Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Paula Jukarainen

Comparison of Real-Time Anti-Aliasing Methods on a Head-Mounted Display

Master's Thesis
Espoo, October 10, 2016

Supervisor:	Jaakko Lehtinen, D.Sc. (Tech.), Professor, Aalto University School of Science
Advisor:	Juha Sjöholm, M.Sc. (Tech.)

Aalto University
School of Science

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author:	Paula Jukarainen		
Title: Comparison of Real-Time Anti-Aliasing Methods on a Head-Mounted Display			
Date:	October 10, 2016	**Pages:**	vii + 82
Major:	Media Technology	**Code:**	T-111
Supervisor:	Jaakko Lehtinen, D.Sc. (Tech.)		
Advisor:	Juha Sjöholm, M.Sc. (Tech.)		

Virtual reality and head-mounted devices have gained popularity in the past few years. Their increased field-of-view combined with a display that is near to the eyes have increased the importance of anti-aliasing i.e. softening of the visible jagged edges resulting from insufficient rendering resolution.

In this thesis, elementary theory of real-time rendering, anti-aliasing and virtual reality is studied. Based on the theory and review of recent studies, multisample anti-aliasing (MSAA), fast-approximate anti-aliasing (FXAA) and temporal anti-aliasing (TAA) were implemented into a real-time deferred rendering engine and the different techniques were studied in both subjective image quality and objective performance measures. In the scope of this thesis, only each methods' ability to prevent or lessen jagged edges and small flickering detailed geometries is examined.

Performance was measured on two different machines; the FXAA implementation was found to be the fastest with 3% impact on performance and required the least memory, the TAA performance impact was 10-11% and 22% to 62% for MSAA was depending on the sample count.

Each techniques' ability to prevent or reduce aliasing was examined by measuring the visual quality and fatigue reported by participants. Each anti-aliasing method was presented in a 3D scene using Oculus Rift CV1.

The results indicate that the 4xMSAA and 2xMSAA had clearly the best visual quality and made participants the least fatigued. FXAA appears visually not as good, but did not cause significant fatigue. TAA appeared slightly blurry for the most of the participants, and this caused them to experience more fatigue.

This study emphasizes the need for understanding the human visual system when developing real-time graphics for virtual reality application.

Keywords:	anti-aliasing, real-time rendering, virtual reality, head-mounted display, computer graphics
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Paula Jukarainen		
Työn nimi: Reaaliaikaisten antialiasiontimenetelmien vertailu virtuaalilaseilla			
Päiväys:	10.10.2016	**Sivumäärä:**	vii + 82
Pääaine:	Mediatekniikka	**Koodi:**	T-111
Valvoja:	Jaakko Lehtinen, TkT		
Ohjaaja:	Juha Sjöholm, DI		

Virtuaalitodellisuus (VR) ja VR-lasit ovat yleistyneet viime vuosina. VR-lasien huomattavasti suuremman näkökentän sekä lähelle silmiä tulevan näytön vuoksi antialiasointi, eli reunojen pehmennystekniikoista, on tullut tärkeäksi.

Diplomityössä tehdään kirjallisuuskatsaus reaaliaikarenderöinnin, antialiasoinnin sekä virtuaalitodellisuuden perusteisiin. Teoriaan sekä viimeaikaisiin tutkimuksiin perustuen kolme antialiasointimenetelmää fast-approximate (FXAA), temporaalinen (TAA) sekä moninäytteistys (MSAA) ovat valittu implementoitavaksi reaaliaikaohjelmistoon ja tarkemmin tutkittavaksi suorituskyvyn sekä subjektiivisesti testattavan visuaalisen laadun puolesta. Diplomityö keskittyy visuaalisessa laadussa tutkimaan vain eri menetelmien kykyä estää tai redusoida reunojen antialiasointia ja esimerkiksi pienien geometristen objektien yksityiskohtien välkkymistä.

Suorituskyvyn mittauksissa FXAA oli menetelmistä nopein (3% menetys suorituskyvyssä), TAA 10-11% menetys suorituskyvyssä sekä MSAA hitain 22-62% suorituskyvyn menetyksellä.

Subjektiivisen laadun testillä mitattiin kokemuksen laatua, joka koostui visuaalisen laadun sekä uupumuksen arvostelusta eri tapauksissa. Ärsykkeet eli eri antialiasointimenetelmät esitettiin reaaliaikaisessa 3D-ympäristössä, jota katsottiin Oculus Rift CV1 -virtuaalilaseilla.

Tulosten mukaan neljän sekä kahden näytteen versiot MSAA:sta olivat selkesti visuaalisesti laadukkaimmat sekä aiheuttivat vähiten uupuneisuutta koehenkilöissä. FXAA havaittiin laadultaan hiekommaksi, mutta ei MSAA:ta enemmän uupumusta aiheuttavaski. TAA aiheutti selkeästi eniten uupumusta sekä oli laadullisesti huonoin liiallisen pehmeyden ja haamuefektin vuoksi.

Tämä tutkimus painottaa ihmisen näköjärjestelmän ymmärrystä kehittäessä reaaliaikagrafiikkaa VR-ohjelmistoihin.

Asiasanat:	antialiasointi, reaaliaikarendaus, virtuaalitodellisuus, virtuaalilasit, tietokonegrafiikka
Kieli:	Englanti

Acknowledgements

The subject of this thesis combined two of my passions: human perception and real-time rendering. What could be a better way to combine those two than virtual reality? This thesis was truly an eye-opening experience, which taught me more than I could imagine.

I would like to thank my employer for this opportunity; professor Jaakko Lehtinen for the supportive feedback and inspiration; advisor Juha Sjöholm for guidance and feedback; Matti Nelimarkka for the last minute introduction to statistical analysis and HCI research; Jani Joki for proofreading; Max Aizenstein and the whole graphics benchmarking team for excellent discussions and ideas; The group of people who participated in the study; friends and family who supported me and cheered me up along the way; and of course Veli-Matti for making me smile even at stressful times.

Espoo, October 10, 2016

Paula Jukarainen

Abbreviations and Acronyms

2D	Two-dimensional
3D	Three-dimensional
AABB	Axis-aligned bounding box
AAGA	Aggregate G-buffer anti-aliasing
BRDF	Bidirectional reflectance distribution function
CFF	Critical flicker frequency
CS	Compute shader
FOV	Field of view
FPS	Frames per second
FXAA	Fast approximate anti-aliasing
G-buffer	Geometric buffer
GPU	Graphics processing unit
HDR	High dynamic range
HMD	Head-mounted display
LDR	Low dynamic range
MSAA	Multisample anti-aliasing
PS	Pixel shader
QoE	Quality of experience
RGB	Red-green-blue
SMAA	Enhanced sub-pixel morphological anti-aliasing
stdev	Standard deviation
TAA	Temporal anti-aliasing
VAC	Vergence-accommodation conflict
VR	Virtual reality
VS	Vertex shader
Z-buffer	Depth buffer

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Computer graphics is a field of study and a collection of various technologies to generate and manipulate digital images. Computer games, medical imaging, virtual reality and movies are good examples of computer graphics applications. Applications that produce images rapidly on a computer utilize *real-time rendering*, which is a process of making images of a virtual world in a rapid pace. This rapid pace of at least 15 digital images, *frames*, per second (FPS) allows interactivity between a user and the virtual world. In order to keep the application responsive the latency between user interaction and the feedback (rendered digital image on the display) has to be low [4, p. 1].

The performance of a real-time rendering application depends on the available computing power, but also heavily on the rendering techniques, the amount of data in the virtual world and the rendering resolution. In every frame, a continuous 3-dimensional (3D) world is sampled in order to have a discrete image of it to be presented on the display [4, p. 117]. The rendering resolution defines how precisely a 3D world can be presented. The higher the rendering resolution is the more information of the 3D world can be rendered to a display with a growing computing cost.

As computing power is not infinite the resolution has to be compromised. Due to this small detailed information such as sharp geometry edges or tight highlights on shiny objects in the 3D world can *alias* when the rendering resolution is not high enough. Aliasing appears usually as ”jagged” edges or flickering highlights (see figure 1.1). As the resolution decreases the chance of aliasing grows, and similarly, the more information of the virtual world is shown the more likely the aliasing occurs.

There are various techniques to prevent and reduce aliasing. These techniques are referred as *anti-aliasing*. Some of the techniques prevent aliasing by increasing the rendering resolution while others by detecting and manipulating aliased patterns from the image, the common goal being the better image quality. However, these methods cost varying amount of computing time.

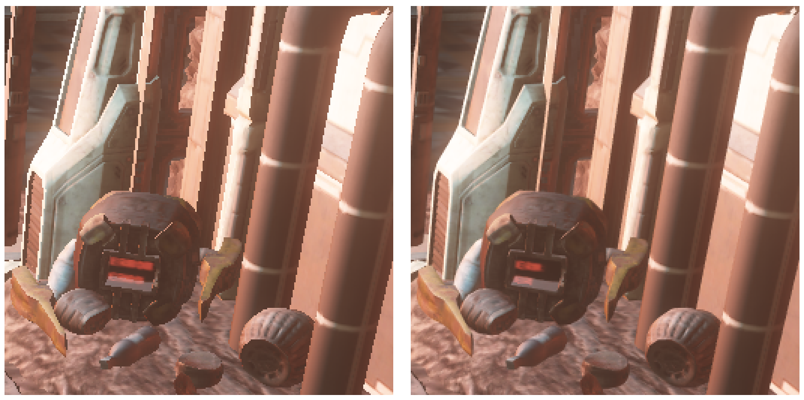

Figure 1.1: Picture on the left show jagged geometry edges. On the left jaggies do not exist due to anti-aliasing. Both images have been rendered with the same resolution.

Virtual reality (VR) head-mounted displays (HMDs) such as Oculus Rift [53] and HTC Vive [25] have become popular in the last few years. These devices have a 90Hz display with nearly 1080p resolution per each eye combined with a large field of view (FOV) [53]. In order for an application to render in real-time for HMDs, it has to render two frames, one per each eye, on 90 frames per second (FPS). If application is unable to maintain the high rendering speed and fast response time to minimize latency, users can experience discomfort (see section 5.2).

A large field of view requires a high resolution for the image to be presented precisely. However, HMDs only have a certain amount of pixels and already high performance goals. This makes anti-aliasing important, but challenging.

This thesis is inspired by the challenges of VR and importance of anti-aliasing. The main goal is to study existing anti-aliasing methods and rank them by image quality and the amount of discomfort experienced by human observers. The results of this thesis indicate that anti-aliasing methods contribute to visual quality, but also to discomfort. Wrong choices in the selection of anti-aliasing method can be disastrous for a VR experience.

1.2 Scope and Contributions

This thesis compares two different anti-aliasing methods against multisample anti-aliasing (MSAA) (see section 4.2), which is considered as a gold-standard for VR applications [78].

This thesis focuses on following questions:

- How different anti-aliasing methods affect subjective image quality and fatigue on HMD?
- Can one of these methods be an alternative for multisample anti-aliasing?

The anti-aliasing methods are evaluated by their applicability for deferred rendering (see section 2.5), performance as well as visual quality. For performance comparison, FPS and memory consumption will be used as the metrics. Visual quality is examined in the subjective image quality study as described in section 7.

The scope of this thesis is limited to only examining the aliasing that occurs on edges and on small details. Shading aliasing, which can appear as flickering highlights [30], is not in the scope and is excluded. For examination, fast approximate anti-aliasing (FXAA) (see section 4.3) and temporal anti-aliasing (TAA) (see section 4.4) were selected to be compared against multisample anti-aliasing (MSAA) (see section 4.2). The reasons for selecting these methods is explained in chapter 4.

The thesis done with a benchmarking company Futuremark (UL). The anti-aliasing methods are implemented in the soon-to-be launched benchmark product VRMark. Two of the methods are included in the final launched product.

Content design has a large impact on aliasing. Different types of content can bring up the advantages or disadvantages of different anti-aliasing methods. However, this thesis focuses on testing anti-aliasing methods on the content that is designed for VRMark specifically and not special features such as sub-pixel or edge aliasing solving. However, the findings should be applicable generally for the most situations.

1.3 Structure of the Thesis

The second chapter introduces the basics of real-time rendering, the graphics processing units and different rendering methods: forward, forward+ and deferred. Chapter 3 gives the theoretical foundations of aliasing and anti-aliasing. Chapter 4 describes the anti-aliasing methods that are chosen for this study more in detail. Chapter 5 focuses on explaining details of current virtual reality headsets and explain their relationship to presence, problems and requirements for VR applications. The VRMark rendering engine and implementations of anti-aliasing methods used in this thesis are presented in chapter 6.

Performance and visual quality testing methods are presented in chapter 7. Both performance and visual quality testing results are show in chapter 8. The final conclusions and future improvements are in chapter 9.

Chapter 2

Rendering in Real-Time

This chapter introduces the basic terms and concepts of real-time rendering, which are used in later chapters.

2.1 From 3D to 2D

The goal of rendering is to present a virtual three dimensional (3D) world as a two dimensional (2D) image. The image consists of a finite number of *pixels*, the smallest element of the image. Each pixel holds a color value described by a three component red-green-blue (RGB) vector. An RGB vector holds an intensity value for the red, green and blue channels. The number of pixels in the image is defined by the resolution.

Virtual 3D world A 3D world is usually presented with a set of objects (or geometry) and lights (see figure 2.1). Each object has a 3D position in the 3D world, a shape and a *material.* The materials describes how the surface of the object appears in different lighting conditions - as in real world it would [4, p. 11]. In other words, it has some properties which define how the light is reflected and/or scattered from the surface of the object. As in the real world, in the virtual world lights can be presented differently depending on the light type. A point light has a 3D position and intensity [4, p. 218] whereas a spotlight also has direction and a cone shape [4, p. 221].

Materials Materials describe the pattern and the color of the surface. The pattern or "bumps" of a surface can be presented by 3D vectors called *normals*, which describe in which direction the small bumps are facing. These 3D vectors are called *normals.* The color of the surface is usually composed of two components: the base color of the surface and the color of highlights. The base color, *diffuse*

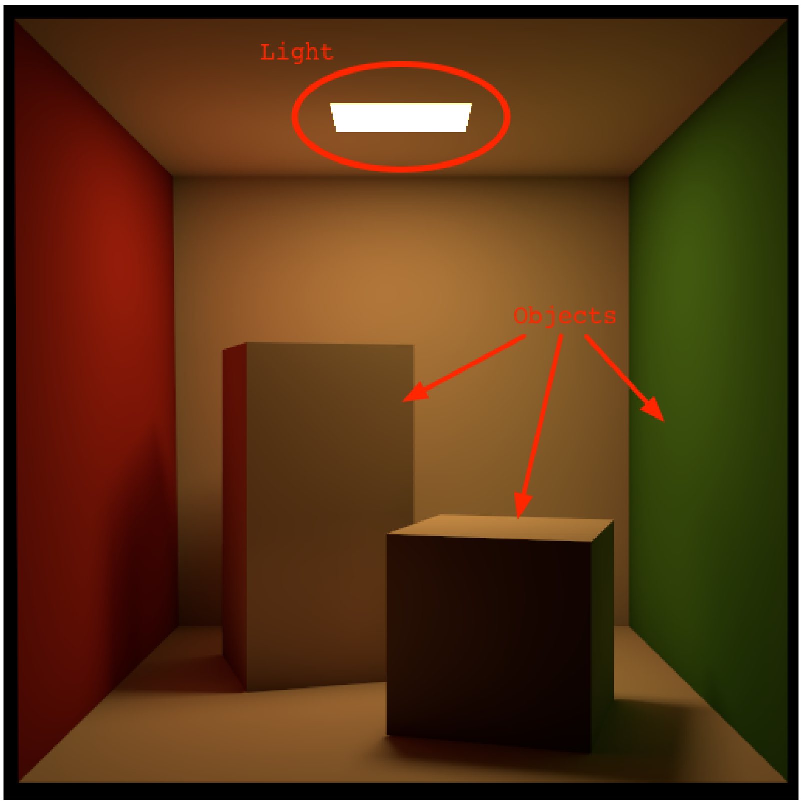

Figure 2.1: Example of 3D world with one light. Image created with Mitsuba renderer [27].

color is the color that the surface reflects uniformly in every direction when lights hits it, whereas the highlight color is defined by *specular color* (see figure 2.3). [4, p. 106]

Visibility When we look at the real world, photons of light are constantly transmitting information of our surrounding environment by hitting our retinas. In image synthesis, a virtual camera functions as our eyes. However, as there are no photons as such the information has the be presented in a different way. To present a 3D world in a 2D image, we have to evaluate what is seen through the virtual camera and present that in a digital image (see figure 2.2). Some objects are occluding others and all the light might not hit the camera lens. This process is called *visibility* evaluation. As every object has a 3D position and a shape, the visibility of each object can be calculated from the camera point of view and the light point of view. If a light hits a visible object, the object can be seen.

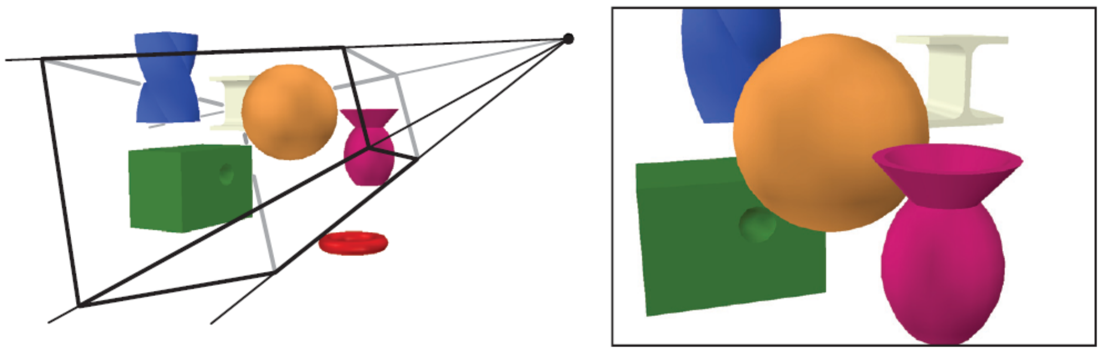

Figure 2.2: Left: A virtual camera is located at the tip of the pyramid (where the four lines converge). Only the primitives inside the view volume, limited by camera's near and far plane, are rendered. Right: The image shows what the camera sees. Objects that are fully or partially outside of the camera's view volume are discarded or clipped. Image from [4, p. 12]

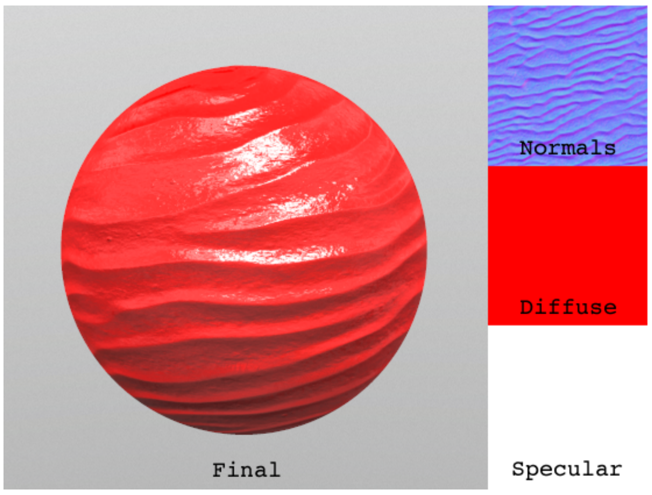

Figure 2.3: A sphere gets its surface patterns by its normals (top). Bright highlights are white (bottom) and the base color of the object is red (middle).) This final image of a sphere on the left and its material properties normals, diffuse and specular colors on the right.

Lighting When a light hits an object its surface gets its visual appearance defined by its material. Determining how all the lights in the scene affect objects is called *shading* [4, p. 17]. The appearance of the object's surface depends on its material properties but also how the light is reflected from its surface. This can be computed using a *bidirectional reflectance distribution function* (BRDF) [48]. The BRDF describes how the light is reflected from the object's surface by the incoming light direction $\mathbf{l}$ and the outgoing view (or camera) direction $\mathbf{v}$.

In order to light the virtual world, we want to evaluate a *reflection equation* for each surface point $\mathbf{x}$:

$$L_0(\mathbf{x}, \mathbf{v}) = \int_{\Omega} f(\mathbf{l}, \mathbf{v}) \otimes L_i(\mathbf{x}, \mathbf{l}) cos\theta_i d\omega_i \tag{2.1}$$

In the equation $L_0(\mathbf{x}, \mathbf{v})$ defines the final pixel color in the rendered image. It is the outgoing radiance, where $\mathbf{x}$ is the surface location in space. Ω is the hemisphere of directions of the outgoing light above the position $\mathbf{x}$. $f(\mathbf{l}, \mathbf{v})$ is the BRDF. $L_i(\mathbf{x}, \mathbf{l})$ is the incoming spectral radiance towards $\mathbf{x}$ from $\mathbf{l}$. θ_i is the angle between $\mathbf{l}$ and the surface normal. [4, p. 327]

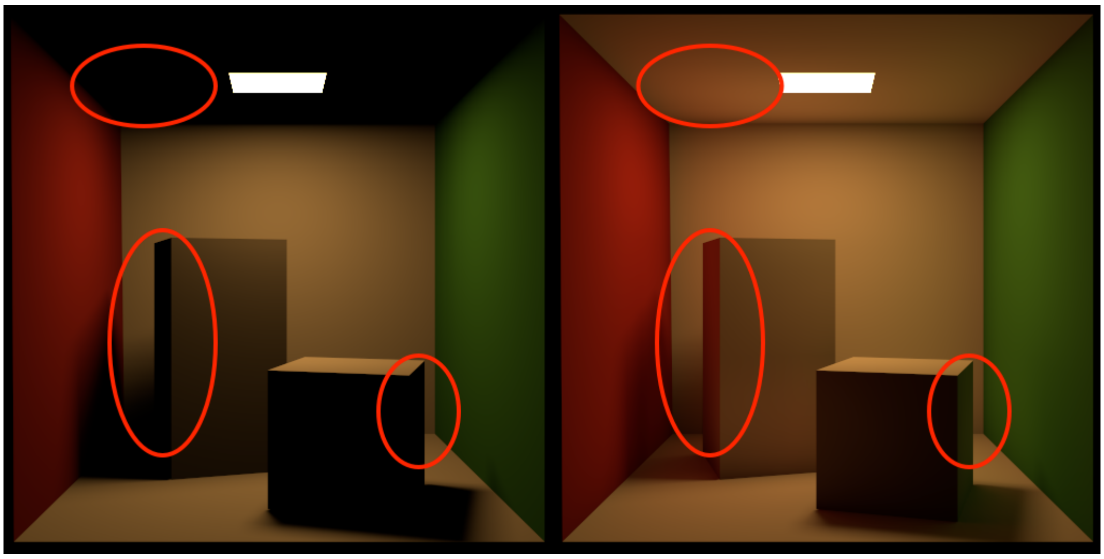

Figure 2.4: Direct and indirect lighting. Image created with Mitsuba renderer [27].

