Marc Aurel Kastner

Supervisor Michael Stengel

Referee Prof. Dr. Ing. Marcus Magnor

Co-Referee Prof. Dr. Ing. Ina Schaefer

Video Streaming for Foveated High-resolution Rendering

Master Thesis

February 24, 2016

Computer Graphics Lab, TU Braunschweig

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Braunschweig, 24. Februar 2016

Marc Aurel Kastner

Zusammenfassung

Applikationen für Virtual Reality nutzen häufig Displays, die nah am Auge des Benutzers abgespielt werden. Um eine hohe Immersion zu erschaffen müssen Videodaten in einem höchstmöglichen Detailgrad übertragen werden. Hierbei kommt es jedoch häufig zu Problemen, da hochaufgelöste Videodaten meist einen Engpass in der Bandbreite zwischen Grafikkarte und Display darstellen. Um einen solchen Engpass zu vermeiden, müssen die Videodaten optimiert und im Umfang reduziert übertragen werden.

Diese Abschlussarbeit entwickelt ein Verfahren zum Darstellen von hoch aufgelösten Videodaten unter Verwendung eines fovealen Ansatzes. Um die Sehschärfe eines menschlichen Auges zu simulieren, sollen sich Videodaten dynamisch in der Auflösung anpassen können. Hierfür ist es notwendig, ein Datenformat zu entwickeln, das dynamisch Regionen in unterschiedlichen Qualitätsstufen auslesen kann. Danach wählt ein Auswahlverfahren auf Basis von fundierten Kenntnissen über Sehschärfe eine optimale Verteilung der Auflösungen für unterschiedliche Bildbereiche aus. Hiermit kann ein Großteil der Rohdaten eingespart werden, da beispielsweise für Objekte im Augenwinkel nur grobe Details übertragen werden müssen. Dies verringert die Bandbreite und verhindert Engpässe in der Übertragung. Um dies umzusetzen wurde ein geeigneter Algorithmus entworfen und implementiert. Die Implementationen wurden ausführlich getestet im Hinblick auf Geschwindigkeit, Robustheit und die Wahrnehmung der Benutzer. In der abschließenden Diskussion wird auf offene Herausforderungen und Potential für zukünftige Arbeiten eingegangen.

Abstract

Virtual reality applications often display video data close to the eyes. A high resolution increases immersion and realism. This comes with drawbacks in terms of bandwidth requirements not sustainable by current hardware, limiting the choice of video resolutions. In order to decrease bandwidth and computation requirements, the transferred data of high resolution video streams needs to be restricted.

This thesis develops an approach for high resolution video streaming utilizing foveated imaging data. To be able to load raw video data in dynamic resolution levels simulating visual acuity, a sufficient data format capable of loading video regions in different quality settings is developed. Furthermore, an algorithm is designed to efficiently incorporate visual acuity to select a set of video data ideally suited for the users' field of view. This way, the amount of raw data streamed to the graphics unit is reduced, as only a small region is displayed in the maximum resolution. This approach is implemented and tested for wide-screen and panoramic video material. The implementations are evaluated in terms of performance, robustness and the users' perception. A discussion rounds up the results and limitations with an analysis of open problems for future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem

Virtual reality applications often display videos or other interactive data close to the eyes. A high resolution can significantly increase immersion and realism. This often comes with drawbacks in terms of bandwidth requirements not sustainable by current hardware, limiting the choice of video resolutions. In order to decrease bandwidth as well as computation requirements, the transferred data of high resolution video streams needs to be reduced.

In this thesis a method for high resolution video streaming is developed. One approach to this is utilizing foveated imaging data. Unfortunately, standard video codecs are not capable of region-based access to video data, which makes performing foveated imaging techniques difficult. Thus, a data format capable of loading video regions in different quality settings needs to be developed. Using this, video data should be loaded in dynamic resolution levels to adapt to the users' vision. The goal is a simulation of visual acuity to minimize the transferred data based on the characteristics of human vision. An algorithm to efficiently incorporate visual acuity needs to select a set of video data ideally suited for the users' field of view.

The implementation needs to be able to visualize very high resolution data in the focal center of the user and comparatively low resolution data in the corner of their eyes. To apply this approach without harming the users' perception, a low-latency rendering and a restricted amount of newly introduced rendering artifacts is preferable.

Furthermore, as eye movements are neither known nor estimated during video pre-processing, the algorithm needs be capable of dynamically adapting to eye movements. This is a major key point discerning this thesis from imaging approaches which mold a preset focal point into existing video data.

The approach must be tested with different real-life wide-screen video data. To increase the applications for this approach, it can be extended to 360-degree panoramic video data.

1.2 Motivation

The resolution and quality of video data gets more and more wide-ranging, with the current generation of even consumer devices being capable of capturing and displaying 4096×2160 (so-called 4K) video data. In the future, formats like 8K, 16K and even higher are thinkable. However, data bandwidth, video processing and consumer displays are amongst the main bottlenecks in technological advancement. As the amount of data increases quadratically with resolution size, the role of data bandwidth as a crucial bottleneck makes very high resolution sizes unfavorable. Furthermore, the standard frame rate of videos tends to increase as well, which makes a low-latency streaming per frame even more desirable.

In virtual reality (VR), most commonly in modern applications involving head-mounted displays (HMD), displays are very close to the users' eyes. A high resolution in the focal point is crucial to increase immersion and decrease simulation sickness. However, in these applications, the users' field of view (FOV) is comparatively huge. Even more so for 360-degree videos, where a large part of the video is always out of the users' viewpoint. As a result, very large parts of the video area are not perceived in the amount of detail they are processed in. In common video rendering techniques, every frame is processed in its entirety. This creates a bottleneck both in terms of bandwidth and computation time.

However, modern HMD devices are capable of pupil-tracking techniques to determine the users' exact point of view. By analyzing the visual acuity of the user a new determinant for choosing video quality can be introduced.

The idea of foveated imaging exploits properties of the Human Visual System[67] to reduce video resolution in the peripheral viewing area. Using this, only the focal center would need to render in the maximum resolution available. Otherwise, a radial decrease of resolution is possible. This gives the rendering algorithm an opportunity to vastly decrease the amount of video data to process and stream.

When being able to decrease the amount of bandwidth transferred, this opens the door to various other applications. For example, a server/client approach with separate devices for decoding and displaying content would allow mobile devices to use high resolution VR videos. Additionally, a vastly reduced bandwidth could allow streaming over the internet or creating cloud-based services for these applications.

1.3 About this thesis

The thesis implements and evaluates different ways to solve real-time video rendering for high resolution data utilizing knowledge of foveated imaging and the human vision system.

Chapter 2 introduces background knowledge and state-of-the-art techniques relevant to this foveated imaging techniques. It starts with a short introduction of virtual reality and head-mounted displays which motivate this thesis. As a direct consequence of modern head-mounted displays, pupil detection techniques and the human vision system are discussed subsequently. There is a focus on visual acuity which will be a main pillar in approaching this problem. Lastly, implementation techniques for high resolution videos and image data, similar to those used to process vast amounts of data in video games and ray tracing, are discussed.

Next, Chapter 3 discusses design decisions and options available for solving the foveated video rendering. It builds a bridge between discussed theory from Chapter 2 and practical implementations in Chapter 4. In the focus of discussion are different approaches for factoring in acuity in video streaming algorithms, so-called foveated imaging. Furthermore, it presents an efficient way to pre-process data based on uniform grids and mip-mapping. Lastly, options for loading and streaming raw video data are discussed, incorporating both wide-screen videos and 360-degree panoramic videos.

In Chapter 4, implementation details based on previous design decisions are explained exhaustively. During the project phases, two prototypes in form of sophisticated frameworks for wide-screen data and panoramic videos respectively are implemented. Each framework tests different design decisions for a comprehensive analysis of each option. In the end, various optimization steps to improve rendering speed and the perceived quality are presented.

The previous implementations are then evaluated in Chapter 5. Technical features of both frameworks are analyzed regarding their performance, memory usage and storage usage. Furthermore, a broad comparison of different design decisions to explore the limits and restrictions of each setting rounds up the results. Lastly, the visual impact, or more specifically the impact to human perception, is evaluated. Artifacts and issues in terms of rendering quality and resolution levels are discussed in relation to the theory of human vision system and its visual acuity.

Afterwards, in Chapter 6 the performance impact on different design decisions is discussed. This includes a comprehensive analysis of each approaches' limitations. Furthermore, an outlook on open questions regarding this problem is presented.

Finally, Chapter 7 briefly summarizes the design decisions, implementations and results. It presents notable insights gained through this approach.

Chapter 2

Related work

This chapter discusses relevant literature and background knowledge, on which the thesis design and implementation are based on. In Section 2.1 an overview of virtual reality (VR) systems, its history and current limitations are presented. It will further emphasize on head-mounted displays which are one of the most ubiquitous approach to virtual reality in recent times. In advanced applications of HMD, pupil detection is used to further improve the users' perception, as discussed in Section 2.2. To design a VR system closely related to the perception of the user, a certain knowledge of the human vision system and its features is necessary, which is reviewed in Section 2.3. Lastly, on a technical side, the implementations are based on various texture techniques discussed in Section 2.4.

2.1 Virtual reality

In layman's terms, virtual reality (VR) describes the simulation of a virtual world. Usually, it describes the process of creating a special environment to be immersed in. In contrast to normal video games, it aims to fully immerse the user in terms of vision, but commonly also include other senses. Therefore, VR feature special input devices tracking the users' movement, head and arm positions and so on. Furthermore, helmets, booths or other techniques are used to fully replace regular eye vision with a virtual world and inhibit the users' perception of the real world while using the device.

There are various applications for VR, incorporating all kinds of hardware. In its first relevant publication, the concept called *The Ultimate Display* was described as "make that (virtual) world in the window look real, sound real, feel real, and respond realistically to the viewers' actions" [73][54]. During the 1960 to 1990, various kinds of machines, simulators and other approaches using multi-sensor input and output were developed and tested.

For input, there are a variety of devices used during history. Using sensors, the users' movements are captured in real-time. To successfully capture three

dimensional motion data, so-called *6DOF* (six degrees of freedom) trackers are used to capture position (3D) as well as orientation (*yaw*, *pitch* and *roll* angles). Motion tracking is usually done using magnetic, mechanical, optical or acoustic sensors. The usage of eye tracking also involves cameras[54]. Furthermore, properties like update rate, latency, accuracy are important as they lead to loss diminished enjoyment perceived by the user which is later discussed in terms of simulation sickness.

As a feedback to users, various output devices are used. Of course, classic displays are used for visual perception. The immersion is enhanced by using head mounted displays (HMD), but there is no conceptual difference. To further increase immersion by using additional senses of the user, elements like force or tactile feedback are popular[69]. The last major sense used is usually audio by incorporating speakers. Senses commonly excluded are smell, taste and non-traditional senses like temperature, balance and pain. However, the classical definition coined by Sutherland[73] generally includes all kinds of perception.

With decreasing size of hardware setups, HMD gained popularity, as their setup is less complex[3]. The first HMD is considered to be *the Sword of Damocles* [72]. These devices slowly gain access into consumer-grade hardware with the release of products like the Oculus Rift[19].

Closely related to VR is augmented reality (AR). While VR simulates a world where the user is fully immersed in, AR adds newly computer-generated elements to the users' perception in real world. It was first described that it "presents a virtual world that enriches, rather than replaces the real world" [20]. This approach, while it seems like a Hollywood movie, is the introduction of computer vision and technology to the users' vision. The discontinued product Google Glass[83] was an augmented reality device in a form of a HMD. It includes a camera and a display, which serves as an overlay on the users' normal eye vision.

The more thorough view on the history of virtual reality is well presented in a thesis[54] fully based on the history and applications of VR.

2.1.1 Simulation sickness

Virtual reality is prone to effects similar to nausea and seasickness[60]. Additionally, various parameters can influence the users' perception in both positive and negative ways[49][42]. Resolution and performance of the system have a great impact on the viewers' pleasure, but can also result in simulation sickness.

For classic VR approaches, resolution, field of view and flickering has been greatly analyzed. When using an HMD system, field of view (FOV) plays a major role as the display is very close to the eye. Thus, the FOV is very high. Note however, that this does not correspond to the scene itself, as the visual distance is still usually around two meters. When using immersive video

material, latency and refresh rate of the rendering are also an important factor.

The refresh rate directly translate to update rate for input devices. If the input is prone to error, the joy of a user is harmed, as input and output are directly connected through user interaction.

A promising parameter is accuracy. For input, it is obvious that some sensors might have inaccurate measurements. In visual output however, this can have several meanings. While resolution is important and a main motivation for this thesis, accuracy also translates to robustness. Thus, newly introduced visual artifacts or rendering issues have a negative effect on this.

Subsequent studies have shown, that these results remain true for modern approaches [45][56] and are valid for various uses of HMDs[50].

In foveated approaches, eye movements are used to directly influence the users' focal center within the scene. This is related to the previous paragraph about input devices. Eye movements are a very delicate measurement, as wrong measurements have a very high impact on the users' viewpoint. Thus, it needs to be reasonably accurate. Pupil detection algorithms are discussed in detail in Section 2.2. When incorporating eye position as a determinant, latency is another critical factor. His latency already depends on the pupil detection algorithm, but can be increased by further processing.

A high delay or stuttering when processing eye movements can have an impact on the video latency and overall perception of the user. This can presumably result in a strong negative impact such as simulation sickness, as the perceived vision does not correctly correspond to the eye movement. Furthermore, rendering issues like artifacts, jitter and rough jumping in between frames will presumably also increase negative effects.

2.2 Pupil detection

Head-mounted displays (HMD) are already an approach to achieve an increased immersion. A single step to further increase the immersion of such systems is the introduction of eye-tracking, or more specifically pupil-tracking. This allows various new features like foveated imaging, which is also called *gaze-contingent rendering* in this interactive context. Applications for this are fully discussed in the later Section 2.3.

In this thesis, the data received from pupil-tracking is used to increase performance by reducing data bandwidth. Likewise, knowledge of pupil position can be used to enhance visual appearance in video games, but can also be interesting for interactive movies or other applications.

There are multiple ways to approach computer vision problems surrounding pupil and eye detection. The input data is usually a captured image of the users' face or eyes. An algorithm assumes the pupil position and returns a 2D value of the users' view point on a panoramic plane. A common pupil

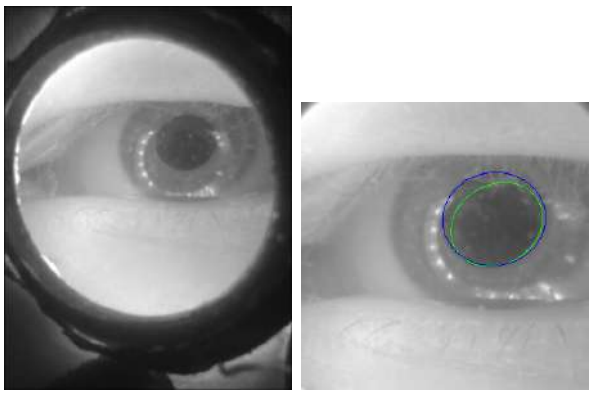

Figure 2.1: Pupil detection algorithm: The input is an image of a human eye (left). Using computer vision algorithms, the position of the pupil is determined and an estimated ellipse is returned (right). The colored ellipses indicate different steps of detection within the algorithm.

detection algorithm is displayed in figure 2.1.

There are various implementations of pupil detection techniques incorporating different computer vision approaches and fields like machine learning and others. It is possible to detect pupil position robustly in real time[27][85] even on embedded devices[38]. Thus, it is possible to create affordable solutions for binocular eye tracking in HMDs[68].

In more advanced cases, an approach using topography-based detection with multiple different resolution pictures was developed[59]. Another interesting work[53] proposes an approach using randomized regression trees to determine the position of a pupil. It is reported to be efficient for real-time usage even on mobile devices.

When designing algorithms for the detection of pupil positions, the nature of voluntary and involuntary eye movements have to be taken into account. So-called saccades[63] are very quick movements of eyes between two phases of fixation. Saccades are not directly controlled by the user and occur in subconsciousness. Thus, quick jumping between different fixations are commonly measurable, even if the user fixes a single point. This needs to be smoothed out by either the pupil detection algorithm or the application using the output.

2.3 Human visual system

The human visual system is a big complex including topics from interdisciplinary research like ophthalmology, physiology and psychology. Thus, there is a history of literature relevant to this topic.

In this thesis, the main interest is publications based on the behavior and

structure of the eye and its visual acuity. This includes the color spaces, edge detection and other factors, which are relevant to the perception of the eye. The developed algorithm needs to select *relevant* frame data based on an educated guess on which regions are not perceived as strongly by the viewer.

Acuity The visual acuity describes the ability to observe detail. A well established psychophysical model is from 1857 by Aubert and Foerster[70].

There is a fall-off in radial distance to the focal center of the eye. This corresponds to the number of neurons within the eye[67], and has a linear decrease within the central vision region[47]. The minimum discernible angular size increases with eccentricity for the first 20–30 degrees and a more rapid fall-off afterwards[82]. Acuity can also have a strong influence on pattern recognition[70].

Unfortunately, visual acuity is strongly user-dependent and can be trained and improved up to sixfold[81]. Medical history like shortsightedness can also have a strong impact on the eccentricity and thus angular size[17].

Furthermore, the actually perceived visual acuity is different in vertical and horizontal direction and can also have different resolutions in the upper and lower part of the eye[17].

However, this model is commonly used due to its simplicity. While it does not perfectly simulate the visual acuity of a human eye, its conservative approximation is close enough to be suited for gaze-contingent rendering.

A simplified model based on this acuity fall-off has been developed[80] and extended in terms of visual resolution[40][78]. It can be roughly categorized in two main parts of vision, called the central vision and the peripheral vision. There is a noticeable fall-off within each field, but a rather strong difference in between. A geometrical visualization of this model is shown in Figure 2.2.

The *retinal integration time* of the eye is approximately 40 ms[14]. This means, there is a time of approximately 3–4 frames, where resolution differences can be delayed and artifacts be smoothed out until the retina is able to fully perceive the highest detail.

Saliency Various approaches regarding the HVS are based on the saliency. It describes the feature of something visually standing out. This can be some kind of large color deviation, a moving object, hard edges or similar effects which have an extra impact on the perception.

