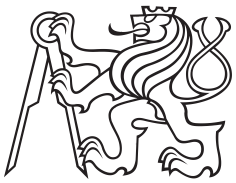
Bachelor's thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Eye-Tracking Tools for Virtual Reality System

Nástroje pro sledování očních pohybů v systému pro virtuální realitu

Radek Nesnídal

Supervisor: Ing. Karel Fliegel, Ph.D.
Field of study: Electronics and communication
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nesnídal Radek** Personal ID number: **491837**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Radioelectronics**

Study program: **Electronics and Communications**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Eye-Tracking Tools for Virtual Reality System

Bachelor’s thesis title in Czech:

Nástroje pro sledování očních pohybů v systému pro virtuální realitu

Guidelines:

Give an overview of recent eye-tracking-based techniques in virtual reality systems with head-mounted displays. Using available open-source libraries, develop software tools to perform related experiments for provided virtual reality system. Focus mainly on eye-tracking integration into the omnidirectional video player pipeline. Deliver support for demonstration tasks.

Bibliography / sources:

[1] Jin, Y., Chen, M., Goodall, T., Patney, A., Bovik, A. C., Subjective and Objective Quality Assessment of 2D and 3D Foveated Video Compression in Virtual Reality, IEEE Transactions on Image Processing, 2021.
[2] Jin, Y., Chen, M., Bell, T.G., Wan, Z., Bovik, A., Study of 2D foveated video quality in virtual reality, Proc. SPIE 11510, 2020.

Name and workplace of bachelor’s thesis supervisor:

Ing. Karel Fliegel, Ph.D. Department of Radioelectronics FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **01.02.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Karel Fliegel, Ph.D.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt | Student’s signature

Acknowledgements

I would like to thank to my supervisor Ing. Karel Fliegel, Ph.D. for guiding this work.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

In Prague, 15. May 2022

Abstract

The aim of this thesis was creation of suitable environment, where eye-tracking experiments can be performed, as well as delivering support for demonstration task. I over-viewed recent techniques for eye-tracking utilization, such as foveated compression or analytical purposes. I focused on developing eye-tracking software tools, with the use of open-source libraries, and also on the visualization of the acquired data. The presented demonstration task consists of watching multiple compressed videos in virtual reality, while the user's gaze would be tracked.

Keywords: eye-tracking, foveation, HTC VIVE Pro Eye, omnidirectional video, virtual reality

Supervisor: Ing. Karel Fliegel, Ph.D.
CTU FEE,
Technická 6,
16000 Prague 6

Abstrakt

Cílem této práce bylo vytvoření vhodného prostředí, kde by mohli probíhat experimenty se sledováním očních pohybů, a zároveň připravení podpory pro demonstrační úlohu. Byl podán přehled moderních technik pro využití sledování očních pohybů, jako foveovaná komprese či analytické využití. Zaměřoval jsem se na vývoj softwarových nástrojů pro snímání očních pohybů, s pomocí otevřených knihoven, a také na vizualizaci získaných dat. Uvedená demonstrační úloha se sestává ze sledování několika komprimovaných videií ve virtuální realitě, zatímco jsou snímány uživatelovy oční pohyby.

Klíčová slova: foveace, HTC VIVE Pro Eye, sledování očních pohybů, virtuální realita, všesměrové video

Překlad názvu: Nástroje pro sledování očních pohybů v systému pro virtuální realitu

Contents

Figures

Tables

Chapter 1
Introduction

Popularity of virtual reality (VR) headsets, as well as embedding eye trackers to them, has risen in recent years. This fact is a result of technological advances, that have been lately achieved, as there was demand for better parameters of the VR systems. The process of tracking user's gaze in VR environment brings new possibilities into this segment, as while we have limited transmission bandwidth, there are needs for better video quality [1]. Chapter 'VR related fundamentals' describes the basics related to VR, eye-tracking and omnidirectional footage.

The fact, that we need to keep in mind and that we want to take advantage of, is that only small part of the full omnidirectional footage can be seen by the user, which is called *viewport* [2]. With this knowledge we come to one of the popular eye-tracking utilization - *foveated compression*. This method of compressing the footage only sends the better-quality footage, where the user's most sharp vision is currently at, degrading the quality with the distance from the viewport [3]. This field is covered in the chapter of this thesis 'Foveated compression'.

Video compression however isn't the only sphere, where eye-tracking is being utilized. As tracing the position of one's gaze carries a lot of information about their visual acuity, eye-tracking is now being widely used for experiments, where we want to know where users are looking during omnidirectional videos, or in any other VR-suited environment [4]. From the collected data various visualisations can be made, where among the most popular ones are *gaze plots* and *heatmaps* [5]. Creation of this environment, as well as analysis of the obtained data are part of the chapter 'Omnidirectional video player with simultaneous eye-tracking'.

In chapter named 'Other fields of use of eye-tracking' I overview previously not mentioned areas, where eye-tracking is being utilized. There I focus on two fields, video gaming and medical research, where in both there have been various ways, how eye-tracking brought new possibilities into those sectors.

Chapter 'Eye-tracking experiment and setting up a demonstration task' focuses on the practical part of this thesis, where the process of setting up an eye-tracking experiment in Unity environment is described, as well as creation of an omnidirectional video player, in the mentioned environment. After that, I write about how the eye-tracking data are being accessed, and

the issues revolving around it. The next section then focuses on processing of the obtained data, where I discuss the way how the data are plotted and various visualisations are showcased. The final section describes the logic behind setting up a demonstration task, where the user would watch video with various degrees of compression, while their gaze would be simultaneously tracked.

Chapter 2
VR related fundamentals

In this chapter I overview the fundamentals related to VR systems. Through this chapter the reader should get familiar with basic concepts revolving around VR, eye-tracking and omnidirectional footage.

2.1 Virtual reality

Virtual reality headsets represent a quite wide range of products with one common feature - they all are able to display an omnidirectional video or any other kind of omnidirectional content. The way it is done is through a head mounted display, that is attached to a headset, typically just a few centimeters from user's eyes. This way, the user can feel way more connected with the content, as VR gives them impression, that they are part of the scene [6].

Among the most popular headsets nowadays are for example Oculus's Quest 2[1], HTC's VIVE Pro[2] or HP's Reverb VR3000 G2[3], which are all meant to be connected to a personal computer. However another fields of use for VR had been growing, such as their use with gaming consoles with PlayStation VR or the usage of mobile phones as the screen, where we put the phone into a special headsets made for this purpose.

The differences between the VR headsets from different manufacturers are for example resolution and size of the built-in screen, its refresh rate or the present sensors in the headset. To deliver satisfactory user experience, it was stated that the resolution should be at least 6K (6144 by 3160 pixels), while the refresh rate should be at least 90 Hz. This however brings a new problem as high bit rate results into high bandwidth consumption [7].

Typical use of VR is for video gaming, where the user usually interacts with the environment by using hand-held controllers. Another utilization can be found for viewing omnidirectional videos or photos, where the user can turn around to see different parts of the footage [3].

[1] https://store.facebook.com/quest/products/quest-2/
[2] https://www.vive.com/us/product/#pro%20series
[3] https://www.hp.com/us-en/vr/reverb-g2-vr-headset.html

2.2 Eye-tracking in VR

While eye-tracking is known technique, that has already been used for years, in VR environment it is quite modern concept, that makes the VR headset a research tool, that can be easily obtained by wide range of people. One of the main advantages here is that the VR environment is highly controlled, while still providing some degree of freedom to the user [8].

The reason why we want to track the gaze of a user can vary - we can use it for research, as data containing information about where and for how long the user is looking in given direction, can tell us a lot about visual attention and mental processes [4]. When analysing the eye-tracking data, we can either visualize the data, or we can use quantitative metrics, that are based on fixations and saccades. Thanks to this we can get information about areas of interest, where we would see longer total fixation duration, or whether the participant has trouble with processing of some information, which could be spotted due to long time of average fixation [9].

In order to setup an eye-tracking experiment, according to [8] following parts are needed

- *VR headset* with embedded or attachable *eye tracker*
- Suitable *software* - most commonly Unity[4] or Unreal Engine[5], where we can create omnidirectional environments, while simultaneously we can collect the eye-tracking data thanks to scripts
- *PC* with sufficient computing power, in order to run the experiments flawlessly
- Other - for example *cable management*, so that the participant has freedom of movement, or *headphones* so that the participant isn't distracted by sounds that do not come from the application

2.3 Viewing an omnidirectional footage

In Unity environment, in order to properly display the video on a sphere surrounding the user, it first needs to be processed. Here the video is viewed like a set of six frames, known as *Cubemap*, which can be seen in Figure 2.1 [6], and which was created from the video available at [10]. Each one of those frames contains information about the front, back, right side, left side, bottom and top of the footage, which together carry all the data about the video while also being a good way of storing spherical data, as it doesn't carry that much redundant data as equirectangulary projected footage [11].

[4] https://unity.com/

[5] https://www.unrealengine.com/en-US

[6] The Cubemap was created using an equirectangular image, which was then converted using https://jaxry.github.io/panorama-to-cubemap/.

Figure 2.1: A Cubemap projection of omnidirectional footage, which is used by Unity.

For my purposes, I need to later view the whole omnidirectional footage on a 2D computer screen, which means that I need to convert the video into suitable way, so that it is clearly visible what is happening - this factor makes the Cubemap a bad choice, as the division into frames might be too confusing for the viewer. In this case we mostly use equirectangular projection, which is the same one used for example for projection of the world onto a flat map. This projection is done by converting spherical coordinates into the equirectangular ones, which then creates the 2D map. This however comes at cost, as the footage now becomes deformed, as parts of the image become stretched. The aftermath of this can be seen in Figure 2.2, where we can see how for different viewing angles the area of viewed space changes on equirectangular map, as well as its shape [12].

However we have to keep in mind, that the projected 2D planes are not just affected by geometric distortion, but also by artificial creation of borders, which causes discontinuity of the projected footage. This leads into a problem, where the traditional compression methods for 2D videos are not applicable and other suited approaches are necessary [13].

In the Figure 2.3 it is shown, how the same video from [10], that was in the Cubemap in Figure 2.1, looks like in equirectangular map.

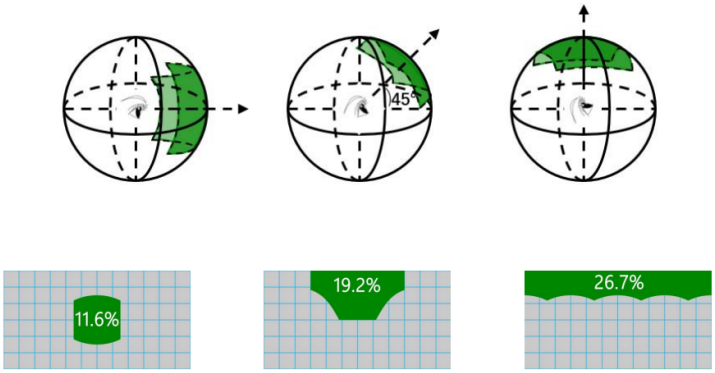


Figure 2.2: Various viewing angles for constant 90° vision, and their impact on mapping onto equirectangular maps [12].

