMover script that is attached to the eye-camera GameObject provided by the SteamVR plugin. As already mentioned, there are other components, which can be added to GameObjects besides scripts or meshes. One of those is the “Rigidbody” component that enables GameObjects to act accurately under physical laws and provides methods to change certain parameters. One of those parameters is the velocity, by which an object moves through space. The velocity is represented by a Vector3 object. The Rigidbody moves along the direction the vector points in, while the vector’s length determines the speed the object moves at. In our simulation, the Rigidbody component is attached to the camera rig, moving the whole rig through the simulation. As the CameraMover is directly attached to the eye camera, it can access its transform’s forward property, which is the vector that points in viewing direction from the eye point. By pressing the right trigger button on the Vive Controller, the user can accelerate. If the left button is pressed, the speed is increased. By multiplying the forward vector with the speed and setting it as the Rigidbody’s velocity, the user is moved through the crystal structure by the following C# code:

Listing 5.1: C#-Code for changing the velocity of a Rigidbody component.

```
rb.velocity = transform.forward.normalized * speed;
```

rb is a reference to the Rigidbody component attached to the camera rig, transform.forward is the camera’s forward vector and speed is a float that determines how fast the user is moving. This way, the player always travels in his viewing direction, experiencing vection in only one direction, as was discussed in the previous chapter.
To avoid VR sickness, it is important to avoid exposing the user to unexpected locomotion. As the simulation’s purpose is to move an electron through a structure of atoms, collisions between atoms and the electron can occur, which then have to be handled accordingly. In our case, the electron is repelled by the atoms, moving it in a different direction than the one it was moving in in the previous frame. This change in velocity would result in serious vection caused for the user, which is not what is desired. That is why we decided against making the user’s virtual self act as electron. Instead, an electron is rendered as spherical object in the center of the user’s field of view. Opposite to the atoms, which are not present as GameObjects in the scene but only exist as objects in the memory, the electron is a GameObject with a spherical mesh attached to it. As the user moves his head, the electron is moved as well in order to remain in the center. If a collision between the electron and an atom occurs, the electron is deflected, therefore pushing it out of the center of the user’s field of view. As soon as no collision is detected anymore, the electron returns to the center. This is realized by the following code:

Listing 5.2: C#-Code for repositioning the electron.

```
Vector3 elecPos = this.transform.position
+ this.transform.forward.normalized * elecDist;
elecBounds.setPosition(elecPos);
Vector3 cVec = collisionDetector.detectCollision(elecBounds);
electron.transform.position = elecPos + cVec;
```

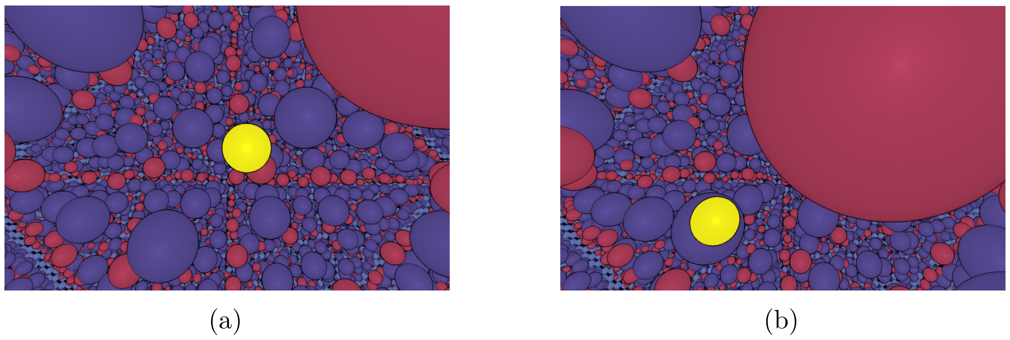


Figure 5.2: Two screen-shots of the electron, visualized as yellow sphere, being pushed away by an atom, visualized as red sphere. While the electron does not collide with any atom in Figure (a) and therefore remains in the center of the user's field of view, it is pushed out of the center by a colliding atom in Figure (b).

The electron's position is calculated using the camera position, its forward vector and a distance to the user that can be set in the user-interface. The “elecBounds”-object is a bounding sphere, that is then used by the CollisionDetector for detecting collisions using an octree. The CollisionDetector returns a vector, “cVec”, by which the electron must be moved in order to not intersect with any atoms in the structure. If all its components are zero, then the electron is not moved, as no collision has happened, and the electron remains in the center of the screen. In case the electron collides with an atom, the user receives additional haptic feedback as the controllers start to vibrate.
The electron can also collide with the atoms of the oxide layers between the single crystals. If the electron moves fast enough, it is able to “tunnel” through the layer, exiting on the other side. If it is too slow, however, it will be reflected by the layer. In this case, the velocity is inverted and the electron, as well as the user observing the electron, move backwards at a certain speed. Although exposing the user to additional vection, he still moves along the viewing ray but with negative speed.
Apart from moving through the structure and colliding with atoms and the oxide layer, there is another way the user can interact with the atomic structure. As the grip buttons on the controllers are held down, the temperature of the scene can be increased or decreased. Depending on a function that converts temperature into parameters for the atom's vibration's frequency and amplitude, which can be fully customized, the atoms behave differently at each temperature value. The user can define one value for frequency and amplitude for each temperature value, which serve as coefficients in the calculation of the vibration. Values in between two temperature values are interpolated linearly. At absolute zero, the atoms do not move anymore, making the regular crystalline structures of the semiconductor particularly visible.