Saliency-based approaches are another way to process scenes with accounting for characteristics of the HVS, as used by [51][13]. Usually, saliency-based algorithm do not factor in gaze and thus only rely on information available within the image data.

Frequency of information has a close relation to perception[36] and refers to the amount of edges and sharpness within an image. It is usually described by a Fourier transform and is used in a variety of filtering approaches.

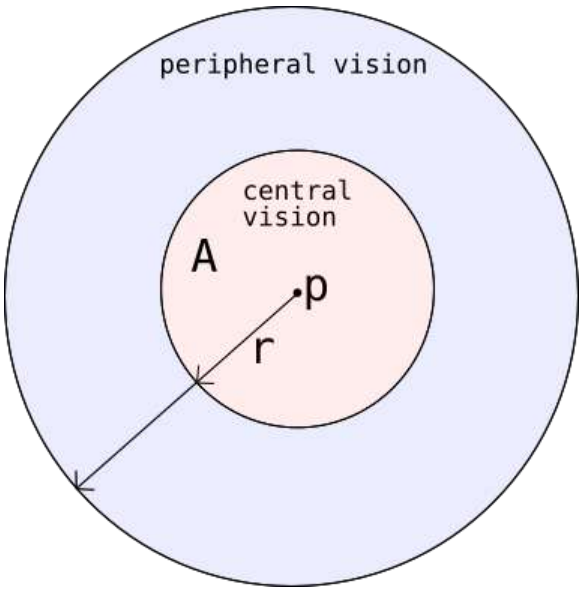

Figure 2.2: Structure of visual acuity: The visual field can be categorized into a central region and a peripheral region. The point p is the fixation point of the eyes. The area A is the foveally focused region. There is a linear fall-off[47] in radius r to p. Additionally, there is a peculiar drop in between these two regions[82].

The influence of temporal variance and other image characteristics like the amount of edges have a strong influence to perception[76][6].

In ray tracing, there are measures to increase the amount of samples based on saliency-based methods to find visually relevant areas[10][71]. In the context of ray tracing, this is also called *selective rendering*.

Color perception When incorporating color, RGB color space turns out to be improper for comparisons based on human perception[74]. HSV, LAB or other[11] especially for human perception developed color spaces can give assumptions suitable for this algorithm. Based on this, in order to optimize the algorithm, color variance is a promising point to analyze.

Active versus passive gaze estimation The idea of a gaze-contingent rendering approach is to factor in eye position for an acuity-optimized rendering. This was first done in the 90s[48][4] and is frequently used since then in various applications.

There are so-called gaze-contingent multi-resolution displays which try to use gaze-contingent approaches to display the highest resolution within the foval point of a user[61][16]. An example result of this approach is visualized in Figure 2.3.

Figure 2.3: This figure[37] visualizes the results of a fovated imaging approach. The fixation point of the users' eye is in the middle left. In this region, the resolution is the highest. There is a fall-off in terms of resolution in distance to the focal point, which results in a blurry peripheral vision.

However, there is an important distinction between active and passive gaze estimations used for these approaches.

In passive approaches, a presumed fixation point is guessed by the algorithm or set by the user in advance of the process. This can be based on saliency, the scene contents or other factors. In a project using passive gaze estimation, single images are pre-processed to mold in an approximation of visual acuity. Note, that this is a static approach as it permanently changes the underlying media data. As a result, it does not allow to *move* the gaze position after the algorithm is done processing video data.

This is typically used in order to improve video data in terms of storage or bandwidth for scenes. Use cases might include news or sport scenes, where the user is expected to watch a certain point of the video. Thus, the rest of the video can be saved in a lower resolution as the focus lies somewhere else.

This passive approach has been tested for various resolution sizes in the past[43] and was implemented in multiple foveated image codecs. The goals are things like an automatic fixation detection[79] or a good compression with optimal rate control[46]. Notice, that neither of these approaches have a dynamic component, as they statically render the eye information into the image data. This also means, they are error-prone if saliency diverges from actual gaze direction.

The alternative are active gaze estimations. This approach is more dynamic, as it accounts for an actual user input[15][52]. Typically, pupil detection algorithms are used to measure the current eye positions. Afterwards, they are used to dynamically set a gaze estimation used for further processing.

An actual example of this is foveated ray-tracing. It uses active gaze estimation within an HMD to render geometric scenes using gaze-contingent rendering techniques[29]. Instead of captured video data it uses three dimensional scenes. As ray-tracing is already sample-based, a foveated approach can easily be added to save calculations in arbitrary regions. This is a huge difference to this thesis' topic, as video data is commonly not prepared for sampling.

This thesis uses an approach using active gaze estimation. Opposed to previous publications for foveated codecs using passive gaze estimation[79][46], this approach does dynamically adapt to each users' eye motion by incorporating an active gaze estimation. Thus, it is not possible to estimate a singular fixation point or remove any data, as high resolution data must be available for *any* region of the video at *any* time. Similar to the ray tracing approach[29], the goal of this approach is a reduction of actual data to be processed and later streamed to the GPU. A major difference to other approaches is the data access as video data is not well structured for foveated processing and needs to get a newly designed data format.

2.4 High Resolution Textures

The core challenge of loading and streaming high resolution data to the GPU is one of the main issues for video games and other kinds of dynamic applications rendered on a graphics card. Thus, this field is well studied and numerous approaches and solutions exist. This is also relevant for other applications, like geo information systems[33].

Mip-mapping The standard approach to different texture sizes in video games is mip-mapping[84]. The key idea is saving downsampled variants of each texture within the forth free quadrant of a RGBA image. This way, video RAM is saved efficiently.

Furthermore, different projects solve efficient texturing by using different mip-map values for mapping regions[30]. Thus, mip-map levels must be selected reasonably[18]. There are other ideas to break off from standard mip-mapping like Octree textures[8] but they usually introduce an overhead in algorithmic complexity.

In this thesis, video RAM is not specifically targeted, but the approach of having downsampled duplicates of video data gives additional opportunities for efficient video access.

High resolution video approaches Similar to the goal of this thesis, previous projects have investigated high resolution video playback within virtual environments[41]. They use a grid to load video tiles based on the head direction. While this is closely related, no analyses of eye position or the simulation of visual acuity are performed.

In another project[64], these ideas are implemented for video game engines to have a procedural technique in content generation. In a related talk by Microsoft[1], these techniques are exploited to enhance texture streaming to the GPU. The idea is to update high resolution texture data in smaller tiles for different rendering regions independently. This allows a dynamic texturing of scenes for different level of details while zooming and using a memory efficient data layout for the textures.

Video decoding Video decoding is an essential pillar in the design of this thesis, as video data is loaded from the hard drive in real time, before then processed and streamed to the graphics unit.

For video decoding, a region or pixel-based access is uncommon or not possible. Thus, a foveated imaging process has trouble accessing data depending on eye position. Region-of-interest can be included into the encoding process[32], but is usually not available for decoding.

There are efforts to make it possible to extract region-based data from a video with only reasonable overhead[2]. However, these techniques are not implemented in commonly available video processing frameworks.

2.4.1 Ray tracing-based approaches

Ray tracing[5] is not common in both virtual reality applications and video processing techniques. Therefore, one might wonder why to include it into the background chapter.

The nature of ray tracing based approaches is based on sampling. In almost every case, some kind of raw scene data is pre-processed in a way to quickly load data. Furthermore, data lookup is usually done based on sampling or looking at certain chunks of the scene.

In this way, these two sets of problems are related. In terms of video loading, there is a need of data pre-processing in order to efficiently decode only relevant parts of the full frames. Furthermore, to incorporate knowledge from the human vision system, it will be necessary to have a look on how other applications in computer graphics solve similar issues.

For data pre-processing, ray tracing uses acceleration structures to have geometrical data in an efficiently traversable format. More modern approaches use sophisticated algorithms incorporating geometrical features of the scene. While this is promising and could be also used in videos by incorporating a saliency-based approach, it is not the focus of this thesis.

However, in the beginnings of ray tracing, simpler data structures like the grids were popular. A grid is a space partitioning data structure for ray tracing, introduced by Fujimoto et al. [23]. In its most common approach, the Uniform Grid, it divides the three dimensional space into $N \times M \times O$ cells of same size, so called *voxels*. For this project, this could be adapted to a 2D approach.

Additionally, ray tracing tries to solve several sampling-based challenges. Sampling errors[55] and performance issues due to the amount of sampling per pixels[77] are common issues. This is strictly related to the Nyquist sampling theorem[65]. This thesis tries sampling-based approaches which are prone to similar issues.

Chapter 3

Method

This chapter discusses design options for developing a foveated high resolution video renderer. Background knowledge from the previous chapter is distilled to form an outline suited for an efficient implementation.

As shown in Figure 3.1, the design scheme can be summarized by dividing the process into three main blocks. Loosely, design decisions are either concerning data pre-processing, video loading or texture streaming. The choice of data type, video codec and method to factoring in acuity influences the former two. The way of transferring pixel data to the GPU and reconstructing the rendered frame is part of latter.

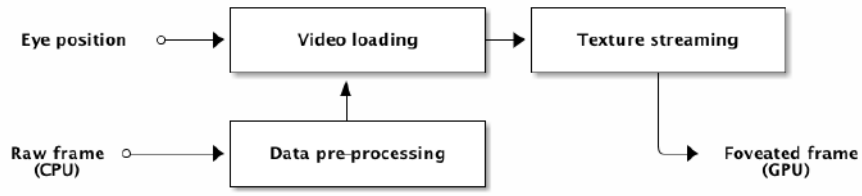

Figure 3.1: Algorithm flowchart: This flowchart visualizes the three main pillars of the design. The algorithm will have some element of data pre-processing which is solely static and can not contain eye information. This is noteworthy, as the algorithm needs to dynamically adjust to eye movements. The video loading process then uses the eye position information as determinant to decrease pixel data and in succession reduced bandwidth demand. Lastly, the texture streaming sends all necessary data to the GPU in a way, that shaders are capable of reconstructing a full frame.

First, section 3.1 will introduce three design patterns applicable to introduce acuity as a determinant into a rendering algorithm. Each technique is explained and compared in terms of assumed rendering quality and algorithmic efficiency. These approaches set an outline for designing data flow within

the algorithm. Next, section 3.2 will determine necessary pre-processing steps in order to successfully load region-based foveated video data. Lastly, section 3.3 will discuss both the design of video loading and texture streaming, as they are closely related.

3.1 Factoring in acuity

As discussed in the introduction, the main motivation of this thesis is a foveated approach to video data streaming. Video bandwidth commonly creates a major bottleneck for media streaming. An uncompressed RGB frame consists of $w \cdot h \cdot 3$ character values for width w and height h. Or in other words, it creates a stream of raw data linear to the videos frame rate and quadratically to its resolution.

When factoring in knowledge about visual acuity (Section 2.3), it follows that only a small percentage of pixels are perceivable in a very high detail. The rest blurs out drastically depending on the distance to the users' focal point. As a consequence a vast majority of pixels, and thus bandwidth, is wasted in regions which are not perceived at highest level of detail.

In case of normal video rendering, this information is not necessarily helpful, as the viewer usually watches videos from a high distance. Due to this, the radial decline may not be strong enough and most of the scene is perceived normally. This of course depends on the size of the display.

However, HMD systems confront the user with displays extremely close to the eye. While the virtual distance to the scene is still about 2 meters, it vastly increases the field of view. Therefore, the decline of visual acuity has a strong impact on corner regions of the overall image.

While this knowledge is handy, both the raw data and the algorithm needs to be designed in a certain way to take advantage of these characteristics. The former will be discussed later in Section 3.2, as its design is largely dependent on the following results.

The input to a foveated imaging algorithm is a raw frame F and a foveated area A corresponding to the focal point p and its radius r. As the main purpose of this algorithm is a reduction of frame data, the output should be a newly computed frame C, so that $size_C \leqslant size_F$. The area A needs to be perceptually equal in both frames, but rendering mistakes become increasingly tolerable with distance to point p.

The minimum discernible angular size increases roughly linear within the first 20 to 30 degrees, but then rises more rapidly[82]. Thus, the vision can be categorized into two regions, called *peripheral vision* and *central vision* with a peculiar acuity drop in between.

For this purpose, three kinds of techniques are discussed in greater detail. Sampling-based approaches have a direct mathematical relationship to the acuity function, while grid-based approaches are a rough estimation and

therefore have a rather loose connection to the radial decrease of visual acuity.

Lastly, analyzing image variance can hint at certain *points of interest* within an image. This can be seen as a supplement to other approaches, as it has no direct connection to the visual acuity by itself. Each one is implemented and exhaustively tested in Chapter 4 and 5 respectively.

3.1.1 Sampling-based approaches

When modelling visual acuity as a mathematical function $F(p_{x,y}) = a$, one gets an acuity value a for each pixel $p_{x,y}$ of an input frame. As previously described in Section 2.3, this results in a radial distribution with a very high resoluted focal center region. When visualized as a heat map, it roughly results in a distribution close to shown in figure 3.2.

Figure 3.2: Visual acuity heat map: This heatmap rendered on top of a video frame visualizes the visual acuity distribution of a human eye. Warm colors indicate a high resolution while cold colors indicate a low resolution. The users' foveal region lies in the center of the image.

In ray-tracing techniques, a ray is casted for each pixel in a given frame. Every ray is evaluated in 3d space to calculate whether it hits an object. Using this, a color value for each single pixel is determined. Using techniques similar to this, visual acuity can be modeled using optical ray tracing for medical applications[26].

The results of the acuity function are constant, as the distribution does not change over time. Moreover, due to its radial nature the values only depend on the radius r relative to the focal center. Thus, it is possible to precalculate all values to a 2 dimensional *acuity mask*. Using this, eye movements can be treated as translation by a vector $\vec{v}$.

While this approach has a close mathematical connection to the way the human eye perceives images, it has some drawbacks when it comes to

data processing. First, the eye perceives these acuity changes continuously while an algorithm unavoidably has discrete units. Second, the eye as a very high amount of cone cells in the foveal region[66]. This can not be simulated by fixed rectangular video resolutions. Thus, it is prone to sampling-based errors [55] known from ray tracing. Furthermore, sampling is sensitive to an introduction of performance bottlenecks[77].

While there are region-based decoding algorithms for some video codecs, no currently available approach is capable of per-pixel extractions. When analyzing video encoding algorithms, it is clear why this would not be efficient. Usually, regional data is woven into a zipped data format by looking at temporal and neighborhood-based variations. Therefore, extracting small regions would create a big data overhead per pixel, rendering encoding efforts useless. As a result, when sparsely sampling only relevant data from previously decoded video frames, every frame, or at least large regions need to be decoded entirely. Therefore, data access can not adapt closely to the sampled acuity value.

In conclusion, a sample-based approach can only efficiently influence the texture streaming process. It is possible to gather samples from decoded video data. However, loading video tiles can not be implemented efficiently based on sampling.

3.1.2 Grid-based approaches

This idea focuses on adjusting the sampling-based approach for a better way to load video data. In order to load video regions in a simplistic and naive approach, incorporating grid-based ideas have been one of the first data structures for ray tracing[23]. The problem is in fact similar, as classic ray tracing tried to solve the same issue: A during pre-processing unknown distributed set of sample points on an arbitrary set of data.

Modern approaches in ray tracing often greatly incorporate the scenes geometric data and its variance. It is an option for foveated imaging too and is discussed in the Section 3.1.3 in greater detail. However, in terms of complexity, there is a focus on a universal approach suitable without further video analysis.

The idea of a grid-based approach is simple. The surface is divided into $t = n \cdot m$ tiles of the same size. For foveated imaging, it would result in a pre-processed video of t small *video tiles*. This is a naive approach to allow loading single video regions without the need of a specialized video codec supporting such functionality. When applying a grid on a video, it results in a spatial subdivision as visualized in Figure 3.3.

The previously discussed sampling-based approach to factor in visual acuity can be roughly approximated with a suitable sized grid. An algorithm selects a quality setting for every grid cell. Based on this, different raw data is loaded for each cell. This can be done efficiently for video loading.

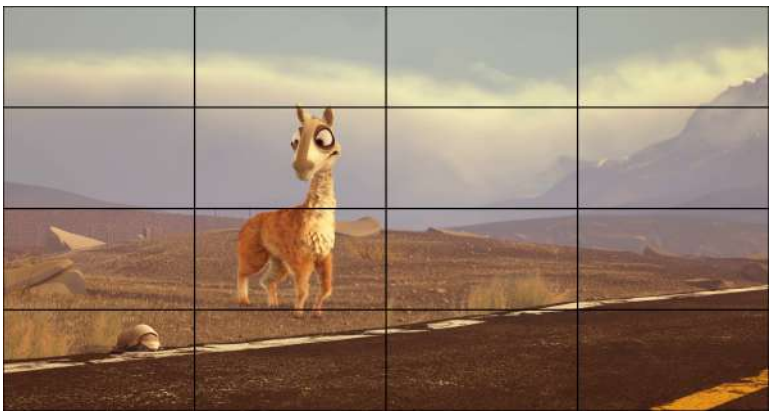

Figure 3.3: Grid based data pre-processing: This scene visualizes a wide-screen video pre-processed in a 4×4 grid. Afterwards, every video tile will be available in different resolution settings.

Furthermore, grid-based approaches are also favorable in terms of texture streaming. Most GPU functions are optimized for batch operations, with a strong disadvantage on every per-pixel access of texture data. Similarly, CPU-based imaging libraries are also heavily optimized towards batch operations, as micro-benchmarking easily reveals. Therefore, updating a full video tile at once has the highest efficiency.

As a result, for foveated imaging techniques a grid-based approach is a rough approximation of sampling-based approaches. They can solve the problem more efficiently in both video loading and streaming. This makes them a main candidate for the implementation.

3.1.3 Saliency-based approaches

This section does not emphasize on visual acuity by itself, but rather the possibility of incorporating the full human vision system. As previously discussed, these approaches sample sparsely and thus their results are prone to sampling errors. Therefore, rendering artifacts, sharp edges and other visual problems can harm the viewers' perception.

They have an impact on the human visual system and thus the viewers' perception. Therefore, image variance is commonly estimated within saliency-based approaches[51]. It shows, that these approaches have promising results when it comes to simulating the behavior of a human eye.