Figure 2.3: An equirectangular projection of an omnidirectional video.

Chapter 3

Foveated compression

Foveated compression is one of the recent eye-tracking-based techniques in VR that keeps on getting more popular. Knowing the position, where user's most sharp vision is, at given time, allows us to compress the quality of the footage, which is outside of it. This part of one's sight with highest visual acuity is called fovea, and the described type of compression is also named after it. In this chapter I write about the human visual system (HVS), how the footage is compressed and also how latency might negatively affect user experience.

3.1 Human visual system

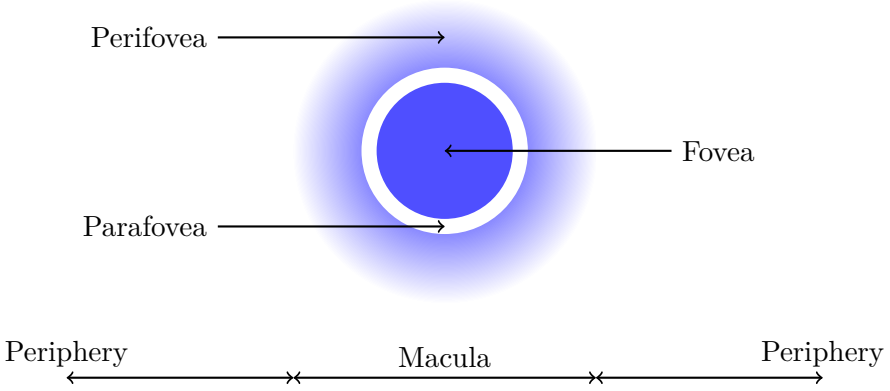


Figure 3.1: Regions of retina (inspired by [2]).

To understand the problematics of foveated compression we need to know how the HVS works. As we can see in Figure 3.1, the retina consists of two main parts - macula and periphery. Macula can be then divided into three smaller parts - fovea and transition region consisting of parafovea and perifovea, where we can see linear density falloff - this effect is shown in Figure 3.1, where the color of the perifoveal part gets gradually lighter. So

can we halve periphery into its near and far sections [2],[14]. The human vision related to the mentioned segments is specified in [2] as

- Fovea - 5° of the central part of eye's vision with high density of cones - this region has highest susceptibility to fine details
- Parafovea - sight for eccentricity between 2.5° and 4° which contains more rods than fovea leading into lightly worsened perception for details
- Perifovea - region related to eccentricity between 4° and 9°, where the rods start to outnumber the cones
- Near periphery - perception for eccentricity between 9° and approximately 30° - linear regression of visual acuity as the eccentricity ascends
- Far periphery - vision beyond eccentricity of 30°, with low density of rods, where the acuity falloff is way steeper

The part of the eye we are especially focusing on is fovea, as it is the most significant part of our vision. In Figure 3.2, we can observe the distribution of light sensitive cells (cones and rods) in human eye. A spike in number of cones can bee seen in the fovea's region. Cones allow us to perceive color and great quantity of visual details, which gives us the ability to do highly detail-oriented tasks, such as reading [15].

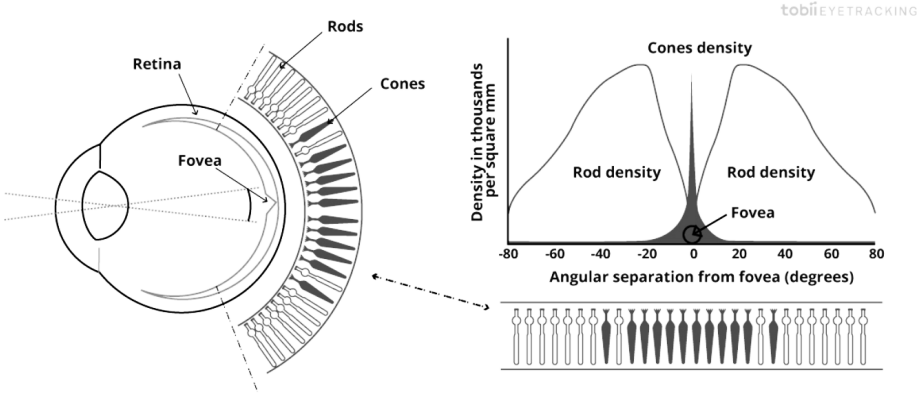


Figure 3.2: Simplified illustration of the distribution of cones and rods on the retina [15].

On the other side rods, which are the cells mainly located in peripheral vision (as can be seen in Figure 3.2), are tuned for situations, where we don't have much light. Rods are not capable of seeing color, but they are able to sense contrast and movement, which is something we have to lookout for when applying foveated compression [15].

In the VR environment people do not tend to behave all the same. While all people give their attention to faces, signs or moving objects, the time they spend looking at them varies. Some people tend to only focus at those objects, but some on the other hand tend to look mainly on the surrounding

and focus on these points just for a short period of time. Another interesting event happens when people watch the same video multiple times, as they start to focus on one point for longer time before moving their head an focusing on other point [9].

3.2 Compression of an omnidirectional footage

To determine the distortion of foveated footage, we need two parameters - inner radius of each region and the level of compression distortion withing the areas. In typical implementation of foveated compression we separate the field of view into three concentric parts, where various degrees of foveation can be applied. It is possible to use more than three concentric regions as it improves the level of foveation, however for research purposes it might be too time-consuming, as quantity of footage that needs to be analyzed, would rise [3].

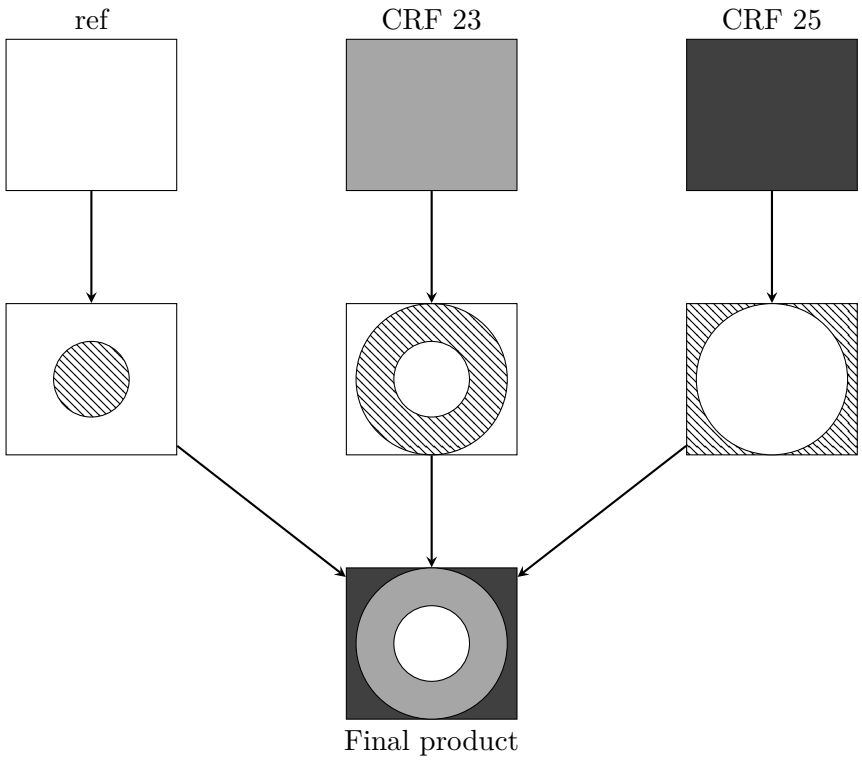


Figure 3.3: Demonstration of creation of foveated footage (inspired by [3]).

In Figure 3.3 we can see the process of creation of foveated footage. In the case of this figure we have three various qualities, with one of them being the reference uncompressed video, and the other two are compressed by H.264 codec with Constant Rate Factors (CRFs) of 23 and 25. Every one of these videos are used for different parts of the final footage, where for the innermost part we use the reference footage, because the user's gaze is the sharpest in

here. After that, the second best quality, which in this case is the CRF 23 compressed video, is then used for the part surrounding the reference video and for the rest of the footage we use the worst quality video - in this case CRF 25. This way we gradually worsen the quality of the footage with the distance from the middle of the circle, which is determined by the position of user's fovea.

However we can use more than three levels of compression and also we can have more radii, that we could work with. Nevertheless, the logic stays the same, and the process still consists of two steps, that result into creation of foveated footage, with the quality being the best in the middle, and then consistently dropping with the distance from the user's fovea. With more parameters we however have to face the difficulty, that the total number of possible variation of foveated videos increases [3].

Various types of foveation techniques can be implemented for peripheral compression - different kinds of techniques have their own specifics that affect the way the footage is compressed. The technicalities, that we mostly care the most about, are how much they can compress the footage and how much computing power is needed for this compression.

In [14] they tested three types of peripheral compression - fCPS, subsampling and Gaussian blur (see Figure 3.4). It was also examined how the methods differ when we change the fixed eccentricity for far periphery - peripheral eccentricities of 5°, 15° and 20° were applied.

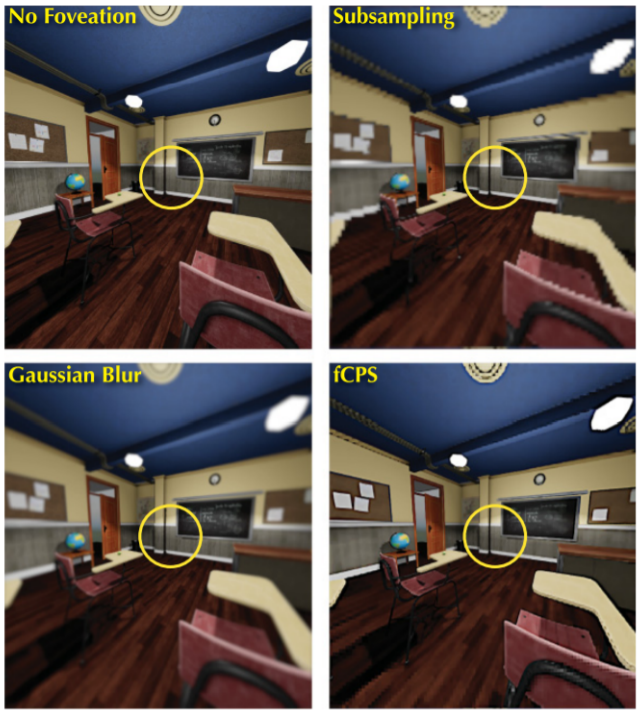


Figure 3.4: Reference uncompressed image and various types of foveation techniques with amplified immensity of foveation for better illustration [14].