5.4 Detecting Collisions

Due to the electron having to be repelled upon colliding with atoms, it is essential to detect those intersections properly while still retaining a frame rate that allows for interaction in real time. To accomplish that, an octree has been implemented and used, as argued in the previous chapter. For the detection to work, bounding volumes have to be implemented. In our case, an axis-aligned bounding box and a bounding sphere are sufficient. The box is defined by two Vector3 variables, one for the box' minimum and one for its maximum. The sphere is defined by its position, which is stored as Vector3, and a radius as float. The sphere offers methods to detect intersections with another sphere or a box, returning true if both objects collide. The electron has a normal bounding sphere and is represented as "Sphere" object for collision detection, as it is re-positioned externally every frame. The atoms, however, take care of their positioning themselves. Therefore, they are represented as "AtomSphere" objects, which are derived from the "Sphere" class. The only difference is that the AtomSpheres move themselves away and towards their position within the atomic grid depending on a sine-function of the current time. As the AtomSpheres are not present in the scene tree as GameObjects and therefore do not have an update function, the vibration is triggered via event system every frame by a GameObject called "VibrationController". However, detecting collisions for both Spheres and AtomSpheres works as mathematically described in the previous chapter.
The hierarchical octree data-structure is specifically designed to contain the Sphere objects and their derivatives only, which represent the structure of a crystal. Due to the properties of a crystal, certain assumptions can be made. For example, the space contained within the octree can be represented by an axis-aligned bounding box. It is also assumed that a Sphere can only intersect with one of the elements contained in the octree to speed up the detection process. The two core elements are the "Octree" class and the "OctreeNode" class. The Octree class offers an interface for other objects to interact with the data-structure and defines, what dimensions the octree has and at which number of elements the spaces are split into sub-spaces. Internally, the sub-spaces of the tree are represented by the OctreeNode class. Objects of these classes either store a list of Spheres that are contained within their space or a list of eight children, that each represent a sub-space that each is one eighth of the parent's size. As it is a recursive data-structure, the Octree class stores the head- or root-node of the tree. If the Octree's add- or detectCollision-methods are called, the calls are delegated to the root-node. The root-node then delegates it to its potential children, which recursively delegate it to their children and so on. While the purpose of the Octree class is mainly storing information about the data-structure, the logic is written in the nodes. A node is characterized by the space it manages, represented by a Box object, and the number of spheres it can contain before it has to split into eight sub-spaces. Of particular interest are the two methods add() and detectCollision().

Listing 5.3: C#-Code for adding a Sphere to an Octree-Node.

```
public void add(Sphere sphere) {

        if(children != null) {

            foreach(OctreeNode child in children) {

                if(child.intersectsWith(sphere)) {
                    child.add(sphere);
                }
            }
        } else {

            elems.Add(sphere);

            if (elems.Count > maxElems) {
                split();
                this.add(sphere);
            }
        }
    }
```

If a new sphere is added to the tree, it is checked whether the node the sphere is inserted into, which is the root-node at first, has already been split or not. This is indicated by the existence of child-nodes. If those are present, it is checked which sub-spaces the sphere intersects with. The sphere is then recursively inserted into all those sub-nodes. The absence of children indicates that the node has not yet been split because the maximum number of Spheres it can contain has not been exceeded. In this case, the new Sphere is added to the list of Spheres stored in the node. Afterwards, the number of elements in the node is checked. If the maximum is exceeded, the split-method is called, which divides the space managed by the node into eight sub-spaces and distributes the Spheres among those by adding them recursively.