As a result, it might be advisable to introduce further determinants based on the HVS into the algorithm, to ensure smooth transitions and sustainable rendering. While the focal center should always have the maximum acuity, this is not true for other parts of the image. Image data by itself could have relevant information on which quality setting is most suitable when selecting acuity values for cornered objects.

For example, an empty house wall without either a peculiar texture or color deviances has less impact on our subconscious mind than repeating checkerboard patterns. Similarly, a white car has probably less impact than a rainbow colored car. Based on this, in order to optimize the algorithm, color variance are an interesting point to analyze.

When determining these factors, RGB color space turns out to be improper for comparisons of human perception[74]. HSV, LAB or other[11] especially for human perception developed color spaces can give better assumptions for this algorithm.

Furthermore, temporal variance and other image characteristics like FFT transformations and the amount of edges per image region can have further impact on perception[76][6].

Variance-based approaches give an interesting second determinant in the choice of image acuity based on the video data itself. While this is not as relevant for the foveal focus region, it can give better assumptions on other image regions. However, this needs a vast amount of additional image analysis within the data pre-processing phase.

The focus of the thesis is based on approaches using visual acuity and without information based on image data. Thus, not all saliency-based approaches have been investigated. While optimizing and fine-tuning the final results, some color variance techniques were added. Other approaches are open for future work.

3.2 Video pre-processing

Depending on the acuity function, video data needs to be loaded in different resolution settings based on their position relative to the eye center. However, no available video codec has functionality relevant to help with this flow of data.

While there are approaches to decode videos based on their region, they are not implemented in commonly available video processing frameworks. This increases the complexity for implementations.

Furthermore, they come with an overhead in accessing a singular region. When only one region is accessed, this is of course much faster than decoding a full frame. However, to fully sample a whole frame based on visual acuity, pixels from every region in a video are needed. As a result, these approaches can not provide a reasonable increase in efficiency, when it comes to loading acuity based data.

Thus, in order to efficiently access video data, the data needs to be pre-processed. This way, the algorithm can circumvent limitations of the codecs.

The pre-processing step can theoretically take as much time as suitable, as it needs to be done only once per video. This way, it does not restrict the capabilities of the rendering system.

Mip-mapping A main bottleneck of video accesses is due to decoding whole frames of tremendous resolutions. For an 8K frame, only a small patch of a few hundred pixels is relevant to the foveal focus region. This is a challenge, as most codecs don't allow easy access to single regions and impede sampling-based approaches.

To solve this, in a first step, mip-map levels for the video should be generated. This can be pure down-sampling, as common in video game engines. Please note, that in video gaming the mip-mapping is commonly done dynamically after GPU streaming. In foveated imaging, it will be done in advance to enhance efficiency up to the level of file access.

The size of mip-mapping is chosen depending on the scene. There is no reason to limit it to certain multipliers or ranges, as most shader-based reconstructing techniques can circumvent any issues with custom texture functions. However, it may lead to anti-aliasing and rounding errors for unfortunate chosen mip-map sizes. This can occur if the original video size is not divisible by multiples of two, where the resulting mip-map can be half a pixel off depending on the video region.

Uniform grid As previously discussed in section 3.1.2, the data can be transformed to a grid-based structure. A simple video or image pre-processing can cut them into single *tiles*.

When having no information about the video characteristics, a uniform approach to grids is the only reasonable choice. There is the possibility to analyze videos in advance and determine a better suited grid approach. However, this has a tremendous additional complexity, bursting the limits and intentions of this thesis. Thus, it is moved to the future work for a more extensive analysis.

A uniform grid has a resolution of $n \cdot m$. For quadratic video input, as popular in panoramic videos using cube mapping, setting $n = m$ seems well suitable. For wide-screen videos, a ratio related to their aspect ratio might be more efficient.

Utilizing a combination of uniform grid and mip-mapping, the video can render in a very high mip-map level on one side of the screen while being in a very low mip-map level on the rest of the screen.

Variance data To implement the ideas from Section 3.1.3, it might be the most efficient way to calculate variance data in advance. It does not change based on the users' eye position, as it is solely based on video characteristics.

This data can be calculated in parallel to re-encodings, but should be performed on the highest mip-map level as it contains the most video data.

3.3 Video handling

This section describes the design of video handling. First, video data is loaded into the memory based on the acuity algorithm. Then, it is transferred to the GPU and reconstructed in order to render a full frame. Transferring the pixel data to the GPU is also called *streaming*. Video game techniques commonly use dynamically changing textures, so this problem is distinct to usual 3D processing done by a GPU.

Video loading The loading is based on the video codec of choice. When introducing common video processing libraries, all of them should share a similar API. Thus, multiple codecs can be used and compared in terms of performance.

Alternatively, a series of single pictures are semantically equivalent to videos. It needs to be tested whether real videos or picture series are advantagous for the algorithm.

Based on an implementation of the acuity function and the focal point as input data, a set of video tiles is loaded. As this process might result in a bottleneck of decoding, a multi-threaded design is advisable.

Eye movements result in a change of resolution settings and thus a change of loaded tiles. In case of video files, this means other video tiles are opened and seeked to the relevant frame to be loaded. Unfortunately, seeking might not always be available in linear time, which complicates the video loading algorithm.

In order to load video tiles efficiently, the multi-threaded approach needs to adapt to this and fill unavailable video tiles with other data. For example, it could load a *wrong* resolution for a few frames, while the right mip-map level is processed in another thread. When ready, it can gradually exchange the video tiles to fully adapt to the users' visual acuity. This is discussed extensively in later sections regarding implementation (Section 4.1.3) and its limitations (Section 5.5).

Texture streaming A texture update, that means an upload of raw data from CPU to GPU, is one of the most expensive operations in graphics frameworks. Therefore, the data needs to be as minimal as possible.

For GPU operations, frame buffer objects tend to be the fastest option, with pixel-buffer objects being a close second depending on the use case. It is cheaper to update sub-regions than updating a whole texture at once. Furthermore, the options of OpenGL allow various custom formats like 1D textures for lookups using 1D indices. This way, sampling based approaches can be optimized by collecting 1D arrays of data and reconstructing their full frames within the shader.

3.3.1 Reconstruct video frames

Rendering a wide-screen video with hardware-acceleration is commonly done by rendering a 2D texture to a panoramic plane in parallel to the view port. This is an easy task for most videos and has no issues with bandwidth for normally sized videos. However, a high resolution video can exceed the bandwidth limit easily, as every frame has a raw data of $w \cdot h \cdot c$ pixels per width w, height h and channels c. This is a quadratical increment with its resolution.

For gaze-contingent rendering to work, the video data is pre-processed on CPU and pixels are selected in a way to ensure a minimum of raw data to be transferred. It sends all data needed for shader-based reconstruction of the full image to the GPU. Afterwards, the shader needs to combine the data efficiently.

On the CPU, the data is compressed as much as possible. This can be easily done by removing irrelevant data and focusing on either the sample data by itself or chunks of different resolution data. As a goal, no duplicate data should be transferred.

Afterwards, on the GPU, there needs to be some kind of interpolation to reconstruct the full frame with only limited data transferred. This can be done using look-up tables and precalculated masks.

As an alternative to decreasing the raw data send to GPU, the texture updates can occur less often. While most videos have a frame rate of 25, GPUs easily exceed this. The data for every region should however to be updated at least once per video frame, as the difference between two frames can be big enough for the user to perceive. Furthermore, in a grid-based approach, the video frame needs to be consistent, as frame inconsistency can result in hard edges depending on video contents.

3.3.2 Handling 360-degree panoramic videos

For panoramic videos, a similar approach has been tested to have interactive rendering of high resolution video[31]. Based on the users' head position, video chunks are loaded from the video data based on their cube mapping. Different mappings suitable for this are visualized in figure 3.4. The video tiling is solved similar to a grid-based approach. To guarantee a fluid streaming, prefetching of non-visible regions was added. Their approach however does not factor in dynamic resolution deviations based on visual acuity. Furthermore, no multiple mip-map levels were used, so the loaded video tiles have an equal resolution.

In this approach, a simulation of the visual acuity is added. This results in resolution deviations which need to be modeled by the renderer in order to sufficiently stream texture data to the GPU.

For displaying panoramic textures on a GPU, most approaches favor

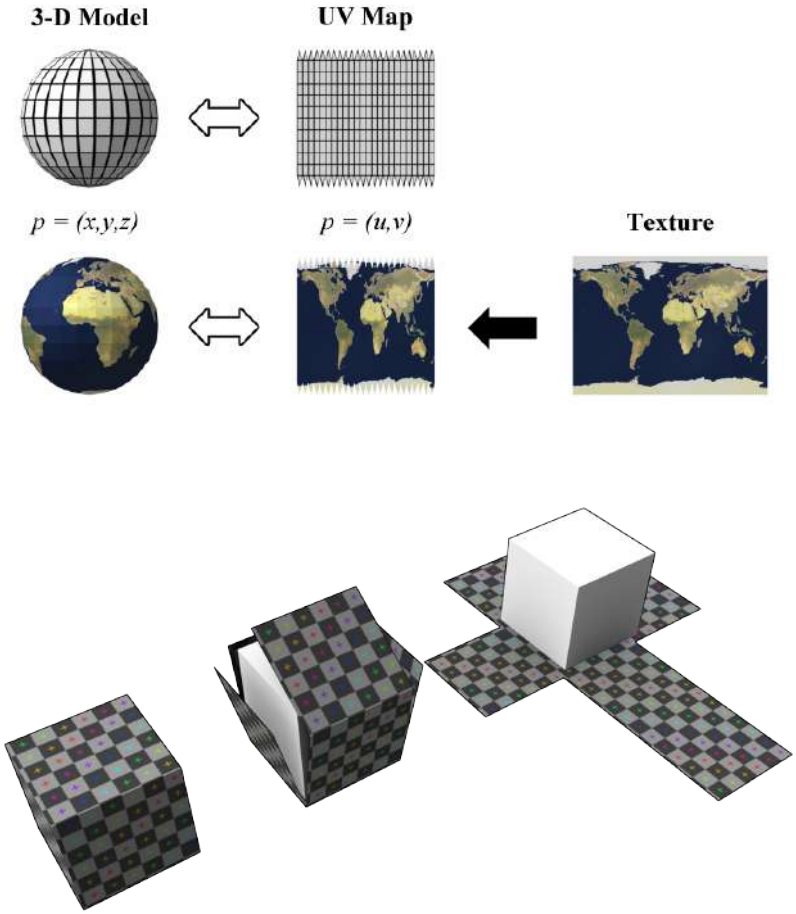

Figure 3.4: Panoramic mappings: The top figure shows a classical UV map which is spherically mapped to its 3D model[75]. In the bottom, it shows a cube mapping[86]. This approach is common for skymaps and similar panoramic approaches, as it allows expressing a 360-degree cube surface as six quadratic 2D textures.

either a spherical or cube-mapped based projection. A cube-mapped based projection consists of six quadratic textures simulating each face of a cube[25]. Cube-mapping is commonly used for skydomes in video games. This approach allows a mobile viewpoint in the scene, making interactive scenes easier to create. Due to this, cube-mapping is heavily integrated in modern graphic cards. As a disadvantage for cube-mapped data, a uniform grid gets distorted towards the corners due to its projection. Therefore, tiles in the seams are ought to be selected more conservatively.

As an alternative, sphere mapping provides another common mapping for 360-degree panoramic videos. A uniform grid mapped on a spherical surface has fewer distortion in terms of perspective. However, this approach needs further transformations when calculating uniform grid positions based on the view vector. This adds a slight complexity to its calculations. Additionally, it has disadvantages for possible usage of mobile scenes.

Prefetching can be introduced for smoother rendering of movements. As pupil movements are prone to having very fast saccades[63], a low resolution version of regions close to the users' foveal center need to be accessible at any time. This is usually already true, as this data is used to render out-of-focus regions. Furthermore, depending on the overall speed of the implementation, quality discrepancy during eye movements might be visible. Real time videos are the main purpose, so changes due to eye movements are delayed a maximum of one video frame.

Additionally, head movements can be added. Implementation wise, this is equivalent to eye movements. However, the angle increases and 180-degree movements would be possible. It is a performance trade-off, which movements are taken care of by the rendering. Again, the maximum waiting time to adjust head movements will also be one video frame. This could have a minor perception impact on low frame rate videos.

Chapter 4

Implementation

This thesis' implementation has an iterative prototype-driven development. Due to this, there have been multiple versions incorporating different phases of design and development. When grouping together, there has been implementations in two directions, resulting in a rewrite half-way during the project. As every approach has his insights in future development phases, they are discussed exhaustively. This points out advantages as well as draw backs of different implementations.

Both resulting frameworks were developed on Unix based systems written in standard C++ '14. They use SDL 2.0[44] and OpenGL 4.1 for rendering. In terms of video and image loading, a mix of OpenCV 3.1[35], FFmpeg 2.8.6[21] and STB[7] were used. Although not tested on other platforms, each code was written with portability in mind.

In the first iteration discussed in Section 4.1, a framework for wide-screen videos based solely on sampling-based approaches was designed. As video loading had various issues due to the nature of their video codec, a hybrid solution using grid tiles was established.

Next, a rewrite discussed in Section 4.2 focused on better support for panoramic videos. Due to afore-mentioned issues, this approach adapted into a solely grid-based solution. This version also adds cube maps as rendering surface. Thus, it is better suited for VR applications.

Lastly, the brief Section 4.3 discusses ways on debugging HMD usage on a computer as well as issues when porting current solutions to a HMD, while section 4.4 emphasizes on optimization steps, both in terms of performance and rendering quality.

4.1 Framework for wide-screen videos

This section summarizes the first iteration of development. For this approach, testing a sampling-based approach to factor in visual acuity was a main goal. To implement video access more efficiently, a grid-based approach is added.

This results in a hybrid model.

In order to load the video files, a complex multi-threaded handler is built to ensure a fluid video rendering. It will sacrifice accurateness in terms of visual perception for a few frames after eye movements, in order to maximize the speed of the system.

For processing pixel data to the GPU, it uses a sophisticated approach with pre-calculated masks: a static mask to determine samples, as well as an interpolation mask which serves as a look up table for nearest neighbor interpolations. It can efficiently transfer video data and later reconstruct it linearly during shading.

4.1.1 Acuity by sampling

As known from ray tracing, in a sampling-based approach there are n samples, which are shot into the scene. Every pixel is assigned a value based on its hit point. For this project this simply means every sample has the color value of the video on that position. There are three things to note.

- Uncommonly, based on the nature of visual acuity, the sampling rate is dynamically changing in radius to the foveal center.
- The visual acuity is however static and does not change when moving relative to the foveal region.
- There are gaps between samples which need to be filled with data.

As a basis for the sampling, an *acuity mask* is calculated. The mask has the size of the viewpoint. It has the highest density in the center region and gets sparse in distance to the center. This mask can be applied on the eye view vector to have a static knowledge about all necessary sample points.

For an efficient solution to interpolation, pre-calculation is an option. In the same fashion as the acuity mask, all interpolation values are static. Thus, an *interpolation mask* is created.

Both masks are created in Matlab and exported to a text format. The acuity mask is simply a list of (x, y)-coordinates for every sample. In this implementation a 1920×1920 mask with 690926 samples is used. The interpolation mask is a matrix, where every (x, y)-coordinate has a specific sample as value. For later convenience, the sample indices are transformed to 1D. Within the interpolation mask, the nearest sampling point to every coordinate is calculated. As a result, the interpolation mask contains the nearest neighbor approximations in a hard-coded manner.

Drawbacks of sampling

Unfortunately, sampling-based appproaches come with a hand full of drawbacks. A prominent issue is aliasing. This often happens in patterns or close

to edges, where sampling points are prone to errors.

Shown by the Nyquist sampling theorem[65], the sample rate needs to be twice as high as the information frequency. In case of the visual acuity approximation, this ratio gets worse with distance to the foveal focus center. Thus, the error gets bigger. This can be an issue as big regions are interpolated, especially in the corner of the eye. Therefore, a sampling-based aliasing issue will result in heavy flickering for corner areas, as those as mostly prone to errors.

Another issue is performance. Most algorithms for software and hardware are optimized for batch access. Access to a single pixel are often only possible with large and slow workarounds, which can introduce another bottleneck. This is already an issue for single pictures, where pixel-accesses can't be optimized in terms of hardware cache for example. When using video materials, this problem gets severe.

There are mathematical solutions for accessing some regions of the video, albeit having an overhead compared full video access in relation to region size[2]. However, these are not commonly available for a wide range of video codecs, and furthermore are not implemented in any high-level video processing libraries. On top of that, this is still a region-based access and does not accelerate pixel based access, which results in very large overhead for sparse corner pixels.

Grid-based acuity for video access

The issues for sampling-based access to the video data is decoding speed, as a major bottleneck of high resolution videos is the performance of decoding full frames. While 4K videos are not an issue anymore, a video with 12K resolution and a high bit rate is unplayable for any consumer grade hardware. Therefore, the video loading needs to adapt to this situation.

When evaluating samples for every region of the video, every frame would need to be decoded entirely, albeit most of the video data is not used and discarded after the sampling process.

In order to avoid full frame decoding, a grid-based video loading is introduced below the sampling algorithm. The video is pre-processed in a new file format containing a uniform grid and mip-map levels for every grid cell. These videos are called *video tiles* and are a major pillar for the video loading process.

The sampling will use the highest mip-map level for the area in the foveal focus region. In distance to this point, lower mip-map levels are used for sampling, providing a faster decoding speed, as the video tiles will have much lower resolution, corresponding to the visual acuity.

As a result, this framework uses a hybrid approach of both a sampling-based algorithm and a grid-based algorithm. The video is loaded in a grid-based manner to avoid issues and performance bottlenecks with full sized

video data. Afterwards, a sampling-based acuity mask is applied to add a realistic approximation of the visual acuity. In the end, an interpolation mask with precalculated value fills in blank areas between sampling points.

4.1.2 Data pre-processing

Every video needs to be in a certain format to satisfy the requirements of an efficient video loading. Both grid-based subdivisions and mip-map values are needed.

As a result, every video gets re-encoded in two smaller mip-map levels. Furthermore, every mip-map level is split into a 4×4 uniform grid. This creates 48 small video files out of the source video. While this comes in hand with a storage overhead, this step is necessary for video loading to perform in efficiently.