The specifics of shown techniques according to [14] are

- Subsampling - reduces image resolution as the retinal eccentricity increases
- Gaussian Blur - utilizes Gaussian image-space blur where blur radius amplifies as retinal eccentricity increases
- fCPS - uses foveated coarse-pixel shading with prefiltered shading, foveated temporal anti-aliasing and post-processed contrast enhancement.

From this list of techniques, the one that would stand out is fCPS, as already for eccentricity of 10° it supported remarkably more foveation than subsampling or Gaussian blur [14] - this probably has connection to the way the footage is compressed here, as the periphery is responsive to sharp edges and contrast changes [15]. For eccentricity of 20° both fCPS and Gaussian blur allow more foveation than the subsampling method [14].

While this is very popular way of creation of foveated footage, other approaches have also been proposed. For example in [16] it was mentioned that we could implement the foveation by preprocessing before we encode the footage. Here we focus on areas that attract human gaze, called salient regions and we would reduce the amount of detail in non-salient areas before the encoding process. This is done by disposing of high frequency components from video frames in the non-salient regions. If using encoder with fixed bitrate, this means that the encoder will allocate more bits for the salient regions to the detriment of the non-salient parts of the footage. This method then consists of two steps - generating saliency maps from eye tracker coordinates and then blurring the non-salient areas.

3.3 Impacts of latency

Among one of the main problems with VR and omnidirectional footage is its high bit rate and limited bandwidth of the system. This means that our goal is to optimize the available system resources to function as efficiently as possible, while still delivering satisfactory user experience [2],[7]. While foveated compression could be the solution, it also comes with its own difficulties that shouldn't be overlooked.

One of them is latency - while some algorithms might provide us better compression, they also might add too much latency into the system, which would lead to worsened overall user experience. When creating compressing methods for omnidirectional footage, we have to try to find the best possible balance between the compression and latency. According to [14], if the eye to image latency would be around 20 to 40 ms, making the system's total latency 50 to 70 ms, the user experience will not be affected by it.

The phenomenon that we have to look out for is called saccadic omission, as it is an event where observer changes their gaze position rapidly between

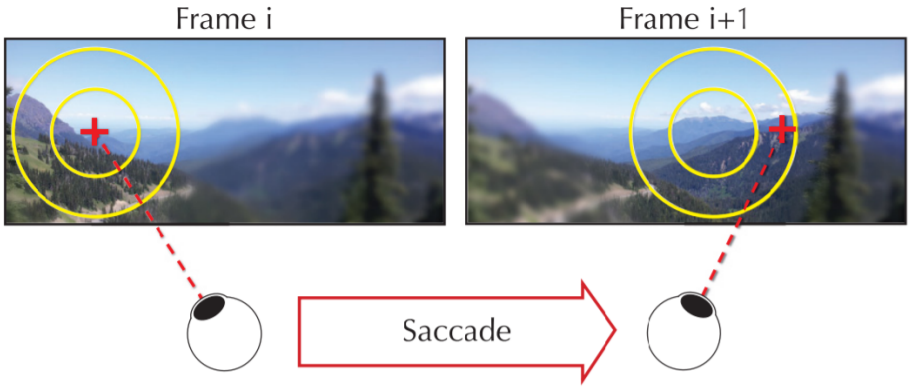


Figure 3.5: Unwanted impact of lag for large eye movements [14].

two locations. If the system's latency is too high, the computer wrongly estimates the gaze position, meaning the user is looking at the compressed part of the footage due to the offset of the viewport with high quality - this can be seen in Figure 3.5. Extreme occurrence of this phenomenon is when a person would turn and look behind them. This means that low-latency data from eye tracker are significant, as lag between large eye movements and display updates can negatively impact the extent of foveation [14].

However it was stated in [17], that most of the time people tend to move their heads to avoid large eye movements. It was also noted that according to their studies, for 95 % of time people keep their gaze inside 20°. This means that mostly saccades are not the main problem that would occur, but it is something that we have to lookout for, as it is situation that could negatively affect the user experience.

Chapter 4

Omnidirectional video player with simultaneous eye-tracking

For data analysis purposes or other eye-tracking related experiments, such as foveated compression, we need a suitable environment, where our experiments can be performed. This environment for me is a video player that is simultaneously able to track the eye movements. In this chapter I overview the process of creation of this video player, analysis of the provided data and also how various factors impact quality of experience.

4.1 Creation of the video player

The process displaying how a 360° video player works is shown in Figure 4.1. The data from the VR headset's eye tracker are sent to Unity, as well as the data about the omnidirectional footage. Unity works here as a 360° video player and simultaneously allows us to process the obtained data and merge them, so that we have precise information about where a person is looking in given time. We can then take advantage of this, and sent back to the VR only parts of the video, that the user is looking at, instead of sending back data about the whole omnidirectional footage. These merged data can also be used for analysis, that could be done in various programs for data analysis.

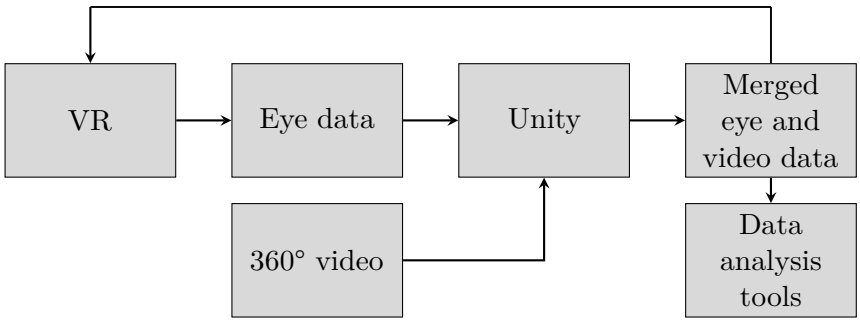


Figure 4.1: The functionality of 360° video player with eye-tracking.

To acquire the gaze position data of a person watching a 360° video, we can use various existing applications or write own code. Among the applications,

that have already been created, are for example iMotions[1], Cognitive3D[2] or Tobii Pro VR Analytics[3], which however is discontinued. While all of these programs are very advanced and are a great solution for analysis of the provided data, their disadvantage is usually price.

The other mentioned way to get eye-tracking data while watching omnidirectional footage is by using Unity, where I have to create or use existing code for a 360° video player and then merge it with a code for tracking of the eye movements. A guide for eye-tracking in Unity at maximal possible rate of 120 Hz is available at HTC's forum at [18]. Implementation of this code and its combination with a code for omnidirectional video player would be the next step. A problem we have to look out for here is correct connection between the data and the video, so that we can credibly interpret the data in further examinations. The biggest downside of creation of own code is absence of additional software for data analysis, that we would have to do by ourselves. Other disadvantage here is time consumption, however a big advantage on the other hand is that writing our own code is way less financially demanding, and also we can easily adjust the codes to our particular needs. As our specific needs could vary, depending on the goal of our task, the modifiable code is way better option for my research needs.

4.2 Analysis of the provided data

The retrieved data from the eye tracker are then ready for analysis, which can be done by various visualisations, where each of them have their differences and advantages. Among the most popular ones are gaze plots and heatmaps, that are both shown in Figure 4.2, or three dimensional scatter plots, that can be seen in Figure 4.3.

Probably the most popular type of visualisation of the eye-tracking data is heatmap - an attention map displaying how looking is distributed throughout the scene. As it can be seen in Figure 4.2a, the warmth of the color indicates which part of the stimulus is the most looked at. This means that the red areas show the most viewed part of the image, where areas that are not covered in any color are parts which was not viewed for a long time, if at all. This method is suitable for both mapping focus of visual attention of multiple people, as well as for visualisation of looking of a single person. However heatmaps are often misinterpreted, which results in misleading findings, so we should be careful when working with them [5].

Another way how to display the data is with gaze plots. These plots give us information about locations, where a person is looking, but also show us the order and time spent looking at various positions of the footage. In Figure 4.2b the size of the circle determines the time spent watching a given location and the numbers in them indicate the sequence in which the positions were seen [5].

[1] https://imotions.com/vr-simulations/
[2] https://cognitive3d.com/
[3] https://www.tobiipro.com/product-listing/vr-analytics/

(a) : A 2D heatmap [5]. **(b) :** A 2D gaze plot [5].

Figure 4.2: An illustrative visualisation of 2D heatmap and 2D gaze plot.

Three dimensional scatter plots are also used for visualisation of eye-tracking data. The plot, as can be seen in Figure 4.3, consist of multiple points with various colors, that indicate how long the viewer spent looking in given direction.

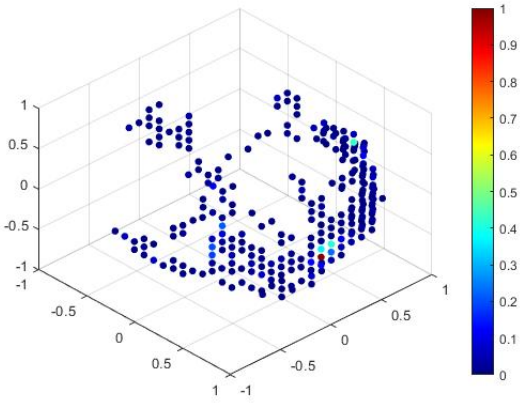


Figure 4.3: A 3D scatter plot, where visual attention is determined by color of plotted points.

During the many analysis of the eye-tracking data that have been done, various interesting discoveries were made. According to findings from [9] people in the VR environment tend not to look in vertical dimension and rather they look around in the horizontal one. Furthermore the degradation of the video quality does not affect VR user's selective attention, meaning that whether the footage is compressed or not, people tend to look at the same things.

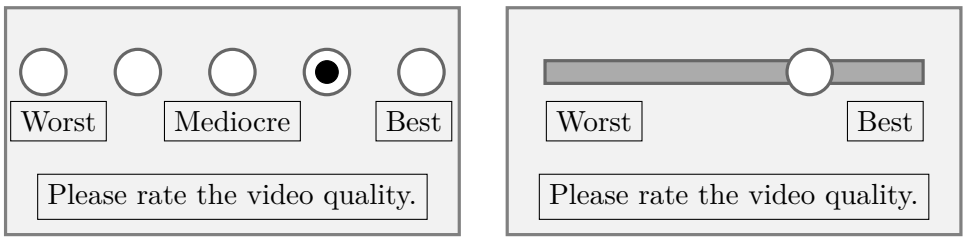
Another way we can analyze the data is through determination of the eye movements. Typically we look for saccadic eye movements - a motion

where the person's gaze changes rapidly between two places. This either is a response to some visual stimulus, but it can also occur without one [19].

4.3 Quality of Experience

Quality of experience (QoE) shows us how well a user perceives given application or service. For an omnidirectional footage the QoE is affected by various factors such as the video quality, its degradation or freezing events (stopping the video for a given time). These factors are different than those for 2D footage, due to the differences between these types of content. For omnidirectional VR footage we have more interactive freedom, as the user determines in which way they want to look, but also there are problematics such as cybersickness, that influence the user experience [9].