The reflectance equation sums up all incoming radiance from all the directions. It does not distinguish between the *direct lighting* and *indirect lighting*. In this thesis, we only focus on direct lighting as is common in real-time rendering. The direct lightning means the direct emitted radiance without contribution of reflected radiance from other objects. In figure 2.4 the left image is rendered with direct lighting only and the image on the right has also indirect lighting. The light has to be "bounced" around in order to compute the indirect lighting. However, "bouncing" usually means some kind of recursive algorithm, which are not a real-time-friendly techniques.

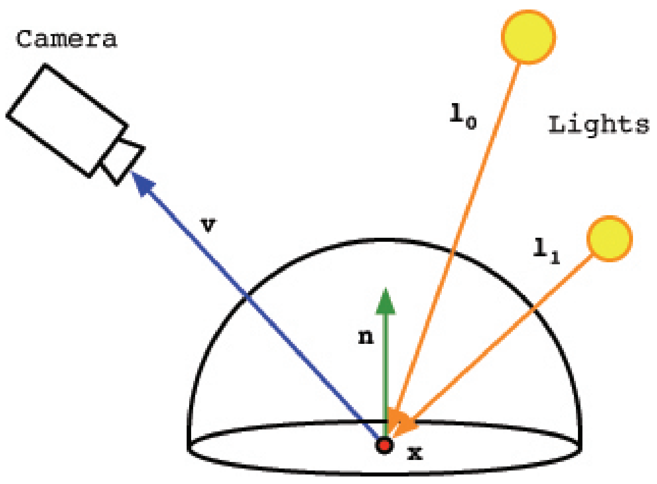

Figure 2.5: Direct lighting is computed in real-time by combining the lighting contribution from all point light sources from directions $\mathbf{l_0}$ and $\mathbf{l_1}$

The reflectance equation cannot be evaluated as such in real-time rendering. As we are only computing direct lighting, the integral have to be replaced by a sum which indicates the contribution of all direct light sources n. All the outgoing light from the surface point $\mathbf{p}$ may not be visible for camera, so visibility evaluation term $V(\mathbf{v})$, where $\mathbf{v}$ is the direction of outgoing light, is added:

$$L = \sum_{i=0}^{n} L_i(\mathbf{x}, \mathbf{l}) f(\mathbf{l}, \mathbf{v}) cos\theta_i V(\mathbf{v}) \tag{2.2}$$

Modified reflectance equation 2.2 is illustrated in figure 2.5.

Enhancing the rendered image The effects that add realism and visual richness are added to a rendered image after lighting. These effects are called *post-processing effects.* Typical post-processing effects that are used in PC games are different camera effects such as depth of field [4, p.486], bloom and lens flares [4, p. 482] [69] (see figure 2.6). However, use of post-processing effects in VR is fairly limited (see section 5.4).

A virtual world might have multiple lights with large differences in their intensities. In such a case the world has a *high dynamic range* (HDR). The rendered HDR image pixel values can require values beyond the range of numerical value that the image format is able to present. Additionally, the display has only a certain luminance range, likely more limited than what the virtual world has. The rendered image has to be fitted to display's luminance range before it can be shown in the display. This process is called *tone mapped.* [4, p.475]

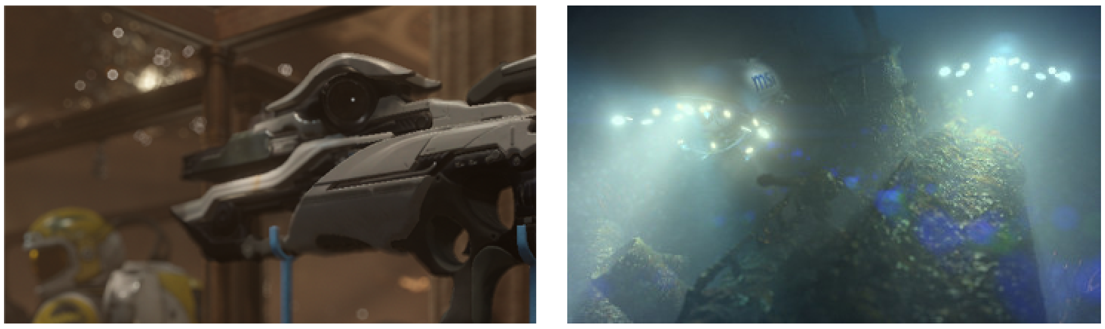
Figure 2.6: Example of depth of field effect on the left: The out-of-focus areas in the back are blurry and in-focus areas on the front are sharp. Lens flare effect example of the right: Incoming light is scattering inside of the camera lens system forming the blue flares.

There are different ways to render the final image from a 3D world. Three of the most common rendering styles are introduced in the further sections 2.3 - 2.5.

2.2 Graphics Processing Unit Pipeline

All the algorithms and processes introduced in this thesis are executed in a graphics processing unit (GPU). Therefore a brief introduction to the GPU pipelines and their programmable parts are necessary to understand the topic.

GPU pipeline has two main programmable pipelines: a *graphics* and a *compute* pipeline. Both of these pipelines can be programmed and they both can use image data and numerical data. Images are referred as *textures* and usually numerical data is stored to *buffers*. Both of these can be generally referred as *resources*. In this thesis, going through the whole graphics pipeline is referred as a *draw call* and going through any pipeline is referred as a *pass*.

Graphics pipeline The graphics pipeline is a pipeline specialized in generating i.e. rendering a 2D image from a 3D world. It takes a 3D world as input and outputs a 2D image. The output image is called a *render target*.

Each unique object in the 3D world is given to the graphics pipeline one at the time. An object is composed of triangles which in turn are composed of a set of 3D points called *vertices* (see figure 2.8). The first programmable stage of the pipeline, the *vertex shader* (VS) handles them. In order to present the 3D world as 2D image, vertices have to be *transformed* from 3D to 2D. In order to achieve this, all the vertices of the object go through multiple different *coordinate systems*

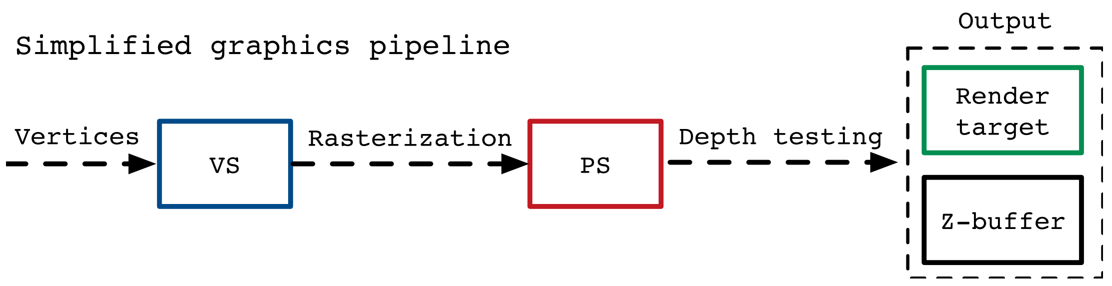

Figure 2.7: Simplified representation of the relevant parts of the graphics pipeline.

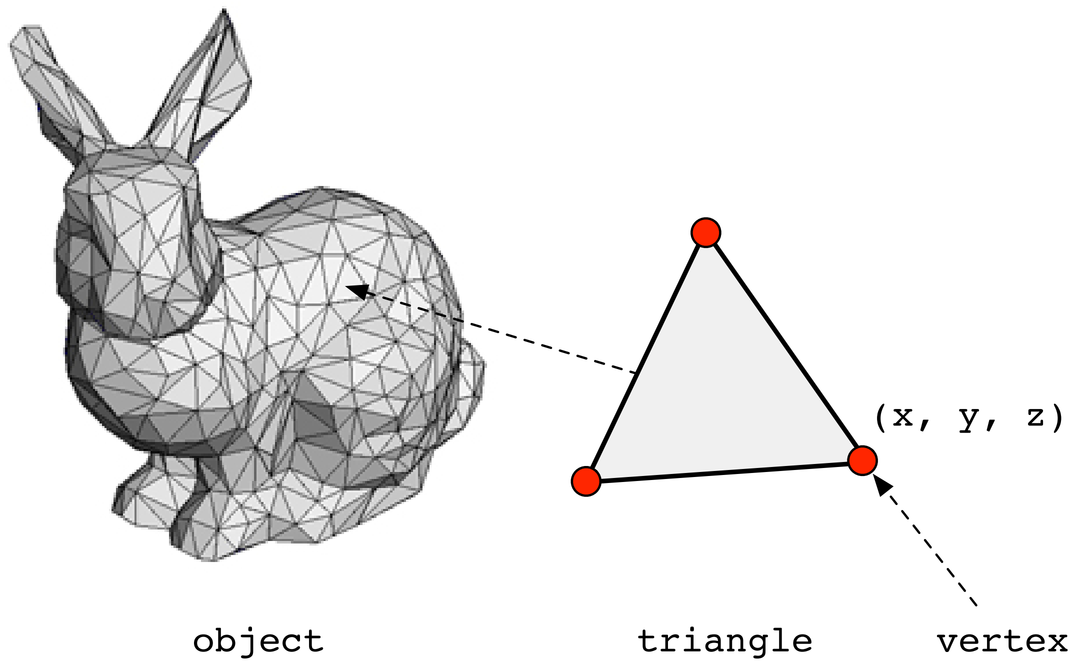

Figure 2.8: The object (bunny) consists of triangles, which consist of three vertices, a 3D points.

referred to as *spaces*. At the start, each object has its orientation and position defined in its *model space*. In order to handle all the objects similarly, the objects have to be transformed to a common space, the *world space* [4, p. 16]

The virtual camera is different from objects in that its ”space” is called a *projection matrix*. The camera projection defines how the camera is oriented, where it is located, where are its near and far planes (recall the figure 2.2), field of view and what is the projection type. Using the projection matrix, we can

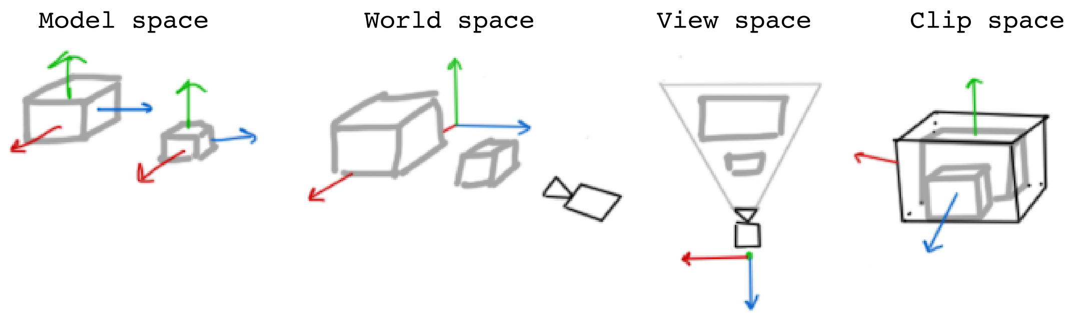

Figure 2.9: Two cubes have their own coordinate systems in model space. When both cubes are transformed to the world space they have common coordinate system. In view space, camera is placed to the origin and the objects according to it. In clip space, objects are clipped inside the camera frustum and scaled between [-1, 1].

determine a *view space* to which the object in the 3D scene can be transformed. View space places the camera at the origin of the coordinate system and aligns objects accordingly. [4, p. 17]

Finally the objects are transformed to a *clip space*. The clip space respects the camera's near and far planes and scales the 3D positions between [-1,1] i.e. represents the objects in *normal devices coordinates*. This chain of transformations is illustrated in the figure 2.9. [4]

However, it is important to notice that these transformations do not necessarily need to be executed in VS only, and also not only from model space to world space etc. Typically it is so, but sometimes, for example with some special effects, these same transformation can be used in any other shader or pipeline (defined later) when needed. Transformations can be executed in the inverted order also.

After VS, the triangles formed by vertices are *rasterized*. The rasterizer handles triangles instead of individual vertices. Each triangle is tested if it is inside the camera's view properties i.e. the *view frustum* (see figure 2.2). The parts that are outside are thrown away and not processed further. The triangles or parts of them which are inside the view frustum are clipped against pixels center in a raster image. If the triangle overlaps it, the visible part of the triangle becomes a pixel i.e. invokes a *pixel shader* (PS) (see figure 2.10). [44]

The PS is the last programmable shader of the pipeline. It is able to perform per-pixel operation such as lighting. PS outputs per-pixel values which are presented in the final color (render) target, if not overlapped by another [45]. Overlapping is evaluated by the depth of the pixel. To evaluate if pixel is overlapping by another, its depth is evaluated against a depth value in a *Z-buffer* i.e. depth

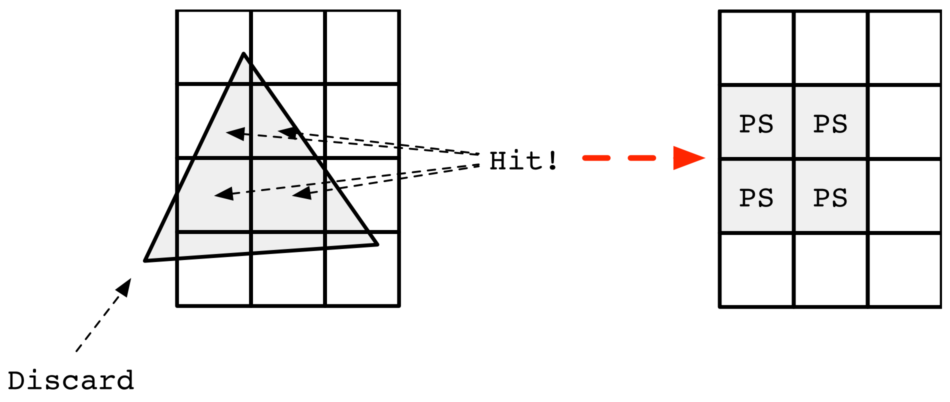

Figure 2.10: A triangle is tested against a raster image pixels. The parts that are outside the image area are discarded. The parts that cover a center of a pixel invoke a pixel shader.

buffer. The depth buffer stores the foremost pixel depth. If a pixel is overlapping a current value in depth buffer, the value is update and if not, the pixel is discarded. The render target has the foremost depth associated color value.

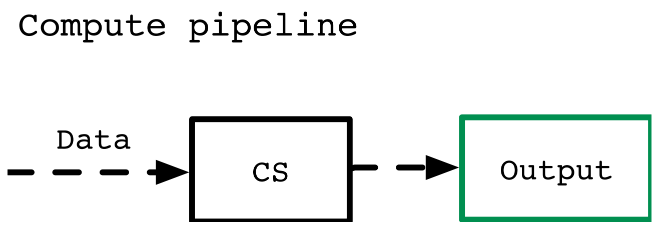

Figure 2.11: The compute pipeline.

Compute pipeline The compute pipeline is a general purpose computing pipeline which utilizing parallel processing. The pipeline can be programmed with *compute shaders* (CSs). A CS can take resources (not vertices) as input and outputs buffers or textures as results. Typical use-cases for computer shaders are light culling and image-based post-processing effect. Illustration of the compute pipeline is presented in the figure 2.11.

2.3 Forward Rendering

The traditional way of rendering is called *forward rendering*. In forward rendering the data is processed so that light intensity, visibility and BRDF are computed per each light for all the objects. In practice, a forward rendering pipeline takes geometry data as input and passes it to the graphics card. Data is processed and transformed from 3D to 2D in a VS. Finally in PS, all the lights in the scene are looped over and lighting is computed for each pixel.

In other words, material properties and lighting are computed while the data goes through the whole pipeline. The geometry is processed for all the lights separately and the results are summed to the same render target or a texture, which is presented on a screen (see figure 2.12).

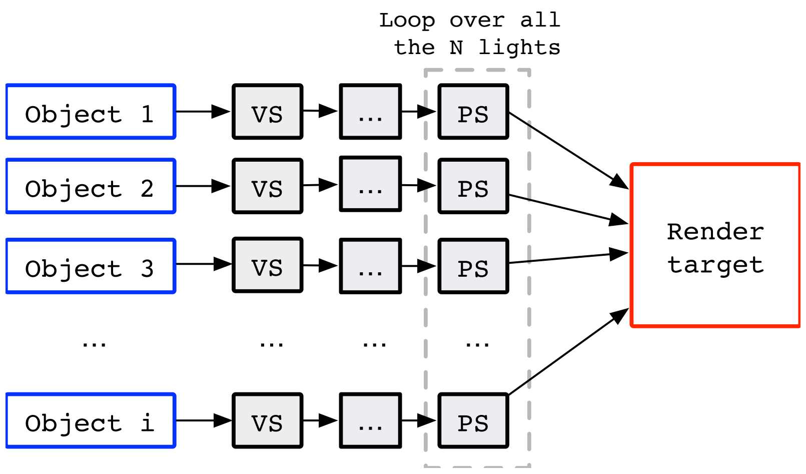

Figure 2.12: Forward shading pipeline: All the objects in the scene are pushed to the graphics pipeline one by one, transformed in VS, shaded in PS and finally the shading results are saved to the same render target.

As we see from the figure 2.12, all the geometry has to be processed with all the lights and thus the number of lights affects the performance of the PS pass. A large number of lights will make forward rendering heavier. Determining which light sources affect which objects is time-consuming and the more complex the light sources get the heavier the computation becomes.

Also, Z-buffer is populated and updated as objects go through the pipeline. This means that it is possible that if multiple triangles are overlapping on a pixel, the same pixel location will be tested and updated multiple times causing rendering to become inefficient. Each pixel is shaded although some might be fully covered by an another pixel. If geometries are sorted in a front-to-back order before rendering the problems can be minimized, but not entirely fixed. Then some of the pixels which are not visible (are covered by an another pixel) are discarded and are not shaded.[4, p. 279]

In contrast rendering an approximation of (semi)transparent objects is a simple task in forward shading. For example, transparent geometries can be shaded after opaque geometries in back-to-front order and blended on top of them. [4, p. 24]

Also, anti-aliasing is straightforward using multisampling approaches. This is discussed in section 4.2).

2.4 Forward+ Rendering

Forward+ is an extended version of the forward rendering method with the ability to render a large amount of lights. Forward+ adds a *depth prepass* and *light culling* to reduce the pixel overdraws and thus simplify the rendering of many lights in the final PS. [23]

First, in the depth prepass the scene depth is rendered to a Z-buffer. The evaluated depth can be used in the light culling and in the final shading pass. The depth optimizes the final shading pass as the foremost pixels are already evaluated there will be no pixel overlapping and unnecessary shading. [23]

Evaluating which light hits which pixel in advance to shading reduces computation on the final shading pass. In the light culling the list of lights overlapping a pixel or a tile of pixels with a chosen size is calculated. Culling lights per pixel is not the most efficient solution as memory footprints can grow quite high. Culling lights only per screen tile reduces footprint and computation costs a lot but is not as accurate and can introduce false-positives. [23]

The shading stage uses the light list that was calculated in the light culling and evaluates the materials using the stored light information. The cost of this stage depends on how many lights overlap a single pixel per tile. [23]

Forward+ has the same benefits as forward rendering. For example hardware accelerated multisample anti-aliasing (see section 4.2) can be used and transparent materials can be supported efficiently. Adding the light culling makes forward rendering more efficient at the larger number of lights which is where normal forward rendering is weaker compared to deferred rendering (see 2.5). As shading is done in the forward style, forward+ rendering theoretically requires less memory traffic than deferred rendering but has a small workload increase compared to traditional forward rendering caused by the separate light culling pass. [23]

2.5 Deferred Rendering

The key concept in deferred rendering is that each pixel is shaded only once [16], which is achieved in current deferred rendering implementations by drawing all the geometry to intermediate textures holding material properties, the *geometry buffer* (G-buffer) [61], and computing the shading is a separate rendering pass.

In deferred shading geometry data is processed in a similar way as it is in forward shading, but in the pixel shader the geometry material attributes are saved to a G-buffer. The G-buffer consists of textures which store all relevant geometry information such as normals, depth (Z-buffer), albedo and other material attributes (see figure 2.15). Lighting is computed on a separate pass using G-buffer information. Deferred rendering is presented in figure 2.14. [4, p. 279]

A larger number of lights can be easily supported as lighting has to be calculated only once per pixel. The problems of overlapping pixels in the forward rendering is solved by the deferred renderer's G-buffer which holds the foremost geometry information. [4, p. 281–282]

The traditional way of implementing lighting in a deferred renderer is light accumulation, where shading is calculated per light similar to forward rendering. The more efficient method is light culling (shown in the figure 2.14) which was originally developed for the forward+ renderer. Culling reduces unnecessary G-buffer loads and simplifies the shading, which also optimizes it as all the visible lights can be shaded at one go [17]. Light culling is described in section 2.4.

Having the G-buffer stored for later use and having a separate lighting pass has several advantages. Having depth information, normals and other material attributes available enables multipass lighting and the use of multiple different screen space the post-processing effects such as depth of field and screen space reflections. [4, p. 279]

One of the drawbacks of deferred rendering is that memory requirements and fill rate costs can be high as G-buffer consist of many textures, which are all filled in the same pixel shader. Using separate passes for material and lighting can cause performance drops and bandwidth problems as many render target reads and writes are required. [4, p. 281–282]

Another disadvantage is the difficulty of rendering transparent geometries. As only the foremost geometry information is stored in the G-buffer for lighting, rendering transparent geometries is impossible in the same pass without storing extra information [32, p. 127]. There are ways to implement transparent geometry rendering for deferred rendering, but extra effort is needed [32, 55, 57].

Anti-aliasing is an issue as well. See chapter 4 for discussion.