The application of a uniform grid on a wide-screen video as resulted by the pre-processing function is visualized in figure 3.3 in previous chapter. It shows a 4×4 grid. A 8×4 grid would be another thinkable option, as it better adapts to the nature of a wide-screen video. However, it also doubles the amount of open file handles and video streams in total.

The pre-processing is done by a bash script utilizing FFmpeg[21] for video processing. The video format is interchangeable and various formats are tested in the evaluation chapter 5.

4.1.3 Texture loading

The acuity and interpolation masks are fully loaded into the memory. Then, an algorithm decides for every sample point in the acuity mask, which video tile needs to be loaded. Video tiles out of view are not loaded. The selection of mip-map levels are based on a thresholding in distance to the center of the acuity mask.

For every video tile, all video files for every mip-map value are open in parallel. In terms of processing usage, video seeking is quite expensive. Therefore, an open file handle for every tile and mip-map will provide fast access to the next frames with the smallest amount of seeking possible.

Video handling This comes in hand with multi-tasking issues and the need for a sophisticated threading model around the frame retrieval phase. The process is visualized in Figure 4.1.

The video handler has a main thread which executes sampling for every video tile. It assumes the open file handle for a tile to be at frame $n-1$ to calculate the frame n. After acquiring frame data, a pixel-based access for each sample determines the sampling results for this tile. This is directly done for every video tile which is *waiting* at frame $n-1$. This would be the

case, if the last frame was calculated beforehand and no eye movements have occurred.

In parallel, the video handler checks which needed open file handles are not *waiting* at frame $n-1$. If this is the case, the relevant video handle was unused in the last frame and did not seek to a recent frame. This happens if eye movements have occurred, which results in a change of the mip-map levels to be loaded for each tile. The video handler will spawn one thread in a thread-pool for each open file handler which needs seeking.

If the sampling thread is in idle, it will start using lower mip-map levels, which might be seeked to a more recent frame, to complete its sampling. This case, the sampling can be done more quickly while sacrificing rendering quality. As the texture streaming has no relevant bottleneck, this streaming can happen for every GPU frame. This means, the quality of sampling can be adjusted from frame to frame, whenever the seeking threads are finished.

As an analysis[14] of the retinal integration time shows, the eye can forgive approximately 3–4 frames of artifacts and jitter before fully perceiving a region change in full detail. Thus, short delays of full resolution frames after eye movements are negligible.

Texture streaming The sampling is done completely on CPU to minimize the bandwidth to the GPU. The interpolation map which contains a 1D index for every sample on every (x, y)-coordinate is transferred to the GPU on the initialization. As it is static and can be shifted by the eye view vector $\vec{v}$, it does not need to be updated. It is implemented as a `GL_TEXTURE_2D`, but uses 1D 32-bit integer values instead of RGB float values, which results in a `isampler2D`.

The sampling data is transferred to the shader as a `usamplerBuffer`. These are 1D arrays containing single elements, but are addressed like textures using methods as `texelFetch()`. Every element is addressed by an index and contains a RGB 8-bit integer value based on its sampling results.

Using these scheme, the interpolation process is a simple chaining of both texture lookups. For every 2D texture coordinate, there is a 1D integer value inside the `isampler2D`. This 1D integer value is the index for the `usamplerBuffer` array containing the sampling data for this point.

```
1 uniform isampler2D interpolationMask;
2 uniform usamplerBuffer samplingResults;
3 uniform vec2 eyeVector;
4 in vec2 texcoord;
5 out vec4 color;
6 [ . . . ]
7 int index = texelFetch(interpolationMask, texcoord - eyeVector).r;
8 color.rgb = texelFetch(samplingResults, index).rgb / 255.0;
```

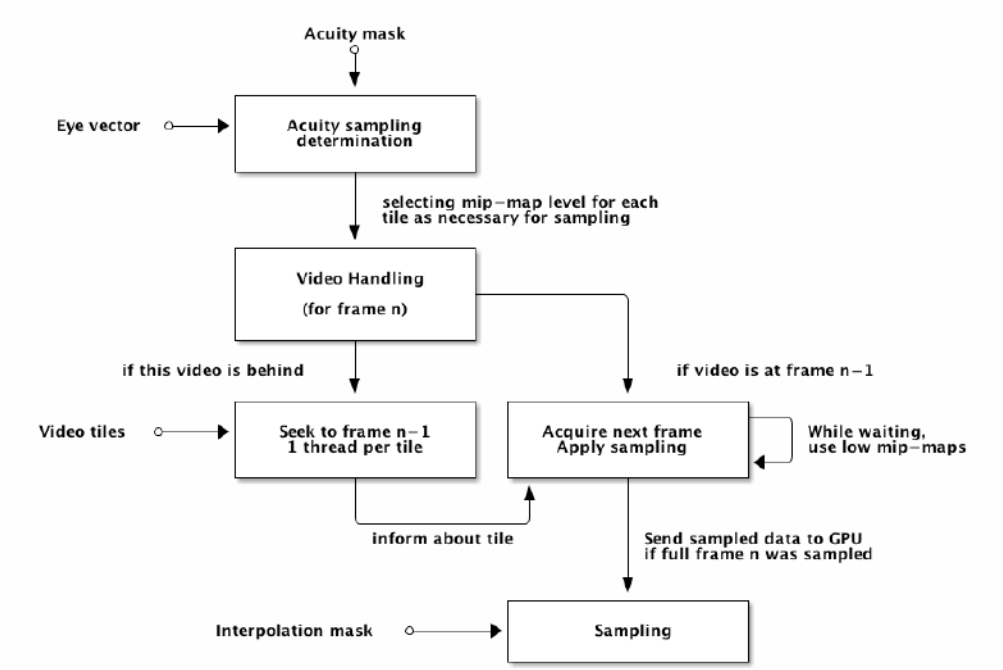

Figure 4.1: Widescreen framework flowchart: This flowchart visualizes the video loading process. Based on the static acuity mask, a suitable mip-map level for each video tile is determined. The video handler loads a frame from every video tile, which is already seeked to the right frame and applies sampling. In parallel, a thread for each unready tile is created for seeking to the current frame. If the sampling thread is in idle, it starts using lower mip-map levels in order to quickly finish the current frame. If any resolution level is finished processing, the data is sent to GPU for interpolation and rendering.

By implementing the masks as static data and chaining them in this way, the shading has no overhead added through interpolation. The amount of bandwidth used by this approach is at an absolute minimum, as only raw sample data is transferred for every frame. This process is visualized in Figure 4.2.

4.2 Framework for 360-degree panoramic videos

This section shows the second iteration of development. The foundation is based on the previous prototype discussed in Section 4.1. However, a majority of the code has been redesigned to more efficiently adapt to panoramic videos.

As the previous framework already used a hybrid model and grid-based pre-processing, the sampling-based approach seemed to be an unnecessary

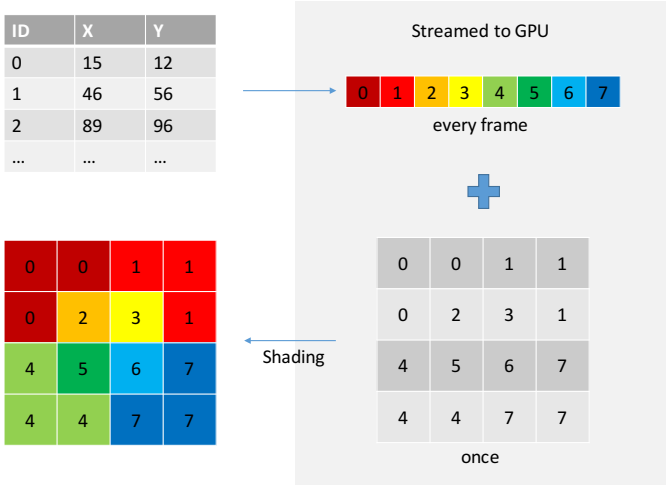

Figure 4.2: Sampling-based frame reconstruction: This figure visualizes frame reconstruction using two masks. The acuity mask is a list of sample points which are evaluated for each frame. In combination with a static pre-calculated interpolation mask, the shader is able to reconstruct a full frame.

drag. It can introduce various kinds of sampling-related issues and also result in flickering and visual artifacts. Thus, the sampling-based approach has been removed as it introduced a handful of unfortunate problems.

Due to the removal of the sampling-based techniques, the shading is completely rewritten. Again, the implementation features an efficient way to minimize texture streaming and reconstruct the frame linearly.

4.2.1 Data pre-processing

As the previous video-based approach introduced a number of issues with multi-tasking, seeking the video and loading a high amount of video tiles at the same time, this approach uses another data format.

Due to its simplicity in loading, picture files are used instead of videos. Therefore, every video is split into single files per frame. Any format will work, although JPEG appeared to be the fastest for decoding.

There are six picture files per frame. This corresponds to the panoramic video being transformed to a cube-mapped projection. Every file is read and sampled down to each mip-map level. Similar to the last approach, three mip-map levels are used. Afterwards, uniformly sized cells are extracted and saved as single files. For simplicity, every cube side gets their own uniform

grid.

For a 4×4 grid, this results in 288 files per frame. Due to the enormous amount so files, they are saved in a folder structure based on their mip-map level and grid cell. This step is purely optional, but facilitates file handling operations within the operating system. This approach can be extended to use some container file format, which might be more convenient, but has no semantic impact on the algorithm by itself.

Additionally, every video tile is analyzed in terms of their RGB color variance. This test is best done on the highest resolution level, as it contains the most information about the frame. For later evaluations, a simple average over the three color channels is used as a *variance determinant*. More advanced approaches are possible, but skipped in order to keep this process simple. The variance data is calculated in parallel and saved in a large matrix for later usage.

The script is based on Python, utilizing Pillow 3.0[12] for image operations.

4.2.2 Acuity function

This approach redesigns the visual acuity algorithm in terms of its fundamental approach. While the previous algorithm was based on a direct mathematical sampling of each pixel and interpolating their values accordingly, this approach acquires data based solely on lookups within the grid. The data is read in chunks, which can usually result in faster access due to better optimized library functions as well as cache efficiency.

The data pre-processing creates three mip-map levels for every grid cell in the cube map. In order to select mip-map levels relating to the users' viewpoint, the view vector based on data from the pupil detection is used:

```
1 side, axis calculateCubeside(vec3 eye):
2     switch eye:
3         eye.x >  0.5: return +X, X
4         eye.x <= 0.5: return -X, X
5         eye.y >  0.5: return +Y, Y
6         eye.y <= 0.5: return -Y, Y
7         eye.z >  0.5: return +Z, Z
8         eye.z <= 0.5: return -Z, Z
```

The function `calculateCubeside` can calculate the side of the cubemap, on which the user is looking. Afterwards, the axis used for calculating the current cube side is not longer used, but the other two axes contain information about the grid. The grid is uniformly distributed over the surface of one cube side. Thus, a simple division and rounding down to the nearest integer returns the grid cell, or the video tile has data for this region. This example uses a uniform grid with 4×4 cells:

```
vec2 calculateGridPosition(vec3):
    side, axis = calculateCubeside(eye)
    otheraxis1 = (getAxisUnequal(eye, axis).a + 0.75) * 2
    otheraxis2 = (getAxisUnequal(eye, axis).b + 0.75) * 2
    return (otheraxis1 / 4.0, otheraxis2 / 4.0)
```

Using the knowledge about the users' viewpoint, an algorithm will select mip-map levels for each tile based on their position relative to the foveal focus. Additionally, a saliency-based approach can use variance data to further enhance the selection process. Note, that this has no further relation to the users' eye position, as this determinant is solely based on pre-processed data. The following pseudo-code visualizes the selection process:

```
void selectMipmap(vec3 eye):
    setquality(ALL, lowest)
    // Grid-based acuity
    vec2 fovealCell = calculateGridPosition(eye)
    setquality(fovealCell, maximum)
    foreach(neighborCell : adjacent to fovealCell):
        setquality(neighborCell, halfway)
    // Saliency-based acuity
    foreach(noiseCell : cell near maximum(color variance)):
        setquality(noiseCell, halfway)
```

Every cell is initially set to the lowest mip-map level. Based on the local distance to the eye position, the setting is increased accordingly. Afterwards, a saliency-based approach can be added to further enhance the results.

Notice, that this approach does not necessarily set tiles behind the user black. Therefore, everything on the mip-map gets at least the lowest mip-map level. This can have an impact on the overall performance, but helps in terms of very fast movements. As a result, the user will always see a current low resolution version of the video, even if turning their head 180 degrees.

When finished, this created a distribution as seen in Figure 4.3. This information is saved in a *acuity matrix* and passed to texture loading and streaming in order to render the video. As all calculations up to this point are computable in linear time, they are no bottleneck and can be computed for each frame. This way, there are no delays for eye movements to be used within the texture loading.

4.2.3 Video handling

This section describes all steps relevant to video handling. As seen in Figure 4.4, a central element of this approach is the acuity matrix calculated in the previous section. Using this, the video handler will load all relevant video tiles and stream them to the GPU. Finally, the shader combines all tiles to a full image.

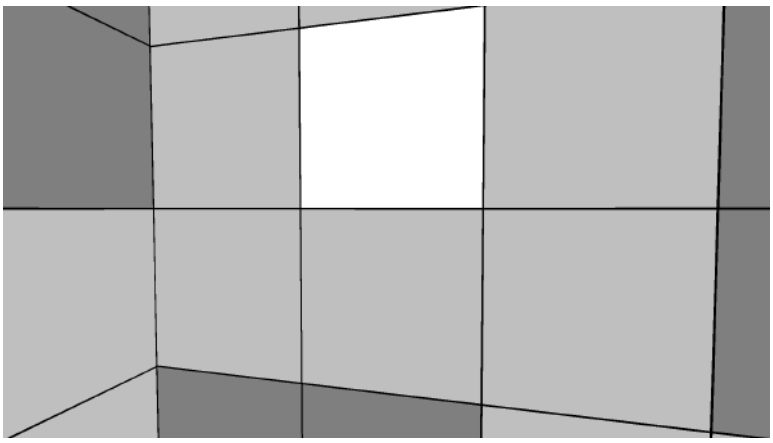

Figure 4.3: Mip-map selection for panoramic video: Based on the view pointer, the algorithm selects a map of suitable mip-map values based on visual acuity. White indicates a full resolution, light grey indicates a $\frac{1}{4}$ resolution and dark grey indicates a $\frac{1}{16}$ resolution.

Video loading

In this framework, video data has been available as pictures series. Panoramic videos were calculated from equirectangular data to a cube mapped format. Therefore, there are 6 pictures per frame. The pre-processing algorithm has created a set of mip-map values and their tiles per frame.

When removing the sampling based approach, there is no further need for image-manipulation functions, as any loaded file can be directly streamed to the GPU. Due to the nature of being very big and versatile libraries, FFmpeg and OpenCV introduce an overhead. As a result, the loading methods switched to *stb_ image* as a lightweight alternative.

In most development phases, a 4×4 uniform grid has been use, as it turns out to be the most efficient. There is one grid for each cube side, and thus there are 96 tiles per frame in maximum resolution, and 288 tiles when counting all three mip-map levels.

The amount of files to be read in parallel are a major bottleneck. However, pictures are easier to handle in these amounts, compared to video files. Previously, Section 4.1 discussed challenges when decoding a vast amount of video tiles in parallel and the need to parallelize this process. As it turns out, this is less severe for picture files.

As every frame has his single file set, the operating system does not need to keep as many files open at the same time. Opening a new picture is cheaper than seeking to a new position in most video codecs. Therefore, file handles can be closed conservatively, which also frees up resources.

In terms of parallelism, several approaches have been tested. However, a sequential algorithm has similar performance than multi-threaded approaches.

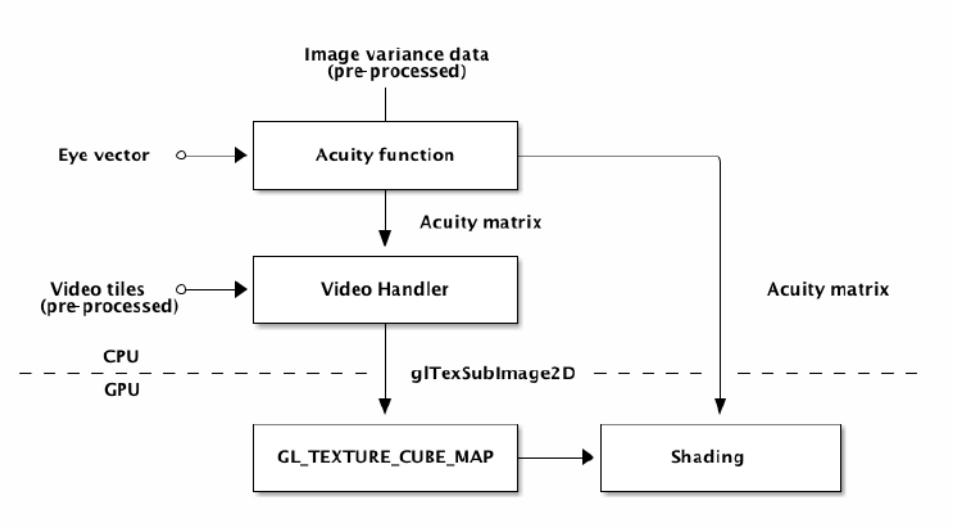

Figure 4.4: Panoramic framework flowchart: This flowchart visualizes the video loading process. Using on the acuity function, a matrix containing a suitable mip-map level for each video tile is determined. The video handler will load one video tile per grid cell and stream it to the GPU. The shader can combine the picture based on the acuity matrix.

This is due to the nature of selected mip-map values. There is a maximum of one tile in the highest resolution. Furthermore, a vast majority of the tiles are loaded in tiny resolution levels. For a 12000×3000 video, the smallest mip-map level has grid cells of 188×188. These are very quickly loaded and do not justify a complex threading system.

Video streaming

The video loading provides an array of video tiles in different resolution. This means, for a cube map with a 4×4 grid, there are 96 tiles.

There are three `GL_TEXTURE_CUBE_MAP` textures. Internally, OpenGL uses an array of six `GL_TEXTURE_2D`, which would be semantically identical. Every texture has a different size, corresponding to the pre-generated mip-map levels.