In the experiments, that would provide us the QoE for omnidirectional VR footage, we usually ask the participants few questions about the video quality or overall experience. Then they answer by typically rating the quality on a scale - the scale can either have fixed points that we can choose from (as shown in Figure 4.4a), or there can be a continuous rating bar (which can be seen in Figure 4.4b). In both cases there is one side representing the best experience and the other side indicating the worst one. The difference between them is the freedom of choice, as the fixed bar allows us to only choose from limited selection of choices, while on the continuous bar let's us to drag the point wherever we want it on the scale, which results into greater freedom of choice.



(a) : Fixed rating bar. **(b) :** Continuous rating bar.

Figure 4.4: Illustration of two typical rating bars used for QoE experiments.

The experiments, that took part in [9], used the integrated eye tracker in VR and asking the participants for feedback to get results of the experiment. The participants were watching multiple 4K videos, with various degree of compression and the questions they were asked related to the quality of the footage and user experience. According to their observations, the QoE of a 360° video would differ among the participants, as some had rated the quality on average high, while some on average low. This means that some people tend not to notice the differences made to the quality of the footage.

The negative effect on the QoE, that is caused by worsening the video quality by compression of the footage, is stronger than linear. Another unwanted impact on the QoE have the freezing events, as it takes just one

freezing event to ruin the user experience and drop the QoE to levels, that are unacceptable for the users, which means that we need adequate computing power in order not to spoil the user's experience. It was stated that for better QoE it would be better to increase the compression, if it result into less freezing events, than keeping the footage in higher quality with higher risk of those events. The videos, where the change in quality of the footage was most noticeable, were the ones with higher motion activity [9].

In [3], it was also noted that for the omnidirectional footage, subjective quality was influenced by the quality of the innermost part that is perceived by fovea, rather than the peripheral quality.

For experiments, where we have static camera position and the user doesn't move, we don't have to worry about cybersickness caused by compression or freezing events, as those factors don't affect it in these experiments [9].

The QoE is very important factor and many researches revolved around this problematics. One of them is [16], where they improved the QoE by using saliency-based foveated compression of the video, which was mentioned in previous chapter.

Among the cases, where the usage of the eye-tracking data is not advised and could result into worsened QoE are for example laser eyes, where the user would shoot rays from their eyes. If we would implement this, it might result into worsened ability of reading or unwanted casting of the beams from eyes, both being unpleasant for the user. Another bad case is using blinking as an input, as we could run into an event called accidental activation, where it is wrongly assumed, that the user blinked on purpose and it also leads to worsening of the QoE [20].

Chapter 5

Other fields of use of eye-tracking

In this chapter I overview other fields of use, where tracking of the eye movements in the VR environment have been implemented or used for research. Among those fields are mentioned as an example medical research and video gaming, where eye-tracking helped make progress in various ways.

5.1 Eye-tracking in medical research

One of the fields where eye-tracking in VR is being utilized is medicine. Here various experiments were performed ranging across many of its branches, from studying Parkinson's disease to ophthalmology.

In [19] the research revolved around neurodegenerative disorders, as due to the ageing of world's population this becomes an important recent problem. In this paper it was tested whether the HTC VIVE Pro Eye could be used in research analyzing the assessment of saccadic eye movement, and the results showed that this VR system is suitable for this type of research and could be used in upcoming experiments.

One of the experiments about neurodegenerative disease, where eye-tracking was used for research of Parkinson's disease was described in [21]. The researchers here collected performance metrics and gaze analytics when the participants were playing their game. The goal of their research was collecting the eye data and head movements of people with Parkinson's disease, and to assess their cognitive function using a game in VR system.

Other medical area that takes advantage of the embedded eye trackers in the VR headsets is ophthalmology. In [22] they reviewed the eye-tracking in VR for ophthalmological purposes and tested the capability of the HTC VIVE Pro Eye for potential online clinical applications. However it was found that this headset has limitations for online usage, but the future headsets might become a better fit for this application.

5.2 Video games and eye-tracking

Another field, where we can expect to see more eye-tracking applications are video games. From social avatars to gaze-aware interface, video gaming might greatly benefit from these eye-tracking implementations.

One of the examples where we can use the eye-tracking data are *social avatars* - those are avatars controlled by the VR user. There we can use the data to match the avatar's and the user's eye movements and blinking. This makes the avatar more human-like and responsive and this implementation will probably become popular in multiplayer video games, where there will be an interaction between multiple human-controlled avatars [20].

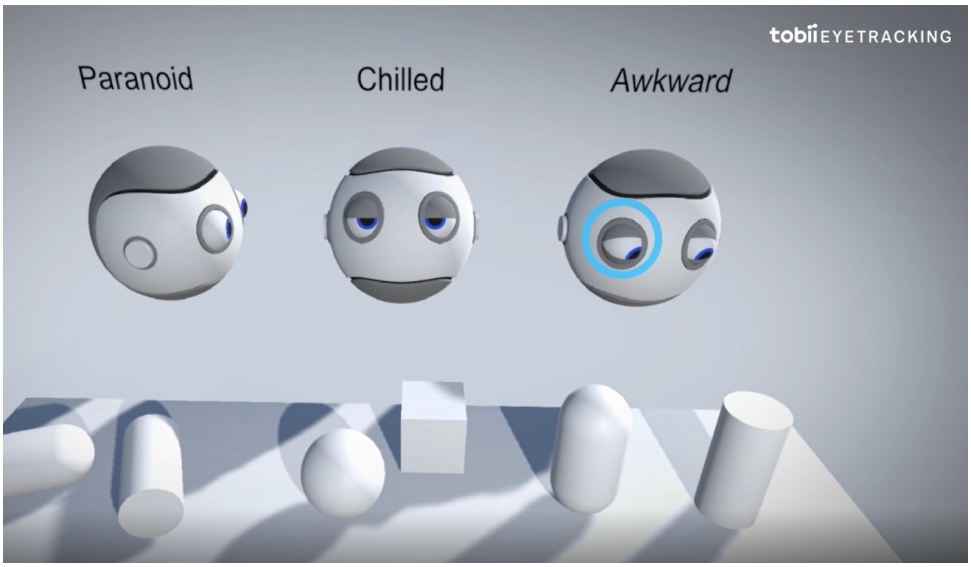


Figure 5.1: Gaze-aware non-playable characters [20].

Another example, where we can use the data from the eye tracker, is creation of gaze-aware non-player characters. This means that those characters will know, when the user is looking at them and they can become more responsive, just as shown in Figure 5.1, where the blue circle indicates the user's gaze and above each of the characters is written, how they will respond when a user looks at them. In case of this image, we can see that the character, that the user is looking at, is supposed to be awkward, so it tries to dodge eye contact with the user [20].

We can also use the eye-tracking data to hide the user interface like character data or other elements, that would distract the user. This means that when we are not looking at the position, where the user interface is located, it is hidden and only shows up when the user looks there [23].

Chapter 6

Eye-tracking experiment and setting up a demonstration task

This chapter focuses on the practical side of my thesis, which mainly consists of the experiments with the VR headset. At the beginning I specify the parameters of the workstation, where all the experiments will take place. In the next section I describe the process of getting the eye-tracking data and how I implemented it. I also focus on creation of demonstration task, that would utilize eye-tracking.

6.1 System parameters

In this section I overview the used devices, as well as their parameters, namely the specifications of the used computer and also parameters of the VR headset.

6.1.1 Computer system parameters

All experiments were performed on computer, with its most important parameters being mentioned in Table 6.1. Those parameters are the operating system (OS), central computing unit (CPU), graphics computing unit (GPU) and random access memory (RAM). The software that I use for the experiments is the 2020.3.27f1 version of Unity[1], with installed SteamVR[2] and SRanipal[3] packages.

OS	Windows 10 Pro
CPU	Intel Core i7-4790
GPU	NVIDIA GeForce RTX 2060
RAM	16 GB

Table 6.1: Selected system parameters.

[1] https://unity3d.com/unity/whats-new/2020.3.27
[2] https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647
[3] https://developer.vive.com/resources/vive-sense/eye-and-facial-tracking-sdk/

6.1.2 HTC VIVE Pro Eye

The VR headset used in my experiments is VIVE Pro Eye, developed by HTC (shown in Figure 6.1). The special thing about this headset is that it is equipped with multiple eye-tracking sensors, which bring new possibilities into the VR world. Those sensors were developed by Tobii[4] and work on infrared basis. In total, there are nine infrared light-emitting diodes and one infrared camera per eye, which then provide us data about eye movements and gaze direction [19].

Figure 6.1: The HTC VIVE Pro Eye headset [24].

The data that the eye tracker provides are timestamp, gaze origin and direction, pupil size and position and eye openness. The eye tracker's binocular gaze data output frequency is 120 Hz, the accuracy within FOV of 20° is between 0.5° and 1.1°, and the rendered FOV is 110° [25]. However the visible FOV is 98° - this means that not all of the rendered footage is visible and is covered by parts of the headset [26]. Another parameters of the headset are stated in Table 6.2, and the minimum required computer specifications are present in Table 6.3.

According to [27] the best case of capturing the gaze changes, for the HTC VIVE Pro Eye VR headset, is for the latency of 0 ms, where the transposition of the gaze direction is captured immediately, while the worst latency would be 8.3 ms, where the change of gaze direction occurs right after the last sampling of the eye tracker.

[4]https://vr.tobii.com/integrations/htc-vive-pro-eye/

Display	Dual OLED 3.5"
Resolution of the display	1440 by 1600 pixels per eye (2880 by 1600 pixels total)
Display refresh rate	90 Hz
Sensors	SteamVR Tracking G-sensor Gyroscope Proximity Eye Comfort Setting Eye tracker
Connections	USB-C 3.0, DP 1.2, Bluetooth
Supported room scale	5 times 5 meters

Table 6.2: Parameters of HTC VIVE Pro Eye, all taken from [25].

OS	Windows 7/8.1/10/11
CPU	Intel Core i5-4590 or AMD FX 8350 equivalent
GPU	NVIDIA GeForce GTX 970 or AMD Radeon R9 290 equivalent
RAM	4 GB
Video output	DisplayPort 1.2 or newer

Table 6.3: Minimum computer requirements for HTC VIVE Pro Eye, taken from [25].

6.2 Setting up an eye-tracking experiment

To access the data, I am using HTC's interface called SRanipal SDK, available from VIVE's developer website[5]. The SDK is compatible with Unity and Unreal - in my thesis I will only use the Unity program.

First step, when setting up the VR system for eye-tracking, is installing application called SR_Runtime, as it enables us to track the eye movements. After starting the application, we can check the computer's notification tray, where a status icon should appear. Depending on the color of shown in the

[5]Download available at https://developer.vive.com/resources/downloads/

icon, we can find out the status of the eye-tracking device - the meaning of the colors is shown in Table 6.4.