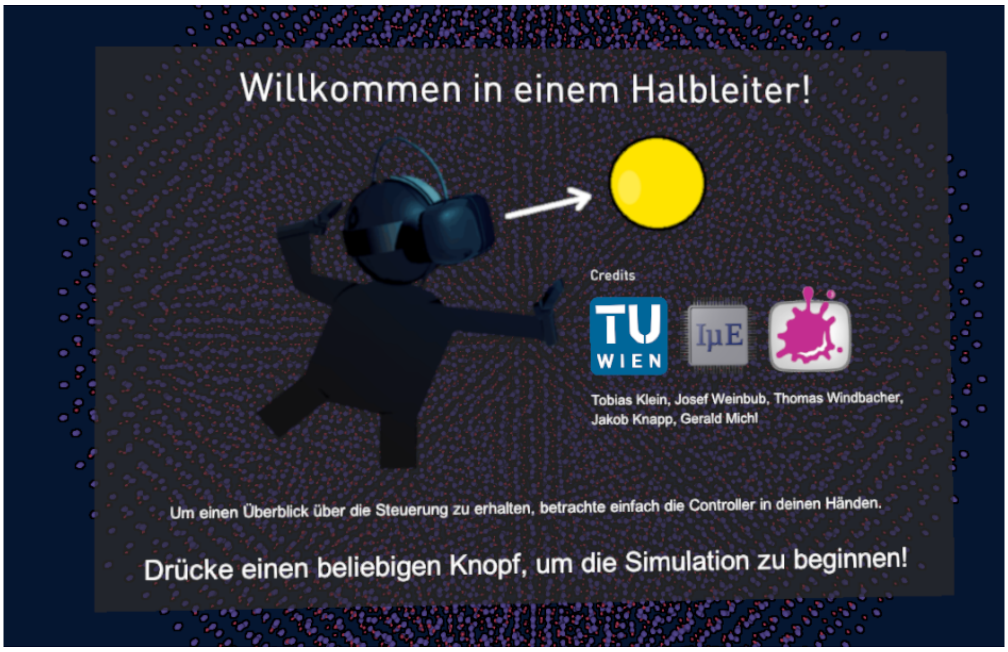
Collisions between any sphere and all spheres stored in the octree data-structure are also detected recursively by calling the root-node's detectCollision-function, which then delegates it to potential children. The function returns a Vector3 that equals to the direction in which the Sphere must be moved to not collide with any elements in the octree anymore. If the space managed by a node is empty and no child-nodes exist, then no collisions can occur and a three-dimensional zero-vector is returned. If the node has no children but contains elements, then all of those are checked for collision with the sphere. In case a collision occurs, the collision vector is calculated and returned, ending the collision detection process. If no collision is found, the zero-vector is returned. However, if the node has children, the detection process is delegated to each child whose space intersects with the Sphere.

Listing 5.4: C#-Code for detecting collisions between a Sphere and an Octree.

```
public Vector3 detectCollision(Sphere sphere) {

        if(this.children == null && this.elems.Count == 0) {
            return Vector3.zero;
        }

        else if(this.children == null) {

            foreach(Sphere elem in elems) {

                if(elem.collidesWith(sphere)) {
                    Vector3 collisionVector = (sphere.
                       ↪ getPosition() - elem.getPosition()).
                       ↪ normalized * ((sphere.getRadius() +
                       ↪ elem.getRadius()) - (sphere.
                       ↪ getPosition() - elem.getPosition()).
                       ↪ magnitude);
                    return collisionVector;
                }
            }

            return Vector3.zero;

        } else {

            foreach(OctreeNode child in children) {

                if (child.intersectsWith(sphere)) {
                   Vector3 collisionVector = child.
                      ↪ detectCollision(sphere);

                   if(!collisionVector.Equals(Vector3.zero)) {
                      ↪ return collisionVector;
                    }
                }
            }

            return Vector3.zero;
        }
    }
```

5.5 Displaying Information

Besides entertaining the user by allowing him to immerse into a virtual world populated with spheres of different sizes and colors, it is also important to inform the audience about what is going on in the scene and how it can be interacted with. Video games often solve this by providing some sort of tutorial, where the player can lay his hands on the controls in a restricted environment. As this simulation is made to be presented at science to public events, where lots of people might be waiting to try it out themselves, adding such a tutorial is too time-consuming. The better way is to add text at the right places, which can be much more difficult in VR than it is in conventional games played in front of a screen. While heads-up displays, abbreviated as HUD, work well in first-person shooters to, for example, display a map in the lower right corner, they are usually difficult to get to work in VR. In comparison to “normal” video games, which are perceived as an image on a surface, VR games aim to eliminate this concept. Players of VR games should ideally be unaware of the screen they are staring at and instead believe that they directly see the virtual world with their own eyes. Adding a HUD to a VR game does therefore not work, as it is not designed to display a screen-overlay, which is always perceived as part of the virtual world. In our simulation, the temperature of the scene was planned to be displayed as a HUD-element. However, as it always felt like a part of the virtual world, we moved it further to the center to associate it with the electron instead, as can be seen in Figure 6.1.



Willkommen in einem Halbleiter!