During initialization, OpenGL filtering techniques and the built-in mip-map functionality is deactivated as it wouldn't fit this approach. OpenGL would create own mip-map levels for the full texture on every update. As this approach is streaming texture updates only in relevant regions, this behavior is unwanted.

Using `glTexSubImage2D`, only relevant regions are updated each frame. This command executes an update of a sub-image. That means, providing an offset as well as a size, only relevant regions are updated per call. As this does

Figure 4.5: Streamed texture levels: For every frame, only updated texture data for every mip-map level are streamed to the GPU. The pictures displaying the transferred cube-map data for the maximum (top left), medium (top right) and lowest (bottom) resolution. The grey area is not updated, and thus either containing data of old frames or being uninitialized.

not trigger any OpenGL post-processing, it is the fastest way to dynamically stream texture data.

In one frame, only one of the three cube-map textures is updated for every video tile, according to its mip-map level. This will minimize the bandwidth used for texture streaming. As shown in figure 4.5, only regions and mip-map levels relevant to the visual acuity of this particular frame are updated.

It results in large parts of un-updated regions for every texture. This is however intended, as the shading process can patch the frame together based on the acuity matrix. Using the tiled approach for different resolution levels, it resembles some ideas of the afore-mentioned talk by Microsoft[1].

Shading

The shader works very similar to the acuity function described in Section 4.2.2. Hence, the GLSL code can use the same `calculateGridPosition` function as described previously.

All three cube-maps are passed to the shader as cube-map textures. The shaders are using 3D cube texture coordinates to simplify the calculations. This is a trick, as now the texture coordinates passed into the fragment shader are corresponding to the eye vector. When passing the texture coordinate into `calculatePosition`, the grid cell corresponding to this video tile is calculated.

Furthermore, a matrix with the grid cell to mip-map level mappings is passed to the fragment shader. Using this data, the shader can use this as a lookup table to decide, which cube-map is used for this texture coordinate. This way, a dynamic switch between the textures based on their texture coordinate is possible:

```
switch(lookupMipmap(calculateGridPosition(texCoord))):
    == 1:
        color = texture(skymap1, texCoord)
    == 2:
        color = texture(skymap2, texCoord)
    == 3:
        color = texture(skymap3, texCoord)
```

4.3 HMD compatibility

In order to be flexible with developing cycles, major parts of development were executed without use of a HMD system. To reliably debug each approach, visuals were rendered to a common computer screen. Pupil movements are simulated by mouse movements, with the cursor being the center of the foveal focus region. Head movements can also be simulated by adding keyboard commands.

While pupil movements are a main concept for this thesis, head movements don't introduce any needed functionality. Large head movements will be equivalent to large eye movements. Furthermore, there are small relocations like breathing or nodding, where the eyes' point of fixation might not change. These can be solved by translating the whole rendering surface in (x, y)-direction without any changes to the visual acuity process.

During finalization of this project, a rough port to a Windows-based framework has been tested. With Oculus Rift[19] having Windows as one of its target platforms, a sophisticated library for HMD solutions (so-called LibOVR) was developed. Focusing on the Oculus Rift platform provides functionality and driver compatibility for a number of HMD systems.

As previous approaches were written with portability in mind, adding their code to new frameworks is easy. This way, support for connected HMD systems is easily solvable by adding LibOVR.

While it has not been tested, stereoscopic rendering will presumably only have a minor impact on the overall performance of the system. In case of a sampling-based approach, the sample must be done twice. However, for a grid-based approach, roughly the same grid tiles can be used. The loaded mip-map levels as assigned by the visual acuity functions is roughly the same for both eyes. Thus, there is no additional overhead introduced by the second eye, in terms of video handling. Other parts of the algorithm are not as

critical, when it comes to performance, so there are no bottlenecks to be expected.

4.4 Optimization steps

This section discusses further optimization steps which have an impact on the rendering quality as well as performance of each approach.

Mip-map selection based on image variance As discussed in Section 3.1.3, there are other determinants except visual acuity which can be introduced. This way, the system can simulate the human vision system more closely.

In order to test this approach, a naive method was implemented. In the pre-processing step, each grid cells average RGB color variance value is calculated and saved in a matrix format. Later, this data is read to modify the mip-map decision step.

Using this, cells with an enormously high color variance are loaded in a higher mip-map level to compensate for possible perception changes due to resolution differences.

Blur To have a closer approximation of visual acuity, a radial gauss blur can be introduced. This can be necessary, especially for the grid-based approach from Section 4.2. While this approach ensembles an efficient way for video-loading and pixel access, it creates a checkerboard pattern. With only three mip-map levels in the current implementation, the jump between two mip-map levels (as seen in Figure 4.3) is comparatively big.

Although these resolution jumps are only present in the corner of the eye, they can create confusion. This can introduce hard edges on locations, where no edges are to be expected.

Adding a blur can easily fix this drawback, removing edges in the corner of the eye and therefore reducing possible points of distraction. It is important to notice, that only regions out of the foveal center are blurred. Due to this, the blur can only have an impact on the subconsciousness. The foveal center will always be displayed in a maximum resolution without post-processing.

The blur is implemented and tested as a sole post-processing step, so it does not change previous implementations. It is important to set up the fall-off by the radial blur similar to the visual acuity function used by the rest of the algorithm.

Performance The performance is variable and can be quickly modified by changing the levels of mip-map. In the default setup, there are three levels of mip-maps for full resolution, $\frac{1}{4}$ resolution and $\frac{1}{16}$ resolution. These parameters

can be easily adapted to different scenes and have a high impact on their performance.

The system tries to load a maximum of one tile with maximum resolution. This is necessary as these tiles have an extreme impact on the frames per second. It can be necessary to load multiple tiles when looking on edges. This is a drawback of the approach and not avoidable. However, the algorithm tries to minimize it.

The second mip-map level is not as critical, although it is still much slower than the third. It is usually used for everything not directly in the foveal center, but close to it. This means, all adjacent tiles are on this level. Additionally, the variance-based approach adds further tiles to this mip-map level based on color variances. This is a pure thresholding process and can be adjusted as convenient.

Another optimization could add further mip-map levels. Especially for panoramic scenes, adjustments in the rendering pipeline would be needed. The current implementation uses a full cube-map for each mip-map level to easily switch between the mip-map levels based on texture coordinates without OpenGL texture functions interfering in terms of filtering and streaming performance.

Due to this, there is an overhead in video RAM. This is negligible as video rendering does not have other objects or textures to render. However, a higher amount of mip-map levels will need additional video RAM. The current implementation can render 12000×3000 panoramic videos on 2 GB video RAM utilizing up to three mip-map levels.

Chapter 5

Evaluation

This chapter discusses an exhaustive evaluation of all approaches and available options presented in the previous chapters.

First, Section 5.1 gives an overview of hardware and software environment used for these analyses. Everything used for implementation and testing are consumer-grade devices and every library is freely available. This makes the frameworks easily portable. Next, Section 5.2 discusses the pre-processing step used for both frameworks. It is analyzed in terms of storage usage as well as runtime. The pre-processed data is loaded based on different design decisions evaluated in Section 5.3. Based on their video codec or image format, a variety of performance footprints are analyzed. Afterwards, the raw performance is analyzed in Section 5.4. This gives a good overall view on the algorithm and its strength. Lastly, the short Section 5.5 will summarize perception differences based on their resolution settings.

5.1 Test environment

This section will describe the test environment all evaluations have been measured in.

In Section 5.1.1 the hardware environment is described. Only common consumer-grade devices are used. Next, Section 5.1.2 describes the software environment used. Both frameworks were developed in a UNIX environment and solely use libraries with free licensing. Lastly, Section 5.1.3 describe test scenes used.

5.1.1 Hardware

The evaluations are done on a MacBook Pro with a 2,5 GHz Intel Core i7 Haswell CPU and 16 GB DDR3 RAM. Although there is a vast amount of parallel encoding processes, the CPU has only a minor impact on performance. More importantly, all calculations are done on a SSD drive.

The grid-based approach results in a high amount of data-accesses on different parts of the hard disk. As longer videos can result in a high amount of storage usage, it would not be viable to pre-cache everything in RAM. Furthermore, as eye movements can dynamically change, there is no way to reliably assume which video tiles are necessary in the next frames.

This results in a very high amount of disk accesses, more specifically a high amount of accesses on different regions of the hard drive. Therefore, using this approach with older hard disk drives using rotating disks is not possible. They would introduce a delay of the moving elements to retrieve different sectors, in contrast to the simultaneous accesses possible with a SSD. This is not sustainable for this approach. This limitation is however negligible, as SSD drives are standard in most consumer level devices nowadays.

The graphics card is an NVIDIA GeForce GT 750M with 2 GB of video RAM. As both approaches have a very low amount of actual data bandwidth and the shader code for reconstructing the frames on GPU are limited, there is no relevant bottleneck when it comes to processing power of the GPU. Therefore, any model capable of OpenGL 4.1 is sufficient.

However, a high amount of video RAM can be necessary for especially the framework for panoramic videos. It is implemented using multiple high resolution cube maps. This results in a high memory footprint on the GPU and can be optimized with a more sophisticated texture model, but memory efficiency has not been focus of this project.

As HMD usage has been simulated, no further hardware is required. The HMD port described in Section 4.3 uses an open library called LibOVR[19] for driver accesses and providing an extensive API for HMD. Therefore, a usage within HMD systems is possible for any device supporting this API.

5.1.2 Framework

The frameworks described in Chapter 4 were implemented as described and tested within Mac OS X 10.11.3. In terms of libraries, SDL 2.0[44], OpenGL 4.1, OpenCV 3.1[35], FFmpeg 2.8.6[21] and STB Image 2.08[7] were used.

Both frameworks are analyzed in terms of their individual settings. As every framework uses different approaches to solving the problem, the evaluation differs slightly. However, providing an extensive view on different design decisions is in the focus here. Thus, comparisons are made where applicable.

The port to a HMD library is completely based on the implementation of the panoramic video framework. Therefore, evaluations for this port have no relevant impact on the analyses. As a result, the port to windows has been excluded from extensive testing, as every measurable parameter is already presented within the other evaluations. As discussed earlier, the mouse position is used as an input data for the acuity-based algorithms. This means, eye position is simulated by user input.

As an abbreviation, from this on, this chapter uses *Framework 1* for the

first implementation using wide-screen videos with video files and a hybrid approach for factoring in visual acuity. Respectively, *Framework 2* is used for the second implementation featuring panoramic videos with a solely grid-based approach and image files.

5.1.3 Scenes

As most evaluations are based on performance, storage and bandwidth usages, the contents of the videos only have a subsidiary influence on each analysis. Despite some variance-based approaches discussed in the optimization step (Section 4.4), no part of the algorithms is influenced by the video contents.

For the purpose of evaluating different approaches, free scenes from various internet platforms were selected. As Creative Commons videos in a 8K resolution are rare, upsampled data has been used to simulate performance behaviors.

For all three scenes, three distinct mip-map levels are used. The mip-map levels are full resolution, the $\frac{1}{4}$ resolution and the $\frac{1}{16}$ resolution. The grid is 4×4 when not other specified.

Scene 1: wide-screen video Caminandes

This scene *Caminandes* (see Figure 5.1) is a video produced as an open project of the Blender Foundation[22]. It is licensed with Creative Commons and thus freely available. The original video is 1:30 minutes and shows a small story about a llama trying to pass a street.

As 4K is the highest resolution available for this video, it is scaled up to 8K and 16K for the purpose of these evaluations. This obviously can not result in an increased quality without at least a post-processing step. However, evaluations regarding performance, storage and bandwidth can be simulated realistically for these video sizes.

It is originally available in h.264. For the evaluations the video is additionally re-encoded to VP 8, VP 9 and h.265, as tests using multiple codecs are necessary.

Regarding pre-processing, a 4×4 grid is used for 8K and a 8×8 grid is used for 16K.

Scene 2: panoramic video Inside the GI tract

This scene *Inside the GI tract* (see Figure 5.2) is an educational 360-degree video[34]. It is licensed with Creative Commons and thus freely available. It shows a flight through the human body, which makes it a good example in context of virtual reality.

The video was already provided as cube-maps in a resolution of 960×960 per cube side. This is scaled up to 4096×4096 per cube side for the purpose of these evaluations. This means, the upscaled panoramic video is 16384×6144.

Figure 5.1: Scene 1: wide-screen video *Caminandes*

It was available in PNG format, but is re-encoded to JPEG for performance reasons.

For most analyses, a subset of 100 frames are used. This allows an easier rough estimate for larger scenes in terms of storage usage and other measurements. The length of the scene has no relevant impact on the performance.

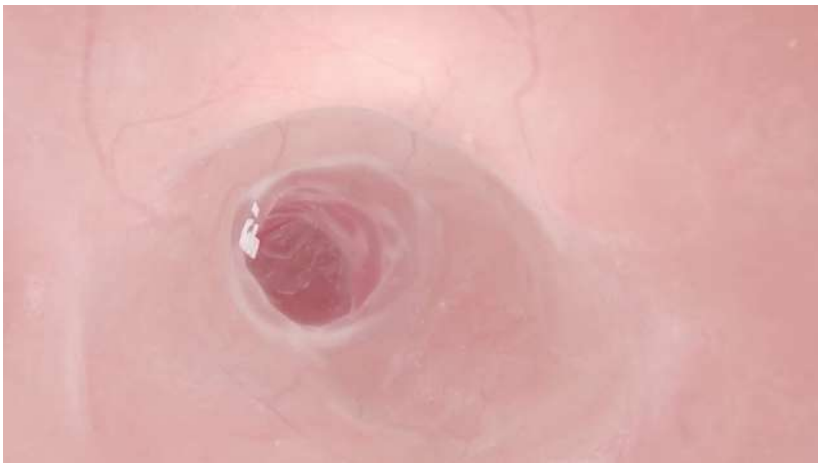

Figure 5.2: Scene 2: panoramic video Inside the GI tract

Scene 3: panoramic video Christmas market

This scene *Christmas market* (see Figure 5.3) is an internally shot scene of a German christmas market, using six GoPro cameras[24] in a 360-degree array. The scene shows a typical town square scene with people walking around and different visual depths due to small houses and booths. It originally consists of 221 frames, but was reduced to 100 frames.

As a special property of this scene, it does not feature a top and down view. Therefore, the $\pm Y$ cube sides are not used and can be skipped for performance reasons.

The raw data has a format of 12000×3000, resulting in a cube-map of 4 faces, each 3000×3000. It is originally in PNG format, but will be re-encoded to JPEG for performance reasons.

Figure 5.3: Scene 3: panoramic video Christmas market

5.2 Pre-processing data

When being guided by the flowchart in the beginning of Chapter 3, there are three main pillars this approach is based on. The evaluation is roughly executed in the way of data flow. First, raw data is pre-processed into an easily loadable format. It consists of a uniform grid and mip-maps for each video tile.

Grid and mip-map The grid size has a minor impact on performance. In small scale tests, it seems to be a trade-off, as a bigger grid size results in much more files to read and decode simultaneously. Thus, a choice between 8×4 and 4×4 for wide-screen videos results in an almost identical performance, or during eye movement it might be even slightly slower. While the grid adapts more closely to the nature of wide-screen videos, the amount of parallel open video tiles is also doubled.

On the other side, a very small sized grid can also harm the acuity approach as it increases the size of a video tile. This defines how quickly the visual acuity can decrease. A very small grid size results in big areas of the same mip-map level loaded.

The number of mip-map levels were set accordingly. Two mip-map levels are too few to successfully adapt to visual acuity within the gaze-contingent rendering approach. The first tests were rendered using up to six mip-map levels. It turns out that within the perceptual vision region the lowest mip-map level is almost always sufficient. Furthermore, the central vision region is simply not large enough to successfully allow more than 2 different levels of grading for a grid of size 4×4 or 8×8. Thus, three mip-map levels are an ideal choice.

Pre-processing step by itself For Framework 1, all video codecs use the *recommended* settings for high resolution data. Usually, this means the slowest preset combined with a sufficient bit rate. Thus, it uses 10 mbit/s for full resolution video tiles (roughly 1080p), 5 mbit/s for the medium mip-map level (roughly 240p) and 2 mbit/s for the smallest mip-map levels (lower than 100p).

Due to the overhead in redundant data, these process can vastly increase the amount of storage used by the videos. As shown in Figure 5.4, the input data has further influence on this ratio.

In the first framework, storage used is around six times larger, due to additional mip-mapping and grid data. In Framework 2, the storage is actually decreasing by around 50% when using JPEG files. This is due to the change of the image format necessary for performance. While PNG has a stronger compression and is providing lossless image data, JPEG generally produces smaller files. The smaller mip-map levels have a very small impact on the overall storage as described in Table 5.1. The addition of two new mip-map levels just increases the overall storage up to 20%.

Note, that scene 2 is much smaller than scene 3 despite the higher resolution. This is due to the characteristics of the scene, as it is one-sided in terms of color variation. Thus, it is better compressible.

Approach	Mip-map 0	Mip-map 1	Mip-map 2
Framework 1 (Scene 1)	1885 MB	365 MB	46 MB
Framework 2 (Scene 2)	721 MB	81 MB	14 MB
Framework 2 (Scene 3)	3520 MB	463 MB	92 MB

Table 5.1: Data usage per mip-map: This table visualizes the data used for every mip-map level. This data is evaluated for one video codec and image format per framework, as there is no significant change in the relative distance between mip-maps depending on their format.

