



EVALUATION OF OPTIMISATION TECHNIQUES FOR MULTISCOPIC RENDERING

Grigor Todorov

This is a digitised version of a dissertation submitted to the University of Bedfordshire.

It is available to view only.

This item is subject to copyright.

EVALUATION OF OPTIMISATION TECHNIQUES FOR MULTISCOPIC RENDERING

Grigor Todorov

A thesis submitted to the University of Bedfordshire in fulfilment of the
requirements for the degree of Master of Science by Research

Institute for Research in Applicable Computing

October 2015

Abstract

This project evaluates different performance optimisation techniques applied to stereoscopic and multiscopic rendering for interactive applications. The artefact features a robust plug-in package for the Unity game engine. The thesis provides background information for the performance optimisations, outlines all the findings, evaluates the optimisations and provides suggestions for future work. Scrum development methodology is used to develop the artefact and quantitative research methodology is used to evaluate the findings by measuring performance. This project concludes that the use of each performance optimisation has specific use case scenarios in which performance benefits. Foveated rendering provides greatest performance increase for both stereoscopic and multiscopic rendering but is also more computationally intensive as it requires an eye tracking solution. Dynamic resolution is very beneficial when overall frame rate smoothness is needed and frame drops are present. Depth optimisation is beneficial for vast open environments but can lead to decreased performance if used inappropriately.

Declaration

I declare that this thesis is my own unaided work. It is being submitted for the degree of Master of Science by Research at the University of Bedfordshire.

Name of candidate: Grigor Todorov

Signature:

Date:

List of Content

Abstract	ii
List of Content.....	iv
Table of Figures	v
Acknowledgements	vi
1. Introduction	7
1.1 Aims and Objectives	8
1.2 Motivation	9
2. Contextual review and market research	10
2.1 Foveated Rendering.....	10
2.2 Dynamic Resolution.....	16
2.3 Depth Reuse	18
3. Artefact Design, Development and Testing	20
3.1 Development and research methodology	20
3.2 Use of the artefact	21
3.3 Design and implementation of the performance optimisations	22
3.3.1 Foveated rendering system.....	22
3.3.2 Dynamic resolution system	31
3.3.3 Depth reuse system.....	36
3.4 Design and implementation of additional systems.....	39
3.4.1 Automultiscopic rendering	39
3.4.2 Performance measuring system.....	41
3.5 Problems encountered	42
3.6 Testing.....	44
4. Evaluation	45
4.1 Traditional rendering.....	47
4.2 Stereoscopic rendering	49
4.3 Multiscopic rendering	51
4.4 Dynamic resolution	53
5. Conclusion and future work	57
5.1 Conclusion.....	57
5.2 Future work	58
References	59

Table of Figures

Figure 1 Human eye rod: cone receptor ratio.....	11
Figure 2 The distribution of rods and cones in the retina (Dubuc, 2002).....	11
Figure 3 All the letters should be equally readable (Anstis, 1974)	11
Figure 4 Aparatus from (Yarbus, 1960) Image from (Yarbus eyetracker , 1960)	11
Figure 5 BlueGain EOG (crsltd, 2015)	11
Figure 6 Tobii TX300.....	11
Figure 7 Gear VR (Gear VR, 2015).....	13
Figure 8 Google Cardboard (Cardboard, 2015)	13
Figure 9 Archos VR (Archos VR, 2015)	13
Figure 10 The Eye Tribe Tracker for tablet (The Eye Tribe, 2015)	14
Figure 11 FOVE eye tracking headset (FOVE Inc, 2015)	14
Figure 12 StarVR headset (Starbreeze Studios, 2015)	14
Figure 13. Battlefield 4 Resolution Scaling (dualshockers.com, 2013)	17
Figure 14 Depth Perception (brainhq)	18
Figure 15 2D rendering camera prefab	23
Figure 16 Drawing order of the render textures	23
Figure 17 Whole screen camera for foveated rendering	25
Figure 18 Foveated region	27
Figure 19 Stereoscopy Interlaced Foveated Rendering.....	29
Figure 20 Multiscopic Foveated Rendering	31
Figure 21 Performance Manager	33
Figure 22 Dynamic Resolution Manager	34
Figure 23 Script Holder Cam game object.....	37
Figure 24 Stereoscopic and Multiscopic implementations depth optimisation	38
Figure 25 Traditional and multiscopic rendering.....	40
Figure 26 Performance manager set to save data to file.....	41
Figure 27 2D Dynamic resolution rendering	54
Figure 28 Stereoscopy - Dynamic resolution.....	55
Figure 29 Multiscopic- Dynamic resolution	56

Acknowledgements

My deepest gratitude to my director of studies Professor Amar Aggoun who gave me both the freedom to explore on my own and the guidance when needed.

I would like to acknowledge James Wood for his continuous support.