Credits

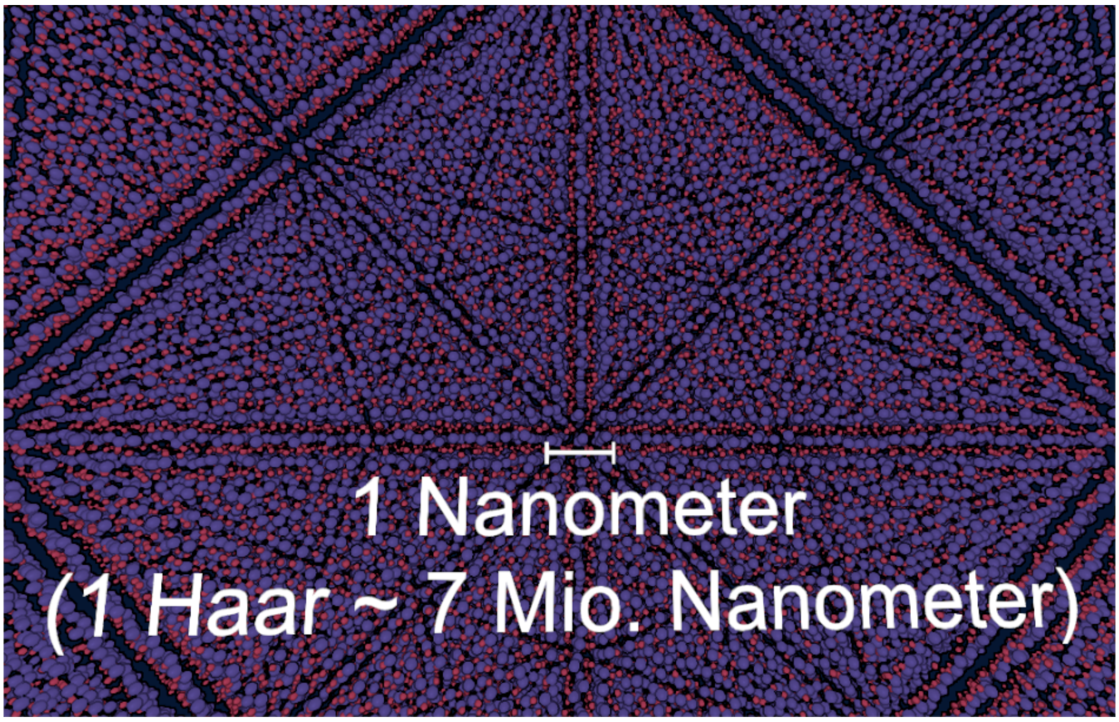
Tobias Klein, Josef Weinbub, Thomas Windbacher, Jakob Knapp, Gerald Michl

Um einen Überblick über die Steuerung zu erhalten, betrachte einfach die Controller in deinen Händen.

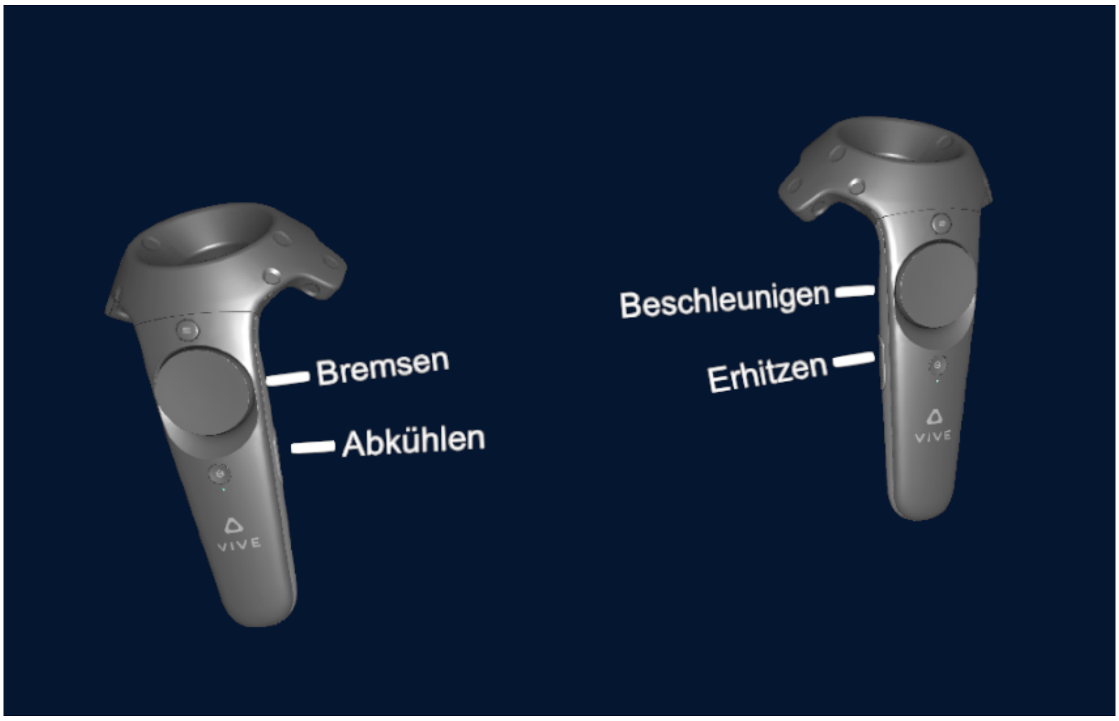
Drücke einen beliebigen Knopf, um die Simulation zu beginnen!