Color	Status
Black	The application is launched but the eye-tracking is unavailable
Orange	Eye-tracking is in idle mode
Green	Eye-tracking data are being retrieved

Table 6.4: Status of the eye-tracker dependent on the shown color.

In Unity we start with creation of 3D project, which provides us three-dimensional workspace area, where the experiments will take part. To build an eye-tracking plugin we have to import the SteamVR and SRanipal packages into the project. This is done by selecting the Asset folder, where than we choose to Import Package, and then select the Custom package option and choose the SteamVR and then we can follow the same steps to import the SRanipal plugin. The usage of VR in Unity is then possible thanks to SteamVR, while SRanipal enables us to work with eye-tracking data.

In order to enable functionality of eye-tracking in the Unity project, we have to add SRanipal Eye Framework to the Unity scene by dragging it into the scene hierarchy from the SRanipal prefab folder. We then have to configure the Framework Settings by checking "Enable Eye" and "Enable Eye Data Callback" [18]. After that our Unity project is ready for experiments with eye-tracking.

6.3 Omnidirectional video player in Unity

To create a video player that is able to replay omnidirectional footage in Unity, I followed steps from [11]. This allows me to use render textures as a Skybox material, meaning that the video will be projected as the sky.

The steps when creating the omnidirectional video player, if we have some 360° footage that can be then replayed (in my case it was video from [10]), are according to [11] following

- Creation of SkyBox material, which is done by creating a material in Unity, and then in the inspector setting the shader to Skybox panoramic
- Creating a render texture, as it is the texture that the Skybox will use to display the video.
- Changing the parameter of the texture is necessary, in order to correspond with the resolution of the used video

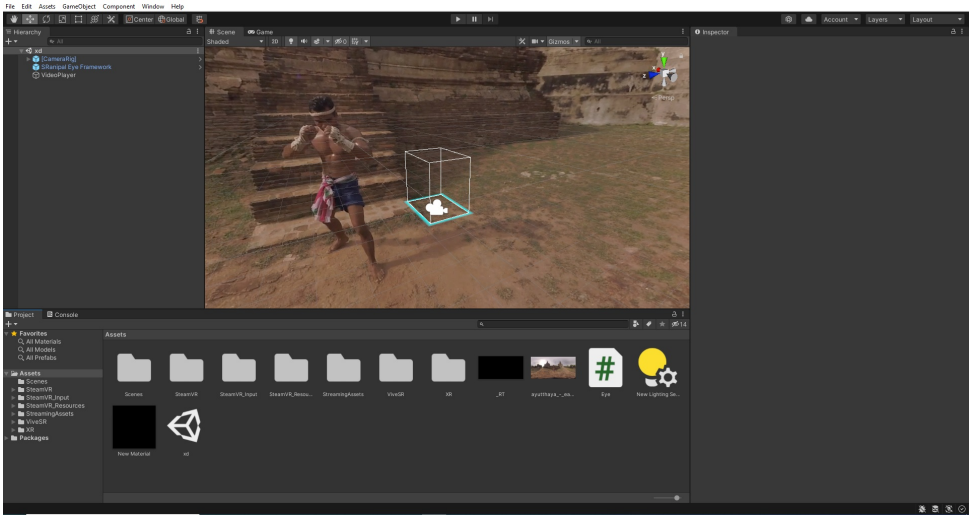


Figure 6.2: A screenshot of the Unity environment with working omnidirectional video player.

- Applying the texture to the Skybox material
- Creation of a GameObject in Unity, called VideoPlayer, where component called *Video Player* is added afterwards
- Inserting the video and the render texture into video player

After this, the video player is ready to display our video. If we want to change the video for another, we need to add it into the video player. However if the resolution of the video is different, than the resolution of the previously viewed one, changes in texture are needed, in order to display the video properly.

6.4 Accessing the eye-tracking data

To access the eye-tracking data, I wrote a `C#` script that I attached in Unity to the CameraRig prefab, which is available from the SteamVR package. The script was inspired by parts of code from [19] and also uses segments of code from [18] and [28]. The logic behind the process of obtaining the gaze data is shown in Figure 6.3.

Before the start of any of my experiments, I first need to call function `LaunchEyeCalibration()` that starts the process of calibration of the eye movements and gaze direction in separate program. This function is called in the `Start()` function, which means that it will only happen at the start of the experiment. There the user calibrates the eye-tracking sensors by looking at and following a blue dot,that moves in various directions.

As mentioned in documentation for the SRanipal SDK[6], before the cali-

[6] Available for download at https://developer-express.vive.com/resources/vive-sense/eye-and-facial-tracking-sdk/documentation/

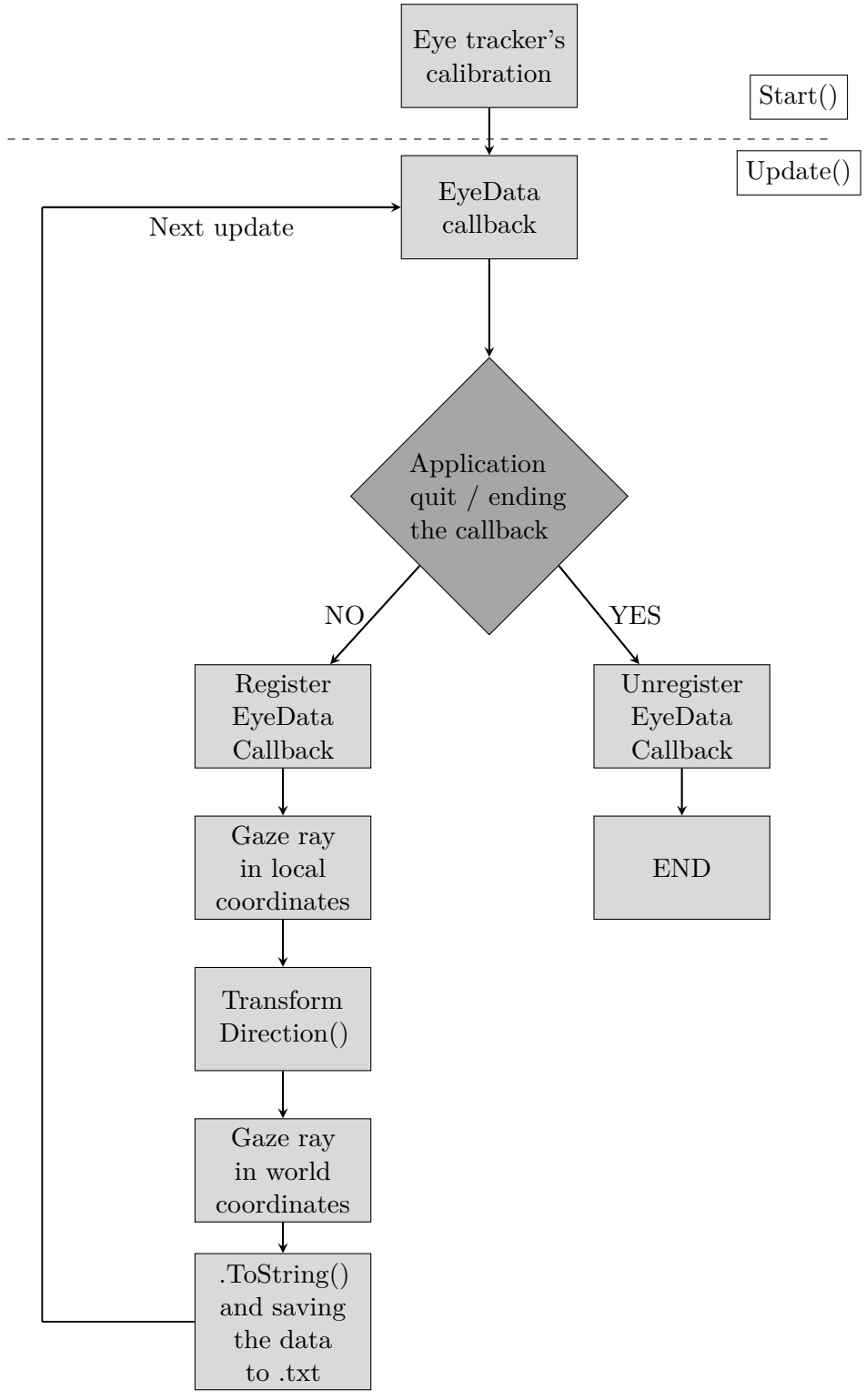


Figure 6.3: Process of obtaining the eye-tracking data in Unity.

bration begins, the program can ask the user to adjust the position of the headset on their head by moving the headset itself. The program can also tell the user to fix the interpupillary distance value (distance between the center of the pupils) by rotation of the knob, that is located on the side of the headset.

In the `Update()` function I check the functionality of the VR system's eye-tracking, and I also register the EyeData callback and unregister it, when the callback is completed or the user quits the application [18].

The EyeData callback however runs on separate thread, as Unity is not thread-safe, meaning that we cannot call any UnityEngine API (Application Programming Interface) from within the callback thread. In the script this means that we have to setup internal class `MonoPInvokeCallbackAttribute` that inherits from the `Attribute` class, which is part of the `System` namespace. This is necessary, as we need this class for IL2CPP scripting backend support [18].

Here I, however, run into a problem - it is not possible to call any functions from Untiy API in the `MonoPInvokeCallbackAttribute` [18]. This is rather a big complication for me, as this would be the only way to get the wanted data at maximal possible frequency, but the fact, that certain functions couldn't be called, makes obtaining vectors showing where user is looking a challenging task. The SRanipal SDK lacks any functions that would help me get the data in the Unity's world coordinate system, so I have to get more creative. I decided to ignore the fact, that I will acquire the data at lower frequency, as for my particular experiment it is not that important, because I will still obtain the gaze data more often, than the frame changes.

To obtain the wanted data, I was inspired by code shown in [29], so I used function `SRanipal_Eye.GetGazeRay()`, with `out GazeOriginCombinedLocal` and `out GazeDirectionCombinedLocal` being the most important parameters for me. This gives me information about gaze ray in local coordinate system, meaning it is linked to users current rotation, so it is related to current screen, seen by the user. I then need to transform it from the local coordinates into the world ones, so that it consists of vectors in the Unity world, that are suitable for data analysis purposes. The local coordinates are useful for non-analytical work, like foveated compression or for observation of saccades, but those data lack information about the rotation of the user and where they are located in the world coordinates, which are necessary for me. The transformation then happens thanks to a Unity function `Transform.TransformDirection()` that converts the vector from the local to the world coordinate system.