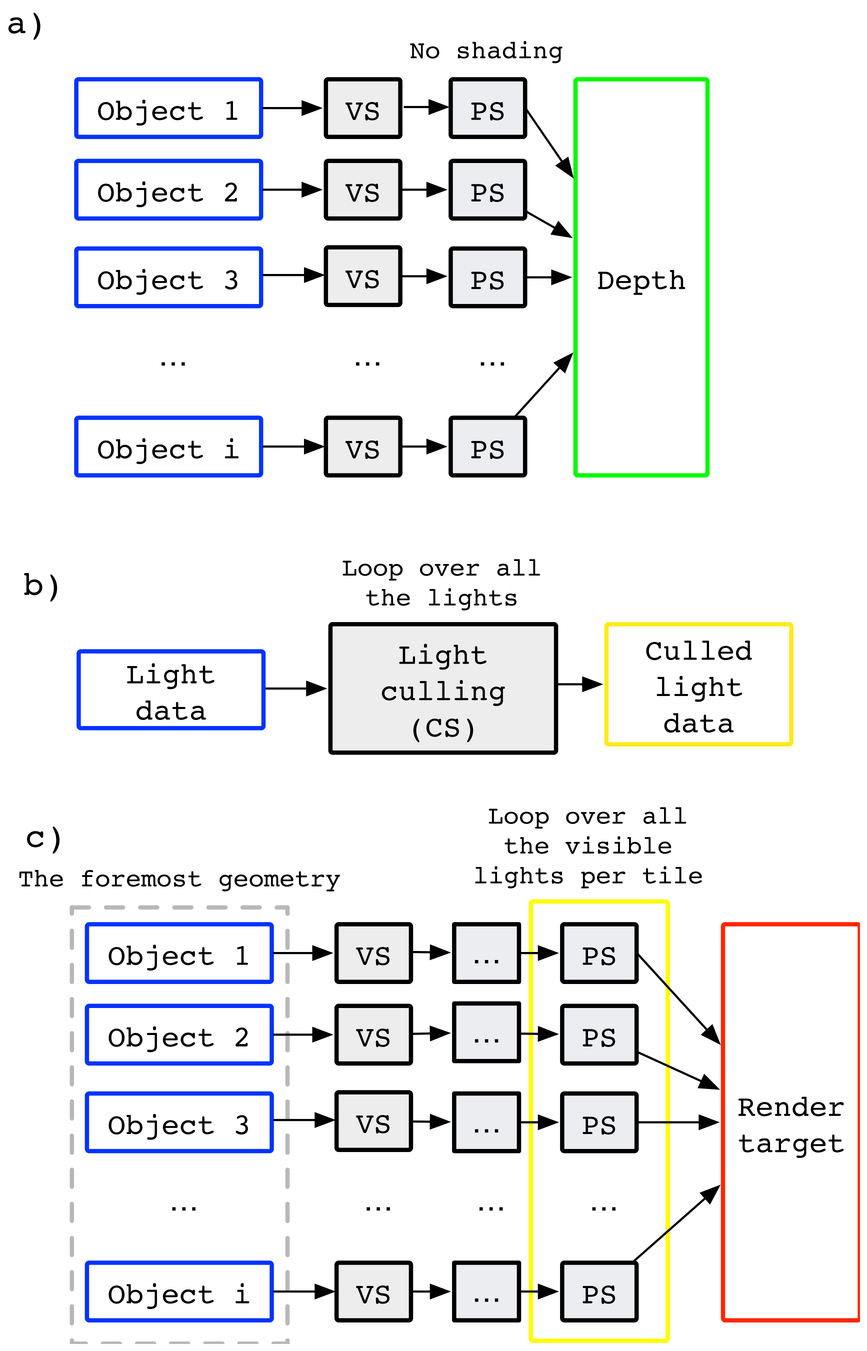

Figure 2.13: Forward+ shading pipeline. a) Depth of the foremost geometries is rendered. b) Lights are computed i.e a list of light indices overlapping a pixel is computed and saved to a buffer. c) The foremost visible geometries are shaded using pre-rendered depth and pre-culled lights. No need to execute the pipeline for each light.

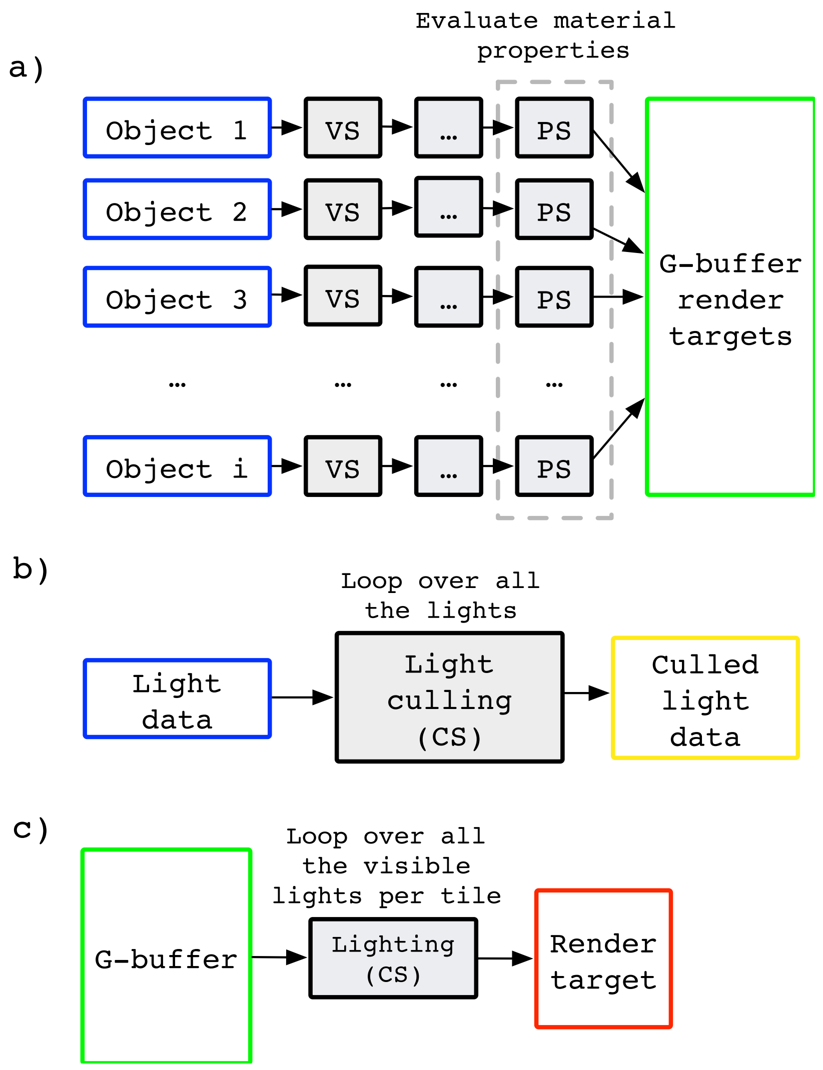

Figure 2.14: Deferred shading pipeline: a) G-buffer is draw by rendering the materials properties on every object to it. b) Lights are culled. c) Lighting is calculated using the G-buffer and culled lights.

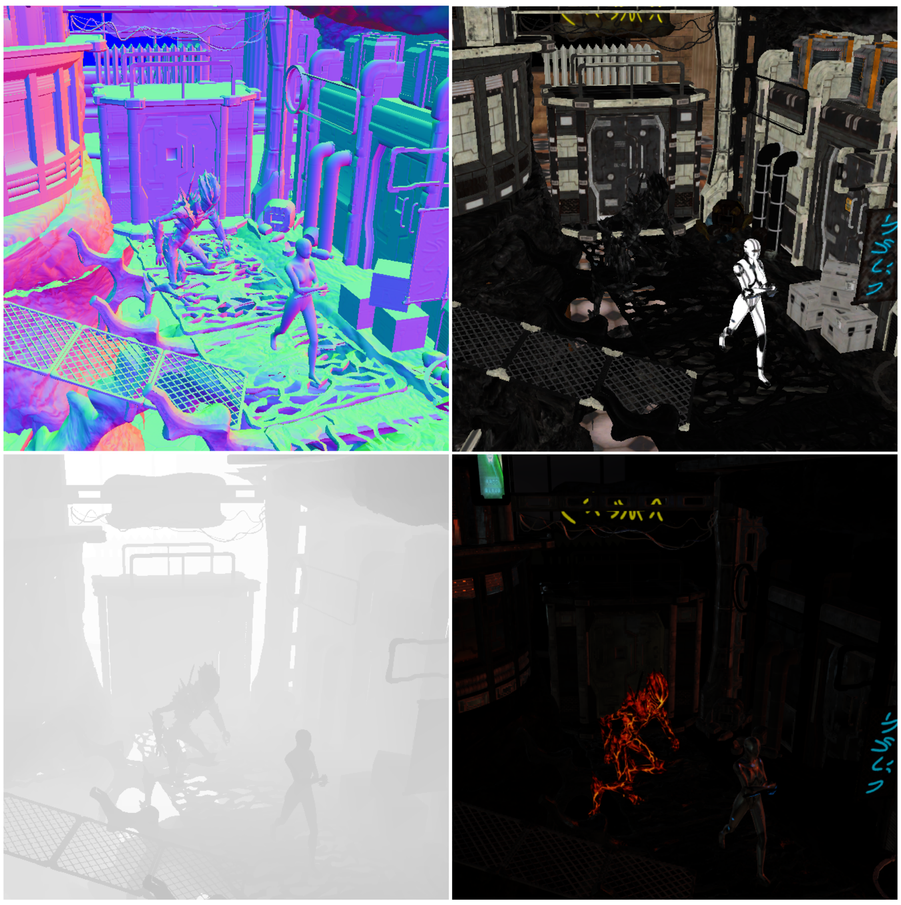
Figure 2.15: VRMark G-buffer: normals (top left), reflectance texture (top right), depth (Z-buffer) (bottom left) and luminance texture (bottom right).

Chapter 3

Aliasing: Theory and Practice

A virtual 3D scene is represented as continuous values, but a physical device requires discrete pixels. Displaying a digital image from a virtual camera's film plane to a physical devices - such as a display - is a process of representing the continuous signal in a discrete form, filtering it and transforming the filtered signal back to a continuous representation. Presenting a continuous signal in a discrete form requires *sampling*. *Filtering* is a process for removing some unwanted features of the signal. Transforming the signal back to the continuous form is called *reconstruction process*. Erroneous sampling, as well as the reconstruction process, results in an aliased image on a physical device. [4, p. 118]

3.1 Sampling

For a continuous function to be presented in a discrete form, the signal has to be sampled. Sampling is usually modeled as a multiplication between the continuous function $f(t)$ and an *impulse train* (see figure 3.1). The impulse train or a sampling function $s_{\Delta T}(t)$ defines a set of impulses ΔT apart. ΔT is defined by the sampling resolution [21, p. 212]. The signal sampling can be formulated to a following equation:

$$\tilde{f}(t) = f(t)s_{\Delta T}(t) \tag{3.1}$$

Aliasing caused by sampling is called pre-aliasing. Understanding pre-aliasing requires understanding of the frequency domain. The frequency domain is a frequency representation of a continuous function (signal) (see figure 3.2). A continuous signal can be transformed from the time domain (or the spatial domain) to the frequency domain by performing the Fourier transformation [21, p. 205]. The Fourier transformation can be calculated with following equation:

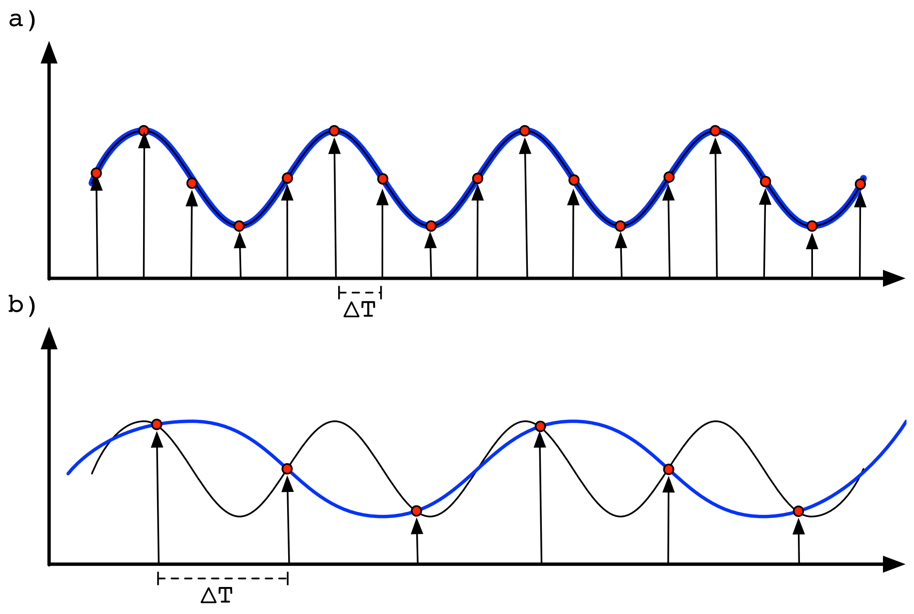

Figure 3.1: A continuous signal (black) is sampled with an impulse train (black arrows) and reconstructed back (blue). a) is an adequately sampled signal and b) is an undersampled signal, which results in a signal that cannot be reconstructed to its original form.

$$F(\mu) = \int_{\infty}^{-\infty} f(t)e^{-j2\pi\mu t}dt \tag{3.2}$$

In the Fourier transformation equation 3.2 the continuous signal $f(t)$, where t is a continuous variable (time or spatial location) is transformed to its frequency domain representation $F(\mu)$, where μ is also a continuous variable (frequency) [21, p. 205].

Recalling the equation 3.1, the multiplication in the time domain can be calculated as convolution in the frequency domain. Thus the sampled signal in the frequency domain can be presented as:

$$\widetilde{F}(\mu) = F(\mu) * S(\mu) \tag{3.3}$$

The Fourier transformation $\widetilde{F}(\mu)$ is an infinite, periodic copies of the $F(\mu)$. Therefore infinite copies create *replicas* in the frequency domain (see figure 3.5) [21, p. 213].

Theoretically in signal processing, the frequency domain is practical in signal analysis and processing. For example finding the highest frequency of a signal or filtering a signal is theoretically easy in the frequency domain. Analysing the highest frequency of a signal is important in terms of aliasing. Pre-aliasing occurs when an insufficient sampling frequency has been used when sampling a continuous function. According to the sampling theorem: The sampling frequency should be at least twice the highest frequency contained in the signal [21, p. 215] [63].

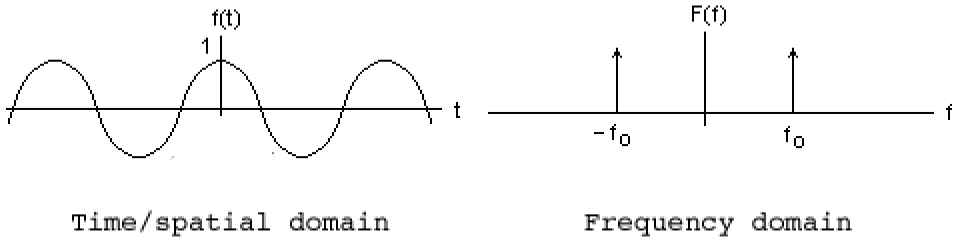

Figure 3.2: Example of a signal in the time/spatial and frequency domain.

In mathematical form

$$f_s \geq 2f_c \tag{3.4}$$

where f_s is the sampling frequency (how often samples are taken per unit of time or space), and f_c is the highest frequency contained in the signal. The sampling theory only applies to a continuous function limited between a certain frequency interval, i.e to *band-limited* function. [21, p. 215]. However, a 3D scene is normally never band-limited. Infinitely high frequencies are produced by perfectly sharp edges, which can appear, for example, on geometry polygon edges or shadow boundaries [13]. The figure 3.3 has hard polygon edges and shadow boundaries which alias. Therefore it is nearly impossible to evaluate an accurate sampling rate that would satisfy the sampling theorem.

The sampling frequency equal to exactly twice the highest frequency of the sampled signal is called the Nyquist frequency [21]. If a signal is sampled at a lower frequency than the Nyquist frequency, undersampling occurs. Higher frequencies are "masquerading" as lower frequencies, which appears as aliasing. Then the signal cannot be reconstructed to its original form. The figure 3.1 shows an example of a signal, which is reconstructed from undersampled signal data.

The reason why an undersampled signal cannot be reconstructed to its original form can be better understood when the signal is observed in the frequency domain. When a band-limited signal is sampled and presented in the frequency domain, it looks as in the figure 3.4. It only has frequencies in certain range $[-B, B]$. f_s is the sampling rate and B is the signal's maximum frequency. Infinite number of signal replicas (green) appear alongside the original signal (blue).

If replicas do not overlap with the original signal, the sampling rate has been sufficient. If replicas overlap with the signal as in the figure 3.5, undersampling

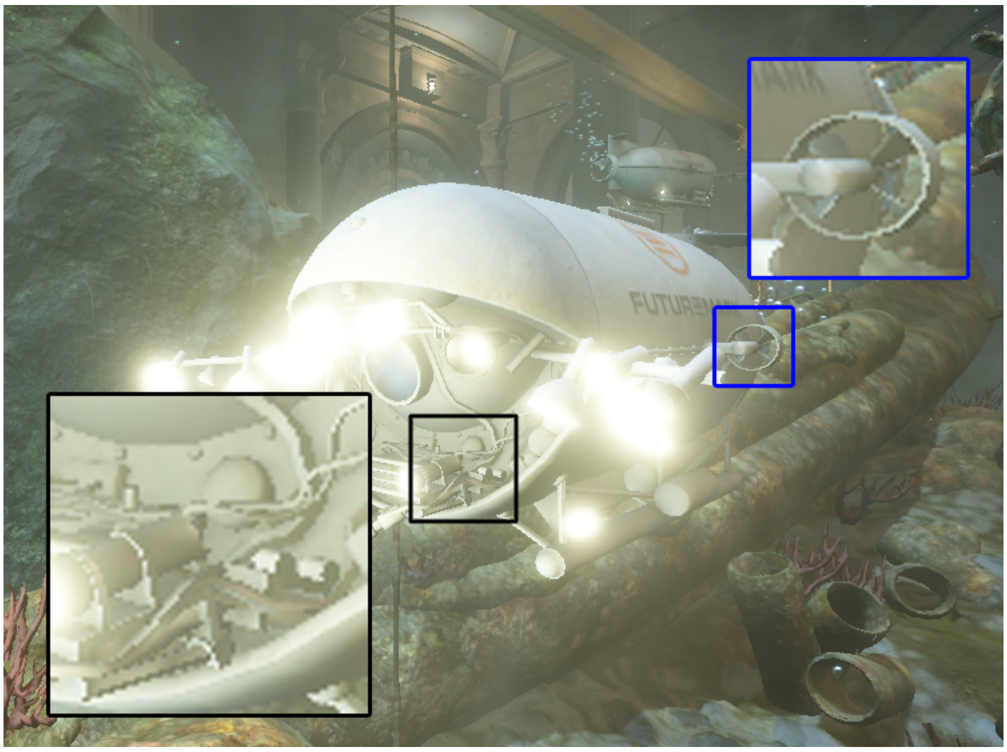

Figure 3.3: Examples of aliasing "jaggies" on the edges.

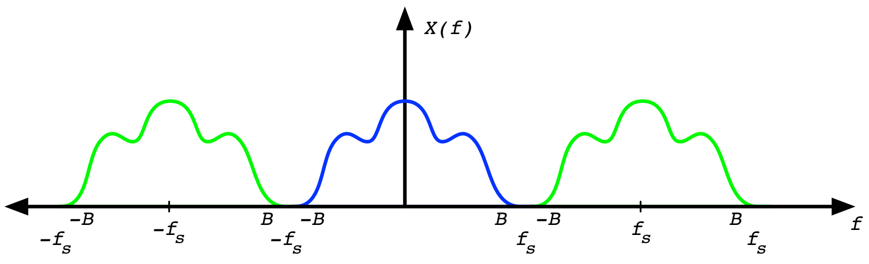

Figure 3.4: A band-limited signal in the frequency domain.

has occurred. When the signal is filtered in reconstruction process replicas will affect the result.

Prealiasing can be avoided with a high enough sampling frequency or by using a pre-filter before sampling. The aim of pre-filtering is to employ a low-pass filter [21, p. 217] to remove high frequencies that would cause aliasing (see next section 3.2). In other words the input signal - the digital image - is blurred before sampling.

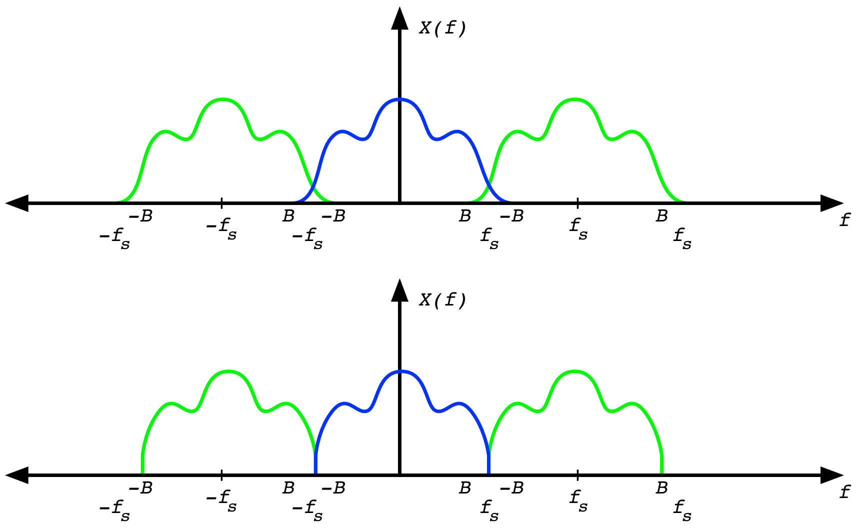

Figure 3.5: Replicas overlap (top) and some frequencies are lost (bottom) due to undersampling.

Then a sampling frequency satisfying the sampling theorem can be used as the highest frequencies of the signal are known. However, blurring before sampling may not be possible in practice as a 3D world cannot be presented as a one signal - it has a multiple signals such as the geometry, lights and the shadows.

3.2 Reconstruction

The purpose of the reconstruction process is to reconstruct a sampled signal to its original form which in practice means removing the replicas.

Reconstruction is done by convolving the discrete signal with a reconstruction filter kernel H. In the spatial domain:

$$f_r(t) = \tilde{f}(t) * h(t) \tag{3.5}$$

In frequency domain:

$$F_r(\mu) = \widetilde{F}(\mu)H(\mu) \tag{3.6}$$

The spectrum of the sampled signal $\widetilde{F}(\mu)$ is the sum of an infinite sequence of shifted replicas of the original signal's spectrum. Here the reconstruction filter $H(\mu)$ is used for eliminating the extraneous replicas of the signal's spectrum and keep the original base-band centered at the origin. [41, p. 3] Successful filtering is

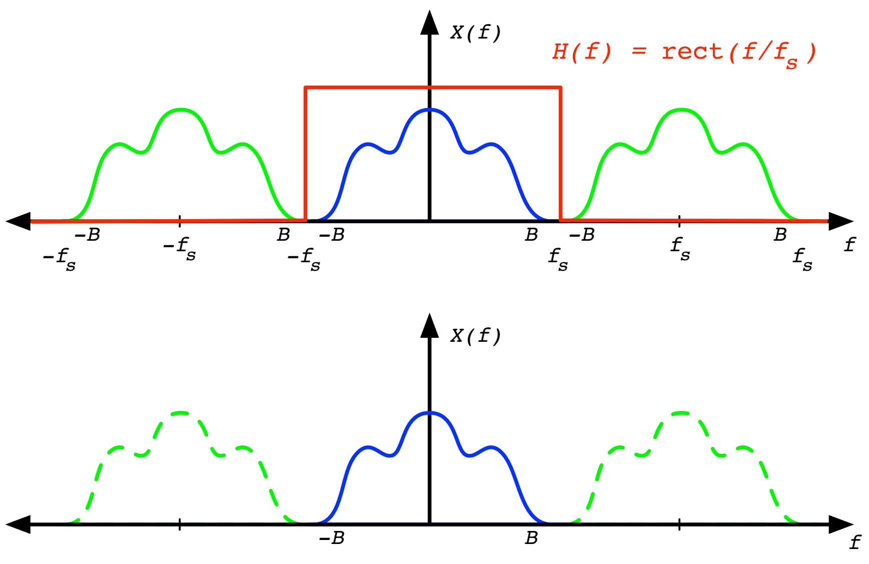

Figure 3.6: When a signal is sampled and filtered adequately (top) replicas can be filtered out successfully (bottom).