Figure 5.3: Screen-shot of the start-screen used in the simulation. The gray rectangle remains stationary within the virtual world as the user moves his head.

A better way to present information in VR is to integrate it into the virtual world as stationary objects, like billboards that are used for displaying advertisements in the real world. We applied this approach for the start-screen seen in Figure 5.3, which is a huge textured rectangle within the scene that appears in front of the user as the simulation is started. Another stationary object is the scale shown in Figure 5.4a that appears right after the start screen is deactivated. Because it should give the user an impression of how small the crystal structure is in reality, it must be anchored in the world too.
For providing information about the controls of the simulation, we implemented a different approach by attaching labels to the models of the Vive Controllers that are rendered in VR (Figure 5.4b). This way, the user is told which button causes which action in the virtual world just by looking at the controllers in his hands.
For providing information about the simulation as a whole, however, text is not the best option. It is important to make it clear that the object surrounding the user is a semiconductor, consisting of millions of atoms arranged in a crystalline structure. Of similar importance is to educate the user about the purpose of the simulation and in which way the simulation can be interacted with. At first, our approach was to add this information to the start screen as well, which was not well received by testers. It takes some time to read the introductory text, which can be exhausting in virtual reality, not least due to the sub-optimal PPI of modern VR headsets. Also, as previously stated, the user's time within the simulation may be limited due to high demand. Wasting this time by reading text that could also be displayed on a poster in the real world, for example, seems meaningless. Because of that, we decided to use audio to read the introductory text to the user as the simulation is already running. This way, he can already discover the structure while the context is still being explained via headphones.



(a) The scale is stationary in the virtual world and slowly disappears as the user gets closer to the crystal from his starting point. It serves the purpose of showing to the user how small the huge crystal is in reality.



(b) This Figure shows the labels, which are attached to the controllers. They inform the user about the functionality of the different buttons.

Figure 5.4: These two Figures show different approaches of visualizing information in VR. Text can be fixed within the virtual world, as shown in (a), or attached to objects as in (b).

CHAPTER 6

Results and Discussion

In this chapter, the results of the project that was implemented in the course of this thesis are presented and discussed. As the goal was to implement a software that will be used by the Institute of Microelectronics at the TU Wien to captivate an audience at public events, it will be evaluated according to how well it is received and how well it performs. The research that was conducted resulted in a Unity-project, which can be built and executed on a computer running Windows 10 and SteamVR. The resulting simulation can be experienced through a VR system, such as the HTC Vive. It satisfies all requirements that were asked from our customers from the Institute of Microelectronics. Those were to design a simulation, which allows a user to interactively maneuver an electron through the crystalline structure of a semiconductor. A screen-shot of the final version can be seen in Figure 6.1. By visualizing a huge structure in VR, the simulation is very immersive, while the underlying cellVIEW framework and the efficient collision detection algorithm allow for interaction in real-time. The fact that it was developed using the Unity engine asserts that the software is extensible and portable as well.

6.1 Reception

Unfortunately, by the time of writing the software has not been used at public events yet. Due to this, no feedback from people without scientific background can be provided. However, the feedback collected during two demo events, which were attended by colleagues from the Institute of Computer Graphics and Algorithms at the TU Wien and the customers from the Institute of Microelectronics, are presented and discussed.
First of all, it can be reported that no syndromes of VR sickness occurred to anyone that tested the simulation. This confirms that the measures that were taken to counter the effects of vection, primarily the introduction of a sphere representing the electron and limiting the user's locomotion to the forward direction, were successful. By applying forces and repulsion to the electron instead of directly to the user, unexpected movements