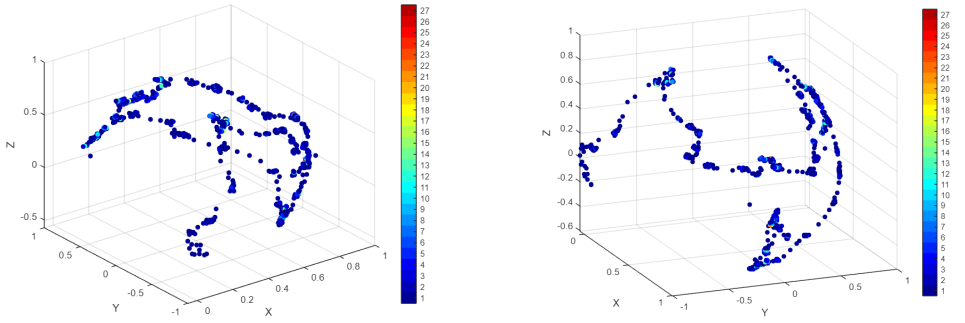
When converting the obtained data, I have to look out for a one small, but very important detail. The data are *Vector3* structure, which rounds numbers to one decimal, when function `ToString()` is applied. This however makes the eye-tracking data very imprecise, which means that I had to work my way around this problem. The solution is in overloading the `ToString()` function with parameter `"F2"`, that makes the function convert the data to String, while the data will be rounded to two decimal places.

6.5 Processing obtained data

The processing of the acquired data was made in the `MATLAB` environment. All the data were stored in text files, where I saved them using scripts in Unity. Those data had already been pre-processed, clearing all unwanted characters like parentheses, making the data ready to be inserted into `MATLAB`.

In `MATLAB`, the vectors, showing me where the user was looking at given time, were loaded into a matrix, where each row represented one of those vectors. I then separated the x, y and z coordinates into separate arrays, and I also loaded the video, using `VideoReader` function, so that I can plot the eye-tracking data on it.

Firstly, I decided to create a scatter plot, where the colors of the points would determine how long a user spent looking there. The code that I used was inspired by [30], and was adjusted for my particular needs. The number telling me how much time user's gaze was at given location, was obtained by `hist` function. Before using this function, I needed to know, how many unique rows there are in my data, which I got by using `unique()` function with parameters `MAT` representing the matrix and `'rows'`.



(a) : A view. **(b) :** A second view.

Figure 6.4: Scatter plot made from acquired data.

To showcase the omnidirectional footage as a 2D one, I use equirectangular projection. To visualize the obtained data this way, I first need to change the Cartesian coordinates into spherical ones. This is done by using following formulas

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \phi &= \arctan\left(\frac{\sqrt{x^2 + y^2}}{z}\right) \\ \Theta &= \arctan\left(\frac{\sqrt{y}}{x}\right). \end{aligned} \tag{6.1}$$

After that I used equirectangular projection to convert the spherical coordinates into two dimensional ones that are suitable for viewing a spherical footage on flat projecting screen. To do this, I used the equations (6.2).

$$\begin{aligned} x &= \frac{r \cdot \Theta}{\pi} \\ y &= \frac{r \cdot \phi}{\pi/2} \end{aligned} \tag{6.2}$$

However, in my case, all the vectors are normalised, so that the radius r is constantly equal to one, and so the equation can be simplified to

$$\begin{aligned} x &= \frac{\Theta}{\pi} \\ y &= \frac{\phi}{\pi/2}. \end{aligned} \tag{6.3}$$

A difficulty, that I faced, was plotting of the data, while using the mentioned equirectangular projection. In `MATLAB`, the coordinate system, that is present while plotting on an image, which is shown below of the plot, can be seen in Figure 6.5. In the same figure, I also show, where I put the starting point, that has fixed position at half of the maximal width and half of the maximal height. The values of height and width of the video are present in *video object*, that has been loaded to `MATLAB` using `VideoReader` function. The position of the actual gaze is then obtained by Equation (6.4), where max height and max width values are needed, as well as values x and y, that I calculated using previously mentioned Equation (6.3). After computing, we obtain this position at $(\xi_{\text{width}}, \xi_{\text{height}})$, where the point's position is then moved, as it can be seen in the mentioned figure.

$$\begin{aligned} \xi_{\text{width}} &= \frac{\text{max width}}{2} - x \cdot \frac{\text{max width}}{2} \\ \xi_{\text{height}} &= \frac{\text{max height}}{2} - y \cdot \frac{\text{max height}}{2} \end{aligned} \tag{6.4}$$

The reason, why I plotted the data this way, is because the x and y values were both in range between -1 and $+1$, and so starting in the middle of the image made sense, where both of the values -1 and 1 would show to one of the borders of the equirectangulary projected image. I also tested, whether this is the right method, on suited data that I created by looking mainly on the person that was right in front of me and then I followed edges of the building, present in the video. I then created a heatmap, by again customising the code from [30] and using scatter plot, for visualisation. The result can be seen in Figure 6.6, where it is clear, that the purposed method is convenient.

Another visualisation of the data was created by plotting them on each frame of the video, so that it showcases where the user is looking at given time. To do that, I use function `videofig` from [31], as this allows me to merge the frames into one figure, that also can be played as video. This figure consists of all the frames of the video, that I used for my experiments,

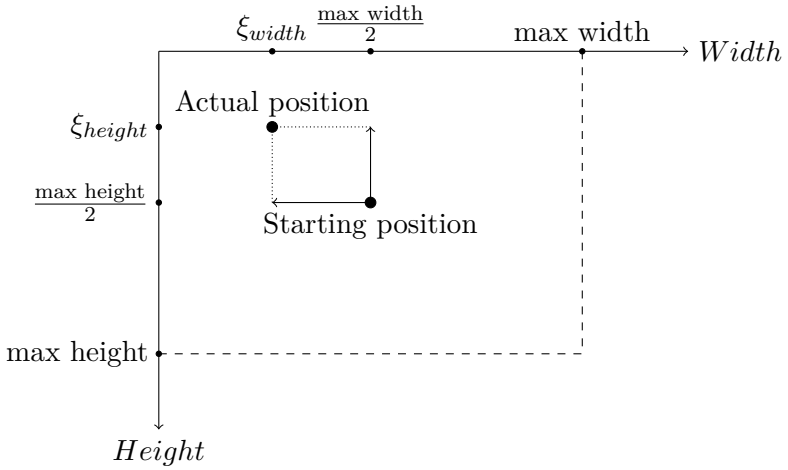


Figure 6.5: Logic behind plotting of the gaze data, with dashed line showing the image's borders.

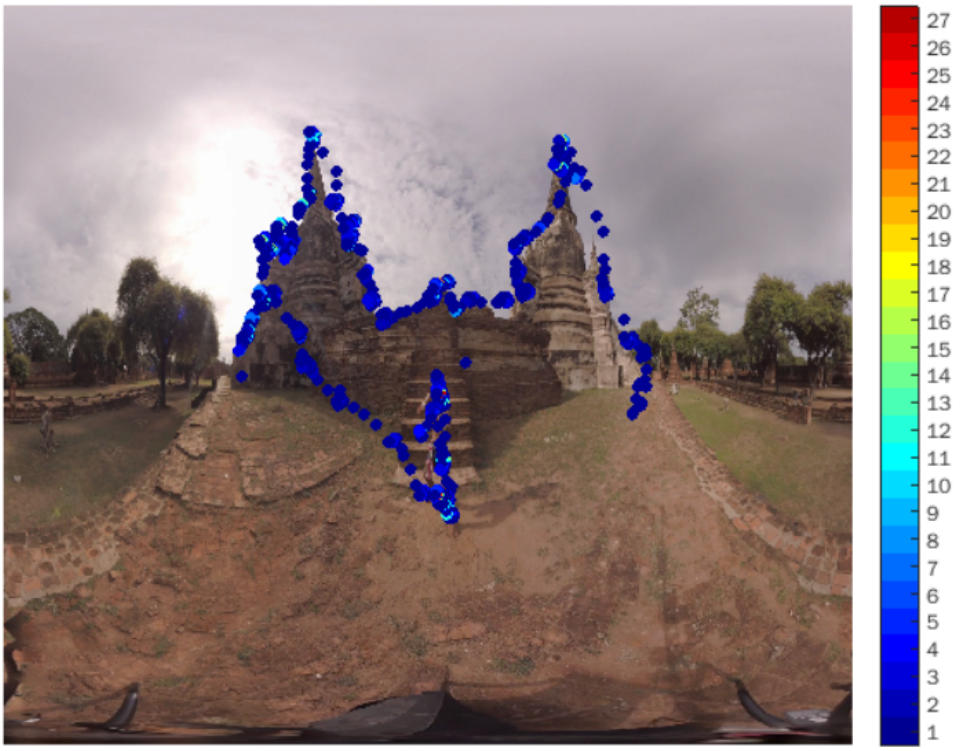
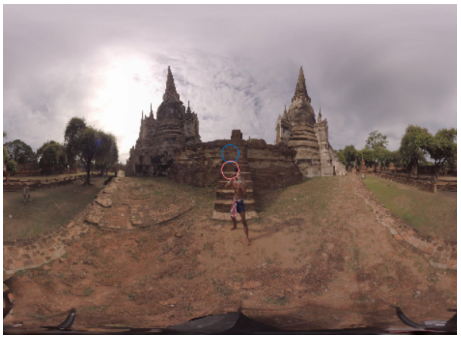
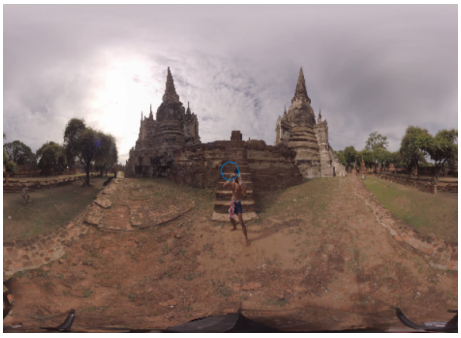


Figure 6.6: Heatmap created from test data, for checking the methodology.

as well as a circle, that showcases where the user is looking at given frame. Here I came across a challenge, whether I should plot multiple circles on one frame (shown in Figure 6.7a), as the sampling rate of the eye tracker is higher than how many frames per second there are, or if I should repeat this frame multiple times, while plotting always only one circle (as shown in Figure 6.7b).



(a) : Multiple circles per frame.



(b) : One circle per frame.

Figure 6.7: Difference between plotting multiple or one circle, showing current gaze, at given frame.

For Figure 6.7a it should be noted, that it actually contains three circles, blue, red and orange one, while the orange and red overlap, meaning that the user's gaze was at those two times at the same position. As this actually happens most of the time, I decided to use the 'one circle per frame' option, and for the video playback I configured it, so that it shows more frames per second to match the speed of the playback of the original video.

Repeating a given frame and plotting the user's gaze was done in separate function `redraw`, where the input parameters were

- *frame* - particular frame, for which I want to plot the multiple gaze locations
- *video object* - loaded with `MATLAB` function `VideoReader`, necessary as this allows me to display the video frame, and also provides various data about the video
- x and y coordinates changes - they carry the information about the user's gaze position, they are in range $[-1, +1]$ and determine the change on given axes
- *time* - indicates when the gaze coordinates were captured, used for synchronisation of the gaze and video.