Unfortunately, the pre-processing step itself has a high runtime. This is however tolerable, as it has no influence on the performance of the rest of the system and is only done once per video. Depending on the video codec or image format, the time can vary. An overview is shown in Table 5.2. For

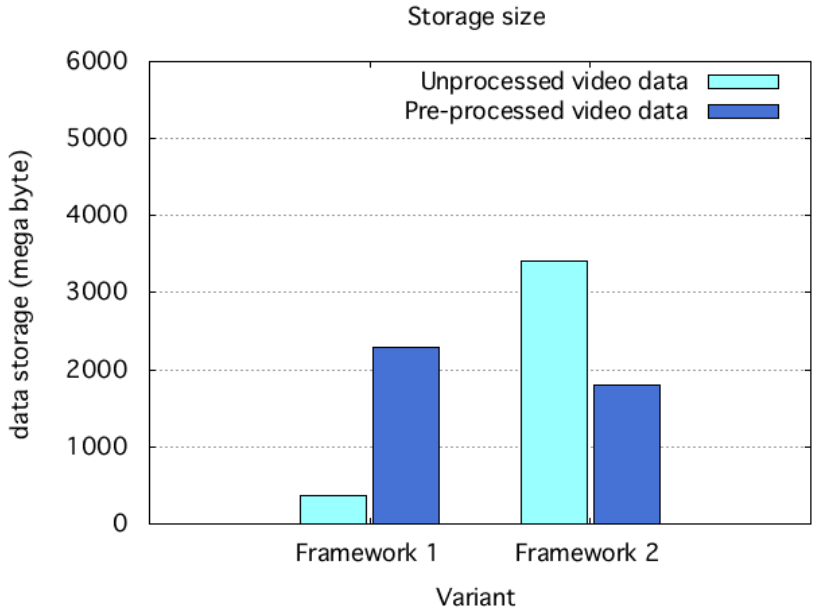

Figure 5.4: Storage usage: This graph shows the difference in storage used by video data before and after pre-processing. The Framework 1 (left) refers to the wide-screen video approach. An increase in storage is natural, as the mip-map and grid data introduces a big overhead in video files. The Framework 2 (right) refers to the panoramic video framework. There is a striking reduction in video storage. This is due to a re-encoding from a lossless format (PNG) to a lossy format (JPEG) due to performance reasons. This had no visual impact on the rendering.

runtime analyses, VP 8/VP 9 unfortunately come with a very high overhead in encoding speed. While h.265 is already about three times slower than h.264, the VP 8 is about 8 times slower and VP 9 around 16 times slower. For the image formats, PNG turns out to be two and a half times slower than both JPEG variants. This is reasonable, as it is the only lossless format, which results in a more complex compressing algorithm. In terms of storage usage, all results seem reasonable. The H.265 can tremendously decrease video size. The high quality JPEG setting ironically uses more space than the lossless PNG due to re-encoding.

Notice, that the pre-processing was done in a single-threaded manner. Depending on the codec, it might be more efficient to use a multi-thread model. Some codecs, like h.264/h.265 already use multi-threaded encoding by the FFmpeg library. Optimizations within the pre-processing part have no priority, so this is not further tested.

To summarize, the first pillar of data flow, pre-processing, has an impact

on storage and has a high runtime by itself. The focus of the project is real-time rendering of the video streaming and the reduction of actual bandwidth. Therefore, there are no restrictions on pre-processing runtime or storage usage, as long as they are reasonable.

Approach	time spent	storage used
Framework 1: VP 8	3 h 58 min	3237 mb
Framework 1: VP 9	8 h 17 min	2294 mb
Framework 1: h.284	36 min	1923 mb
Framework 1: h.285	1 h 30 min	104 mb
Framework 2: PNG	50 min	1608 mb
Framework 2: JPEG q=90	17 min	816 mb
Framework 2: JPEG q=100	21 min	2500 mb

Table 5.2: Ressource usage for pre-processing: This table visualizes the time and storage used for pre-processing videos. The numbers of Framework 1 are using the first scene (1:30 min, 8K). For Framework 2, it references scene 2 (100 frames, 16K). It scales roughly linear with other video lengths.

5.3 Video handling

After evaluating data pre-processing, handling the video data is the next step to look at. This section does not directly analyze parts of the algorithm, but rather underlying technologies.

Depending on the framework, video data can be loaded in various different formats, which are analyzed in Section 5.3.1 and Section 5.3.2, respectively.

Finally, the Section 5.3.2 gives a broad analysis of actual used bandwidth, video memory and GPU processing power. The exact amount of pixels transferred to the GPU for each frame is discussed and compared to the full resolution. This shows the real bandwidth advantage of visual acuity as an approach to video streaming, which was one of the key motivations of this thesis.

As a side note, this evaluation assumes most decoding to be fully done on CPU. The underlying library FFmpeg is not capable of GPU acceleration for most codecs. A GPU acceleration could greatly improve the loading speed, as the GPU is mostly unused and would have spare processing power available.

The key conclusions drawn from this part of the evaluation are related to bandwidth usage and the way texture streaming was implemented. These parts are unrelated to the video loading process and therefore remain true for other codecs or image formats.

5.3.1 Video codecs

An analysis of different video codecs is visualized in Figure 5.5. This shows the performance deviations when switching codecs due to different decoding speeds. The error bars try to visualize stuttering due to eye movements.

All codecs result in a similar speed with the VP 8 and VP 9 codecs having a slight advantage. However, there are several things worth mentioning, which can not be reflected in the diagram.

The h.264 has various significant issues with frame-perfect seeking. This seems to be an open issue for both the current version OpenCV and FFmpeg. Due to this, it is not possible to implement a frame-perfect seeking using the available API.

Every call has a chance to seek to a *close* frame, which might be one or two frames away from the actually requested frame. This unfortunate behavior can create visual gaps between video tiles, as they a rendering different frames of the original video. This however does not correspond to an *inter frame* (or *i-frame*), which is an internally used frame within the codec. Inter frames are added to provide redundancy by duplicating full frame information in certain intervals while encoding. These issues are measurable even if the codec is set to save an i-frame for every single frame. Thus, it seems to be a bug within the decoding process. There is an open bug ticket for OpenCV regarding this behavior[58].

Additionally, this is not consistent through all video tiles and can vary based on the frame. As a result, h.264 produces regular flickering and hard edges when loading wrong frames for adjacent video tiles during scene changes.

In h.265, similar issues were noticeable, albeit less severe. The h.265 API and functionality does not seem as well implemented within FFmpeg, which is due to h.265 being a closed standard. Unfortunately, h.265 decoding also introduces a massive stuttering, which furthermore rules out this codec.

Overall, VP 8 and VP 9 seem to perform very well, both in terms of performance and the amount of issues and hiccups surrounding the multi-threaded tiling approach. There is no major difference between these two, but VP 9 performs best in terms of performance deviations and stuttering due to eye movements. Thus, VP 9 is the codec of choice for other evaluations.

5.3.2 Picture formats

For Framework 2, the data format of choice switched to picture files, which makes a second evaluation regarding the file format necessary. The performance for different formats is shown in Figure 5.6. The figure shows average performance when rendering the acuity approach with different formats. JPEG turns out almost as twice as fast as PNG when using $q = 90$ setting. The maximum setting $q = 100$ is slightly slower, but still around 70% faster than PNG.

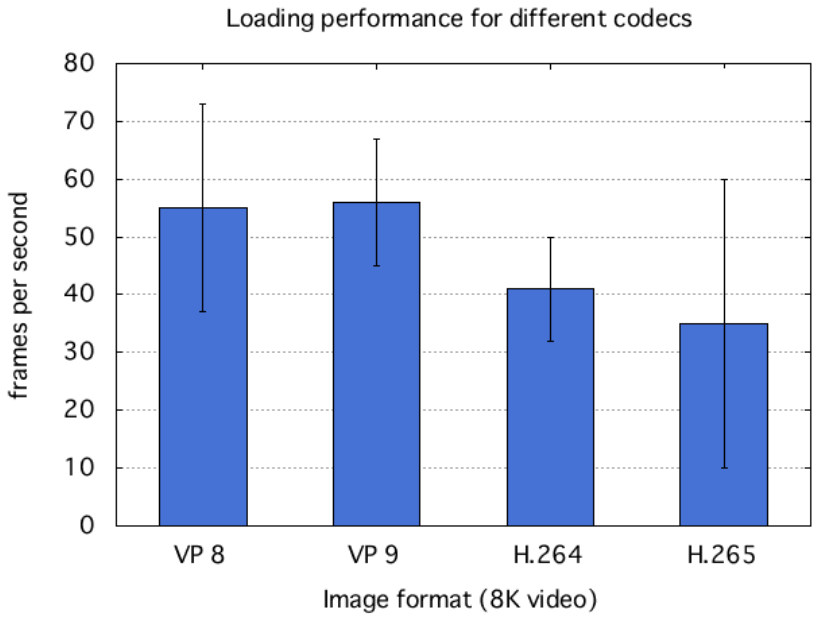

Figure 5.5: Performance for different codecs: This graph shows performance changes when using different codecs for decoding the video tiles. It is purely based on the performance, and different codecs can have additional draw backs not reflected by this graph, as discussed in section 5.3.1. The error bars show performance derivations due to eye movements, as the video handler receives a sudden peak of data to process.

Notice, that PNG is a lossless format. Standard JPEG is always lossy, even at quality set to the maximum. There is a lossless variant, but it was not tested as it is uncommon and thus not well implemented in most libraries.

There are other interesting formats like TGA, TIFF or DDS (among others), which are possible candidates for this approach. However, most imaging libraries can not handle these files properly. Usually, these formats can not be saved, and even the loading methods are much less optimized. Due to this, they are skipped in these evaluations.

5.3.3 Texture handling

The previous section described the loading of video files, where the algorithm does not use the GPU. In contrast, the following section will describe all interfaces to the GPU. The CPU/GPU interface is where bandwidth will play a key role. Furthermore, the memory footprint and necessary GPU processing power are ought to be analyzed to get an informed knowledge of the system.

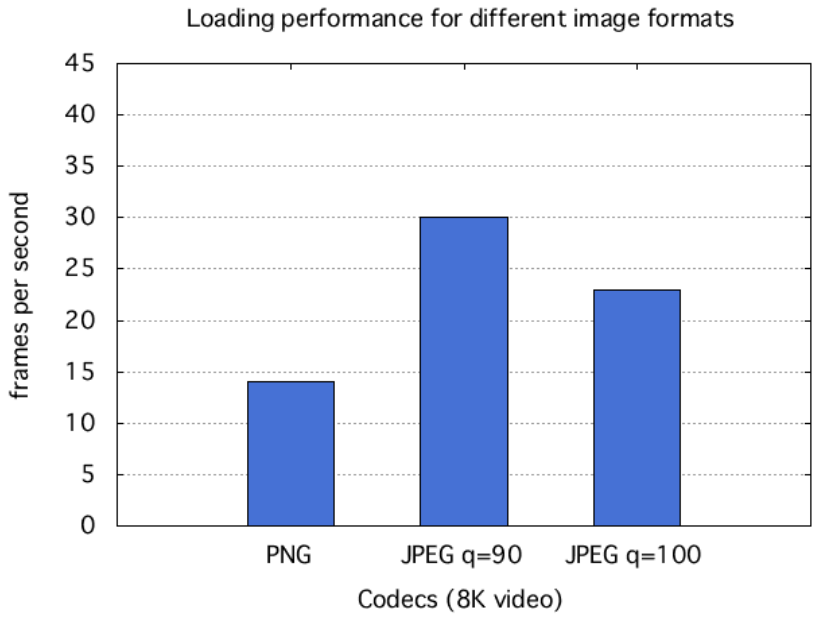

Figure 5.6: Performance for different codecs: This graph shows performance changes when using different image formats for decoding the video tiles. This just shows differences in performance without factoring in quality. For example, while PNG is slower, it always provides a lossless decoding, while JPEG is not. The tested scene is scene 2.

Bandwidth Both approaches are mainly designed to result in a data bandwidth as minimized as possible. The results are shown in Table 5.3. It analyzes the pure pixel data transferred to the GPU. This is unrelated to the previously decoded video data, as not all decoded data is necessarily also transferred to GPU.

The Framework 1 shows the highest decrease possible based solely on visual acuity. As a result, a reduction to 2 percent of the maximum bandwidth is possible. The sampling-based approach results in a mathematically optimized minimum of bandwidth. However, sampling-errors and slow single based pixel accesses are a major drawback.

Despite the grid-based approach being only a very rough estimate of the acuity function, the Framework 2 can show almost comparable bandwidth results averaging between 2.9 and 4.9 percent. This is due to the nature of panoramic videos, where a whole 360-degree surface is rendered. Thus, a high amount of bandwidth is saved in corner regions which are relevant for fast head movements. Notice, that the saved bandwidth in the foveal focus region is less optimal as the grid-based approach can as efficiently adapt to the scene.

The evaluation for Framework 2 is describing an average frame. Depending on the grid size and the gap between mip-map levels, it might be advisable to have a more conservative rendering in order to decrease possible artifacts. In this case, the algorithm can render multiple video tiles in the highest resolution when being close to edges of a grid cell. Using this, the bandwidth naturally increases. A worst case scenario using this approach results in a bandwidth of 7–8 percent. This is the case if the eye vector points directly on the edge of four adjacent grid cells. However, this is rare and in most directions a rendering of one or two tiles in the highest resolution is sufficient.

This greatly reconfirms results of other publications[4][29], which analyzed bandwidth savings to have a very similar order of magnitude.

	transferred	percentage
Sampling-based acuity (1080P acuity mask)	690926	2.08
minimum resolution (smallest mip-map)	129600	0.39
full resolution (8K wide-screen)	33177600	100
Grid-based acuity (with variance)	1773229	4.93
Grid-based acuity (without variance)	1043483	2.89
minimum resolution (smallest mip-map)	212064	0.59
full resolution (12K panoramic)	36000000	100

Table 5.3: Amount of pixels transferred: This table visualizes the amount of pixel data transferred by these approaches. The upper rows describes Framework 1, where all results are constant. The lower part describes Framework 2, where Grid size or the view direction can have a further impact on the results. Therefore, the table indicates average case, which is close to the best case. The worst case can increase the bandwidth of the grid-based approach to around 7–8 percent.

Video RAM The Framework 1 has a huge decrease in video RAM usage compared to a normal video rendering. A normal video surface would use a texture of the full video resolution. Compared to this, the highest texture used in this framework is the interpolation mask at a size of 1920×1920. This is obviously a rough approximation for interpolating a 8K area. However, it can be scaled accordingly and does not need to be very detailed for corner regions. Due to this, the memory usage is a mere 12 percent of the classic memory usage.

On the other side, the Framework 2 has a slight increase in video RAM usage. The use of multiple cube maps for reconstructing the frame within the shader results in a natural increase of memory usage. A normal panorama would only use one cube map. As the resolution of the lower mip-map levels is very decreased, the impact is however negligible. Thus, it reaches a maximum of a 7 percent increase compared to a classic cube map. A comprehensive

view on this data is shown in Table 5.4.

	data (MB)	percentage
Acuity mask	2.76	0.69
Interpolation mask	44	11.06
total used by approach	46.76	11.75
total classic video surface	398	100
Cube map (high resolution)	1152	100
Cube map (mid resolution)	72	6.25
Cube map (low resolution)	4.5	0.39
total used by approach	1228	106.64
total classic cube map	1152	100

Table 5.4: Amount of memory usage: This table visualizes the amount of texture data in use. This directly translates to the usage of video RAM. The upper part assumes a wide screen video with a size of 8K. The lower part assumes a panoramic video with a size of 16K.

Shading Despite it being hard to quantize the amount of shading power used by an algorithm, it is worth noting that no complex shading algorithms are used. Both frameworks fully rely on linear operations per fragment and combining multiple look up tables to successfully reconstruct the full frame.

This ensures, that there is no additional computation overhead added on the GPU. This leaves most GPU computing power unused and available for decoding if applicable. Furthermore, more advanced post-processing shaders are possible.

Naturally, the Gauss blur post-processing which is a possible optimization for the Framework 2 can have an impact on shading performance depending on the kernel size.

5.4 Performance

This section will evaluate the overall performance of the frameworks. After the analysis of actual transferred bandwidth in Section 5.3.3, this is the second most important data.

The results will show how smooth the implementation can render and stream a video. Based on the previous results, each framework is tested with the most promising set of options and codecs.

The overall goal is a performance higher than 25 frames per second (fps) for each scene. This would allow a real-time rendering for most videos, as 25 fps is the most common video format.

Notice, that the rendering is not capped to the actual speed of the video. Therefore, a faster rendering would result in the video playing faster than

recorded. This behavior is implemented purely to analyze the performance capabilities of each system and can be changed easily.

5.4.1 Framework for wide-screen videos

There were no usable video data available in 8K or 16K formats. Therefore, the wide-screen scene Caminandes is upscaled accordingly for these measurements.

The VP 9 encoding has the most promising result, as it suffers the fewest amount of issues with the multi-threaded model and overall algorithm design. Unfortunately, h.264 and h.264 had various issues with seeking, which were not fully resolvable. Thus, they are excluded from these overall analyses.

The framework can perform easily on real-time videos, as shown in Figure 5.7. For both the 8K and 16K scene it has a smooth performance of around 60 to 40 fps respectively. This means, it is fast enough to even have idle times when processing 25 fps video. This allows catching up of eye movements and other changes to the view port. Furthermore, videos in 60 fps are almost playable in real-time for 8K videos.

Note, that there might be small delays and lag spikes when moving the eye position. Every eye movement results in a sudden queue of video files to seek, which the algorithm tries to handle it efficiently by relying on old resolution levels in parallel to new frames being loaded. This can result in a higher workload for the CPU to handle shortly after an eye movement.

The approach seems to scale reasonable, making it well suited for higher resolutions in newer hardware. For VP 9 videos, as the error bars in 5.5 suggest, stuttering effects are limited and it can maintain steadily maintain 25 fps during slowdowns

5.4.2 Framework for panoramic videos

This framework can render panoramic videos in real time, as shown in Figure 5.8. Using the acuity approach, both scenes can easily reach 29 and 31 fps respectively. In contrast, a render of the full resolution has less than 1 fps. It results in a frictionless rendering of a 360-degree surface around the viewer. Thus, it provides a good example for the capabilities for panoramic videos within a HMD system.

For providing a fast loading speed, only JPEG-based data is used. This is a performance trade-off. For these tests, a high quality setting of $q = 90$ has been used. The switch from PNG to JPEG has no noticeable visual differences.

The results are post-processed with a radial blur filter to smooth out possible rendering issues. This radial blur filter renders faster than the underlying video loading process, so it does not influence the overall performance.