can be avoided. Additionally, the electron serves as a point of reference for the user, like the bird's beak in "Eagle Flight". Moving it around by orienting the head in the real world was very intuitive for the the testers. However, the electron's appearance and its behavior were not well received by all testers. One point of criticism was that the bright yellow color, which was chosen for its material, was too bright and the player's eyes might get tired of staring at a bright yellow sphere after a few minutes. Furthermore, it always stays at the center of the user's field of view, which further increases the induction of "yellow-blindness", as it was called. This could be counteracted by pushing the electron farther away as it collides with atoms, resulting an a longer period of time the electron takes to return to the center. While this is a valid point to consider when improving the software, it is not disruptive considering the purpose the simulation was designed for, which is showing it to people for only a few minutes.
In the version that was presented at the first demo event, the simulation included textured boundaries limiting the semiconductor. Because they were received as unnecessarily limiting the size of the structure as a whole, they were removed shortly after. Instead, a sphere around the player was introduced, which allows to only display atoms that are within the sphere. As the player moves, more and more crystals in front of him become visible, giving the impression of an endless structure.
Another thing to consider is adding more game-like mechanics to the game that "spice up" the experience for the user. As it is now, the user should focus on flying through the semiconductor, observing the structure of the crystals. Collisions with atoms should be avoided as much as possible and the oxide layer should be broken through with enough speed. However, there is no particular goal that has to be reached and collisions with atoms are not punished in any form. Success and punishment are important parts of video games though, which can influence simulations to "gamify" them, in turn creating motivation. As the project is a tightrope between scientific simulation and video game, certain trade-offs have to be made. Including features like high-scores and time limits to reach a goal might make the project more of a video game and less of a simulation that is presented at public events. On the contrary, focusing on complete physical accuracy might lead to an unpleasant experience with little to no learning effect or fun. Therefore, we decided to limit the vibration of the atoms to a sine-wave and not simulate the Brownian motion, as it might confuse the user and even cause VR sickness. However, during the second demo event it was suggested to add a feature where the path the electron has taken during the last simulation is visualized. This is an approach that is also implemented in the Virtual Reality game "Drift VR", where the user's last path is visualized to allow for improvement and comparison. As long as no goal for the electron to reach is defined, there is nothing that can be "improved" by following another path, it just adds another detail that may additionally entertain users.
The last point of critique included the way information was visualized. While some feedback was addressed in the final version, like the large amount of text on the introductory screen that was described in the previous chapter, some remains to be implemented in the future to further improve the simulation. To some testers, it was not clear what atoms the structure consists of. Despite the color coding, it might be a good idea to add

labels with names to the single atoms. When the atoms are vibrating, those should be deactivated as they cannot be read with ease. Additionally, connections between the single atoms could be visualized similar to the popular ball-and-stick model. Therefore, the rendering pipeline of cellVIEW has to be adjusted and a new collision detection system might have to be implemented if collisions between the electron and the "sticks" is desired. Another proposal for improvement similar to this was to implement a guided tour through the crystal. This is contradictory to the aim of the simulation, which is to interactively move an electron through a semiconductor. However, this could very well be a meaningful addition to the application in the future. It would also allow to implement some sort of vehicle the user travels in, like the capsule in "The Body VR", and go well with the concept of narrating the journey instead of presenting text to the user.
Apart from those points of critique, the simulation was very well received and described as funny, intuitive, impressive and exciting, especially by the customers. The great visibility of symmetries within the crystal was highlighted and the interaction between the electron and the atoms of the crystal and the oxide layer, as well as the functionality to change the temperature worked as expected.

6.2 Performance

The project was developed and tested on a Windows PC containing an Intel i7-920 processor, 12 Gigabytes of RAM and an NVIDIA Geforce GTX 1070 graphics card. Using an HTC Vive as output device, an average frame-rate of 44 frames per second (FPS) could be achieved when outside the crystal or when the detection of collision was deactivated. Although the performance could be increased to over 60 FPS when using the newer Geforce GTX 1080 graphics card for reference, this is still below the 90 FPS suggested by LaValle [LaV17]. However, this is mostly due to the cellVIEW rendering process. Detecting collisions using an octree did not heavily influence the performance when the right maximum value for the number of objects an octree-node can store was chosen.

As seen in table 6.1, finding the correct value is crucial to the effectiveness of the algorithm, even if one crystal only contains about 10000 atoms. Already splitting the node upon

Split Value	Frame Time	FPS
4	49.5ms	20.2
8	27.1ms	36.9
16	26.4ms	37.8
32	24.5ms	40.8
64	22.8ms	43.7

Table 6.1: This table shows the influence of a good splitting value on the performance of the collision detection algorithm using an octree. The values were gathered flying through a crystal containing 10890 atoms.

containing eight spheres decreases the performance by about seven frames each second. Splitting it at 64, the frame-rate is increased to about 43.7 frames per second, with each frame taking about 22.9 milliseconds to be generated. This is almost equal to the 44 frames per second that are achieved when collision detection is deactivated, indicating that the collision detection is not the bottleneck, which is lowering the performance. Despite the performance being sub-optimal in comparison to the 90 FPS proposed by LaValle, the simulation still offered a pleasant experience and none of the testers complained about the application's frame-rate.