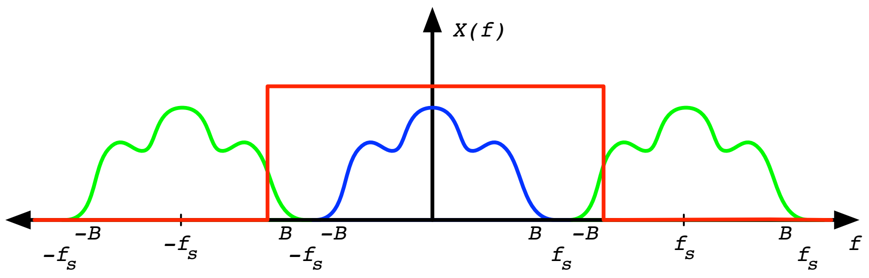

Figure 3.7: Properly sampled signal is filtered with inadequate filter. Replicas effect the filtered result.

shown in the figure 3.6. The filter lays fully only over the original signal and not the replicas as in the figure 3.7.

The figure 3.7 shows an example of a poor filter design. However, in this particular case the signal is sampled with a proper frequency filter that overlaps with replicas and energy is leaked into a constructed signal, which appears as aliasing.

Aliasing that happens in reconstruction process is called *post-aliasing*.

3.3 In Practice

The target resolution of rendering plays the role of sampling resolution. When rendering a frame, a 3D scene is sampled with the texture resolution and processed in order to be saved to a texture for later use or shown on the display. Especially in the real-time rendering, the render target resolutions are carefully selected considering the amount of memory available in hardware and the performance goals.

Rendering content also tends to have features that might cause infinite frequencies such as sudden changes in color, shadow boundaries and small details. In such cases, evaluating the sampling frequency that would satisfy the sampling theorem is impossible and performing low-pass filtering first can also be impractical.

Therefore, the resolution in most of the cases cannot be chosen freely. For example double the display resolution could be used to minimize the aliasing, but this may not be possible due to performance reasons. It should be understood that in practice all the sources of aliasing cannot be avoided in sampling. Other ways to prevent aliasing must be considered.

Chapter 4

Anti-Aliasing for Real-Time Rendering

This chapter briefly introduces real-time anti-aliasing approaches in general and discusses more in-depth the methods chosen for this study.

The methods chosen for this study are multisample anti-aliasing (MSAA) (see section 4.2, fast-approximate anti-aliasing (FXAA) (see section 4.3) and temporal anti-aliasing (TAA) (see section 4.4). MSAA is the recommended anti-aliasing solution for VR [78], to which the other methods will be compared. FXAA is a widely used anti-aliasing method which can be easily implemented regardless of the renderer type. It provides a fairly good image quality with a small impact on performance. The third method is TAA, which has gained popularity especially during last few years and some parties speculated on whether it will work well in VR. [30]

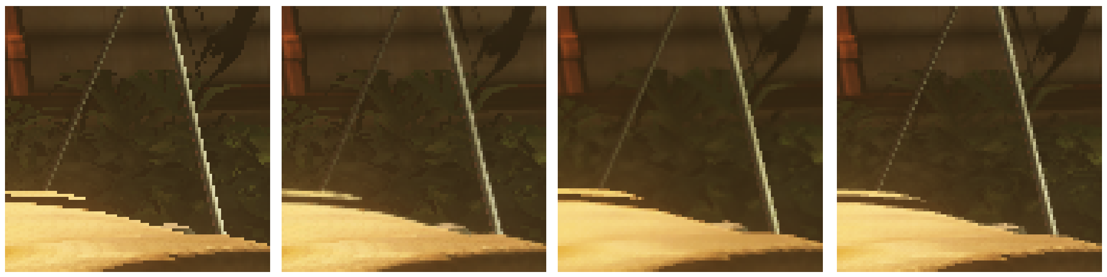

Figure 4.1: Anti-aliasing method comparison for a static image. From left to right: no anti-aliasing, FXAA, 4xTAA and 4xMSAA.

4.1 Overview

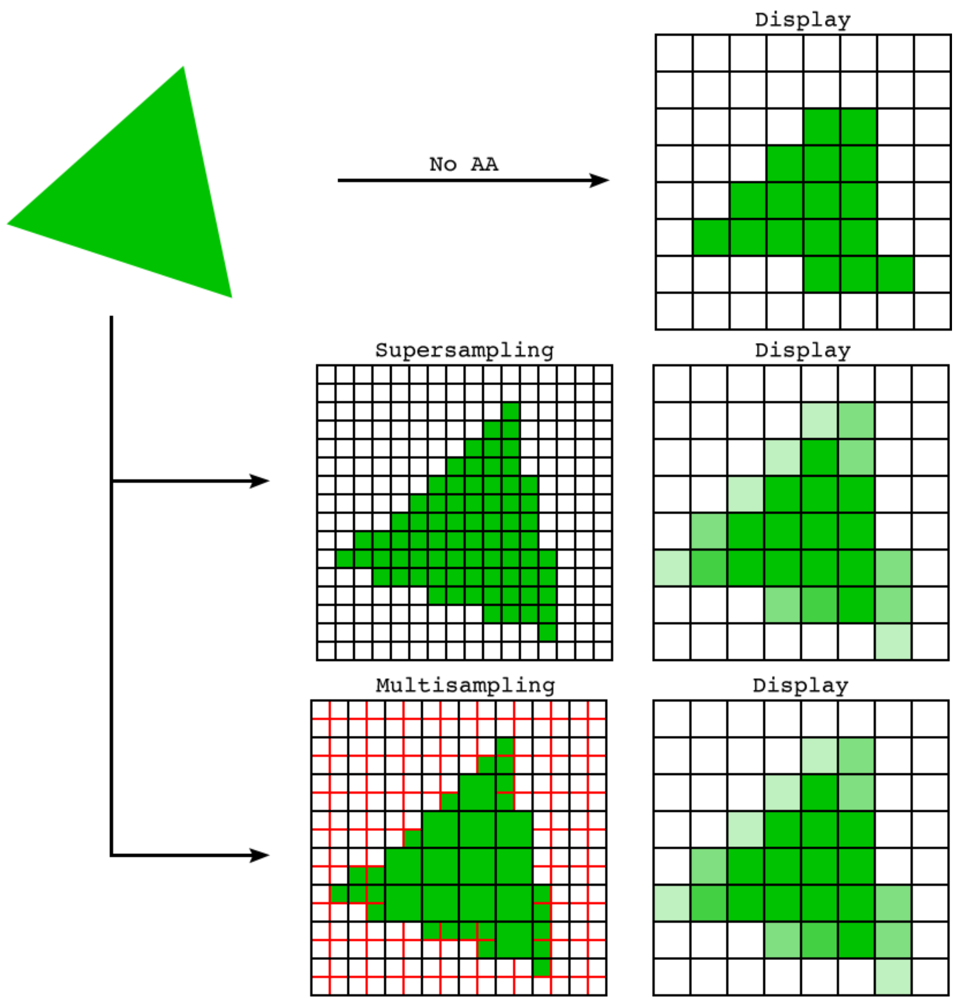

Figure 4.2: Example of how supersampling and multisampling work in a simple case. The triangle on the left is rendered without anti-aliasing (top), with supersampling (middle) and with four (4) sample multisampling (bottom). The supersampling result (middle-middle) is downsampled before shown on the display. The triangle that is rendered on the display with supersampling resembles the original one better than the one rendered without it. In multisampling (bottom-middle), each pixel has four samples (red). The pixels that are fully covered with the triangle do not need to save the triangle color for each sample, one pixel value is enough. The pixels that are partially covered have in color per sample. The results (bottom-right) is evaluated similarly as in supersampling.

Methods that increase the sampling frequency i.e. the resolution or the amount of samples taken within a pixel are called *super-* and *multisampling* methods (see figure 4.2). These methods are usually high quality, but can be costly in memory and performance depending on the chosen rendering style. [4, p. 126]

To reduce the costs, other types of methods have been developed and solving aliasing in the post-processing stage has become popular. Most of the modern anti-aliasing methods use post-processing for reducing aliased patterns.

Morphological methods find edges and analyse shapes on the rendered image (see figure 4.3) [15, 60]. Detected edges are smoothed, which softens the jaggies. As morphological methods function in the image-space they are usually relatively easy to implement. Most of these methods can be just added as a one extra independent effect before any other post-processing effects [15]. However, they lack the ability to handle the whole object with surroundings instead of just the edges or some certain shapes. Surroundings and context of the object affect how humans perceive an object [9, p. 147]. As only the edges or certain shapes are blurred the whole object might end up looking different from how a human would have perceived it from the original image.

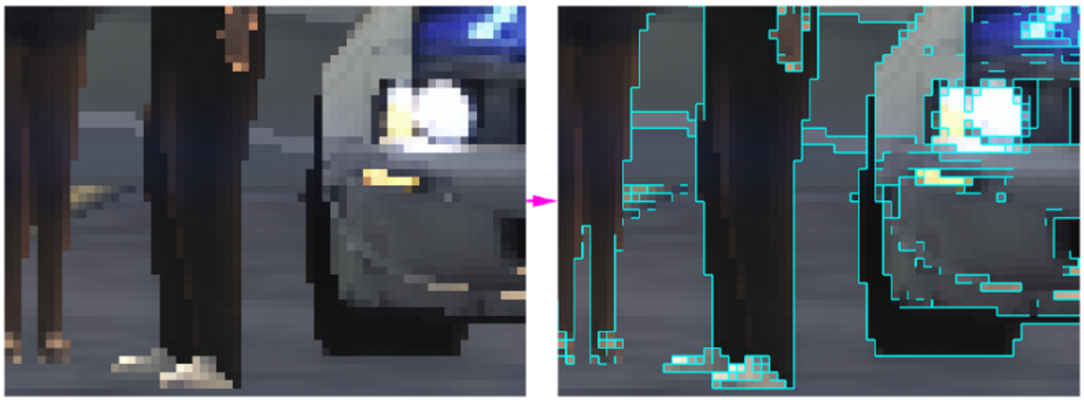

Figure 4.3: Example of edge detection result. Image from [15].

Temporal anti-aliasing methods take advantage of information from the previous frame. The previous frame information can be combined with the current frame information in order to smooth jaggies or reduce flickering. In the best circumstances the quality of temporal method can be close to quality of multisampling methods, but in the worst case next to nothing. [71, 81]

There are also a number of methods which combine multiple different approaches [18, 28].

4.2 Multisample Anti-Aliasing

Like supersampling, MSAA uses a higher resolution to test if multiple triangles cover sub-parts i.e. sub-samples of a pixel. For example 4xMSAA implies that four (4) samples are taken from different locations within a pixel. This requires a render target to be four times the original size, but will not necessarily make the entire rendering four times heavier.

When vertices are rasterized they are tested against all N sub-sample positions if they hit a sub-sample or multiple of them. If at least one sub-sample is covered by a triangle, PS is executed for that rasterized triangle. PS has to be executed only once per pixel per triangle regardless of how many sub-samples the rasterized triangle covers [43]. This saves the computing costs compare to supersampling.

After execution of the pixel shader, all the pixels in MSAA texture are averaged to determine one value for each pixel. This process is called *resolving*.

MSAA and rendering techniques The forward rendering can benefit the most from MSAA due to implementation simplicity and the ways forward rendering is meant to be used: with only a limited number of lights. The shading from one light to one pixel costs as much regardless of the rendering technique. The smaller the number of lights is used, the less computation cost there is per pixel. As MSAA in the worst case calculates lighting N times per pixel, the light count affects the performance considerably.

From the implementation and memory point of view, the pipeline uses only one render target (and Z-buffer) and geometry is processed and shaded in the same draw call. When MSAA is used, the memory usage grows as a multiple of the sample count. Rendering becomes only slightly heavier as triangles are tested against all the sub-samples (if a triangle covers a sub-sample), but the pixel shader, the shading, can be executed according to the coverage. After shading the MSAA render target has to be resolved, which is a fairly fast operation.

In the deferred rendering MSAA is not as straightforward. The deferred renderer requires a more complex implementation and a lot more memory due to the G-buffer. The memory requirements grow high as every texture in the G-buffer has to be created in a size multiple of the sample count. The separation of geometry processing and shading increases complexity of implementation and performance costs of MSAA.

Deferred renderer loses the ability to share per pixel shading computations for sub-samples covered by the same triangle. This happens because the G-buffer stores only a color per pixel per texture. The information of wheter a pixel was overlapped with different triangles, i.e if it was *complex* or not (normal pixel) can be detected and saved separately. Then shading can be evaluated for each sub-samples on complex pixels and per pixel on normal ones. [49]

However, this approach is still lacking the benefit of sharing shading results. If a complex pixel has a couple sub-samples covered by the same triangle, the shading has to still be computed for all those samples and cannot be shared. This problem is enlarged by a large number of lights.

In order to improve the performance of shading, the complex detection is usually separated to its own rendering pass, where G-buffer normals, depth and possibly a color buffer are used for finding complex pixels from the image (see section 6.3.2 for more details) [49]. This reduces lighting costs, but increases implementation and has a performance impact.

Advantages Regardless of wheter we are using a forward or deferred renderer, multisampling is still a high quality method which can only increase the amount of information and does not introduce artefacts or change how to content appears. MSAA can tackle jagged edges and sub-pixel aliasing as these both can be treated with adding samples [4, p. 128].

Disadvantages The main disadvantages in MSAA are the performance and memory requirements and its poor applicability for the deferred rendering. Although MSAA is improved from supersampling performance-wise, it is still quite heavy in terms of performance and memory.

MSAA requires also special treatment when it is used in a high dynamic range (HDR) pipeline. As the trend in computer graphics is to aim photorealistic look and use of physically based attributes and functions, HDR imaging has become popular. In the real-time HDR imaging, the HDR render target have to be tone mapped (unless the display is HDR capable) before displayed on the screen.

In the MSAA resolve, the pixel color is evaluated by averaging the sub-samples. Averaging is linear space operation and it works well when shading is calculated in low dynamic range (LDR) as LDR is always limited to a certain intencity range [65, p. 29]. Commonly, post processing effects are executed after shading and they function in HDR. This requires tone mapping to be executed for the MSAA-texture before resolving. The resolved texture has to be inverse tone mapped before the HDR post-processing effects, or use a LDR post-effects without an inverse tone mapping in-between. The other option is to make post processing effects support MSAA-texture and resolving is executed last after tone mapping. Both options have separate disadvantages in terms of performance and implementation.

4.3 Fast Approximate Anti-Aliasing

Fast approximate anti-aliasing (FXAA) is a post-processing anti-aliasing method developed by Timothy Lottes at NVIDIA [37]. This morphological anti-aliasing method is based on and inspired by MLAA [60], subpixel recontruction anti-aliasing (SRAA) [12], directionally localized anti-aliasing (DLAA) [5] as well as some other anti-aliasing methods.

The basic idea of FXAA is to detect image edges and smooth possible "jaggies". It is a proper method for solving aliasing on the edges of geometries and shader aliasing by reducing single and sub-pixel aliasing. [37]

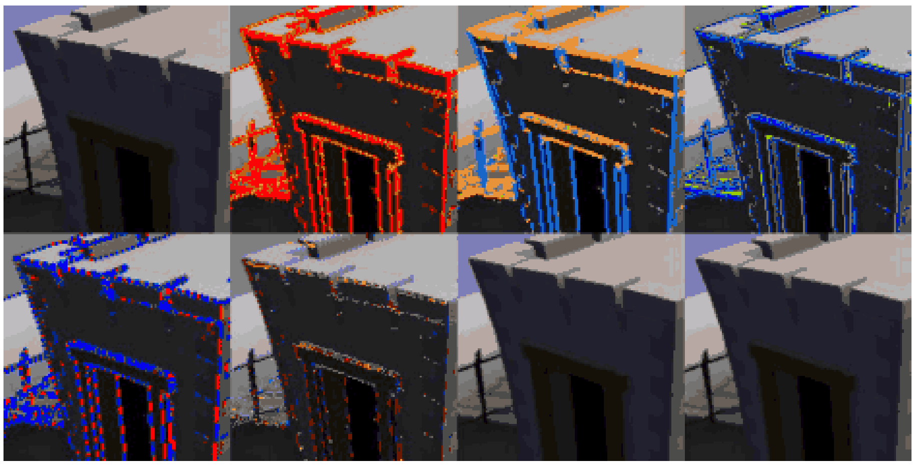

Figure 4.4: FXAA algorithm: steps 1 to 7 and final image from left to right. [37]

Algorithm works as following:

1. A non-linear perceptually encoded RGB color data input is converted to a scalar estimate of luminance. Estimated luminance is calculated only from red and green color channels as pure blue aliasing rarely appears in practice.

2. Local contrast is checked to avoid non-edge processing. The check uses pixel and its north, south, east and west neighbours to calculate minimum and maximum luminance differences (contrast) for the pixel and sub-pixel aliasing test. Pixels that pass the test are classified as horizontally or vertically oriented edges.

3. Given edge orientation, the highest contrast pixel pair 90 degrees to the edge is selected.

4. End-of-edges are searched along the negative and positive direction of the edge until a predefined search limit is reached or the average luminance of the highest contrast pixels pair which is moving along the edge changes significantly.

5. Given the end of the edge, pixel position on the edge is transformed into a sub-pixel shift 90 degrees perpendicular to the edge to reduce aliasing.
6. The input data (texture) is re-sampled given the sub-pixel offset.
7. A lowpass filter is blended in depending on the amount of detected sub-pixel aliasing.

The steps of FXAA algorithm are illustrated in the figure 4.4.

Advantages Compared to MSAA, FXAA is easy to implement regardless of renderer type. FXAA functions as a single-pass filter which only needs one full-screen pixel shader or compute shader applied to a post-processing chain. That is why the method does not require any extra memory cost and reduces visible aliasing in a light way. It smooths edges in all pixels on the screen including alpha-blended ones. [37, 40].

Disadvantages However, FXAA has the same issues with HDR imaging as MSAA. FXAA is designed to be applied to a post-processing chains after a LDR conversion. Therefore FXAA needs the same procedure than MSAA. [37]

Also, FXAA is not perfect in finding edges. As with any morphological method, "false positives" are possible and in some case, the anti-aliased image might end up appearing blurry as some edges are incorrectly smoothed.

4.4 Temporal Anti-Aliasing

Temporal anti-aliasing (TAA) or temporal supersampling has become a hot topic during the past few years. The main idea of TAA is to use previous frame information to reduce aliasing [30, 56]. However, TAA is not as straightforward as MSAA or FXAA. There are multiple different recommended implementations ([30, 40, 56] to mention a few). This section will present the commons ideas among these methods. TAA is illustrated in the figure 4.5.

In multisampling, each pixel is rendered slightly from different positions in order to get more information of the sampled signal. TAA utilizes similar technique, but samples are distributed over frames. For example in the figure 4.6, four samples are distributed over four frames and the anti-aliasing result is gained by combining these frames.

In order to gain more samples, the frame is rendered in slightly different positions. For example by using the MSAA sample pattern [39]. Offsetting is achieved by moving the camera position slightly, in a sub-pixel scale, i.e. *jittered* prior to

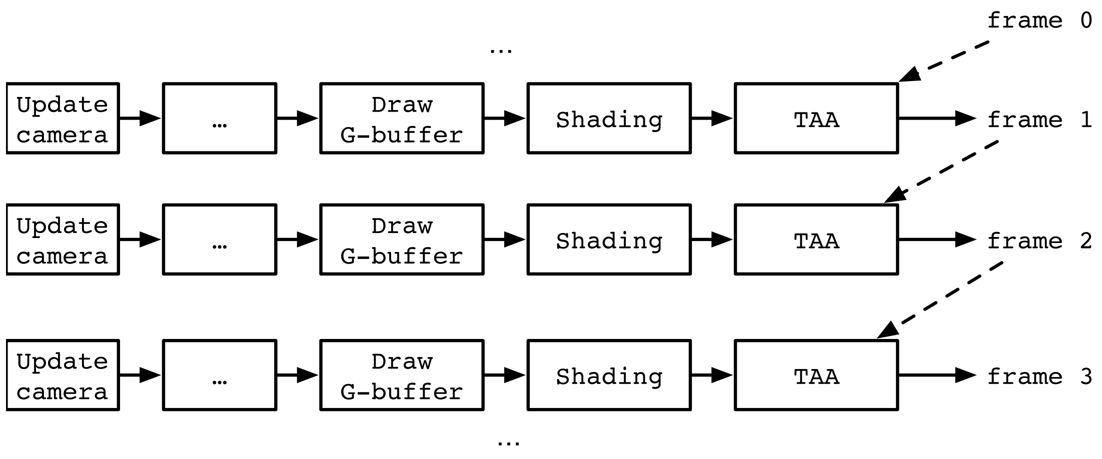

Figure 4.5: A rendering pipeline with TAA. At the start of the pipeline camera position is moved slightly. During the geometry draw task the velocities of objects are drawn to G-buffer. After shading the TAA pass is executed. In TAA pass, the previous frame is used for anti-aliasing the current frame.

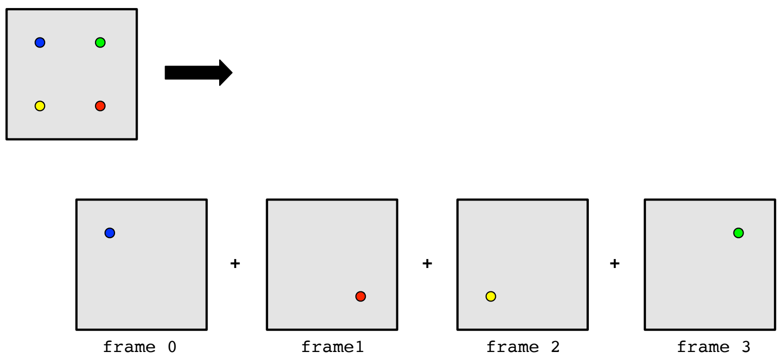

Figure 4.6: Every frame is the camera position is offset by a certain amount and anti-aliasing results are integrated over the frames. Image inspired by [71]

rendering. With a slight jittering the scene is sampled from slightly different positions which increases, and in a way doubles the amount of samples without extra costs.

TAA is based on using information from the previously rendered frame and usually in games objects and camera itself are moving. In order to get to the domain of the previous frame we need to know where the camera previously was compared to the current location. However, the objects can be animated too, and to get their previous frame locations the object velocity has to be computed.

For the object velocities, a *velocity buffer* is added to the G-buffer. The velocity buffer can be computed during the G-buffer draw from the the difference between the previous frame and current frame position. Using a velocity buffer in addition to using only camera reprojection (discussed in the next paragraph) in a screen space can increase the anti-aliasing quality on dynamic objects. If velocity buffer is not used, ghosting might occur on dynamic objects. In this case the objects that are moving "too fast" have to be left untouched. [30, 40, 56, 72]

After other parts of the rendering such as shading are executed (recall the figure 4.5), the TAA pass can be executed. Here the aim is to smooth the current frame using the information from the previous frame. In order to get more information for the current frame from the previous frame, the corresponding position on the previous frame for the current frame pixels have to be solved. The pixel position on previous frame position can be solved by *reprojecting* the current frame pixel position [56].