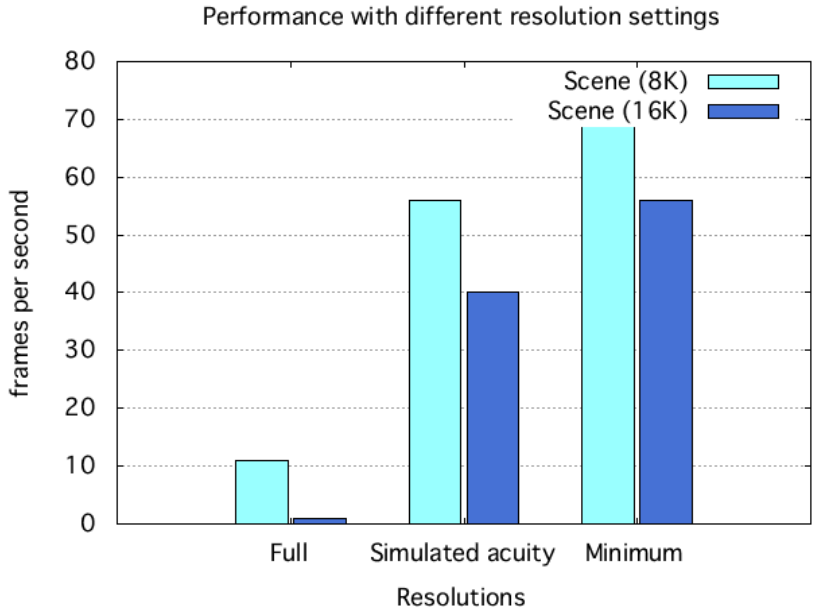

Figure 5.7: Performance for wide-screen videos: This graph shows the performance using different mip-map settings for the wide-screen scenes. *Full* references to a full resolution of the cube-map. *Minimum* is a render of the smallest mip-map level available. It corresponds to a rendering in 540p (for 8K) or 1080p (for 16K) using the same approach. This is results in a very low resolution and can render over 100 frames per second depending on the scene. Thus, it is not fully displayed within the chart. *Simulated acuity* uses the acuity mask and gaze-contingent rendering to increase performance with a minimum impact on perception. Its perceived resolution should equal *full*.

5.5 Perception

The current implementation unfortunately suffers from various small rendering issues which can have a slight impact on the viewers' perception. An example of possible artifacts is shown in figure 5.9.

Framework 1 Framework 1 utilizing sampling-based acuity has typical errors due to the nature of sampling. As described by the Nyquist theorem[65], such errors are only avoidable for a sampling frequency higher than double the information frequency. However, in a radial decreasing function, it is unavoidable to reach this at some point.

While the interpolation tries to fill the gaps with reasonable data, the static approach can not fully fix sampling issues. This can result in flickering noticing from the corner of the eye. An additional post-processing step with an additional gauss blur helps in terms of reducing the perception of an

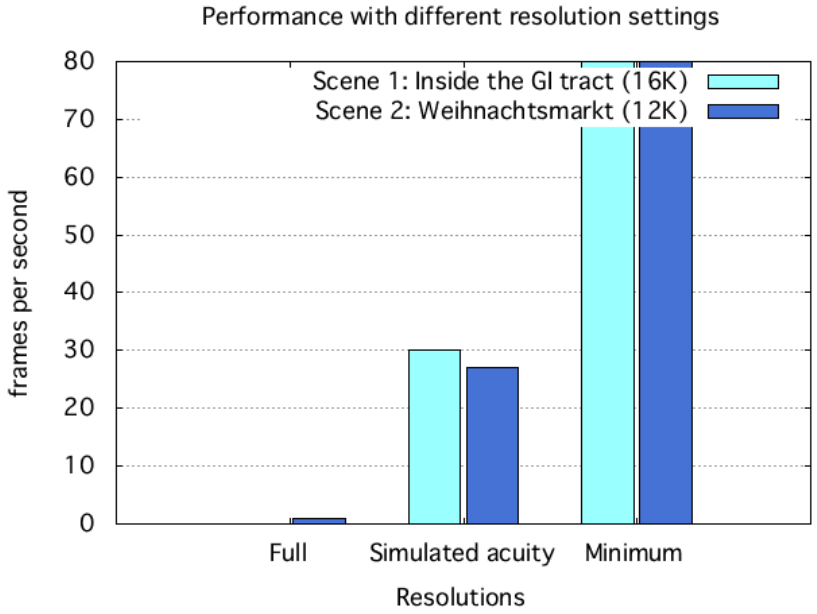

Figure 5.8: Performance for panoramic videos: This graph shows the performance using different mip-map settings for both panoramic scenes. *Full* references to a full resolution of the cube-map. *Minimum* is a render of the smallest mip-map level available. This results in a very low resolution and can render at up to 150 frames per second depending on the scene. Thus, it is not fully displayed within the chart. These show the minimum and maximum performance for each scene, and the gaze-contingent rendering will be anywhere between both limits. *Simulated acuity* uses gaze-contingent rendering to increase performance with a minimum impact on perception. Its perceived resolution should equal *full*.

artifact.

Framework 2 Framework 2 fully utilizes grid-based approaches for streaming the data. This can unfortunately result in hard edges on the border of two grid cells. This is not avoidable, as the data is pre-processed and the edge can not dynamically move with the movements of the eye. In the worst case situations, that means, the eye view vector is close to a grid edge, this is an issue. It can lead to a perceivable quality gap.

This can be adjusted by loading additional tiles in a higher resolution. However, this is a trade-off between performance and quality, which needs to be adjusted on the scene and the viewers' preferences.

Another option is using a higher resoluted grid like 8×8 or 16×16, or the usage of more mip-map levels. For this framework, there is no additional

Figure 5.9: Rendering artifacts: These pictures show possible rendering artifacts for both frameworks. Framework 1 (left) has flickering edges due to the interpolation mask. Framework 2 (right) can introduce hard edges relatinng to the transition between video tiles of different resolutions.

computational overhead due to more mip-map levels, as there is no need for video seeking when using single image files. Note, that mip-map levels will still have an impact on the video RAM and of course the pre-processing step in terms of storage and runtime.

Additionally, a radial gauss blur in a post-processing step smoothes out edges and makes the scene more aesthetically appealing. It is configured to approximate the acuity function.

Variance approach The current version of variance does have a minor impact on the perception. Some complex regions of the video are given a higher resolution, which results in a slightly lower amount of artifacts. However, in RGB color space this approach does not seem to be overly reliable, as there are other regions ought to be affected by this, which the algorithm does not detect successfully.

As discussed earlier in Section 3.1.3, other color spaces especially developed to model human perception like HSV or LAB as well as approaches using detection of meaningful edges, might further enhance this idea. However, this was not further evaluated and is left open for future work.

Chapter 6

Discussion

This chapter will thoroughly discuss the result of previous design and implementations. Based on Chapter 5, a great insight on the overall performance as well as different settings for each framework are known. However, each approach has their individual drawbacks and limitations. In Section 6.1, promising ideas and untested approaches for future work are presented.

Framework 1 Due to limitations in video and picture access, it is not viable to develop a solely sampling-based approach. However, the wide-screen video framework shows the capabilities of an efficient hybrid approach involving both sampling-based pixel access and a grid-based video loading.

Despite the sampling approach not being suited well for video data, an efficient multi-threaded loading scheme based on pre-processed video tiles was built for wide-screen videos. The data was processed in a way suiting the infrastructure of GLSL shading language. This results in an efficient shading with no added overhead through sample interpolation.

The bandwidth reduction is around 98 percent and constant, as it is fully based on the mask sizes. However, the masks can be changed easily and thus the algorithm is able to adapt to different bandwidth requirements.

Framework 2 The second approach involving panoramic videos showed capabilities of a full implementation based on grid-based approaches.

Furthermore, a sophisticated framework for 360-degree videos was implemented. It utilizes a dynamic visual acuity function and a picture loader. Only visible mip-map tiles were transferred to GPU to minimize bandwidth without sampling. Video data was then combined on GPU based on previously determined acuity matrices. To limit rendering artifacts and give a stronger impression of visual acuity, radial blurs turned out to be a cheap and efficient optimization.

Based on this, a panoramic video with a real resolution of 12000×3000 pixels could be rendered in real time. The bandwidth is reduced tremendously,

which makes image loading a new major bottleneck. Using the grid acuity approach, the reduction of bandwidth averages around 97 percent, but still stays over 92 percent in a worst case scenario. When introducing saliency-based techniques like using color variance, this number may increase.

Tile size and mip-map levels During all evaluations except one, a solid grid size of 4×4 and three mip-map levels were used. For the 16K wide-screen video a 8×8 format was used. Prior to evaluations, there were various tests using different grid sizes. Unfortunately, the length of the pre-processing, especially for the VP 9 codec which is mostly used, makes thorough testing stressful.

For Framework 1, a very high grid size has a major impact on the performance. While it would be possible that a higher resolution grid is always better as the gaze-contingent rendering can adapt more easily for higher grid sizes, this does not hold true. The amount of grid cells directly corresponds to the amount of open video tiles multiplied by the amount of mip-map levels. Thus, additional grid cells introduce a higher amount of stuttering and can quickly over-exert the CPU with decoding processes. This can introduce various stuttering effects for larger grid sizes.

When using three mip-map levels, a 8×8 grid already has 192 open video files while a 4×4 grid only has 48. Each video stays open for a reasonable time as it is more efficient in terms of seeking. Furthermore, for every eye movement 16 (for 4×4) or 64 (for 8×8) video tiles need to seek. This results in the same amount of parallel decoding processes for each frame. Due to this, it can often be faster to use a smaller resolution grid, even if it can not adapt as well to the gaze-contingent rendering approach.

Framework 2 has less impact by a higher grid size. When loading single picture files instead of full videos, there are no issues regarding seeking video material and providing the correct frame. However, there is still an impact as there is a higher amount of video tiles to be decoded. For a higher resolution grid, a single video tile has a lower resolution. However, the decoding process does not necessarily scale linear to resolution in terms of performance. Thus, there is a sweet spot when it comes to balancing the grid size for each format. In cube maps, every cube side is exactly quadratic. This means, an identical width and height per cube side is reasonable.

On the other side, a very low grid size is also unfavorable. For a 2×2 grid, it would not be possible to use gaze-contingent rendering. As an example, a view into the middle of the video (where all 4 cells of a 2×2 grid intersect) would need to render a full resolution frame and thus making the approach ineffective.

For mip-map levels, there were also design decisions to be made. Obviously, there is a need of at least two levels. While two levels can already simulate the fall-off between center vision and the peripheral vision quite closely, it

is simply not necessary to load the full resolution for the full central vision region. Due to the linear fall-off[47] along the central vision, it is possible to add further sub levels.

The choice of three mip-map levels for the evaluation directly corresponds to the grid size of 4×4. For a 4×4 grid there is not enough room to exploit more than three mip-map levels. Previous tests with up to six mip-map levels have shown, that three is a very efficient size.

Within the focal point and roughly 5–10 degrees around it, the highest level is always preferable. Furthermore, the peripheral region does not need further sub levels and can always be rendered within the lowest level available. For all zones in between, usually one further mip-map level seems sufficient. However, to smooth out hard edges and possibly other artifacts, introducing a forth or fifth mip-map level and a slightly higher grid size could overall improve perception with a minor impact in performance.

Perception This approach could show the great capabilities both in terms of performance and bandwidth reduction. The human perception has been judged subjectively, as no extensive user study has been performed.

Depending on the used video format, the algorithm has almost no stuttering during movements. This is a great success, as stuttering could harm the perception[42].

Unfortunately, there are various artifacts and sampling errors which can result in flickering effects and strong perceivable edges within the peripheral region, as shown in Section 5.5.

As perception is strongly user-dependent[81], it is not viable to simply judge the results in terms of perception without a user study. A perceptual study is necessary to successfully conclude the overall perceptual results of this approach.

Afterthoughts A major disadvantage of this approach is the pre-processing step. A playback of new videos is not possible without at least creating the intermediate data format giving access to both a uniform grid and mip-mapped data. This creates a vast overhead of storage, which is necessary to access video data in chunks. Without this, every frame would need to be decoded in full resolution. This would create an infeasible load on the CPU, which would render real-time performance on current hardware impossible. In terms of performance, the pre-processing is a one-time process, making it negligible for the user.

Both the sampling-based and the grid-based approaches for factoring in visual acuity has been tested exhaustively, in form of hybrid approaches. Variance-based approaches were designed and tested lightly. A determinant based on RGB color space variations could have slight improvements on the quality of mip-map selections.

6.1 Further ideas

The approach incorporates a broad view of options available to solving actuiy-based video streaming. Of course, not every idea and design decision could be tested exhaustively. Therefore, the following gives a summary on open questions for this topic.

Storage optimization To reduce storage usage, approaches from video games to save texture space could be applied. When saving the data in a more compressed format, the storage will decrease. However, this will likely be a performance trade-off, as higher compression or more sophisticated data-packing usually also increases complexity. As the focus of the thesis has been real-time performance, this has not been further investigated.

Video and image formats The evaluations are all based on standard video and picture formats. Currently, there are no promising formats better suited for this usage. H.265 seems to be a step in the right direction and future formats building on it might have functionality which help for this approach. However, they are currently closed formats and are thus not well implemented in freely available libraries and frameworks commonly used in research.

In order to decrease the amount of files, a container format can be added. This makes file handling and pre-processing more convenient, but has no promising effect on the overall performance.

For video games, special picture formats for fast loading of high resolution textures are in development. In current evaluations, there was no format available which could beat standard JPG in $\frac{loading\ speed}{storage}$-ratio. TGA and TIFF are still used in modern video game engines, but seemed to be not well supported by most libraries. These among other rare formats can be evaluated more extensively.

Another option are graphics cards formats like KTX[28] for OpenGL. The current approach uncompresses other formats like JPEG or various video codecs to work on uncompressed raw data. The KTX format is a compressed texture format that already corresponds to the way a texture is saved within video RAM. Thus, the format can be loaded and streamed more efficiently, as it could skip certain parts of decompression and re-compression within the algorithm. The corresponding DirectX format is called DDS.

Video loading The video loading process is solely based on CPU and accelerated by SIMD as the API of OpenCV, FFmpeg and STB Image already provides. Moving more parts of the algorithm to the GPU could enhance performance. This will increase bandwidth, as the full video data needs to be moved to GPU to start the decoding process. However, it might be a

fraction of the full raw frame data, as the encoded frame is obviously better compressed. In the current approach, the GPU has no major workload despite the linear reconstruction shading and some post-processing like the radial blur.

Perception and saliency An additional focus on variance-based approaches could improve the perceptual results. Using knowledge about significant edges and factoring in other color spaces, similar to results from [76][6], could add robustness. This will however drastically increase data pre-processing step and result in a much more complex algorithm.

As discussed earlier in Section 3.2, ray tracing use knowledge of scene geometry to closer adapt data structures like kD trees[9] or BVH trees[62][39] to the scene. Similarly, a Grid could use saliency-based approaches to more closely adapt to the actual video data. For example, for regions with no derivations less grid cells might suffice, while other regions might need a very high resolution grid.

Lastly, the frame rate presumably does not need to be equal for every video tile. The current approach has effectively the same frame rate for every video tile. However, the Framework 1 already limits this by not guaranteeing frame correct mip-map level changes due to the multi-threaded nature of the implementation. Similarly, frames in the peripheral region could simply just update less often than within the central vision region.

6.2 Other applications

The current approach can reduce the bandwidth by up to 97 percent. This creates great possibilities for various applications. The main motivation are consumer setups for virtual reality, as soon available with Oculus Rift[19]. However, other applications are thinkable.

Due to the very low bandwidth, remote applications are possible. Pupil detection is possible even on mobile and embedded devices[38]. The algorithm needs just (x, y) input data of the current eye position to process a frame. When processed, it has a byte array of raw video data to be send to a GPU for reconstruction.

However, this does not necessarily need to be on the same computer. For example, a mobile device could detect pupil movements, upload them to a server which has an implementation of the algorithm and raw video data. The server can then process a frame and just send back the foveated image. While the bandwidth was reduced to make GPU streaming more efficient, this also results in a very small network bandwidth for downloading the image data back to the mobile device.

Thus, when encapsulating the pupil detection and rendering from the video loading, this will open the door for various kinds of remote and mobile

applications.

While client and server approaches within the same room or house are already a great application, the reduced bandwidth can even help with streaming the data over the internet. Various cloud gaming provider like OnLive[57] (though suspended and acquired by Sony Entertainment) already proved the capabilities of streaming interactive media through the internet. Combining these technologies with a HMD system and this algorithm, such a service could easily be implemented in a much higher resolution using gaze-contingent rendering.

Chapter 7

Conclusion

First, Chapter 3 compared various ideas based on different literature and other existing approaches (discussed in Chapter 2) in terms of whether they fit gaze-contingent rendering. An approach for both sampling-based and grid-based visual acuity was in the focus. Furthermore, a video format for loading high resolution data and some techniques for texture streaming with high efficiency were developed.

Incorporating these approaches to software design, Chapter 4 uses this knowledge to efficiently implement two different frameworks: one for wide-screen videos and one for 360-degree panoramic videos. Each framework uses a variety of different design ideas in order to gain a good grasp on different decisions. A focus has been the decrease of bandwidth used for texture streaming.

The analysis of Chapter 5 shows great capabilities of both frameworks. Each implementation was easily able to play videos in up to 30 frames per second in resolutions up to 16K. This means it is capable of sustaining a real-time rendering for common 25 frames per second video material. The wide-screen video framework could even reach 60 frames per second for 8K video material. The evaluation successfully shows, that the approach can scale well enough to smoothly render high resolution media like 16K panoramic videos.

Furthermore, the bandwidth could be decreased by up to 97%. This confirms the results of other publications and shows the potential of the underlying acuity approach. There is an increase of only 6% video RAM for the panoramic videos. In terms of shading, both approaches use only linear operations within the fragment shader to successfully reconstruct the frame. This means, the graphics card is not working to its capacity and can be used for other things like post-processing. There are some drawbacks and limitations to the system and the overall state is thoroughly discussed in Chapter 6. Furthermore, there are a variety of ideas and other applications which can be build upon this thesis, as discussed in Section 6.1.

Bibliography

[1] Microsoft Virtual Academy. *Massive Virtual Textures for Games: Direct3D Tiled Resources*. https://channel9.msdn.com/Events/Build/2013/4-063. June 2013.

[2] G. Agarwal, A. Anbu, and A. Sinha. "A fast algorithm to find the region-of-interest in the compressed MPEG domain". In: *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*. Vol. 2. July 2003. DOI: 10.1109/ICME.2003.1221571.