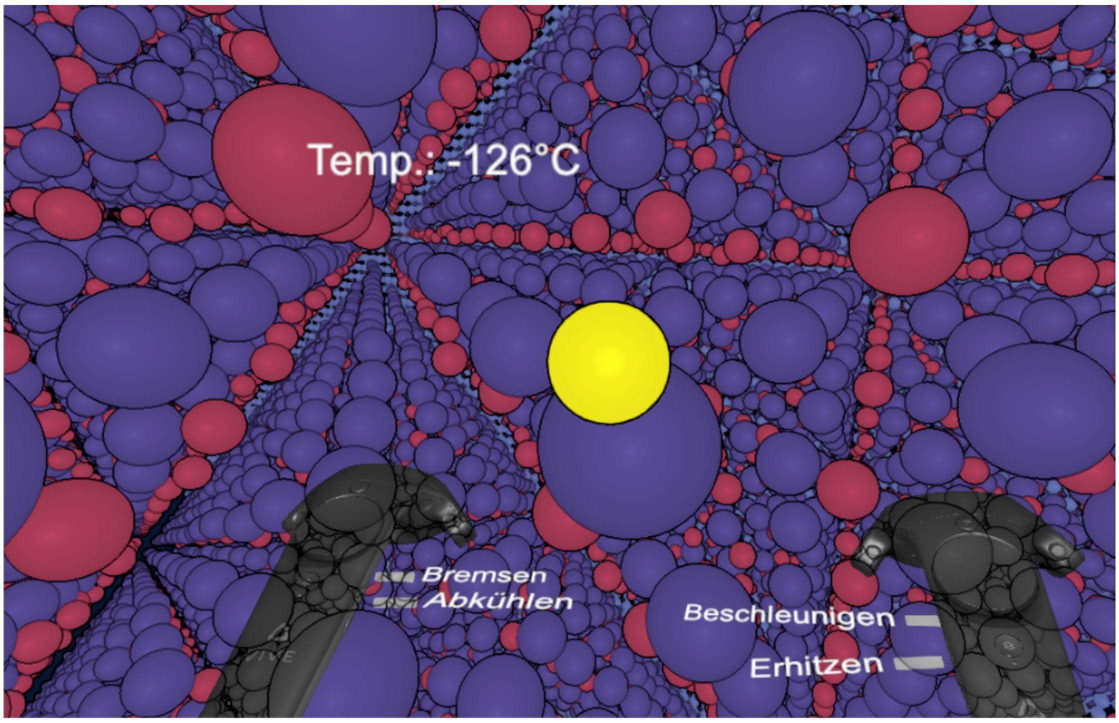


Figure 6.1: Screen-shot taken in the final version of the application. The yellow sphere in the center of the user's field of view represents the electron, which is controlled by the orientation of the user's head. The purple and red spheres are atoms, which are part of a silicon's crystal structure. The light-blue spheres in the back represent an oxide layer. The text above the electron is used to inform the player about the scene's current temperature, while the labels next to the controllers display the different possibilities for interaction.

CHAPTER 7

Conclusion and Future Work

In the course of this thesis, a Virtual Reality application was devised, which allows the user to maneuver an electron through the atomic crystal structure of a semiconductor by moving his head. This application, which was commissioned by the Institute of Microelectronics at the TU Wien, will be presented to a non-scientific audience at “science2public”-events and is designed to raise interest, to entertain and to educate at the same time. This makes it important to balance physical accuracy with elements borrowed from video games. To implement such a software, thorough research on Virtual Reality and the human perceptive system is necessary to avoid pitfalls, which make the experience unpleasant for the user by causing what is called VR sickness. Steven M. LaValle’s book “Virtual Reality” from 2017 served as a main source of information concerning details that need to be considered when implementing software that supports VR. According to LaValle, the two most severe causes for syndromes like nausea and dizziness are bad performance and bad interaction design. While we utilize the cellVIEW framework based on Unity for fast rendering of a large amount of spheres that represent atoms, there are still other factors to consider when it comes to achieving good performance. To assure that the detection and handling of collisions, which is needed in order to create an immersive experience, do not lower the frame-rate, an octree data-structure was implemented. To prevent the user from feeling vection, which can occur due to a mismatch of the visual and vestibular senses caused by moving the user’s virtual self through the virtual world, several different approaches were researched by analyzing popular VR games. Afterwards, these approaches were implemented, tested and compared. The best solution, which was automatically moving the user in his viewing direction, was then included in the final version.

Although the application was very well received by our testers, there is still room for improvement. Especially the way that information is presented to the user is an issue in Virtual Reality, as popular methods of doing that, like using HUDs, are not available. This could be improved by labeling the atoms at low vibration. Another approach would be to implement a guided tour through the crystal, which could eventually be combined with the “power-of-ten”-approach, where different objects are shown one after the other, with each one being several times smaller or bigger than the previous one. Besides visualizing the ball-and-stick model, it would be an improvement to visualize different semiconductors separated by oxide layers. As for now, only one crystal structure, which is duplicated to generate the whole structure, can be visualized at a time.

When it comes to performance, there is still a lot that could be improved, mainly regarding VR itself, but also concerning the application devised in this thesis. The frame-rate was around 60 frames per second on a Geforce GTX 1080 graphics card, which is below the 90 that are desired to fully utilize the 90Hz refresh rate of modern VR headsets. Software-based solutions like multi-resolution shading, foveated rendering and predictive rendering could further increase performance and credibility of the virtual world . Progress must, nevertheless, also be made in the field of hardware. Better graphic cards will inevitably increase performance. Retinal displays with higher pixel density and increased field of view have the potential to display images that cannot be distinguished from the real world by the human eye. If this can be achieved and the artificial virtual world behaves exactly as the real world is expected to behave, then one of the only things to work on is the stimulation of other human senses, like the vestibular system, to create a perfectly immersive VR experience.