If the scene has no dynamic objects the reprojection can be done by transforming the current pixel position to world space, and then to previous frame pixel position (see the figure 4.7). In turn, if scene has dynamic objects, the previous frame pixel position can be calculated using the corresponding velocity in the velocity buffer. [56]

The current frame is sampled with the current pixel location and the previous frame with the reprojected position. However, the previous sample may not be valid due to possible occlusions. The objects in the scene or the camera might have moved so that the visibility of the objects was changed between frames (see figure 4.8). Therefore the sample has to be validated to diminish the possible artefacts such as ghosting.[71] The sample validation can be performed in different ways, which are discussed in section 4.4.1.

After validating the previous frame sample, the final color value for the screen can be composed. The current sample can be blended with the validated color value based on the distance between the luminances of the samples [56]. Another approach is to calculate a blend factor based on the sample velocity and other features [40]. The faster the sample moves, the less it affects the final result. Or the current and clipped color can be just blended using a static blend factor α: $P_n = \alpha * P + (1 - \alpha) * P_{n-1}$, where α is recommended to be 0.1. [71]

Finally, the composed image is saved to be used as a previous frame in the next frame. [40, 71, 75]

Advantages The advantage of TAA in comparison to MSAA is the lower cost in both memory and performance. The required memory heavily depends on how

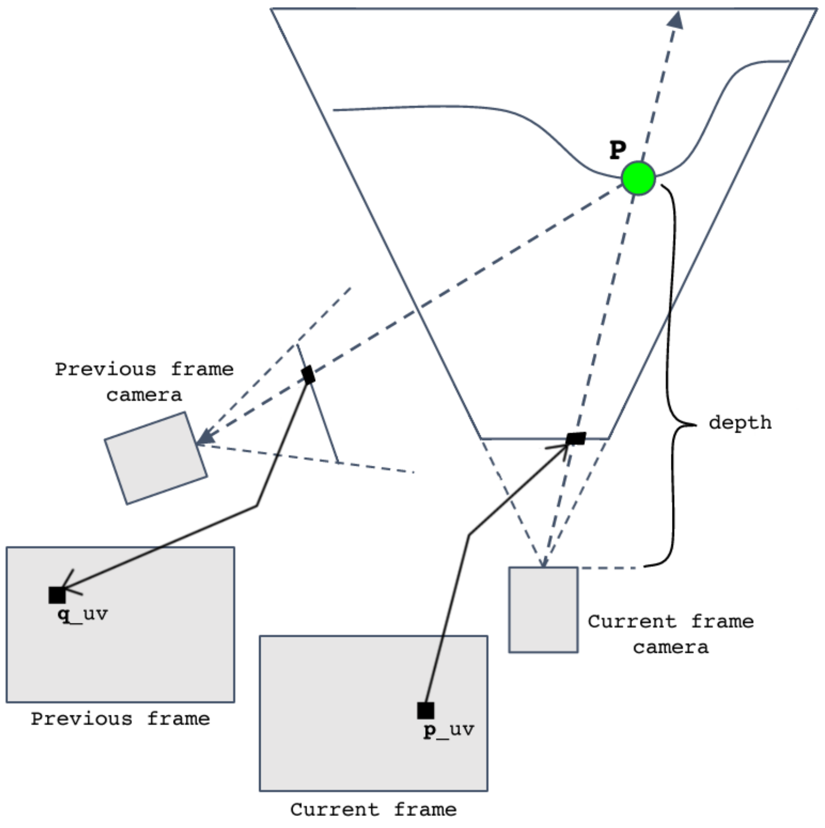

Figure 4.7: The previous frame position **q_uv** can be solved by reprojecting the current frame position **p_uv** to world space position **p** and then to the previous frame. The original image from [56].

much data is needed and saved for the next frame. Usually, only one extra texture for G-buffer (velocity) and one full-screen size texture or buffer for the previous frame or for the accumulation history buffer is enough. Also, reprojection and anti-aliasing are faster than performing multisampling, edge detection and resolving for deferred MSAA. [30]

Unlike MSAA, TAA is simple to implement regardless of the renderer type. Usually the same velocity buffer or the previous frame data can be used to boost the quality of effects like screen space reflections, bloom, ambient occlusion, depth of field or to double the samples in volumetric lighting. Motion vectors can be also found useful for upsampling resolution. [56, 74, 75]

TAA has been speculated to be a great fit for VR applications. In VR camera

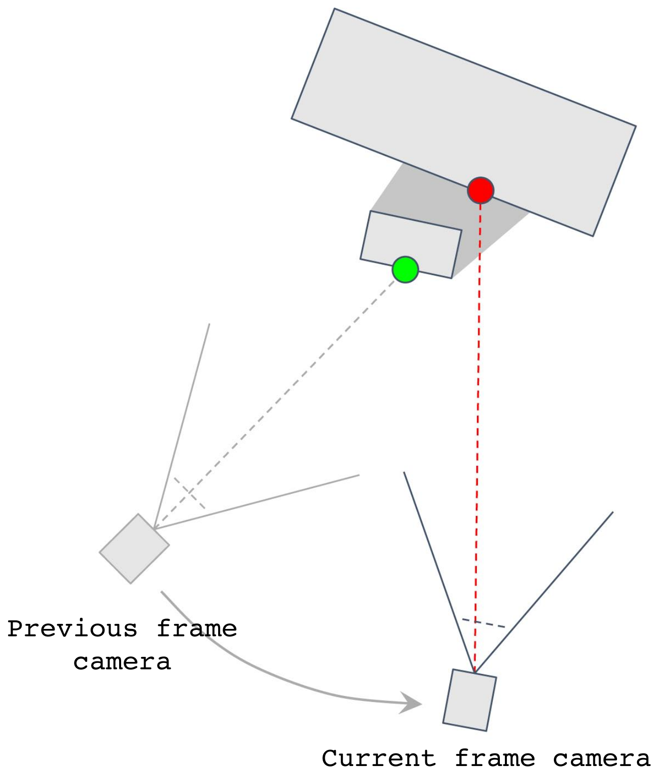

Figure 4.8: The camera sees behind the small box. However, when the position is reprojected to previous frame, the position is occluded by the small box and only the green dot can be seen. The samples have no relation. The original image from [56].

is constantly moving and temporal stability becomes really important. TAA can increase temporal stability greatly.[30, 56]

Disadvantages TAA is delicate for artefacts. As mentioned, ghosting or trailing artefacts can happen easily if implementation does not validate the previous frame samples properly. Robust validation of the previous frame sample might be even impossible or at least impractical to validate against depth, normals, other material properties or material ID. That is why simpler validation is a better option but that has its downsides too. Validation can have artefacts and usually kills some details from the image. Details can also be easily killed by cumulation of the history buffer. As the history buffer is cumulated frame by frame numerical diffusion becomes a problem. This error can be reduced by a back and forth error compensation and correction method. [30, 56, 71, 74]

Translucency or other techniques requiring multiple layers might be a problem.

TAA is usually run only to opaque surfaces and translucent materials are rendered on top of it. That results aliased appearance on the translucent materials. Another option would be to run the temporal pass to all different material layers and compose the final image, but this will come with a cost of performance. [30, 36, 56, 71]

Finally, TAA is a good solution for tackling flickering frames in other words temporal aliasing, but a downside of it is that it may also introduce some flickering. This can especially happen with a static camera. Flickering happens when neighborhood samples are missing some sub-pixel details and this history will be clamped out. When in next frame details appear frames starts flickering as the resolved color alternates between details missing and appearing, and will never converge. [30]

4.4.1 Validation Methods

The previous frame sample can be validated by clamping [70] or clipping [30, 71] it to the current frame sample neighborhood. In neighborhood clamping, neighboring pixels of the current position and the previous position are also sampled. The final result is calculated by linearly interpolating between the current and the previous frame samples which are first clamped between their minimum and maximum neighbors. [70]

Another solution for validation is neighbourhood clipping [30]. An axis-aligned minimum bounding box (AABB) made from the current sample's neighbourhood minimum and maximum values and the previous frame sample can be clipped against it (see the figure 4.9 right). However, clipping against AABB can introduce poor results, if AABB fits the neighbourhood colors too loosely (compare against the reference on the left in the figure 4.9). If the color sample is distant from the current sample ghosting might appear [71].

The clipping can be improved by taking *rounded* neighbourhood (figure 4.10 right) minimum and maximum values and calculating average of for minimum and maximum. This may help with AABB to be tighter and clipping and clamping results to be more accurate. [30]

Variance clipping is the third method for validating the sample to get even tighter AABB to clip the previous frame sample against. In variance clipping the box to clip against is calculated by calculating the first and second order moments from neighborhood sample values. [71]

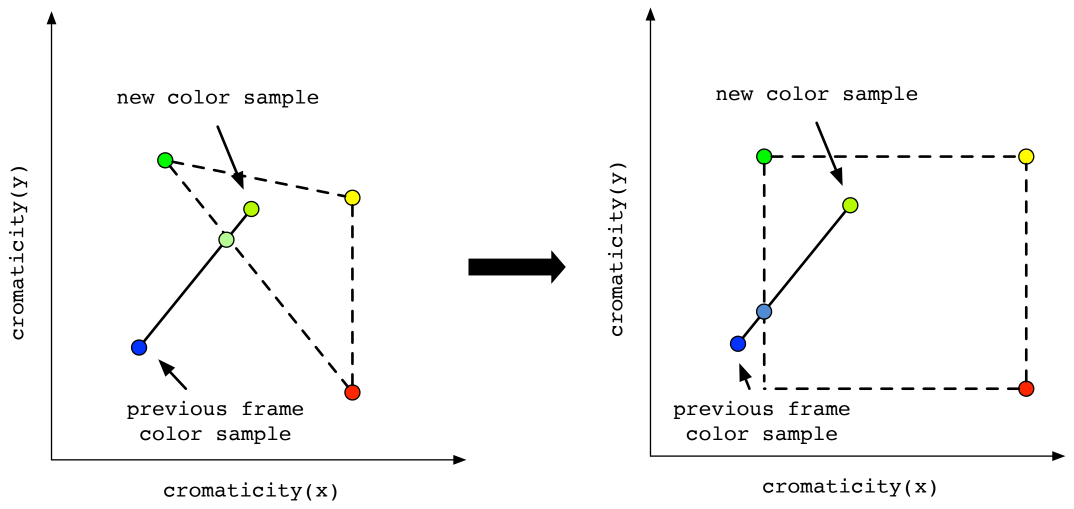

Figure 4.9: Left: The previous frame sample is clipped against the ideal convex hull of the current neighbour samples. The clipped value is close to the current color sample. Right: Clipping the previous frame sample against AABB box made of current frame neighborhood minimum and maximum values. The clipped value is far off the current color sample. Image inspired by [71]

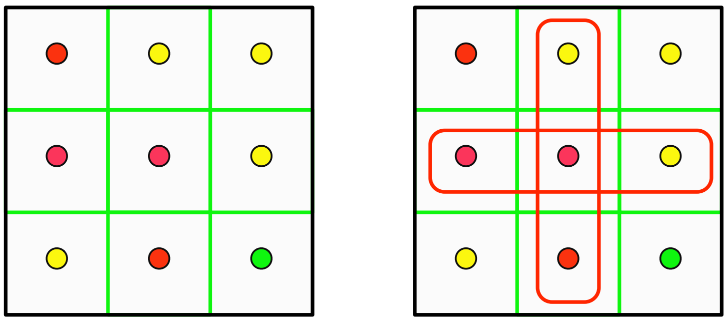

Figure 4.10: Sample neighborhood minimum and maximum values can be calculated from the whole neighborhood (left) or from rounded neighborhood (right)

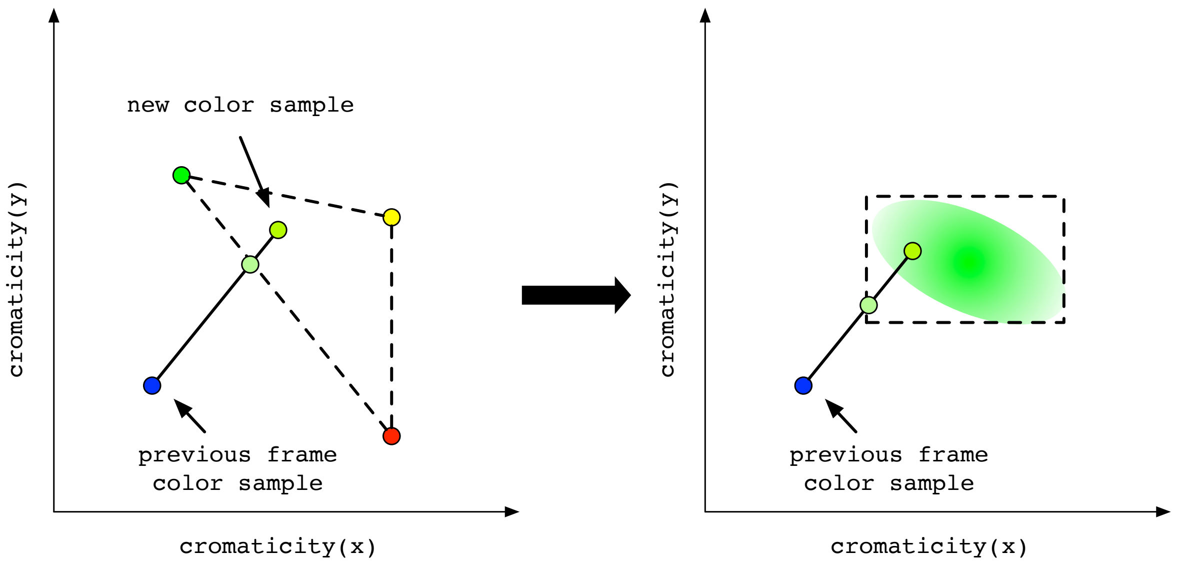

Figure 4.11: By creating AABB from variance, a tighter box can be created and better clipping results are achieved.

Chapter 5

Virtual Reality and Head-Mounted Displays

This chapter introduces what VR HMDs are, what they require from the hardware (section 5.1) and software (section 5.4) and also explain the common problems (section 5.2) as well as *presence*, a sense of being somewhere while in VR [64] is discussed (section 5.3).

5.1 Head-Mounted Displays

HMDs are used as a display for VR games and applications. The advantage of HMDs is that they enable presence (see section 5.3). Generally HMDs are small and light-weighted devices with wide field of view and lenses [47]. HMDs achieve stereoscopy by showing a slightly different image per eye. However, this is not done automatically. The application is fully responsible of what is shown in the lenses of HMD.

Currently, there is two major types of commercial head-sets: PC headsets and mobile headsets. The PC headsets are independent devices which are plugged into PC such as the Oculus Rift CV1 in figure 5.1. Mobile headsets use a mobile phone as a VR display. A mobile phone is mounted to a headset rig to function as an HMD, as shown in figure 5.2.

The recommended hardware requirements are listed in tables 5.1 and 5.2 for Oculus Rift CV1 and HTC Vive. The hardware requirements are high. Similar high-end PC hardware would be suitable for playing AAA games with normal PC, but not with HMD.

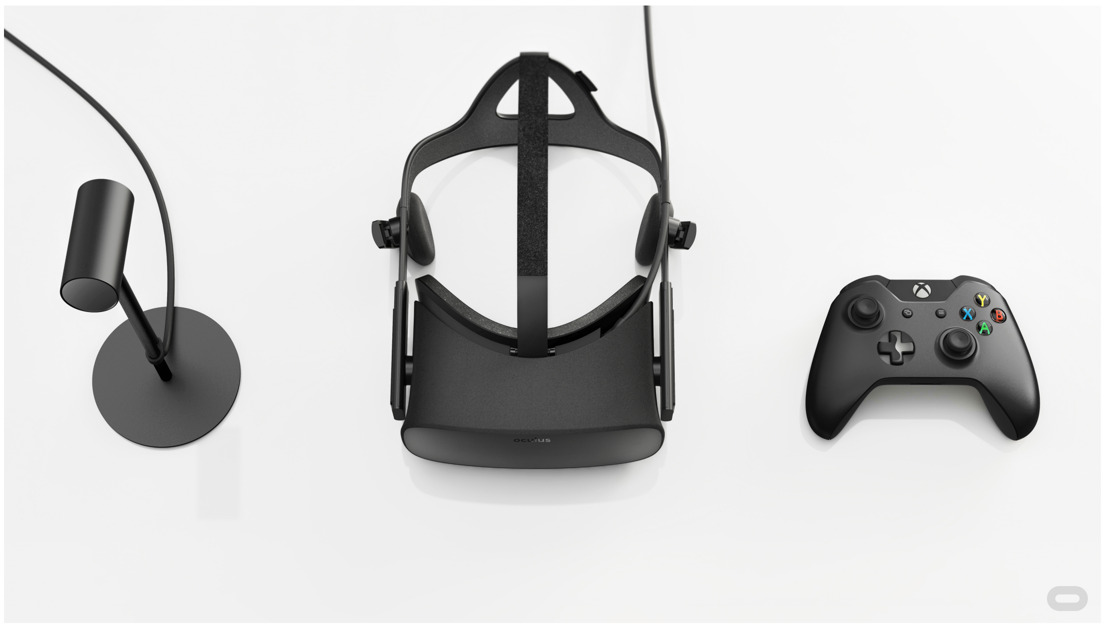

Figure 5.1: Oculus Rift CV1: the camera, the headset and the controller.

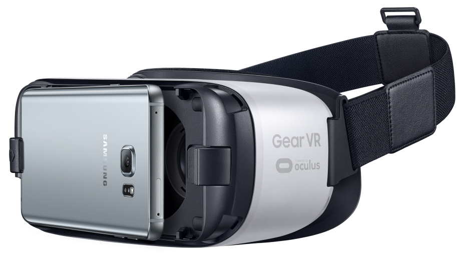

Figure 5.2: A mobile device partly plugged in Samsung GearVR headset.

5.2 Problems with HMDs

A downside of HMDs (for Oculus Rift or HTC Vive) is that they require a lot of computing power and if it is not available problems start to arise. Having an insufficient amount of rendering power can lead to poor FPS or latency which can break presence (see section 5.3). When an application cannot render frames at a consistent 90 FPS for a 90 Hz HMD images may ”judder”. When a new frame is not ready at the right time, the headset will show the previous frame which

	Oculus Rift CV1
GPU	NVIDIA GTX 970, 4GB / AMD R9 290, 4GB
CPU	Intel i5-4590
RAM	8GB
OS	Windows 7 SP
Video output	HDMI 1.3
USB Ports	3x USB 3.0 ports plus 1x USB 2.0 port

Table 5.1: Oculus Rift CV recommended PC specifications [53]

	HTC Vive
GPU	NVIDIA GTX 970, 4GB / AMD R9 290, 4GB
CPU	Intel i5-4590 or AMD FX 8350
RAM	4GB
OS	Windows 7 SP1
Video output	HDMI 1.4 or DisplayPort 1.2
USB Ports	1x USB 2.0 or greater port

Table 5.2: HTV Vive recommended PC specifications [25]

is warped with the current rotational pose of the headset in HMD. The problem of this technique it is only able to compensate for the rotational pose, not the movement of dynamic objects or user movement in the virtual world and for that reason the image can appear "juddery" (see figure 5.3) [78].

Low frame rates as well as the high latency can make users feel nausea. When virtual world is not responding fast enough it conflicts between visual and bodily senses, which leads to motion or simulator sickness. [51]

Oculus Rift's and HTC Vive's safety manuals report many possible health and safety issues and warnings. PC headsets are not recommended to use if a user is pregnant or elderly or has issues with heart, anxiety disorders or post-traumatic stress disorders[52, 77].

A lot of possible discomfort issues are also reported. Users might experience seizures, loss of awareness, convulsions, involuntary movements, dizziness, disorientation, nausea, light-headedness, drowsiness or fatigue. Different forms of eye strain and discomfort are known to be pretty usual. [52, 77]

One known reason for eye strain and discomfort is vergence-accommodation conflict (VAC). Both accommodation and vergence are oculomotor depth cues,

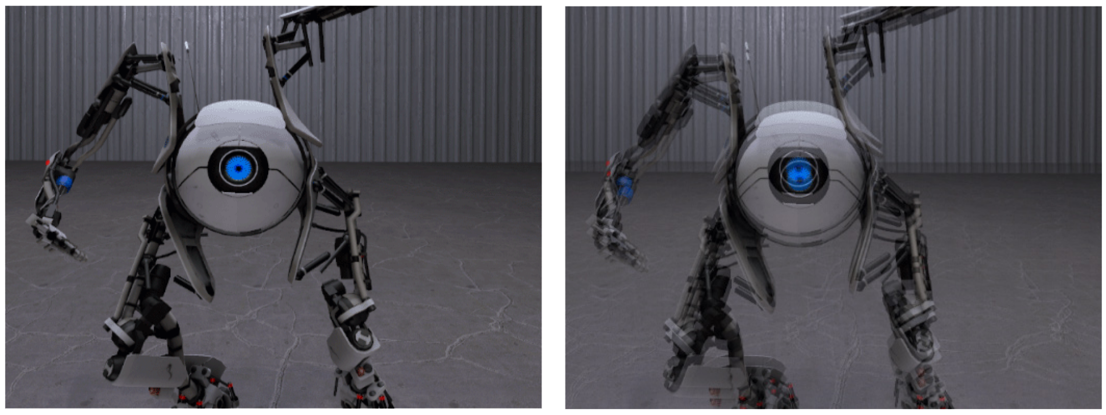

Figure 5.3: Left: Correctly presented image. Right: Image could not been finished under 90Hz, and previous frame was warped current pose. Image is ”juddering” due to change in animation between frames. Image from [78]

which are especially useful for perceiving the depth of near-by objects. Both are closely related to the retinal cues of blur and disparity. The retinal blur drives the muscular process of accommodation for adjusting eye lenses to focus at different objects to minimize the blur. Vergence is the simultaneous movement of the eyes to obtain or maintain fixation to an object, and is droved by the retinal disparities. These motor responses are normally coupled: a change is one is accompanied by a change in the other. They are still sensed individually. [9]

The current HMD design is such that VAC happens without exception, but it differs from person to person how discomforting it is. VAC happens when focal distance is fixed at the physical display while vergence distance varies depending on the virtual content and where viewer chooses to fixate. This forces accommodation and vergence to function independently and conflict with retinal cues. See figure 5.4. [54, 66]

For example, Oculus Rift CV1 has a fixed focus on certain distance. VAC can happen when the viewer is trying to focus on other distances, but blur or disparities remain constant.