[3] P. Agre and D. Schuler. *Reinventing Technology, Rediscovering Community: Critical Explorations of Computing as a Social Practice*. Folktales of Their People]. Ablex Pub., 1997. ISBN: 9781567502589. URL: https://books.google.de/books?id=LwTYjYH-GNwC.

[4] S. Anstis. "Picturing peripheral acuity". In: *Perception* 27.7 (1998), pp. 817–825.

[5] Arthur Appel. "Some Techniques for Shading Machine Renderings of Solids". In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). Atlantic City, New Jersey: ACM, 1968, pp. 37–45. DOI: 10.1145/1468075.1468082. URL: http://doi.acm.org/10.1145/1468075.1468082.

[6] Tunç Ozan Aydin et al. "Visually Significant Edges". In: *ACM Trans. Appl. Percept.* 7.4 (July 2010), 27:1–27:15. ISSN: 1544-3558. DOI: 10.1145/1823738.1823745. URL: http://doi.acm.org/10.1145/1823738.1823745.

[7] Sean Barrett. *Sean's Tool Box*. https://github.com/nothings/stb. Feb. 2016.

[8] David Benson and Joel Davis. "Octree Textures". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. San Antonio, Texas: ACM, 2002, pp. 785–790. ISBN: 1581135211. DOI: 10.1145/566570.566652. URL: http://doi.acm.org/10.1145/566570.566652.

[9] Jon Louis Bentley. "Multidimensional binary search trees used for associative searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: http://doi.acm.org/10.1145/361002.361007.

[10] Mark R Bolin and Gary W Meyer. "A perceptually based adaptive sampling algorithm". In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM. 1998, pp. 299–309.

[11] Hamilton Y. Chong, Steven J. Gortler, and Todd Zickler. "A Perception-based Color Space for Illumination-invariant Image Processing". In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 61:1–61:7. ISSN: 0730-0301. DOI: 10.1145/1360612.1360660. URL: http://doi.acm.org/10.1145/1360612.1360660.

[12] Alex Clark. *Pillow*. https://github.com/python-pillow/Pillow. Feb. 2016.

[13] Nitin Dhavale and Lativent Itti. "Saliency-based multifoveated MPEG compression". In: *Signal processing and its applications, 2003. Proceedings. Seventh international symposium on*. Vol. 1. IEEE. 2003, pp. 229–232.

[14] Piotr Didyk et al. "Apparent display resolution enhancement for moving images". In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM. 2010, p. 113.

[15] Andrew T Duchowski. "A breadth-first survey of eye-tracking applications". In: *Behavior Research Methods, Instruments, & Computers* 34.4 (2002), pp. 455–470.

[16] Andrew T Duchowski, Nathan Cournia, and Hunter Murphy. "Gaze-contingent displays: A review". In: *CyberPsychology & Behavior* 7.6 (2004), pp. 621–634.

[17] Asieh Ehsaei et al. "Visual performance fall-off with eccentricity in myopes versus emmetropes". English. In: *Journal of Optometry* 6.1 (2013), pp. 36–44. DOI: 10.1016/j.optom.2012.07.001.

[18] J.P. Ewins et al. "MIP-map level selection for texture mapping". In: *Visualization and Computer Graphics, IEEE Transactions on* 4.4 (Oct. 1998), pp. 317–329. ISSN: 1077-2626. DOI: 10.1109/2945.765326.

[19] Facebook. *Oculus*. https://www.oculus.com/en-us/. Jan. 2016.

[20] Steven Feiner, Blair Macintyre, and Dorée Seligmann. "Knowledge-based augmented reality". In: *Communications of the ACM* 36.7 (1993), pp. 53–62.

[21] FFmpeg. *FFmpeg*. https://www.ffmpeg.org/. Feb. 2016.

[22] Blender Foundation. *Caminandes*. http://www.caminandes.com/. Feb. 2013.

[23] A. Fujimoto, T. Tanaka, and K. Iwata. "ARTS: Accelerated Ray-Tracing System". In: *Computer Graphics and Applications, IEEE* 6.4 (Apr. 1986), pp. 16–26. ISSN: 0272-1716. DOI: 10.1109/MCG.1986.276715.

[24] GoPro. *GoPro Official Website - Capture + share your world.* https://gopro.com/. Jan. 2016.

[25] N. Greene. "Environment Mapping and Other Applications of World Projections". In: *Computer Graphics and Applications, IEEE* 6.11 (Nov. 1986), pp. 21–29. ISSN: 0272-1716. DOI: 10.1109/MCG.1986.276658.

[26] J. E. Greivenkamp et al. "Visual acuity modeling using optical raytracing of schematic eyes". In: *Am. J. Ophthalmol.* 120.2 (Aug. 1995), pp. 227–240.

[27] Steve Grogorick. "Binokulares Eye-Tracking fuer Linsenbasierte Head-Mounted Displays". Master's Thesis. Germany: TU Braunschweig, Mar. 2015.

[28] Khronos Group. *The KTX File Format and Tools.* https://www.khronos.org/opengles/sdk/tools/KTX/. May 2013.

[29] Brian Guenter et al. "Foveated 3D Graphics". In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 164:1–164:10. ISSN: 0730-0301. DOI: 10.1145/2366145.2366183. URL: http://doi.acm.org/10.1145/2366145.2366183.

[30] Taosong He et al. "Voxel Based Object Simplification". In: *Proceedings of the 6th Conference on Visualization '95.* VIS '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 296–. ISBN: 0818671874. URL: http://dl.acm.org/citation.cfm?id=832271.833850.

[31] S Heymann et al. "Representation, coding and interactive rendering of high-resolution panoramic images and video using MPEG-4". In: *Proc. Panoramic Photogrammetry Workshop (PPW).* 2005.

[32] Yiqun Hu, Liang-Tien Chia, and Deepu Rajan. "Region-of-interest based image resolution adaptation for mpeg-21 digital item". In: *Proceedings of the 12th annual ACM international conference on Multimedia.* ACM. 2004, pp. 340–343.

[33] Tobias Hüttner. "High resolution textures". In: *Visualization'98-Late Breaking Hot Topics Papers* (1998), pp. 13–17.

[34] HybridMedical. *360° VR | inside the GI tract.* https://www.youtube.com/watch?v=upSnH436ya8. Sept. 2015.

[35] Itseez. *OpenCV.* http://opencv.org/. Feb. 2016.

[36] J. H. Jamar, L. F. Kwakman, and J. J. Koenderink. "The sensitivity of the peripheral visual system to amplitude-modulation and frequency-modulation of sine-wave patterns". In: *Vision Res.* 24.3 (1984), pp. 243–249.

[37] JeffPerry. *Foveated Imaging*. https://en.wikipedia.org/wiki/Foveated_imaging, Public domain. June 2008.

[38] Marc Aurel Kastner. "Robust binocular pupil detection for a mobile computing platform". Project Thesis. Germany: TU Braunschweig, Sept. 2015.

[39] Timothy L. Kay and James T. Kajiya. "Ray tracing complex scenes". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 269–278. ISSN: 0097-8930. DOI: 10.1145/15886.15916. URL: http://doi.acm.org/10.1145/15886.15916.

[40] Joyce L. Kerr. "Visual resolution in the periphery". In: *Perception & Psychophysics* 9.3 (May 1971), pp. 375–378. ISSN: 1532-5962. DOI: 10.3758/BF03212671. URL: http://dx.doi.org/10.3758/BF03212671.

[41] Han Suk Kim and J.P. Schulze. "High Resolution Video Playback in Immersive Virtual Environments". In: *Virtual Reality Conference, 2009. VR 2009. IEEE*. Mar. 2009, pp. 247–248. DOI: 10.1109/VR.2009.4811038.

[42] E.M. Kolasinski, U.S. Army Research Institute for the Behavioral, and Social Sciences. *Simulator sickness in virtual environments*. Technical report (U.S. Army Research Institute for the Behavioral and Social Sciences). U.S. Army Research Institute for the Behavioral and Social Sciences, 1995. URL: https://books.google.de/books?id=7qwrAAAAYAAJ.

[43] Philip Kortum and Wilson S. Geisler. *Implementation of a foveated image coding system for image bandwidth reduction*. 1996. DOI: 10.1117/12.238732. URL: http://dx.doi.org/10.1117/12.238732.

[44] Sam Lantinga. *Simple Directmedia Layer*. https://www.libsdl.org/. Feb. 2016.

[45] Joseph J. LaViola Jr. "A Discussion of Cybersickness in Virtual Environments". In: *SIGCHI Bull.* 32.1 (Jan. 2000), pp. 47–56. ISSN: 0736-6906. DOI: 10.1145/333329.333344. URL: http://doi.acm.org/10.1145/333329.333344.

[46] Sanghoon Lee, Marios S Pattichis, and Alan Conrad Bovik. "Foveated video compression with optimal rate control". In: *Image Processing, IEEE Transactions on* 10.7 (2001), pp. 977–992.

[47] Dennis M Levi, Stanley A Klein, and AP Aitsebaomo. "Vernier acuity, crowding and cortical magnification". In: *Vision research* 25.7 (1985), pp. 963–977.

[48] Marc Levoy and Ross Whitaker. "Gaze-directed volume rendering". In: *ACM SIGGRAPH Computer Graphics* 24.2 (1990), pp. 217–223.

[49] J.J.-W. Lin et al. "Effects of field of view on presence, enjoyment, memory, and simulator sickness in a virtual environment". In: *Virtual Reality, 2002. Proceedings. IEEE.* 2002, pp. 164–171. DOI: 10.1109/VR.2002.996519.

[50] Gerard Llorach, Alun Evans, and Josep Blat. "Simulator Sickness and Presence Using HMDs: Comparing Use of a Game Controller and a Position Estimation System". In: *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology.* VRST '14. Edinburgh, Scotland: ACM, 2014, pp. 137–140. ISBN: 9781450332538. DOI: 10.1145/2671015.2671120. URL: http://doi.acm.org/10.1145/2671015.2671120.

[51] Peter Longhurst, Kurt Debattista, and Alan Chalmers. "A GPU based saliency map for high-fidelity selective rendering". In: *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa.* ACM. 2006, pp. 21–29.

[52] Robert Gabriel Lupu and Florina Ungureanu. "A survey of eye tracking methods and applications". In: *Bul Inst Polit Iaşi* (2013), pp. 71–86.

[53] Nenad Markuš et al. "Eye pupil localization with an ensemble of randomized trees". In: *Pattern Recognition* 47.2 (2014), pp. 578–587. ISSN: 0031-3203. DOI: http://dx.doi.org/10.1016/j.patcog.2013.08.008. URL: http://www.sciencedirect.com/science/article/pii/S0031320313003294.

[54] Tomasz Mazuryk and Michael Gervautz. *Virtual Reality - History, Applications, Technology and Future.* 1996.

[55] Don P. Mitchell. "The antialiasing problem in ray tracing". In: (1990). DOI: 10.1.1.587.6122.

[56] SARAH NICHOLS, CLOVISSA HALDANE, and JOHN R. WILSON. "Measurement of presence and its consequences in virtual environments". In: *International Journal of Human-Computer Studies* 52.3 (2000), pp. 471–491. ISSN: 1071-5819. DOI: http://dx.doi.org/10.1006/ijhc.1999.0343. URL: http://www.sciencedirect.com/science/article/pii/S1071581999903439.

[57] OnLive. *OnLive Games on Demand.* http://onlive.com/. Feb. 2016.

[58] OpenCV. *Bug Tracker: Erroneous frame seek in VideoCapture::set.* https://github.com/Itseez/opencv/issues/4890. Dec. 2014.

[59] V. Ponz et al. "Topography-Based Detection of the Iris Centre Using Multiple-Resolution Images". In: *Machine Vision and Image Processing Conference (IMVIP), 2011 Irish.* Sept. 2011, pp. 32–37. DOI: 10.1109/IMVIP.2011.15.

[60] James Reason and J. J. Brand. *Motion sickness.* English. Academic Press London ; New York, 1975, xi, 310 p. : ISBN: 0125840500.

[61] Eyal M. Reingold et al. "Gaze-contingent multiresolutional displays: An integrative review". In: *HFES Journal* 45.2 (2003), pp. 307–328.

[62] Steven M. Rubin and Turner Whitted. "A 3-dimensional representation for fast rendering of complex scenes". In: *SIGGRAPH Comput. Graph.* 14.3 (July 1980), pp. 110–116. ISSN: 0097-8930. DOI: 10.1145/965105.807479. URL: http://doi.acm.org/10.1145/965105.807479.

[63] Alexander C. Schütz, Doris I. Braun, and Karl R. Gegenfurtner. "Eye movements and perception: A selective review". In: *Journal of Vision* 11.5 (2011), p. 9. DOI: 10.1167/11.5.9. eprint: /data/Journals/JOV/933487/jov-11-5-9.pdf. URL: +%20http://dx.doi.org/10.1167/11.5.9.

[64] Graham Sellers et al. "Rendering Massive Virtual Worlds". In: *ACM SIGGRAPH 2013 Courses*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 23:1–23:88. ISBN: 9781450323390. DOI: 10.1145/2504435.2504458. URL: http://doi.acm.org/10.1145/2504435.2504458.

[65] C.E. Shannon. "Communication In The Presence Of Noise". In: *Proceedings of the IEEE* 86.2 (Feb. 1998), pp. 447–457. ISSN: 0018-9219. DOI: 10.1109/JPROC.1998.659497.

[66] A. Shroff. *An Eye on Numbers: A Ready Reckoner in Ophthalmology*. Postscript Media Pvt Ltd, 2011. ISBN: 9788192112312. URL: https://books.google.de/books?id=mngaB6jFVS0C.

[67] L. Spillmann and J. S. Werner. *Visual Perception: The Neurophysiological Foundations*. Elsevier Science, 2012. ISBN: 9780323138147. URL: https://books.google.de/books?id=zJp6s%5C_YkqpwC.

[68] Michael Stengel et al. "An Affordable Solution for Binocular Eye Tracking and Calibration in Head-mounted Displays". In: *Proc. ACM Multimedia 2015*. Won the 'Best Student Paper Award'. Oct. 2015, pp. 15–24.

[69] Robert J. Stone. "Haptic Human-Computer Interaction: First International Workshop Glasgow, UK, August 31 — September 1, 2000 Proceedings". In: ed. by Stephen Brewster and Roderick Murray-Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. Chap. Haptic feedback: a brief history from telepresence to virtual reality, pp. 1–16. ISBN: 9783540445890. DOI: 10.1007/3-540-44589-7_1. URL: http://dx.doi.org/10.1007/3-540-44589-7_1.

[70] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. "Peripheral vision and pattern recognition: A review". In: *Journal of Vision* 11.5 (2011), p. 13. DOI: 10.1167/11.5.13. eprint: /data/Journals/JOV/933487/jov-11-5-13.pdf. URL: +%20http://dx.doi.org/10.1167/11.5.13.

[71] Veronica Sundstedt et al. "Visual attention for efficient high-fidelity graphics". In: *Proceedings of the 21st spring conference on Computer graphics*. ACM. 2005, pp. 169–175.

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

[73] Ivan E Sutherland. "The ultimate display". In: *Multimedia: From Wagner to virtual reality* (1965).

[74] M. Tkalcic and J.F. Tasic. "Colour spaces: perceptual, historical and applicational background". In: *EUROCON 2003. Computer as a Tool. The IEEE Region 8*. Vol. 1. Sept. 2003, 304–308 vol.1. DOI: 10.1109/EURCON.2003.1248032.

[75] Tschmits. *The application of a texture in the UV space related to the effect in 3D*. https://en.wikipedia.org/wiki/UV_mapping, CC-BY-SA-3.0. Jan. 2008.

[76] Chun-Shun Tseng et al. "Significant edge finding in view of human perception". In: *System Integration (SII), 2011 IEEE/SICE International Symposium on*. Dec. 2011, pp. 84–89. DOI: 10.1109/SII.2011.6147424.

[77] Eric Veach and Leonidas J. Guibas. "Optimally Combining Sampling Techniques for Monte Carlo Rendering". In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '95. New York, NY, USA: ACM, 1995, pp. 419–428. ISBN: 0897917014. DOI: 10.1145/218380.218498. URL: http://doi.acm.org/10.1145/218380.218498.

[78] V. Virsu and J. Rovamo. "Visual resolution, contrast sensitivity, and the cortical magnification factor". In: *Exp Brain Res* 37.3 (1979), pp. 475–494.

[79] Zhou Wang, Ligang Lu, and A.C. Bovik. "Foveation scalable video coding with automatic fixation selection". In: *Image Processing, IEEE Transactions on* 12.2 (Feb. 2003), pp. 243–254. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.809015.

[80] T. Wertheim and T. Wertheim. "Peripheral visual acuity: Th. Wertheim". In: *Am J Optom Physiol Opt* 57.12 (Dec. 1980), pp. 915–924.

[81] Gerald Westheimer and Suzanne P McKee. "Stereoscopic acuity for moving retinal images". In: *JOSA* 68.4 (1978), pp. 450–455.

[82] Frank W Weymouth. "Visual Sensory Units And The Minimum Angle Of Resolution". In: *Optometry & Vision Science* 40.9 (1963), pp. 550–568.

[83] Wikipedia. *Google Glass*. https://en.wikipedia.org/wiki/Google_Glass. Feb. 2016.

[84] Lance Williams. "Pyramidal Parametrics". In: *SIGGRAPH Comput. Graph.* 17.3 (July 1983), pp. 1–11. ISSN: 0097-8930. DOI: 10.1145/964967.801126. URL: http://doi.acm.org/10.1145/964967.801126.

[85] Joanna Wiśniewska, Mahdi Rezaei, and Reinhard Klette. "Computer Vision and Graphics: International Conference, ICCVG 2014, Warsaw, Poland, September 15-17, 2014. Proceedings". In: ed. by Leszek J. Chmielewski et al. Cham: Springer International Publishing, 2014. Chap. Robust Eye Gaze Estimation, pp. 636–644. ISBN: 9783319113319. DOI: 10.1007/978-3-319-11331-9_76. URL: http://dx.doi.org/10.1007/978-3-319-11331-9_76.

[86] Zephyris. *Representation of the UV mapping of a cube*. https://en.wikipedia.org/wiki/UV_mapping, CC-BY-SA-3.0. Feb. 2016.