The function is programmed, so that for every set of coordinates, I check to which frame they belong to. This is done by division of the time, when the data were captured, by inverse value of the video's frame rate. This calculated value was then rounded towards positive infinity, using `ceil` function and as a result I obtained the responding number of the frame.

When we have the correct video frame, then the gaze is displayed in form of a circle placed on top of it. This process repeats for every set of coordinates, until all the eye-tracking data aren't plotted, and as a result we get set of multiple figures with visible current gaze, which can also be played as a video.

6.6 Demonstration task

Merging all the information, that I mentioned in the previous sections of this chapter, lead up to a demonstration task, that can easily be reproduced. The task consists of eye-tracking experiment that focuses on gaze position of user who watches an omnidirectional video, that has been compressed. The whole process is shown in Figure 6.8.

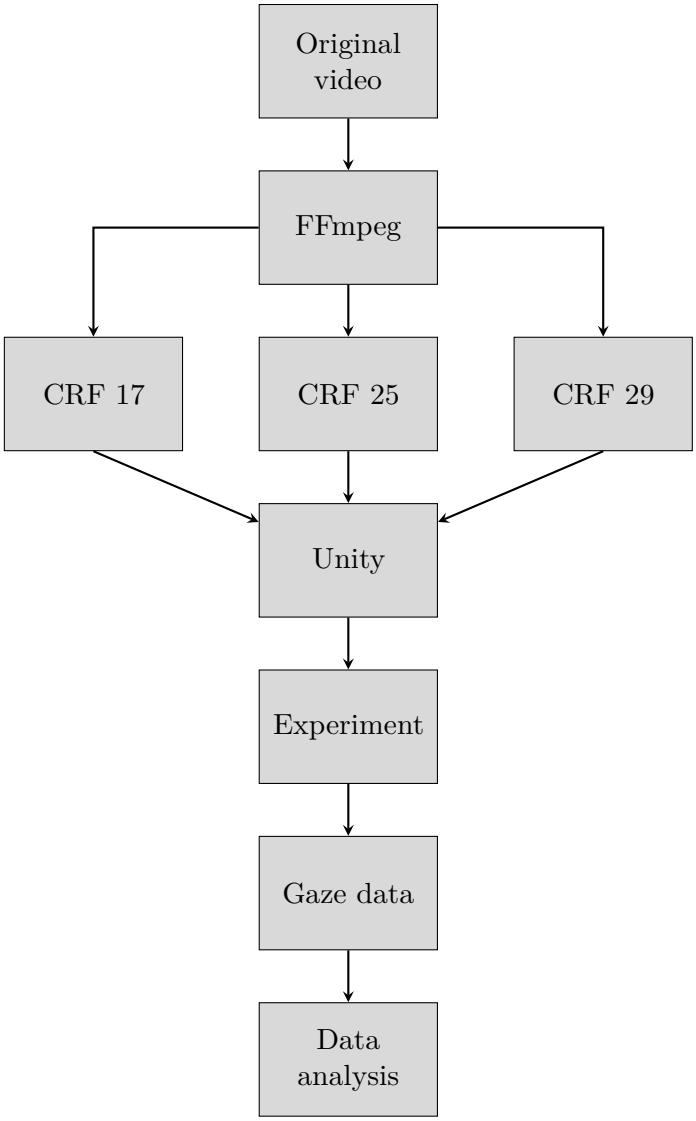


Figure 6.8: Process of demonstration task.

The video, which I used, was in .mp4 format, which is one of the most common containers both for normal and omnidirectional videos. For the demonstration task, I chose to work with this format only, because it is probably the one, that most people will come across. List of some of the most

popular video formats used for 360 footage are mentioned in Table 6.5.

Video format	Compression codecs
.mp4	H.265, H.264
.mkv	H.265, H.264, VP9
.webm	VP9, VP8

Table 6.5: Common formats used for omnidirectional footage according to [32], and their specifications.

If we don't have the video, which we would like to use, in this format, it can easily be converted using `FFmpeg` application, where in Listing 6.1 an example code for conversion between .mkv and .mp4 can be seen. Here the video doesn't have to be encoded and is just simply copied into different container.

```
1 ffmpeg -i video.mkv -codec copy video.mp4
```

Listing 6.1: Conversion of .mkv file into .mp4 one.

As we have our wanted video in MP4 container, we can start working with it. In my demonstration task I chose to encode the video, using *H.264* codec with CRFs of 17, 25 and 29. This provides me the same video in three different qualities, and the goal should then be to subjectively compare the quality by the VR user, while simultaneously their gaze would be tracked. In Figure 6.9 it can be seen the difference, between uncompressed video, and video, that has been encoded with CRF 29 at medium speed.

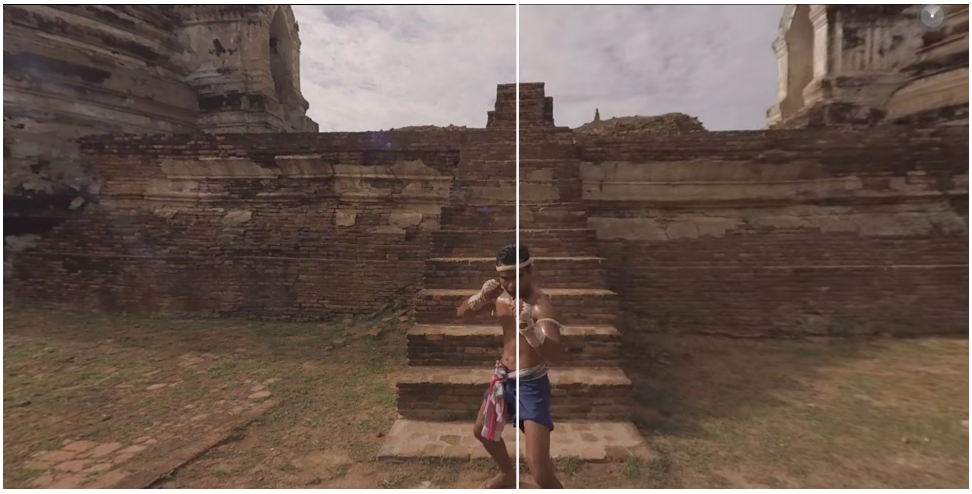


Figure 6.9: Original video (left) vs H.264 CRF 29 compressed video (right).

During the demonstration task, the eye-tracking are being collected, and their analysis is done, after all the experiments are over. Here, difference between user's behaviour for each compression can be seen, as they will watch the same video three times, and when evaluating the data, the researcher should keep this in their mind.

The analysis can be done by using the processes, that were mentioned in Section 6.5. This means that heatmaps can be created, so we can see, where the highest visual acuity could be spotted. Another way can be done by creating gaze plot, which we would produce by plotting the gaze location on the video.

After the video has ended, the participant is asked to rate their QoE. This is done by selecting one of the five cubes in Unity scene, where each one of them represent value form one to five, one being the best and five the worst. The cubes appear after the video is over, and their selection is done thanks to the knowledge of which object is currently in participants focus. There I used code inspired by [28], where function `SRanipal_Eye.Focus()` is utilized. Here as a `return` value, I receive information about gaze ray's collision with a `gameObject`.

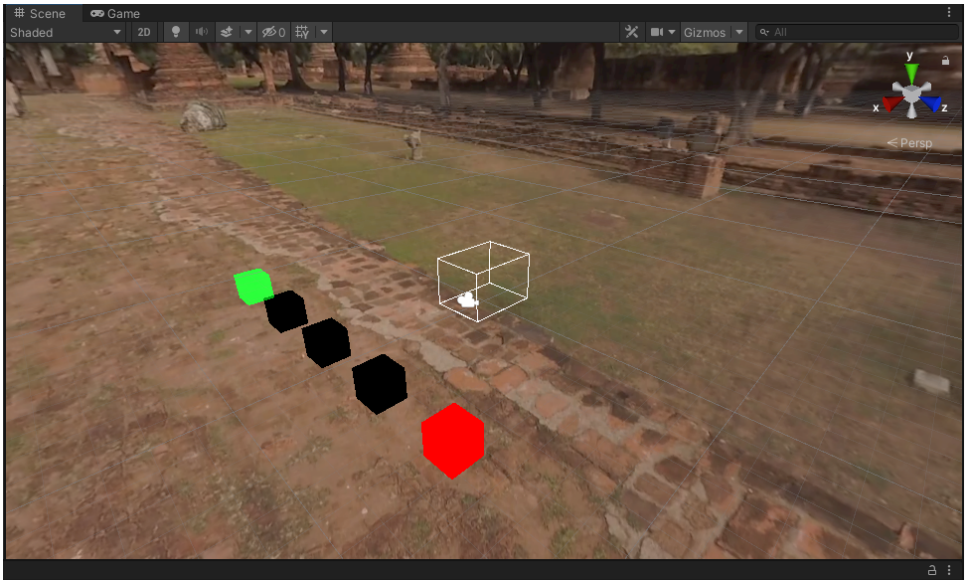


Figure 6.10: Cubes in Unity scene, working as a fixed rating bar.

The actual scene with the cubes displayed can be seen in Figure 6.10. The way as the cubes are shown in this figure however isn't optimal, and if wanted to be implemented in a real experiments, I suggest that the researchers should try to make this interface more user-friendly. The way as it is exhibited here is just showcasing possible implementation and doesn't follow any guidelines for interface design. Also the code is only showcasing how to obtain the given data, where the researcher can either just get strings containing the name of the current focused gameObject for every update, or further implementation can be done, by using `bool` obtained by checking if two strings are equal. The first mentioned way needs to be processed afterwards, while the second one enables us to process the data in the script.

6.7 Discussion

In this chapter the creation of suitable environment for eye-tracking experiments was showcased. I integrated eye-tracking into omnidirectional video player pipeline in Unity, where the process of obtaining the data is secured by a script attached to CameraRig. In the script I utilized function `SRanipal_Eye.GetGazeRay()`, which provided me vectors of the gaze ray in local coordinate system, that I then converted into world coordinates. Those data then were analysed and I created various plots, in order to showcase the differences between them. I then presented demonstration task, that can be reproduced and adjusted for the needs of future works. The further experiments can for example focus on compression approaches, that are mentioned in [13], where methods from various papers have been summarised.

Chapter 7
Conclusion

In this work I analyzed several eye-tracking-based techniques in VR systems and their state of the art. At the beginning I wrote about fundamentals, that revolve around this subject, where functionality of both VR and its embedded eye tracker were described. I also mentioned the way an omnidirectional footage can be viewed and how equirectangular projection affects an image.