Another problems for HMDs and VR applications in relation to human visual system is *binocular rivalry*. Binocular rivalry means that the brain alternates between the views that both eyes see if the views could not be fused [9, p. 292]. For example if left and right eye see slightly differently oriented object, the images cannot be fused and bistable vision occurs. Differences in color, luminance, motion, velocity, form, size and contrast polarity can trigger rivalry [8]. However, brain is able to fuse slightly miss-matched images. Some studies have tried to take advantage of this feature by studying subjective quality of asymmetric stereoscopic

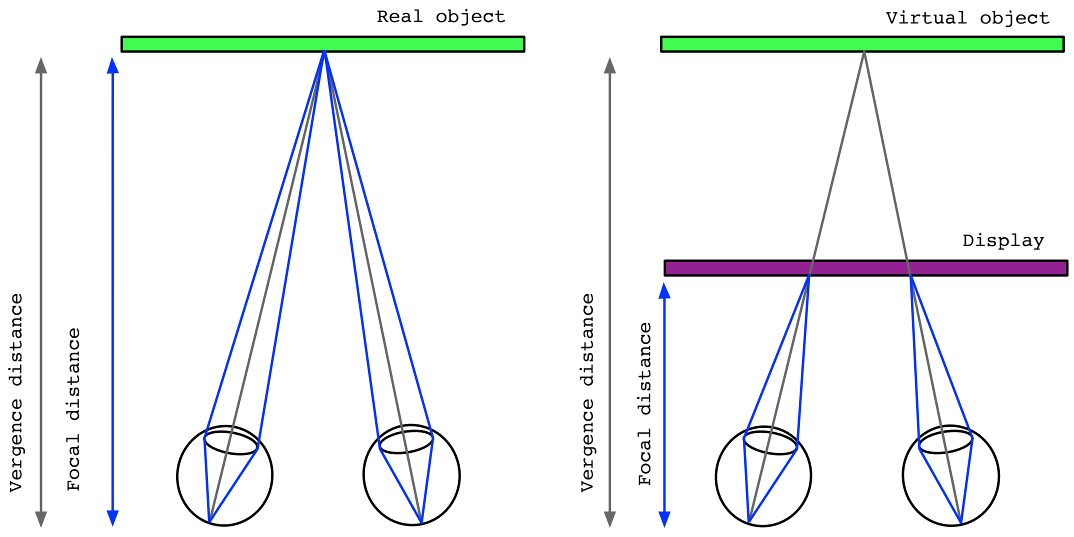

Figure 5.4: Left: Accommodation and vergence behavior on real-world situation. Right: Accommodation and vergence are forced to decouple with a stereoscopic display.

video (for example [2]).

The requirements and restrictions set by binocular rivalry are discussed in the section 5.4.

5.3 Presence

The quasi-magical feeling of presence is the sense of being in and belonging to a virtual world while located in physical world. Presence is build from *focus* [19], *involvement* and *immersion*. The virtual world has to have something that catches user's attention and takes his/hers focus. The more focused the user gets, the more involvement he/she has. Fatigue and discomfort lessens the involvement, which lessens the sense of presence. [80]

Immersion refers to the objective level of sensory fidelity provided by a VR system, whereas presence is a human reaction to it [67]. Usually, immersion is more likely to happen with head-mounted displays than with regular monitors. With regular monitors users tend to feel more being outside of the virtual environment while HMDs provide isolation from the physical world. Also, if users feel awkward immersion is reduced although involvement stayed the same. [80]

Besides focus, involvement and immersion there are a set of requirements for display, optics and tracking and application.

Display requirements

- Display resolution [1, 10]
- Adequate refresh rate [1, 10]
- Low level pixel persistence [1]

Display resolution should be large; the larger the better. One reason for a large resolution is how HMDs are used. HMDs are worn near-to-eye, which causes humans to detect individual physical pixels. Although the pixels can be seen resolutions starting from 1080p per eye can enable presence [1]. In principle, as long as human eye can detect physical pixels, HMD resolution should be increased.

Adequate display refresh rate is generally considered to be at least 60 FPS and preferably 95 FPS [1]. Originally adequate refresh rate is derived from the critical flicker frequency (CFF) [9, p. 331]. CFF defines the highest frequency of flickering light which is seen as such. The higher frequencies are seen as a continuous light. The threshold of fusion depends on many variables such as light intensity and size. Therefore there are conditions when humans can perceive flickering of higher frequencies or experience fusion at lower ones. Under the best conditions human CFF is around 60Hz. But still, there has been studies that show that higher frequency flickering, in this case, 120Hz, can cause cognitive deficits and headaches in some circumstances [76]. All in all, the CFF is not an exact measure. It depends at least on the intensity and the size of the stimulus [24].

The time pixel remains lit is called pixel persistence. The longer pixel persistence time will smear pixels especially in fast motions. This is seen as blurring. Pixel persistence of less than 3ms is required for presence [1].

Optics

- optics [1]
- optical calibration [1]

Human visual system is extremely sensitive to small deviation, as explained in section 5.2 for binocular rivalry. It is essential to proper optics and correctly calibrated system [1].

Tracking requirements

- Accuracy of position tracking [1, 10]
- Low latency [38]

To enable presence and to avoid motion sickness, an accurate low latency position tracking is needed. Tracking should be accurate: of a millimeter in position and a quarter-degree in orientation. Also, a user action from movement to an updated display image should not take more than 20 ms. [1]

Application requirements

- Field of view [10, 58]
- Frame rate [10]
- Realism (of lighting) [10, 62]

A wide horizontal FOV starting from 80°[20] is required for immersion and to perceive peripheral visual cues, which contribute to balance and gaze stability [1, 35] and frame rate should respect the display refresh rate. If frame rate is lower than display's refresh rate "juddering" problems can occurs (see section 5.2).

Early studies showed that spatial realism or environment naturalness was a factor of presence [80]. For example enhancing virtual environment with dynamics shadows and reflections [68] might increase the felt presence as well as better quality textures and lighting [82]. The presence is strengthened by realism or naturalness of the virtual environment. However, a lack of realism does not prevent users feeling it [62].

5.4 Virtual Reality Applications

Virtual reality sets high requirements on applications. The application has to be able to render high speed to a large render target, use a large FOV and ensure that rendering techniques are minimizing discomfort and maximizing the visual quality.

On desktop, an application has to be able to render a single display-sized image (usually from 720p to 1440p) at a framerate of 30 to 60 FPS. This means 55M to 221M pixels to be rendered per second. For VR application the requirements are much higher. VR application should render two images in a large resolution (1332x1586 per eye recommended on Oculus Rift CV1) in 90 FPS. This results 380M pixels per second. The high amount of pixels per seconds explains why rendering for HMD is challenging in performance-wise.

However, in VR all the pixels do not necessarily need to be rendered. A large number of pixels are actually blurred or lost when the rendered image is distorted to HMD lenses (see figure 5.5). Only a limited area in the center of the image is seen sharply. There are ways to prevent those pixels from being rendered. The simplest ones simply discard the pixels on certain areas [78] and more advanced are rendering the outermost areas with lower resolution with help of external APIs [50].

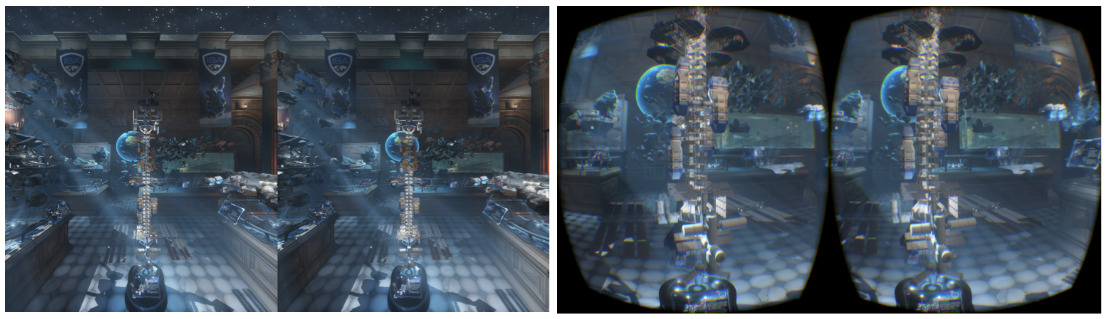

Figure 5.5: Left: Rendered images for left and right eye. Right: Left and right eye images after distortion.

The rendering techniques has to be chosen so that the performance goals can be achieved. The frame time budget to render both eyes is approximately 10ms framerate target being 90FPS and as 10% head room should be left for the VR API [78]. Executing the whole rendering pipeline as such for both eyes is a simple approach, but it also wastes performance. There are aspects that are not going to change per eye, such as drawing shadows and physical or particle simulations. Otherwise it is mostly matter of used rendering features and developer efforts, what can be shared and what cannot. Chapter 6 gives an example how the pipeline was designed in order to reach performance goals.

Usually the recommended style of rendering for VR is forward rendering. The main reason for this is that image quality can improved greatly and cheaply with MSAA [78]. Also, heavy post-processing effects are not used for both performance and suitability reasons. VR applications do not usually implement them because in most cases post-effects are screen-space effects. Effects that function in screen-space, i.e. manipulate image data can introduce binocular rivalry and should be avoided or used very carefully. Also, different post-processing camera effects such as depth of field and lens reflections are popular on the desktop side, but should not be used in VR for obvious reasons: in VR, user is not watching the virtual world through camera lens, he/she is watching it through his/her own eyes. Thus camera effects do not make sense.

Chapter 6

Performance Tests for VR: VRMark

This chapter introduces the software used in this study: VRMark. The rendering engine as well as the implemented anti-aliasing methods are described.

6.1 VRMark

VRMark is a benchmarking software targeted to test the performance of VR ready systems. VRMark has two different benchmarking performance tests: Orange Room and Blue Room. The Orange Room measures whether the tested system meets the recommended hardware requirements for Oculus Rift. The Blue Room, the more demanding benchmark of the two, is designed as a high-performance PCs paired with futuristic HMDs.

Performance tests can be run without the VR HMD on a desktop monitor. The tests simulate the same workload which would be running if HMD would be attachedd. Two views are rendered to a split window. The tests are run at high resolutions just like in real HMD use. The target frame rate on the performance tests is 90 FPS as both for Oculus and HTC Vive the display refresh rate is 90 Hz. Hitting that rate ensure the best experience for the user.

6.2 Technical Details

VRMark is implemented on DirectX 11 and supports both OpenVR and Oculus SDKs. VRMark hardware requirements are described in table 6.2.

	Orange Room	**Blue Room**
OS	Windows 7 SP1	Windows 7 SP1
Processor		
RAM	2GB	2GB
GPU	DirectX11	DirectX11
GPU Memory	2GB	2GB

Table 6.1: VRMark hardware requirements

6.3 Rendering Engine

VRMark uses a deferred rendering engine with VR optimized pipeline. Simplified illustrations of the rendering pipeline is shown in the figure 6.1. In the figure, green covers the rendering tasks that are executed only once per frame: scene update, shadow map draw, particle simulations, physics simulation and geometry visibility evaluations. After shared tasks, the left eye is rendered. First the G-buffer is filled in geometry drawing task, then shading is computed and particles drawn on top of the shading. Finally post-processing effects, i.e. anti-aliasing (FXAA or TAA) and tone mapping are executed for the rendered frame. If MSAA is used, only tone mapping is executed. Then the same is executed for the right eye. Lastly, the rendered image is presented to a screen or an HMD.

6.3.1 Fast Approximate Anti-Aliasing

FXAA is implemented in the post-processing chain as described in the FXAA white paper [37]. FXAA is computed to a tone mapped texture after all the other post-processing effects.

6.3.2 Multisample Anti-Aliasing

MSAA is implemented in the following fashion:

- Multisampled G-buffer is drawn
- Complex pixels are solved and a single sample luminance and depth is outputted
- Illumination is multisampled on the edges
- Rest of the pipeline (particles, post-processing) uses single sampled resources

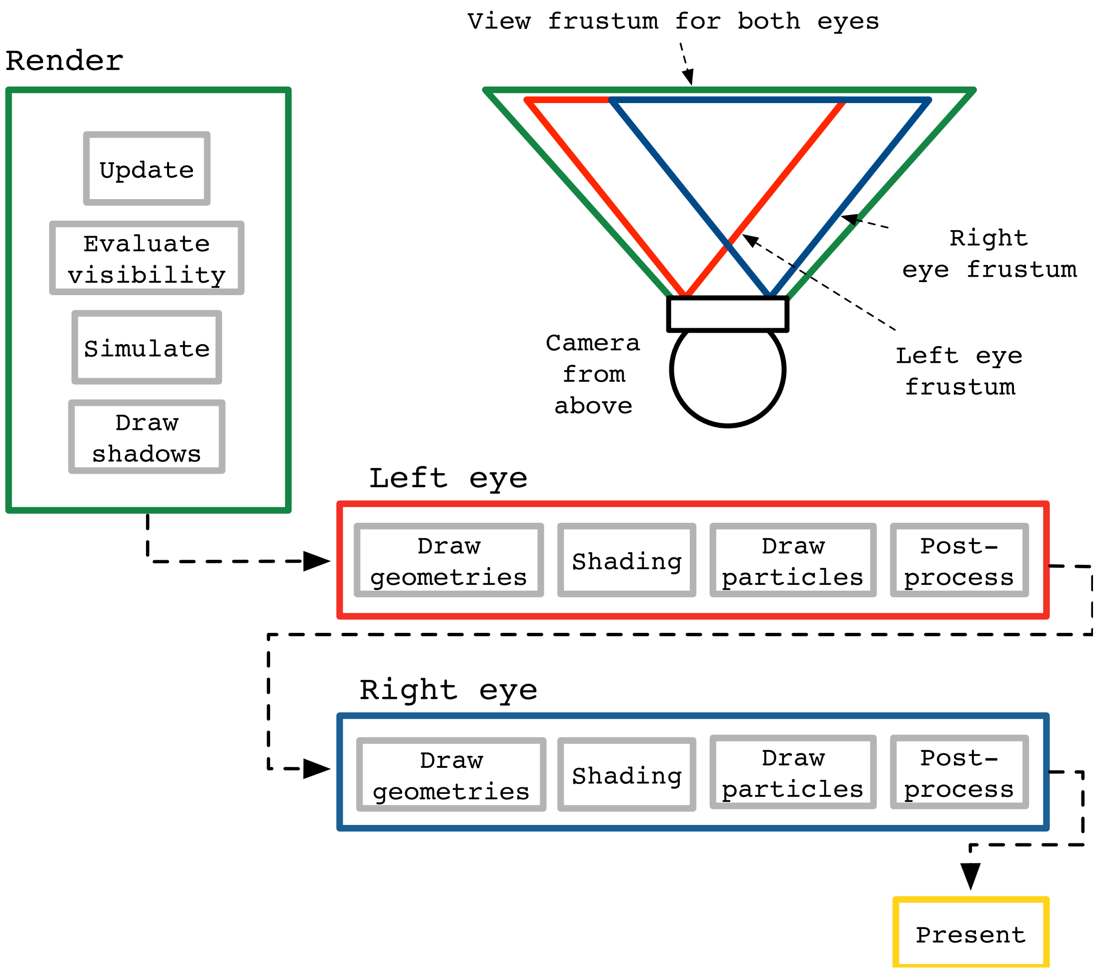

Figure 6.1: Simplified illustration of VRMark rendering pipeline. In the green box are the tasks that are executed in the beginning of the frame only once per frame. The green view frustum covers both left and right eye views and can therefore be used for geometry visibility evaluations. Also shadows are drawn and physics and particles are simulated only once per frame. Drawing geometries, shading, drawing particles and post-processing are executed twice per frame. First the left eye is rendered and then the right. Finally the rendered frame is presented to a screen.

Multisampled G-buffer In the beginning of every frame a multisampled G-buffer is created with a selected sample count. Supported sample counts are 2, 4 and 8. Multisampled textures are drawn in geometry draw tasks.

Detecting complex pixels Aliasing happens most commonly at the parts of the image where colors change rapidly, which usually happens on the edges of different object. However, different triangles can cover sub-parts of a pixel for other reasons as well. If multiple lights are used with deferred renderer the shading can be costly if all the samples covered by different triangles within a pixel are shaded. A more efficient way is to analyze the pixels that are covered by multiple triangles and have some relevance i.e. are complex. Complex pixels are detected using depth, normals, reflectance and luminance texture. Detection is performed in a separate edge renderer pass, which takes the multisampled G-buffer as resources and finds the geometry edges. Edges are searched first by comparing samples in the normals against a tolerance value. Then also depth, luminance and reflectance textures are analyzed in a similar way.

Illumination After the complex pixels have been detected shading can be executed. The shading (illumination) is evaluated for all the samples in complex pixel and only once per pixel for non-complex pixels.

The pseudo code implementation of the algorithm can be found from appendix A.

6.3.3 Temporal Anti-Aliasing

Temporal anti-aliasing is implemented in the VRMark engine in the following fashion:

- Camera position is jittered according to the sample count
- Velocity buffer is populated in the G-buffer draw
- TAA is solved in the post-processing chain before any other post-processing effects

TAA supports three different sample counts: 2, 4 and 8. The sample counts 2 and 4 follow MSAA sample patterns [39] and 8 samples using a random number sequence, 2,3 Halton sequence [22].

Camera position jittering In the beginning of every frame when camera is updated the previous frame matrices are stored and the current ones are jittered according to the chosen jittering pattern. The unjittered matrices are stored and used later in the TAA post-processing pass.

G-buffer velocity The G-buffer velocity buffer is populated. Velocities are calculated from the difference between the objects previous position and the current position in VS.

Post-processing TAA is performed in a compute shader pass and is the first effect in the post-processing chain. It uses the current G-buffer depth and velocities, surface illumination and the previous frame resolved color texture as a resources.

First, velocities are sampled for the current pixel and for its neighbors. The longest velocity vector among the neighbors is found and moved to the current pixel [71]. If the object was static i.e. had zero velocity, the camera i.e. pixel velocity is calculated from the difference of current and previous frame pixel position using world space. Details how this is done is describe in pseudo code in appendix B.

The previous frames resolved color texture can be sampled with the velocity. Then the current pixel neighborhood is sampled and minimum and maximum values are computed. Variance clipping is used for creating a box for the neighborhood colors of the current sample. The previous frame sample is then clipped against it.

Finally the color can be resolved. The blend factor is calculated to match the pixel velocity. Fast moving objects are really prone to ghosting, so only minimal amount of the previous frame information is used to anti-alias them.

The resolved color is saved into two different textures: one that is saved for the next frame and to one that is forwarded to the next post-processing effects.

Details of the implementation are in appendix B.

Chapter 7

Testing Methods

This chapter describes the methods for measuring and comparing different anti-aliasing methods by performance, memory usage and subjective visual quality.

7.1 Performance Testing

The performance of different anti-aliasing methods was measured by running VR-Mark Orange Room test (see chapter 6) with each anti-aliasing method. The test outputs the average FPS for the one-minute-length timeline run and is performed three times per each anti-aliasing settings. Average FPS is calculated from the runs to minimize variance.

The test is ran without an HMD, but the content is rendered to a split screen view (see figure 7.1), in a manner which emulates VR HMD use. The rendering resolution used in testing is 2664x1586. The CPU simulation that is present in the benchmark is disabled as this test is only interested in GPU performance.

The total memory consumption includes memory needed for all resources needed during the test.

Hardware configurations used for testing is described in table 7.1.

	System 1.	**System 2.**
GPU	AMD Radeon RX 480, 8 GB	NVIDIA GTX 1060, 6 GB
CPU	Intel Core i5-2500, 3.3GHz	Intel Core i5-2500, 3.3GHz
Memory	4 GB	4 GB
Operating system	Windows 10 (x64)	Windows 10 (x64)

Table 7.1: The performance test hardware configuration 1. and 2.

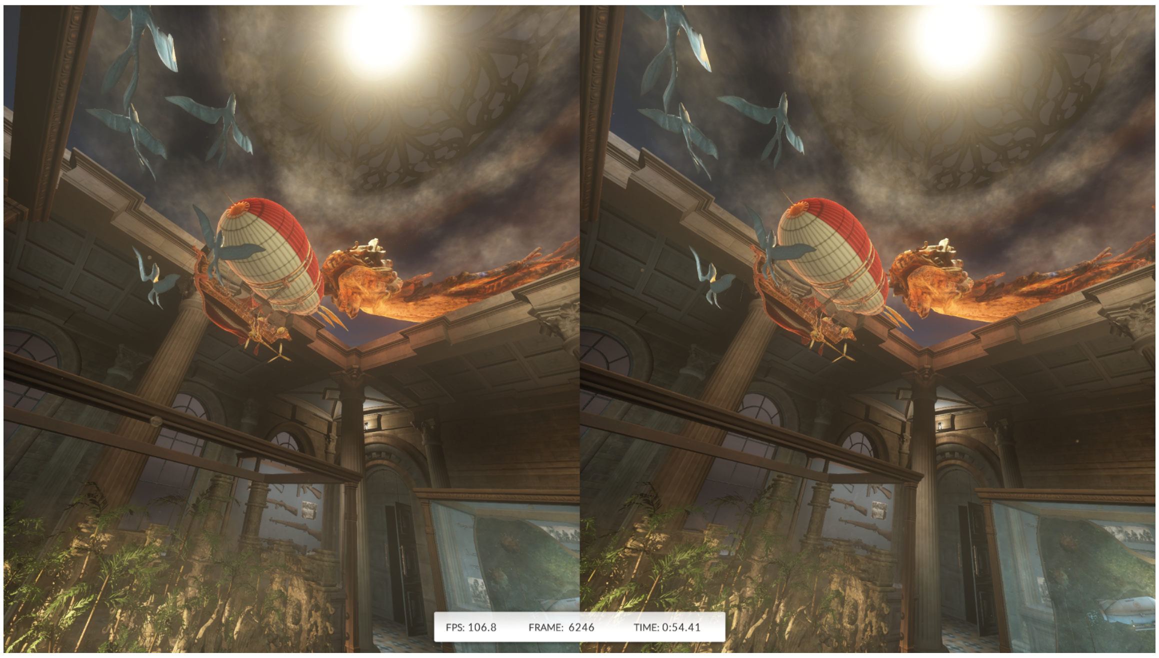

Figure 7.1: Screenshot of the VRMark 'Orange Room' benchmark test.

7.2 Subjective Quality Testing

Usually in the image or video quality studies the image quality may be measured by calculating differences to a reference image or a video stream. The difference to a reference image still may tell nothing about how *natural* or how *likeable* the image look and this is why subjective image quality test becomes useful.

Different anti-aliasing methods have their strengths and limitations. In the worst case, some of the methods might cause artefacts in the rendered image which lowers the image quality and can lead to a sense of fatigue. To be able to compare a methods ability to reduce aliasing on the edges or sub-pixel details, a scene used in subjective tests must be constructed in a way to ensure that aliasing on these certain areas is emphasized and other forms are excluded. To get an overall quality analysis, both *visual quality* and sense of *fatigue* are measured.

In this study the quality of experience (QoE) is measured for each anti-aliasing method. Research question for the subjective image quality test is: Which anti-aliasing method can produce the highest QoE?

The details of the subjective quality test are described in the following paragraphs. The study is designed according to ITU-T[26] and especially ITU-R BT.500-13 recommendations as well according to approaches described in a relevant previous research paper [3].

Test system	
GPU	Nvidia GTX 1080, 4.0 GB
CPU	Intel Core i7-6700K 4.00 GHz
Memory	16 GB

Table 7.2: Hardware components listed for visual quality testing system.