Bibliography

[Afn] Sofia Afnaan. An Introduction to Virtual Reality (VR). http://www.technobyte.org/2016/03/introduction-virtual-reality/. Accessed: 2017-08-14.

[BC03] Grigore C Burdea and Philippe Coiffet. *Virtual reality technology*. John Wiley & Sons, 2003.

[CH14] Tuncay Cakmak and Holger Hager. Cyberith virtualizer: a locomotion device for virtual reality. In *ACM SIGGRAPH 2014 Emerging Technologies*, page 6. ACM, 2014.

[Cha] Brad Chacos. Nvidia's radical multi-resolution shading tech could help VR reach the masses. http://www.pcworld.com/article/2926083/. Accessed: 2017-08-13.

[CNSD+92] Carolina Cruz-Neira, Daniel J Sandin, Thomas A DeFanti, Robert V Kenyon, and John C Hart. The cave: audio visual experience automatic virtual environment. *Communications of the ACM*, 1992.

[COM98] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122. ACM, 1998.

[DDKN11] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pages 9–15. ACM, 2011.

[hal] halbleiter.org. Semiconductor Technology from A to Z - Oxidation. https://www.halbleiter.org/en/oxidation/. Accessed: 2017-08-13.

[Hei62] Morton L Heilig. Sensorama simulator, 1962. US Patent 3,050,870.

[IJs05] Wijnand A IJsselsteijn. History of telepresence. *3D Communication: Algorithms, concepts and real-time systems in human centred communication*, 2005.

[Jag] David Jagneaux. EVE: Valkyrie Review: Space Ace. `https://uploadvr.com/eve-valkyrie-review-space-ace-rift-launch/`. Accessed: 2017-08-14.

[Kit05] Achim Kittel. Festkörperphysik. `http://www.physik.uni-oldenburg.de/Docs/epkos/Festkoerperphysik.pdf`, 2005. Accessed: 2017-08-13.

[LaV17] Steven M LaValle. *VIRTUAL REALITY*. Cambridge University Press, 2017.

[LDP$^+$02] JJ-W Lin, Henry Been-Lirn Duh, Donald E Parker, Habib Abi-Rached, and Thomas A Furness. Effects of field of view on presence, enjoyment, memory, and simulator sickness in a virtual environment. In *Virtual Reality, 2002. Proceedings. IEEE*, pages 164–171. IEEE, 2002.

[LG98] Ming Lin and Stefan Gottschalk. Collision detection between geometric models: A survey. In *Proc. of IMA conference on mathematics of surfaces*, pages 602–608, 1998.

[LMAPV15] Mathieu Le Muzic, Ludovic Autin, Julius Parulek, and Ivan Viola. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In *VCBM*, pages 61–70, 2015.

[LYKA14] Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. Head tracking for the oculus rift. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 187–194. IEEE, 2014.

[Mea82] Donald Meagher. Octree generation, analysis and manipulation. Technical report, RENSSELAER POLYTECHNIC INST TROY NY IMAGE PROCESSING LAB, 1982.

[SN06] Simon M Sze and Kwok K Ng. *Physics of Semiconductor Devices*. John Wiley & Sons, 2006.

[Spe55] Eberhard Spenke. *Elektronische Halbleiter: eine Einführung in die Physik der Gleichrichter und Transistoren*. Springer-Verlag, 1955.

[SSMA$^+$14] Nancy A Skopp, Derek J Smolenski, Melinda J Metzger-Abamukong, Albert A Rizzo, and Greg M Reger. A pilot study of the virtusphere as a virtual reality enhancement. *International Journal of Human-Computer Interaction*, 2014.

[Sut68] Ivan E Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. ACM, 1968.

[The] The Body VR LLC. The Body VR LLC. https://madewith.unity.com/games/the-body-vr-llc. Accessed: 2017-08-14.

[Tom] Tomislav Bezmalinovic. Virtual Reality: Ubisofts majestätisches Eagle Flight im Test. https://vrodo.de/virtual-reality-ubisofts-majestaetisches-eagle-flight. Accessed: 2017-08-14.

[Wad02] Nicholas J Wade. Charles wheatstone (1802–1875), 2002.

[Web] Andrew Webster. PlayStation VR surpasses 1 million units sold. https://www.theverge.com/2017/6/5/15719382/playstation-vr-sony-sales-one-million. Accessed: 2017-08-13.

[Whi] WhiteTimberwolf. Schematic drawing of an octree, a data structure of computer science. https://commons.wikimedia.org/wiki/File:Octree2.svg. Accessed: 2017-08-14.