Foveated compression was another covered field in this thesis. In the chapter, where I reviewed this issue, I described its connection to HVS, how the foveated footage is created, how foveation techniques affect the way the footage is compressed, and also how latency can negatively affect this whole process.

Then I focused on omnidirectional video player, as an suitable environment for eye-tracking experiments, where the relationship between the eye-tracking and the video player was described. I also mentioned various types of visualisations of the eye-tracking data, namely heatmaps, gaze and scatter plots. Another covered issue revolved around QoE, where various influences that affect the user experience were stated.

I also overviewed some other fields, where the eye-tracking in VR has been utilized. One of them was medical research, where the researchers used those techniques for experiments in ophthalmology or in studies about Parkinson's disease. I also focused on video gaming, as it is another sector, where eye-tracking is being used, whether it is by creation of gaze-aware non-player character, or gaze-responsive user interface.

In the practical part of the thesis, I focused on the experiments, that I performed. Among them was setting up the Unity environment, so that the eye-tracking tasks can be performed, as well as creation of a video player, that is capable of replaying omnidirectional footage. I described how I accessed the eye-tracking data and how I dealt with issues, that occurred. To acquire the data I used function called `SRanipal_Eye.GetGazeRay()`, that allowed me to access the ray of user's gaze in local coordinate system, which I then converted into world coordinates. Processing of the obtained data was then explained, and as an outcome I created a scatter plot, as well as a heatmap from the acquired data. In all two dimensional plots I used equirectangular projection, as it is probably the best one for visualisation of omnidirectional data. The issue revolving around plotting of those data on top of an image in

`MATLAB` environment was also described. I further created a series of figures, that can be replayed as a video, where I plotted the gaze data for each frame. Then I described a demonstration task, which consists of watching multiple videos, that have various degrees of compression. The goal of this task is to track the user's gaze, and also to obtain a review of the user's experience, which is acquired by selection of one of the cubes in Unity, using eye-tracking.

In future work further research into the VR and eye-tracking techniques would be necessary, as it is modern and rapidly developing field. Study on actual subjects can also be arranged, using the presented demonstration task, that would be adjusted for its specific needs. The field of the study can range from implementation of foveated compression to medical research.

Appendix A
Bibliography

[1] Y. Jin, T. Goodall, A. Patney, R. Webb, and A. C. Bovik, "A foveated video quality assessment model using space-variant natural scene statistics," in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021. doi: 10.1109/ICIP42928.2021.9506032 pp. 1419–1423.

[2] H. T. T. Tran, D. V. Nguyen, N. P. Ngoc, T. H. Hoang, T. T. Huong, and T. C. Thang, "Impacts of retina-related zones on quality perception of omnidirectional image," *IEEE Access*, vol. 7, pp. 166 997–167 009, 2019. doi: 10.1109/ACCESS.2019.2953983

[3] Y. Jin, M. Chen, T. Goodall, A. Patney, and A. C. Bovik, "Subjective and objective quality assessment of 2d and 3d foveated video compression in virtual reality," *IEEE Transactions on Image Processing*, vol. 30, pp. 5905–5919, 2021. doi: 10.1109/TIP.2021.3087322

[4] B. T. Carter and S. G. Luke, "Best practices in eye tracking research," *International Journal of Psychophysiology*, vol. 155, pp. 49–62, Sep. 2020. doi: 10.1016/j.ijpsycho.2020.05.010. [Online]. Available: https://doi.org/10.1016/j.ijpsycho.2020.05.010

[5] Tobii, "How to work with heat maps and gaze plots - tobii pro," Sep 2015, accessed on 19.1.2022. [Online]. Available: https://www.tobiipro.com/learn-and-support/learn/steps-in-an-eye-tracking-study/interpret/working-with-heat-maps-and-gaze-plots/

[6] I. Wohlgenannt, A. Simons, and S. Stieglitz, "Virtual reality," *Business & Information Systems Engineering*, vol. 62, no. 5, pp. 455–461, Jul. 2020. doi: 10.1007/s12599-020-00658-9. [Online]. Available: https://doi.org/10.1007/s12599-020-00658-9

[7] Z. Chen, Y. Li, and Y. Zhang, "Recent advances in omnidirectional video coding for virtual reality: Projection and evaluation," *Signal Processing*, vol. 146, pp. 66–78, May 2018. doi: 10.1016/j.sigpro.2018.01.004. [Online]. Available: https://doi.org/10.1016/j.sigpro.2018.01.004

[8] V. Clay, P. König, and S. U. König, "Eye tracking in virtual reality," *Journal of Eye Movement Research*, vol. 12, no. 1,

Apr. 2019. doi: 10.16910/jemr.12.1.3. [Online]. Available: https://doi.org/10.16910/jemr.12.1.3

[9] A. Kasteren, K. Brunnström, J. Hedlund, and C. Snijders, "Quality of experience of 360 video – subjective and eye-tracking assessment of encoding and freezing distortions," *Multimedia Tools and Applications*, pp. 1–32, 02 2022. doi: 10.1007/s11042-022-12065-1

[10] Mettle, "Ayutthaya - easy tripod paint: 360/vr master series: Free download," accessed on 20.3.2022. [Online]. Available: https://vimeo.com/214401712?embedded=true&source=video_title&owner=3032121

[11] Unity, "Play 360 video with a skybox in unity," accessed on 20.3.2022. [Online]. Available: https://learn.unity.com/tutorial/play-360-video-with-a-skybox-in-unity#

[12] G. He, J. Hu, H. Jiang, and Y. Li, "Scalable video coding based on user's view for real-time virtual reality applications," *IEEE Communications Letters*, vol. 22, no. 1, pp. 25–28, 2018. doi: 10.1109/LCOMM.2017.2764021

[13] M. Xu, C. Li, S. Zhang, and P. L. Callet, "State-of-the-art in 360° video/image processing: Perception, assessment and compression," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 1, pp. 5–26, 2020. doi: 10.1109/JSTSP.2020.2966864

[14] R. Albert, A. Patney, D. Luebke, and J. Kim, "Latency requirements for foveated rendering in virtual reality," *ACM Transactions on Applied Perception (TAP)*, vol. 14, no. 4, pp. 1–13, 2017.

[15] Tobii, "The eye," accessed on 19.1.2022. [Online]. Available: https://vr.tobii.com/sdk/learn/eye-behavior/the-eye/

[16] A. Polakovič, R. Vargic, G. Rozinaj, and G.-M. Muntean, "An approach to video compression using saliency based foveation," in *2018 International Symposium ELMAR*, 2018. doi: 10.23919/ELMAR.2018.8534631 pp. 169–172.

[17] Tobii, "Visual angles," accessed on 6.1.2022. [Online]. Available: https://vr.tobii.com/sdk/learn/eye-behavior/visual-angles/

[18] Corvus, "Vive eye tracking at 120hz," Feb 2021, accessed on 19.1.2022. [Online]. Available: https://forum.vive.com/topic/9341-vive-eye-tracking-at-120hz/

[19] Y. Imaoka, A. Flury, and E. D. de Bruin, "Assessing saccadic eye movements with head-mounted display virtual reality technology," *Frontiers in Psychiatry*, vol. 11, 2020. doi: 10.3389/fpsyt.2020.572938. [Online]. Available: https://www.frontiersin.org/article/10.3389/fpsyt.2020.572938

[20] Tobii, "Recommendations," accessed on 6.3.2022. [Online]. Available: https://vr.tobii.com/sdk/learn/interaction-design/use-cases/recommendations/

[21] G. Adlakha, S. Singh, A. Patil, K. Nuthalapati, P. Khandve, P. Bhattacharyya, S. Manoharan, S. M. Santhanam, I. J. Lachica, J. M. Finley, and V. Lympouridis, "Development of a virtual reality assessment of visuospatial function and oculomotor control," in *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, 2021. doi: 10.1109/VRW52623.2021.00259 pp. 753–754.

[22] A. Sipatchin, S. Wahl, and K. Rifai, "Eye-tracking for clinical ophthalmology with virtual reality (vr): A case study of the htc vive pro eye's usability," *Healthcare*, vol. 9, p. 180, 02 2021. doi: 10.3390/healthcare9020180

[23] Tobii, "How to use eye tracking in games," Aug 2019, accessed on 6.3.2022. [Online]. Available: https://gaming.tobii.com/onboarding/how-to-use-eyetracking-in-games/

[24] ——, "Htc vive pro eye," accessed on 6.1.2022. [Online]. Available: https://vr.tobii.com/sdk/products/htc-vive-pro-eye/

[25] HTC, "Vive pro eye specs: Vive united states," accessed on 1.3.2022. [Online]. Available: https://www.vive.com/us/product/vive-pro-eye/specs/

[26] VRcompare, "Htc vive pro eye: Full specification - vrcompare," accessed on 1.3.2022. [Online]. Available: https://vr-compare.com/headset/htcviveproeye

[27] Y. Jin, M. Chen, T. G. Bell, Z. Wan, and A. Bovik, "Study of 2d foveated video quality in virtual reality," in *Applications of Digital Image Processing XLIII*, vol. 11510. International Society for Optics and Photonics, 2020, p. 1151007.

[28] Corvus, "How to detect object that is being focused ?" Mar 2022, accessed on 9.5.2022. [Online]. Available: https://forum.vive.com/topic/9142-how-to-detect-object-that-is-being-focused/

[29] HTC, "Assets/vivesr/scripts/eye/sample/sranipal_avatareyesample.cs," accessed on 9.5.2022. [Online]. Available: https://gitup.uni-potsdam.de/mm_vr/vr-klassenzimmer/blob/a5132a7dd0a97b0ef43e2a82991210dae21ed7d1/Assets/ViveSR/Scripts/Eye/Sample/SRanipal_AvatarEyeSample.cs

[30] J. Doke, "Coloring scatterplots based on frequency of point occurrence in input," accessed on 4.5.2022. [Online]. Available: https://www.mathworks.com/matlabcentral/answers/28849-coloring-scatterplots-based-on-frequency-of-point-occurrence-in-input

[31] J. Henriques, "Figure to play and analyze videos with custom plots on top," accessed on 3.4.2022. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/29544-figure-to-play-and-analyze-videos-with-custom-plots-on-top

[32] B. Miller, "360 video format: Video audio formats for 360 vr video," Dec 2018, accessed on 13.5.2022. [Online]. Available: https://www.macxdvd.com/online-video/360-video-format-vr-audio.htm

Appendix B

List of electronic attachments

The electronic attachments are following

- Eye.cs - code written in `C#`, used for data acquisition and preprocessing, so that the data could be easily worked with in `MATLAB`. The creation and basic work with cubes, used in demonstration task, is also present in this code.
- scatter.m - creation of three and two dimensional scatter plots was done thanks to this `MATLAB` code, where for the 2D plot I used equirectangular projection
- video.m - `MATLAB` code, where usage of function `Videofig` is utilized, and I created series of figures with gaze position being present on top of the current video frame. Equirectangular projection was again used for 2D plotting.