7.2.1 Preparation

Test Stimuli Stimuli are presented using a slightly modified version of the 3D scene in the Orange Room test from VRMark. Part of the content from the scene is removed so that participants can focus on specific parts of the scenes and to exclude unwanted stuttering which could happen because of low performance with heavy anti-aliasing methods. The scene offers highly detailed content, sharp edges and moving objects which can bring up different strengths and weaknesses of used anti-aliasing methods.

The scene will be rendered using following anti-aliasing settings:

1. FXAA (see 4.3)
2. 2xTAA (see 4.4)
3. 4xTAA (see 4.4)
4. 2xMSAA (see 4.2)
5. 4xMSAA (see 4.2)

where the number indicates sample count used with a method.

Extremes of the stimuli, the lowest and the highest quality, are a scene without any anti-aliasing and a scene with 8xMSAA respectively.

Material and Devices The hardware used for the subjective quality testing is presented in table 7.2. The virtual reality headset used in the test is Oculus Rift CV1. The hardware was chosen according to the performance needs of different methods. Performance on each stimulus was required to be above 90 FPS to exclude possible issues. For example, participants might feel more fatigue with lower frame rates when image starts juddering.

7.2.2 Test Procedure

Participants 26 male participants joined the study. Distribution of participants was not controlled in any way according to the answers. They were asked for

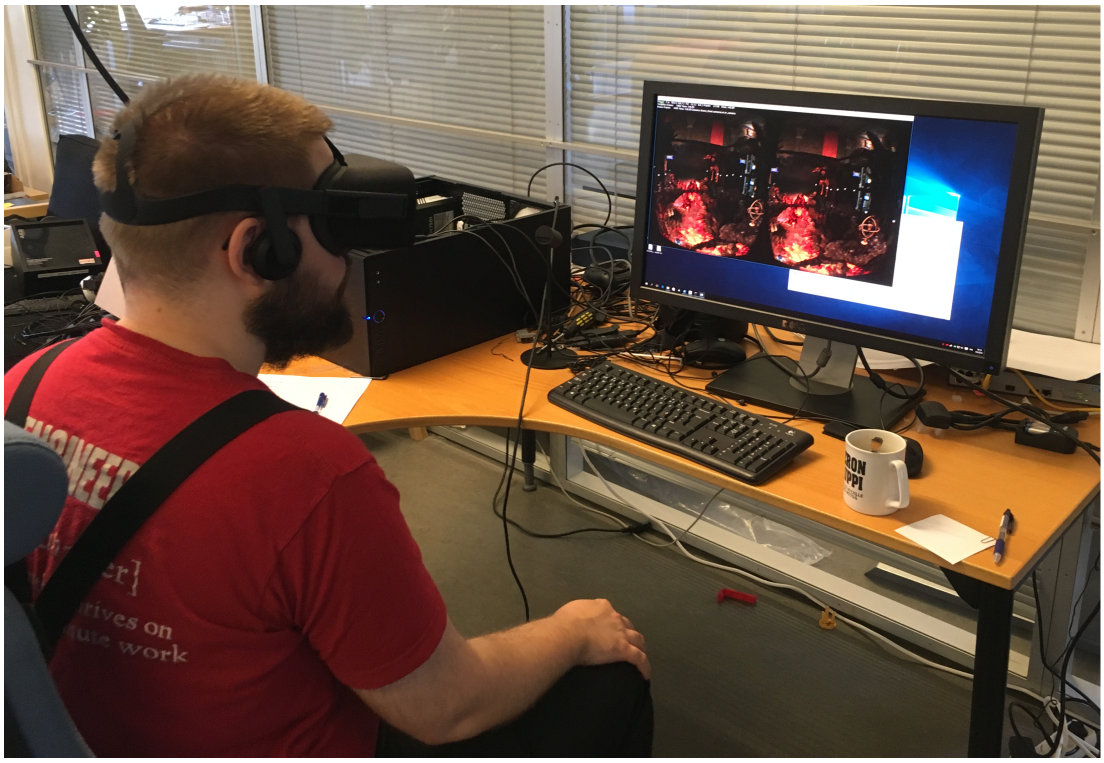

Figure 7.2: Participant taking the test.

their age and previous experience with HMDs. Distribution of participants by background information is shown in figure 7.3.

Visual and Stereo Acuity Before taking the test, a visual and a stereo acuity was tested. Near vision Lea numbers test [34] was used to measure visual acuity and TNO stereo test was used for measuring stereoscopic vision. Criteria for participation was normal near vision acuity (Lea-numbers ≥ 0.5) and normal stereo acuity $\leq$240 secs-of-arc (plate V; TNO test plates V-VII).

Instructions and Training Instructions of the test were given orally to participants. The test had ten (10) test clips which included five (5) unique stimuli with each shown twice in a random order. Participants had 60 seconds to view each stimulus and after that they entered their answer to an answer form. Each stimulus was rated according to the general image quality and fatigue. Visual quality was rated in integer scale from -2 to 2 (”very bad” to ”very good”) and fatigue with using emojis ”smiley”, ”neutral” and ”sad” meaning ”no fatigue”, ”some fatigue” and ”bad fatigue” (corresponding numerical values 1, 0 and -1). Participants were

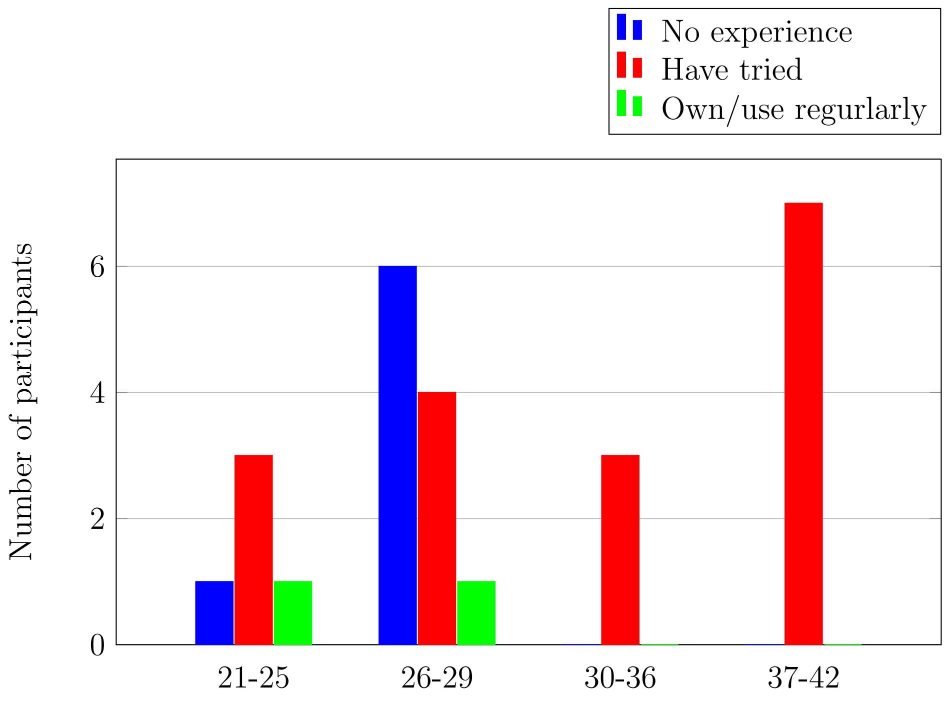

Figure 7.3: Distribution of participants by experience of headsets and age

allowed to take a break at any time between the test clips. They were also encouraged to comment their feelings and the comments were documented. Participants were made sure they understood the evaluation criteria fully.

After acknowledgement of instructions and rating, subjects were asked to adjust the HMD and its lenses. Subjects learned how to move in the test scene and they were shown the extremes of the stimuli to let participants know what to expect. They were also directed to look around if they did not move proactively during the training.

Training time was not strictly limited but lasted usually no longer than 15 minutes.

Test Each participant took one of the six (6) randomized stimuli sequences, which were produced before the study.

The test session lasted 35-45 minutes in total. Participants took the test individually with the test leader being present all the time. Besides rating visual quality and fatigue, participant's comments were documented during the test.

Statistical methods for results analysis The subjective test data was manipulation tested and analyzed with Kruskal-Wallis test [33] and Wilcoxon rank sum test [79]. The statistical difference i.e. the significant difference of means between multiple groups was tested with Kruskal-Wallis test for stimuli and visual quality, stimuli and fatigue, and between background variables. Also, as each stimulus was repeated twice per participant, the possible effect of instance occurrence was analysed: Was the first instance of stimulus always rated differently than the second one? For this purpose Wilcoxon rank sum test was used.

Chapter 8

Findings

This chapter presents the performance test results as well as the results from the subjective image quality tests.

8.1 Performance Test Results

Total memory usage for the benchmark test when using different anti-aliasing methods is presented in table 8.1. Performance test results for the test system 1 and 2 are represented in table 8.1.

FXAA has the smallest negative impact on memory and performance among the compared methods. FPS was reduced by only 3%. MSAA with any sample count had the largest performance (from 22% to 62%) and memory impact. TAA performance and memory impact are same for all sample counts. The performance was reduced by 10-11%. These results are independent from the hardware configuration.

Setting	**Total memory (MB)**
No anti-aliasing	930.0
FXAA	938.1
2xTAA	970.3
4xTAA	970.3
8xTAA	970.3
2xMSAA	980.4
4xMSAA	1044.9
8xMSAA	1173.8

Table 8.1: Total memory usage.

Setting	**System 1. (FPS, stdev)**	**System 2. (FPS, stdev)**
No anti-aliasing	116.9, $\sigma = 0.05$	135.2, $\sigma = 2.35$
FXAA	112.8, $\sigma = 0.11$	126.9, $\sigma = 0.09$
2xTAA	104.1, $\sigma = 0.30$	121.4, $\sigma = 0.00$
4xTAA	104.0, $\sigma = 0.07$	121.0, $\sigma = 0.49$
8xTAA	104.0, $\sigma = 0.25$	121.4, $\sigma = 0.08$
2xMSAA	89.1, $\sigma = 0.00$	106.1, $\sigma = 1.69$
4xMSAA	69.4, $\sigma = 0.00$	78.8, $\sigma = 0.00$
8xMSAA	47.0, $\sigma = 0.00$	51.9, $\sigma = 0.14$

Table 8.2: Performance test results for test systems 1 and 2: mean FPS and standard deviation (stdev)

Stimulus	**Visual Quality**	**Fatigue**
FXAA	-0.20	0.78
2xTAA	-0.76	0.46
4xTAA	-0.52	0.40
2xMSAA	0.46	0.76
4xMSAA	1.38	0.92

Table 8.3: The mean visual quality and fatigue per stimulus.

8.2 Subjective Quality Test Results

Manipulation Tests The subjective quality study results were analyzed against the background variables (age and VR experience) and checked if randomized test sequence or instance of stimulus occurrence had an effect.

Age ($p = 0.131, W = 3$), test sequence ($p = 0.559, W = 5$) or previous experience of VR HMDs ($p = 0.952, W = 2$) did not have any effect on visual quality scores. However, an effect was found between fatigue and age ($p = 0.007, W = 3$) and test sequence ($p = 0.0003, W = 5$).

There is a possibility that training may contribute to reduction of fatigue [73], but no effect was found between VR experience ($p = 0.776, W = 2$) in this study.

Each stimulus occurred twice during and the occurrence of the stimulus did not have effect on how it was rated. Wilcoxon rank sum test revealed no significant difference between stimulus instance of occurrence and rated visual quality ($p = 0.799, W = 7670$) or fatigue ($p = 0.786, W = 7670$).

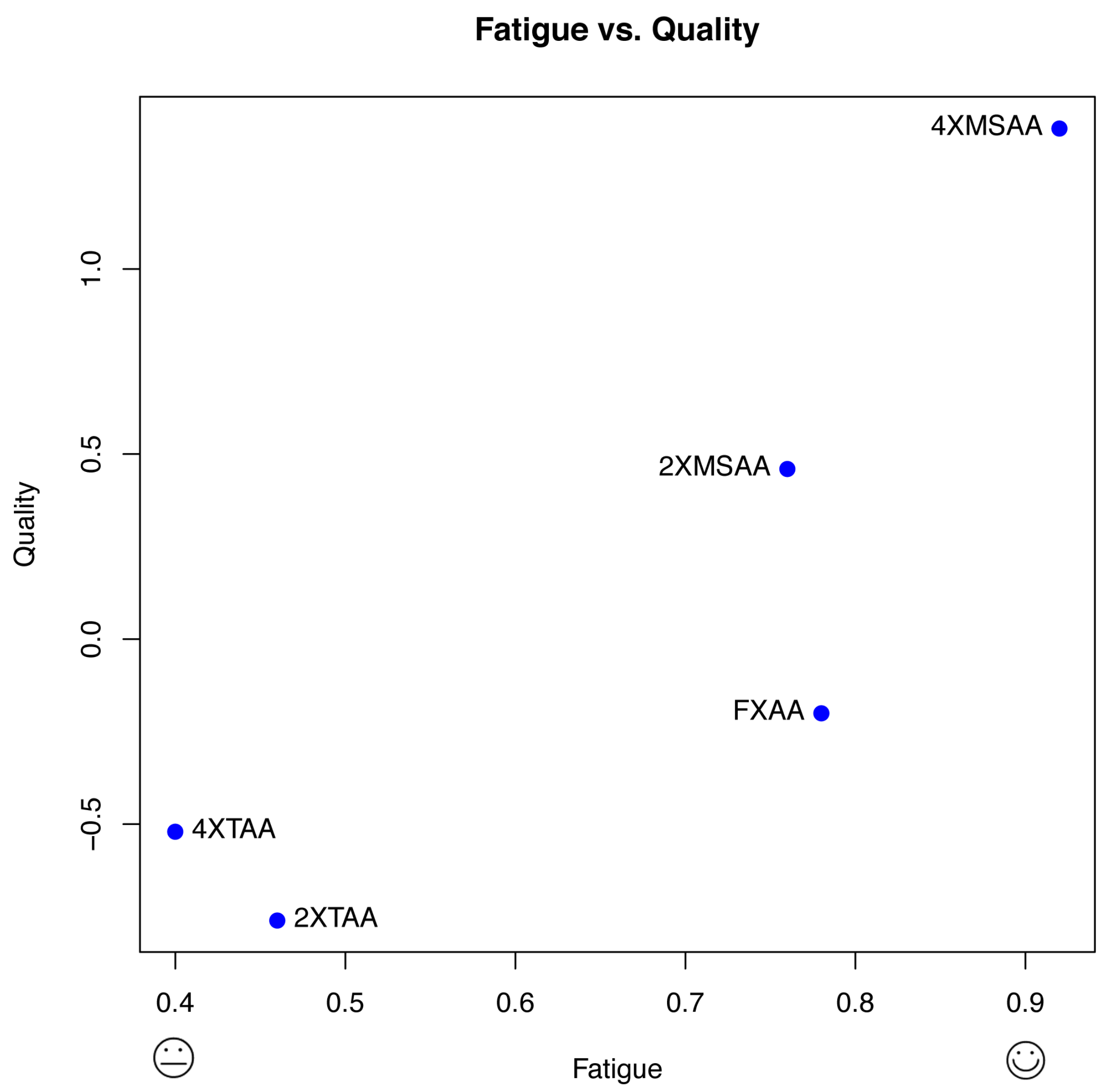

Figure 8.1: Mean score (fatigue, quality) per stimulus. Higher score is better in quality and less fatigue.

Result Analysis The test results for visual quality and fatigue of the stimuli are in table 8.3 and figure 8.1. The Kruskal-Wallis test revealed that visual quality ($p < 0.0001, W = 4$) and fatigue ($p < 0.0001, W = 4$) results are statistically significant i.e. quality and fatigue were not rated randomly.

4xMSAA was clearly the best in quality. The mean of the reported visual quality was 1.38 of the possible maximum score of 2.0. Also 2xMSAA achieved a fairly good results with visual quality mean of 0.46. It was surprising that participants could still notice the difference between different MSAA sample counts as some of them commented during the study that catching the differences of some methods was really hard during the test.

FXAA visual quality was considered worse than that of MSAA, but better than the quality of TAA. Visual quality mean for the method was -0.2, which is significantly lower than MSAA. Surprisingly some of the participants could identify FXAA during the study. They did not necessarily like the quality of FXAA but considered it better than TAA. Against the expectations, participants did not comment FXAA appearing blurry to them.

TAA was considered the worst in terms of quality in this study. The mean visual quality for 4xTAA was -0.52 and -0.76 for 2xTAA -2 being the lowest possible score. During the study many participants commented that they had difficulties to focus their eyes or that the image seemed annoyingly blurry to them when TAA was used. Some of them said afterwards that they would prefer a scene without any anti-aliasing to a scene with slightly blurry anti-aliasing.

Participants experienced nearly no fatigue with 4xMSAA (0.92), FXAA (0.78) and 2xMSAA (0.76). Unlike the others, TAA caused significantly more fatigue. The mean fatigue score for 4xTAA was 0.40 and 0.46 for 2xTAA. Interestingly the mean fatigue 4xTAA was worse than 2xTAA's, but visual quality was rated vice versa. This might have happened because 4xTAA integrates samples over 4 frames and thus gives a more stable result. Drawback of having more samples is possibility of increased ghosting because of the longer feedback chain, which seems to lead to greater fatigue.

8.3 Summary

The highest QoE in the subjective study was obtained with the highest performance impact and with the largest memory requirements. 4xMSAA lowered the performance by 41-42% (compared to no-anti-aliasing case) and this is a very considerable impact of performance. Memory requirements might not matter that much as high-end GPUs with large amount of memory are usually recommended or required for VR, and such card invariably are equipped with sufficient amounts of memory (recall section 5.1 tables 5.1 and 5.2).

FXAA is an interesting method. The method was the fastest (only 3% performance impacts) and participants did not or experienced little to no fatigue with it. The image quality was notably less than MSAA, but this may not matter much as long as the experience is not discomforting.

TAA provided the worst QoE among the methods, with small differences between different sample counts. The advantage of TAA is that the performance is predictable and the same across all different sample counts. However, the most worrying aspect is that participants experienced the most fatigue with this method and when visual quality increased (4xTAA compared to 2xTAA) the fatigue increased as well.

Chapter 9

Conclusions

9.1 Discussion

An alternative anti-aliasing method with a comparable quality to MSAA could not be found in this study. However, this study proves that there is a relation between anti-aliasing, visual quality and fatigue. 4xMSAA was significantly the best tested method in visual quality and caused the least fatigue in participants. The worst quality and the most fatigue was experienced with TAA. Where MSAA did not cause any negative reaction from users, TAA made them complain.

From the performance point of view the results are the opposite. MSAA (2x, 4x and 8x) were significantly the heaviest among these methods, whereas both FXAA and TAA impacted the performance only a little.

The reasons behind why different people perceived a stimulus better as quality than other are not clear. This study does not give detailed information of the features or properties of visual data in order to determine what makes one method better in quality over another. Further research should be performed this evaluation. Section 9.3 will discuss this in more detail.

Visual fatigue or discomfort can be cause of many different reasons [73]. For instance, MSAA was visually the best and it caused nearly none fatigue. This might be because MSAA does not introduce artefacts. It can leave some details aliased, but it won't add any blurriness or erroneous patterns, both of which can happen with FXAA or TAA. Theoretically, it is also possible that a screen-space methods - such as FXAA and TAA can introduce slightly different images for left and right eye, which would then introduce binocular rivalry. However, this may not be a cause of fatigue as at least for FXAA people felt nearly the same level of fatigue as they did with 2xMSAA.

In general, VAC can cause discomfort and eye strain regardless of the anti-aliasing method. TAA might be an extreme case of the phenomenon. Some of the participants were complaining during that TAA appeared blurry to them and they were not able to focus their eyes, and that was the primary reason for fatigue. It is also possible that as VAC, as a focal distance was available, but as the virtual scene seemed blurry eyes could not find focus.

Some participants noticed ghosting effects while watching TAA. It is not in this case the same effect as *crosstalk*, but it might appear similar. Crosstalk is an technology artefact which appears on active or passive 3D stereoscopic displays when information is leaked from one eye to another [29]. As said, crosstalk appears as ghosting or blurring. Crosstalk is known to cause visual discomfort depending on how much of it occurs [31].

Considering the results and the possible reasons behind them, MSAA is a primary candidate for further development. A few initial ideas are discussed in the section 9.3.

9.2 Limitations

This study had a few limitations on the anti-aliasing method implementations and in the subjective quality study.

Implementation limitations Temporal anti-aliasing implementation can possibly have issues or lack some details which would have produced better quality. TAA is not a standardized method and not as well documented as MSAA on deferred renderer or FXAA, which makes it hard to verify the correctness of the implementation.

Dependencies in the subjective study The manipulation test results of the subjective quality study revealed a few unwanted dependencies. Kruskal-Wallis test revealed a significance ($p = 0.006841, W = 3$) between reported fatigue and age of the participant. It is known that age has some kind of effect on cybersickness symptoms but the results of the studies are contradictory [6]. In this study, results show that participants born in close to median birth year (1988) suffered the most fatigue. As the participant age group size is only 6 to 7 this can be a coincidence.

Also some of the randomized test sequences were found to be harder than others. The reported fatigue and a randomized test sequence had a correlation in Kruskal-Wallis test ($p = 0.0002717, W = 5$). This could have possibly been avoided by performing a test study with a smaller group of participants to choose equally heavy test sequences. Although, this would have required more participants and time.

Visual fatigue versus visual discomfort Term 'visual discomfort' could have been used to address fatigue experienced during a stimulus. Term (visual) 'fatigue' usually means the discomfort experienced after a visual stimulus [73] and in this study is was used to address the immediate feeling while the stimulus was shown. This was realized after the study has already been conducted. The effect of this should be small as the meaning of the term 'fatigue' was explained to participants.

Gender of participants Gender distribution of the participants was not controlled and by chance the participant group was all male. Using both genders would have require a larger number of participants. However, testing only with a single gender might limit out some possible dependency between gender and reported fatigue. There has been studies that show some difference in motion-sickness or nausea between males and females. Though, the results of number of studies are also contradictory [59].

Expert participants Some of the participants had some or even extensive knowledge of anti-aliasing and computer graphics in general. This might have an effect on the reported quality. Experts might know where to look to be able to catch the mistakes and artifacts of the different anti-aliasing methods easier. This possible effect is impossible to measure as level of expertise was not documented. Number of participants was also too small to group the participants into non-experts and experts.

Visual environment An ideal solution for evaluating quality of anti-aliasing method would be isolated 3D scenes with particular content highlighting certain advantages and weaknesses of anti-aliasing methods. This approach did not suit the other parameters of the study.

9.3 Future Development

In order to bring the highest possible image quality and minimize the fatigue, the MSAA implementation for a deferred renderer should be improvement. It seems that the most of the performance cost comes from the multisampled shading. To reduce it the detection of complex pixels could be improved. This subject was tentatively discussed and experimented with M. Aizenshtein (personal communication, August 26, 2016). Preliminary experiments showed promising results. The current algorithm for finding complex pixels returned 15% of pixels. For comparison, the verbose version of Canny edge detector [11] returned only 6% of all pixels to be complex when the G-buffer normals were analysed. The normal version of the Canny edge detector reported 3% of pixels as complex from normals and 2%

from depth. Figure 9.1 shows a visual comparison for current implementation versus the Canny edge detector with normal settings using the G-buffer normal and depth textures.

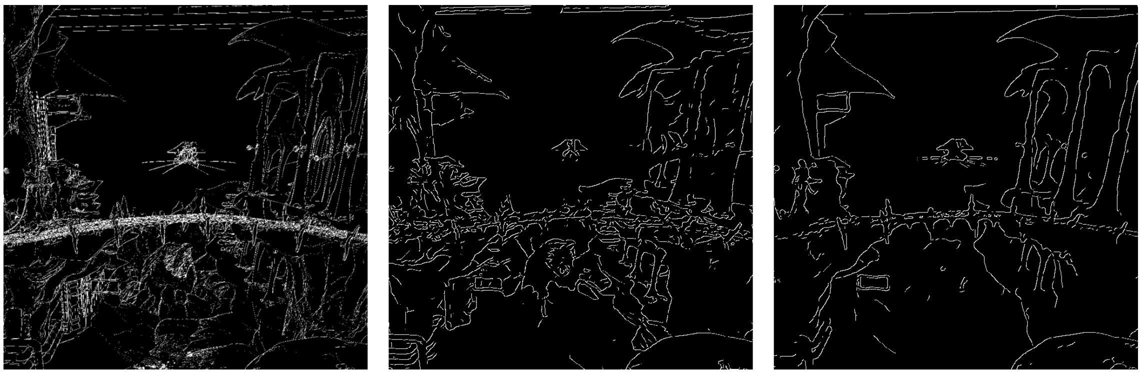

Figure 9.1: Edge detector comparison: current method (left), Canny detector with normal settings using normal texture (middle), Canny detector with normal setting using depth texture (right).

Another performance improvement would be possible by using DirectX 12 [46], or other graphics APIs with similar features. The complex pixels detection could be implemented using compute shaders and computed asynchronously at the same time as graphics pipeline work [42]. This performance gain was estimated to be as high as 30% to 50% (M. Aizenshtein, personal communication and ideation, August 26, 2016)

There are many anti-aliasing methods which could have been investigated further. For example SMAA and aggregate g-buffer anti-aliasing (AGAA) [7, 14] are methods which would be interesting to try. SMAA is quite widely used, it is easy to implement or even inject to an engine, and it does not require a lot of frame time. Most recent versions of it use multisampling and temporal reprojection. AAGA works as an image-space pre-filterer before lighting. It designed to work especially well with deferred rendering and promises quality similar to 4x or 8xMSAA. However, as screen-space methods these can have downsides as FXAA and TAA in terms of fatigue.

This study raised a lot new interesting questions and hypothesis to be researched in the future. The reasons behind exactly what features or properties of the image make image quality to look worse or better could be researched, and same for fatigue. Visual quality could be divided into features such as *image sharpness* or *detail visibility*. Also a better larger rating scale, for example from 1-9 could be then used to be able to get more precise use mean for analysis. Also a scenes with isolated features like dynamic objects and a lot of sub-pixel details could be used to ensure more exact result.

This study could be also performed using a regular display. It would be interesting to know if different aspects of image quality become more or less important as instead of binocular information there was monocular. During the development some informal static image comparison was done using regular desktop monitor, and quality of TAA was seemingly better on the desktop monitor than HMD, especially with a static camera.

And finally, an overall image quality could be researched further and not only aliasing on the geometry edges and sub-pixel details.

9.4 Final Thoughts

Finding a feasible anti-aliasing method with quality comparable to MSAA is a challenge especially when working on VR. On HMDs, the possible weaknesses of different methods seem to be emphasized. When these weaknesses are visible they can really easily make user feel fatigued. The results of this thesis indicate that MSAA is currently the superior anti-aliasing method for VR.

In a sense, this thesis also questions the need of anti-aliasing. Of course, the subjective study did not include the non-anti-aliased stimulus, so exact conclusions cannot be made. Many of participants were surprised by the good quality of the original (non-anti-aliased) image, which was shown as a reference before the actual test. They were surprised how small differences between the original image and the MSAA image had. Also some commented that the image quality is never very good as they were able to detect individual pixels and artifacts, which are caused by the optics of the lenses.

More importantly, the results of this study emphasize the fact that human visual system is extremely accurate and delicate, and HMD's quality is still not as good as people would want it to be. When developing for VR developers should be aware of how human visual system works. Most of the rules commonly used in the desktop applications do not apply to VR. In order to deliver great VR experiences developers should make sure that images provided to left and right eye views are as they would appear naturally and sharp as possible to ensure that users feel that they can focus their eyes.

Bibliography

[1] Abrash, M. What VR could, should, and almost certainly will be within two years, 2014. Presentation at Steam Dev Days.

[2] Aflaki, P., Hannuksela, M. M., and Gabbouj, M. Subjective quality assessment of asymmetric stereoscopic 3d video. *Signal, Image and Video Processing 9*, 2 (2015), 331–345.

[3] Aflaki, P., Hannuksela, M. M., Häkkinen, J., Lindroos, P., and Gabbouj, M. Subjective study on compressed asymmetric stereoscopic video. In *2010 IEEE International Conference on Image Processing* (Sept 2010), pp. 4021–4024.

[4] Akenine-Möller, T., Haines, E., and Hoffman, N. *Real-Time Rendering 3rd Edition.* A. K. Peters, Ltd., Natick, MA, USA, 2008.

[5] Andreev, Dimitry (AND). Anti-aliasing from a different perspective: Directionally localized anti-aliasing (dlaa), 2011. Presentation at Game Developers Conference.

[6] Arns, L. L., and Cerney, M. M. The relationship between age and incidence of cybersickness among immersive environment users. In *IEEE Proceedings. VR 2005. Virtual Reality, 2005.* (March 2005), pp. 267–268.

[7] Bavoil, Louis (Nvidia), Crassin, Cyril (Nvidia). Aggregate G-buffer Anti-Aliasing in Unreal Engine 4, 2016. Presentation at SIGGRAPH.

[8] Blake, R. A primer on binocular rivalry, including current controversies. *Brain and Mind 2*, 1 (2001), 5–38.

[9] Blake, R., and Sekuler, R. *Perception.* McGraw-Hill higher education. McGraw-Hill Companies,Incorporated, 2006.

[10] Bowman, D. A., and McMahan, R. P. Virtual reality: How much immersion is enough? *Computer 40*, 7 (July 2007), 36–43.

[11] Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6 (June 1986), 679–698.

[12] Chajdas, M. G., McGuire, M., and Luebke, D. Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 15–22.

[13] Chan, E., and Durant, F. *Fast Prefiltered Lines.* Addison Wesley, 2005, pp. 345–359.

[14] Crassin, C., McGuire, M., Fatahalian, K., and Lefohn, A. Aggregate g-buffer anti-aliasing. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2015), i3D '15, ACM, pp. 109–119.

[15] Davies, L (Intel). Conservative morphological anti-aliasing (cmaa) - march 2014 update. `https://software.intel.com/en-us/articles/conservative-morphological-anti-aliasing-cmaa-update`. Accessed: 4.8.2015.

[16] Deering, M., Winner, S., Schediwy, B., Duffy, C., and Hunt, N. The triangle processor and normal vector shader: A vlsi system for high performance graphics. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 21–30.

[17] Dmitry Zhdan (NVIDIA). Tiled shading: light culling - reaching the speed of light, 2016. Presentation at Game Developers Conference.

[18] Drobot, M. Hybrid reconstruction anti-aliasing, 2014. Presentation at SIGGRAPH.

[19] Fontaine, G. The experience of a sense of presence in intercultural and international encounters. *Presence: Teleoper. Virtual Environ. 1*, 4 (Oct. 1992), 482–490.

[20] Furness, T. A. Creating Better Virtual Worlds. Tech. rep., University of Washington, Human Interface Technology Laboratory, 1989.

[21] Gonzalez, R. C., and Woods, R. E. *Digital Image Processing 3rd Edition.* Pearson Education Inc., New Jersey, 2008. Third edition.

[22] Halton, J. H. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2*, 1 (1960), 84–90.

[23] Harada, T., McKee, J., and Yang, J. C. Forward+: Bringing Deferred Lighting to the Next Level. *Eurographics 2012 - Short Papers* (2012).

[24] Hecht, S., and Shlaer, S. The influence of intensity, color and retinal location on the fusion frequency of intermittent illumination. *The Journal of General Physiology 19* (6 1936), 965–977.

[25] HTC. Vive. `https://www.htcvive.com/eu/`, 2016. Accessed 2016-07-27.

[26] International Telecommunication Union. ITU-T Recommendations. `http://www.itu.int/ITU-T/recommendations/index.aspx`, 2016. Accessed 2016-08-23.

[27] Jakob, W. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[28] Jimenez, J., Echevarria, J. I., Sousa, T., and Gutierrez, D. Smaa: Enhanced subpixel morphological antialiasing. *Comp. Graph. Forum 31*, 2pt1 (May 2012), 355–364.

[29] Jing Li, Marcus Barkowsky, P. L. C. Visual discomfort in 3dtv: Definitions, causes, measurement, and modeling. In *Novel 3D Media Technologies*, T. D. Ahmet Kondoz, Ed. Springer, New York, 2015, ch. 10, pp. 185–209.

[30] Karis, B. High Quality Temporal Supersampling, 2014. Presentation at SIGGRAPH.

[31] Kooi, F. L., and Lucassen, M. Visual comfort of binocular and 3d displays. *Proc. SPIE 4299* (2001), 586–592.

[32] Koonce, R. *Deferred Shading in Tabula Rasa.* Addison Wesley, 2007, pp. 115–129.

[33] Kruskal, W. H., and Wallis, W. A. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association 47*, 260 (1952), 583–621.

[34] Lea-Test Ltd. Lea Numbers Near Vision Card. `http://www.lea-test.fi/index.html?start=en/vistests/instruct/2709-10/index.html`, 2012. Accessed 2016-09-23.

[35] Leibowitz, H. W. Recent advances in our understanding of peripheral vision and some implications. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 30*, 6 (1986), 605–607.

[36] LOTTES, T. Mixing temporal aa and transparency. `http://timothylottes.blogspot.fi/2015/11/mixing-temporal-aa-and-transparency.html`. Accessed: 15.6.2016.

[37] LOTTES, T. White paper: FXAA. Tech. rep., NVIDIA Corporation, February 2009.

[38] MEEHAN, M., RAZZAQUE, S., WHITTON, M. C., AND BROOKS, JR., F. P. Effect of latency on presence in stressful virtual environments. *Proceedings of the IEEE Virtual Reality 2003* (2003), 141–.

[39] MICROSOFT. Direct3D 11 Reference. `https://msdn.microsoft.com/en-us/library/windows/desktop/ff476218(v=vs.85).aspx`, 2016. Accessed 2016-07-27.

[40] MICROSOFT. DirectX Graphics Samples. `https://github.com/Microsoft/DirectX-Graphics-Samples`, 2016. Accessed 2016-06-14.

[41] MITCHELL, D. P., AND NETRAVALI, A. N. Reconstruction filters in computer-graphics. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 221–228.

[42] MSDN. Executing and synchronizing command lists. `https://msdn.microsoft.com/en-us/library/windows/desktop/dn899124(v=vs.85).aspx`. Accessed: 8.10.2016.

[43] MSDN. Rasterization rules. `https://msdn.microsoft.com/en-us/library/windows/desktop/cc627092(v=vs.85).aspx`. Accessed: 16.7.2015.

[44] MSDN. Rasterizer stage. `https://msdn.microsoft.com/en-us/library/windows/desktop/bb205125(v=vs.85).aspx`. Accessed: 2.10.2016.

[45] MSDN. Shader stages. `https://msdn.microsoft.com/en-us/library/windows/desktop/bb205146(v=vs.85).aspx`. Accessed: 2.10.2016.

[46] MSDN. What is direct3d 12? `https://msdn.microsoft.com/en-us/library/windows/desktop/dn899228(v=vs.85).aspx`. Accessed: 8.10.2016.

[47] NAGAHARA, H., YAGI, Y., AND YACHIDA, M. Wide field of view head mounted display for tele-presence with an omnidirectional image sensor. In *Computer Vision and Pattern Recognition Workshop, 2003. CVPRW '03. Conference on* (June 2003), vol. 7, pp. 86–86.

[48] NICODEMUS, F. E., RICHMOND, J. C., HSIA, J. J., GINSBERG, I. W., AND LIMPERIS, T. Geometrical considerations and nomenclature for reflectance. In *Radiometry*, L. B. Wolff, S. A. Shafer, and G. Healey, Eds. Jones and Bartlett Publishers, Inc., USA, 1992, pp. 94–145.

[49] Nvidia. Antialiased deferred rendering. http://docs.nvidia.com/gameworks/content/gameworkslibrary/graphicssamples/d3d_samples/antialiaseddeferredrendering.htm. Accessed: 17.7.2015.

[50] NVIDIA. NVIDIA VRWorks. https://developer.nvidia.com/vrworks, 2016. Accessed 2016-09-30.

[51] Oculus. Documentation. https://developer.oculus.com/documentation/intro-vr/latest/, 2016. Accessed 2016-07-27.

[52] Oculus. Health and Safety. https://static.oculus.com/documents/310-30023-01_Rift_HealthSafety_English.pdf, 2016. Accessed 2016-06-17.

[53] Oculus VR, LLC. Oculus. https://www3.oculus.com/en-us/rift/, 2016. Accessed 2016-07-27.

[54] Oskam, T., Hornung, A., Bowles, H., Mitchell, K., and Gross, M. OSCAM - Optimized stereoscopic camera control for interactive 3D. *ACM Trans. on Graphics (Proc. SIGGRAPH) 30*, 6 (2011), 189:1–189:8.

[55] Pangerl, D. *Deferred Rendering Transparency.* Course Technology/Cengage Learning, 2009, pp. 217–225.

[56] Pedersen, L. J. F. Temporal Reprojection Anti-Aliasing in INSIDE, 2016. Presentation at Game Developers Conference.

[57] Persson, E. Deep deferred shading. http://www.humus.name/index.php?page=3D&ID=75. Accessed: 31.7.2015.

[58] Prothero, J., and Hoffman, H. Widening the field of view increases the sense of presence in immersive virtual environments. *International Journal of Human-Computer Interaction* (1995).

[59] Rebenitsch, L., and Owen, C. Review on cybersickness in applications and visual displays. *Virtual Reality 20*, 2 (2016), 101–125.

[60] Reshetov, A. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 109–116.

[61] Saito, T., and Takahashi, T. Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph. 24*, 4 (Sept. 1990), 197–206.

[62] Sanchez-Vives, M. V., and Slater, M. From presence to consciousness through virtual reality. *Nat Rev Neurosci 6*, 4 (04 2005), 332–339.

[63] Shannon, C. E. A mathematical theory of communication. *Bell System Technical Journal 27*, 3 (7 1948), 379–423.

[64] Sheridan, T. B., and Furness, III, T. A. Musings on telepresence and virtual presence. *Presence: Teleoper. Virtual Environ. 1*, 1 (1992).

[65] Sherrod, A. *Game Graphic Programming.* Cengage Learning, 2008.

[66] Sherstyuk, A., Dey, A., Sandor, C., and State, A. Dynamic eye convergence for head-mounted displays improves user performance in virtual environments. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 23–30.

[67] Slater, M. A note on presence terminology. *Presence Connect 3* (2003).

[68] Slater, M., Khanna, P., Mortensen, J., and Yu, I. Visual realism enhances realistic response in an immersive virtual environment. *IEEE Computer Graphics and Applications 29*, 3 (May 2009), 76–84.

[69] Spencer, G., Shirley, P., Zimmerman, K., and Greenberg, D. P. Physically-based glare effects for digital images. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 325–334.

[70] Tiago Sousa. CRYENGINE3 Graphics Gems, 2013. Presentation at SIGGRAPH.

[71] Tim Foley, Anton Kaplanyan, M. S. From the Lab Bench: Real-Time Rendering Advances from NVIDIA Research, 2016. Presentation at Game Developers Conference.

[72] Timonen, V. Developing The Northlight Engine: Lessons Learned, 2016. Presentation at Game Developers Conference.

[73] Urvoy, M., Barkowsky, M., and Le Callet, P. How visual fatigue and discomfort impact 3d-tv quality of experience: a comprehensive review of technological, psychophysical, and psychological factors. *annals of telecommunications - annales des télécommunications 68*, 11 (2013), 641–655.

[74] Valient, M. Reflections and Volumetrics of Killzone Shadow Fall, 2014. Presentation at SIGGRAPH.

[75] Valient, M. Taking Killzone Shadow Fall Image Quality Into the Next Generation, 2014. Presentation at Game Developers Conference.

[76] Veitch, J. A., and McColl, S. L. Modulation of fluorescent light: Flicker rate and light source effects on visual performance and visual comfort. *Lighting Research and Technology 27* (12 1995), 243–256.

[77] Vive. Safety and regulatory guide, 2016. Accessed 2016-06-17.

[78] Vlachos, A. (Valve). Advanced VR Rendering, 2015. Presentation at Game Developers Conference.

[79] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics Bulletin 1*, 6 (1945), 80–83.

[80] Witmer, B. G., and Singer, M. J. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoper. Virtual Environ. 7*, 3 (June 1998), 225–240.

[81] Wronski, B. Temporal supersampling and antialiasing. `http://bartwronski.com/2014/03/15/temporal-supersampling-and-antialiasing/`. Accessed: 17.7.2015.

[82] Zimmons, P., and Panter, A. The influence of rendering quality on presence and task performance in a virtual environment. In *Proceedings of the IEEE Virtual Reality 2003* (Washington, DC, USA, 2003), VR '03, IEEE Computer Society, pp. 293–.

Appendix A

Multisample Anti-Aliasing Pseudo Implementation

Detecting complex pixels:

```
void PS()
{
  ...
  bool is_normal_edge = false;

  float3 normal = NormalTexture.Load(position, 0);
  float3 reference_normal = normal;

  for all samples
  {
    normal = NormalTexture.Load(position, i);
    is_normal_edge = is_normal_edge ||
                     Differs(normal, reference_normal);
  }
  ...
```

Edge detection:

```
  ...
  float reference_depth = DepthTexture.Load(position, 0);
  float3 reference_luminance = LuminanceTexture.Load(position, 0);
  float max_depth = reference_depth;
  float3 average_luminance = reference_luminance;

  is_material_edge = is_normal_edge;

  for all samples
  {
    float depth = DepthTexture.Load(p, i);
    float4 reflectance = ReflectanceTexture.Load(p, i);
    float3 luminance = LuminanceTexture.Load(p, i);

    is_material_edge = is_material_edge ||
                       Differs(reflectance, reference_reflectance);
    is_material_edge = is_material_edge ||
                       Differs(depth, reference_depth);
    is_material_edge = is_material_edge ||
                       Differs(luminance, reference_luminance);
    max_depth = max(max_depth, depth);
    average_luminance += ToneMap(luminance);
  }
  ...
```

Luminance output:

```
  ...
  average_luminance /= msaa_sample_count;
  average_luminance = InverseToneMap(average_luminance);

  // Write out edges, solved depth and solved luminance
  output_edge = is_material_edge;
  output_depth = max_depth;
  output_luminance = average_luminance;
}
```

Shading with multisampled G-buffer:

```
  ...
  float3 illumination = 0;

  bool is_edge = EdgeTexture.Load(position) > 0;

  GbufferData data;
  LoadGBufferData(data, position, 0);

  IlluminateSample(data, illumination);

  if (is_edge)
  {
    illumination = ToneMap(illumination);

    for all samples
   {
      float3 hdr_illumination = 0;
      LoadGBufferData(data, position, sample_index);
      IlluminateSample(data, hdr_illumination);
```

```
        illumination += ToneMap(hdr_illumination);
    }

    illumination /= msaa_sample_count;
    illumination = InverseToneMap(illumination);
}
```

Appendix B

Temporal Anti-Aliasing Pseudo Implementation

Calculating the object velocity:

```
void VS()
{
  ...
  output.position = WorldToViewClipMatrix() * vertex.position_in_world);
  float4 previous_position = PreviousWorldToViewClipMatrix() *
                             vertex.previous_position_in_world);

  float2 current = output.position.xy / output.position.w;
  float2 previous = previous_position.xy / previous_position.w;
  output.velocity = current - previous;
  ...
}

void PS()
{
  ...
  output_velocity = input.velocity;
  ...
}
```

Using the longest velocity of neighborhood for the pixel velocity:

```

void CS()
{
  float2 uv = GetPositionInTexture(position);
  float2 velocity = FindLongestVelocityVector(uv);
  ...
```

Calculating the pixel velocity for a static object from the camera movement:

```
if (velocity == 0)
{
  float depth = DepthTexture[position];
  float4 position_in_view_H = NoJitterTargetToViewMatrix *
                              float4(position + 0.5, depth, 1);
  float3 position_in_view = position_in_view_H.xyz / position_in_view_H.w;
  float3 position_in_world = ViewToWorldMatrix *
                             float4(position_in_view, 1.0f);

  float4 previous_position = NoJitterPreviousWorldToViewClipMatrix *
                             float4(position_in_world, 1.0f);
  float2 previous = previous_position.xy / previous_position.w;

  float4 position = NoJitterViewToViewClipMatrix *
                    float4(positionInView, 1);
  float2 current = position.xy / position.w;
  velocity = (current - previous) / 2.0f;
}
```

Sampling the current pixel neighborhood and clipping the previous frame pixel against it:

```
...
float3 current_sample = IlluminationTexture.Sample(uv);
float3 previous_sample = PreviousResultTexture.Sample(uv - velocity);

float2 du = float2(OneOverTargetSize.x, 0.0f);
float2 dv = float2(0.0f, OneOverTargetSize.y);

// Sample current pixel's neighborhood
float3 tl = IlluminationTexture.Sample(uv - dv - du);
float3 tc = IlluminationTexture.Sample(uv - dv);
float3 tr = IlluminationTexture.Sample(uv - dv + du);

float3 ml = IlluminationTexture.Sample(uv - du);
float3 mc = current_sample;
float3 mr = IlluminationTexture.Sample(uv + du);

float3 bl = IlluminationTexture.Sample(uv + dv - du);
float3 bc = IlluminationTexture.Sample(uv + dv);
float3 br = IlluminationTexture.Sample(uv + dv + du);

float3 moment1 = tl + tc + tr + ml + mc + mr + bl + bc + br;
float3 average = moment1 / 9.0f;

float3 moment2 = tl * tl + tc * tc + tr * tr +
                 ml * ml + mc * mc + mr * mr +
                  bl * bl + bc * bc + br * br;

float3 sigma = sqrt(moment2 / 9.0f - average * average);

float gamma = 1.0f;
float3 min = average - gamma * sigma;
float3 max = average + gamma * sigma;

float3 clipped_color = ClipAABB(min, max, average, previous_sample);
 ...
```

Blending the current pixel color and the clipped pixel color:

```
...
  // Limit velocity to max 5 pixels
float max_velocity = length(5.0f * OneOverTargetSize);
float blend_factor = saturate(length(velocity) / max_velocity);
float a = lerp(0.9f, 0.3f, blend_factor);
float3 reprojected_color = (1.0f - a) * current_sample + a * clipped_color;
...
```