1. Introduction

Modern video games rely on a wide variety of optimisation techniques in order to reach the photo realistic rendering quality expected by the users. With the increase of available computational power the demand of more beautiful, bigger and denser virtual worlds also increases. Use of techniques such as occlusion culling, tessellation, mipmapping and level of detail switching is vital for the smooth performance and photorealism, but such techniques are mainly designed to work for traditional rendering of one view. With the constant increase of available stereoscopic enabled hardware such as head-mounted displays and also future trends like multiscopic displays the use of old performance optimisation techniques is not nearly enough to meet the demands. The utilisation of more advanced performance optimisation techniques is required. The techniques evaluated in this project can provide additional performance increase for specific game situations. The first section of this report provides some academic and commercial background information on techniques that could be beneficially used for applications that require rendering of multiple views. Scrum development methodology is used to implement all the performance optimisations in the Unity game engine. Quantitative research methodology is utilized to design numerous experiments that measure the performance of the optimisation techniques. This thesis report provides an evaluation of the data measured as well as a summary of all the different performance optimisations and their appropriate uses. The end of this report features a section that provides some recommendations for future work.

1.1 Aims and Objectives

The aim of this project is to implement a number of optimisation techniques for multiscopic rendering and evaluate their performances. This project has two end products: an artefact and a thesis report. The artefact is open sourced and is intended to be a good starting point for the implementation of any of the performance optimisations discussed. The thesis documents the development process, analyses the commercial and academic viability and evaluates the performance optimisation techniques.

The objectives for both the thesis and the artefact are as follows:

Thesis:

- Outline multiple performance optimisation techniques that could be potentially beneficial for multiscopic rendering.
- Explain how each performance technique is implemented and how it works.
- Outline the benefits, requirements, cost of implementation and practicability for each optimisation technique.
- Evaluate the optimisation techniques by measuring and comparing their performance benefits.
- Suggest future work.

Artefact:

- Implement the performance optimisations discussed in the thesis and create a Unity plug-in.
- Follow good development practices and provide additional functionality.
- Create a system for performance measurement.

1.2 Motivation

The main goal of this project is to bring value for both academic and commercial applications. Some of the performance optimisations discussed in this thesis have never been evaluated for stereoscopic and multiscopic rendering and their application for such situations is novel. The artefact of this project will enable developers to implement such optimisation techniques in both commercial and academic applications.

The development of this project was motivated by the lack examples of such techniques implemented and openly available. The optimisations evaluated in this thesis are implemented with the Unity game engine and C# code. Since the artefact is open sourced the performance optimisations could also easily be adapted for game engines other than Unity and can even be used within custom development environments.

The artefact is designed to be easy to use and modify. The code is fully commented and provides additional tooltips in the Unity user interface for ease of use.

This thesis is freely available online to further aid future works on the optimisation techniques and is designed to be a useful starting point for anyone interested in implementing them. It also evaluates all the performance optimisations and provides suggestions on best use scenarios for each optimisation.

2. Contextual review and market research

This chapter outlines the findings after academic and market research was conducted for each performance optimisation technique.

2.1 Foveated Rendering

People have a field of view of 135° vertically and 160° horizontally (NASA , 1964), but the human eye does not have uniform distribution of optic nerves (Figure 2). A small region of the human eye called “fovea” and located in the middle of the retina (Figure 1) contains half of the optic nerves (Figure 2) and a field of view of 5° (Guenter, et al., 2012). As seen in Figure 3 the ability to perceive information is rapidly decreasing away from the gaze location in the middle of the image and in order to accommodate for this the size of the letters in the outer region of the image is increased. People use motions called “saccades” (Ebisawa & Suzu, 1994) to perceive their environment in high field of view. The quality of the peripheral vision gradually degrades in areas further from the fovea. These areas, called “parafovea” and “perifovia”, have far fewer cone receptors and more rod cells (Figure 1). Cone cells are good at perceiving colour while rod cells are better at distinguishing motion. People are good at detecting motion with their peripheral vision but fail at distinguishing colours or shapes as the quality of vision degrades outside of the foveated region.

Traditional displays are created with the assumption that the user can perceive every region of the screen at the same time, which is clearly not the case. When only one user is interacting with the screen a small fraction of the pixels is fully comprehended (Guenter, et al., 2012) at any given moment.

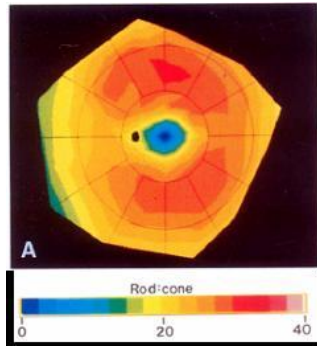


FIGURE 1 HUMAN EYE ROD: CONE RECEPTOR RATIO (CURICO, ET AL., 1990)

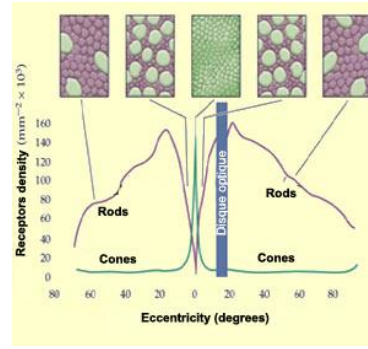


FIGURE 2 THE DISTRIBUTION OF RODS AND CONES IN THE RETINA (DUBUC, 2002)

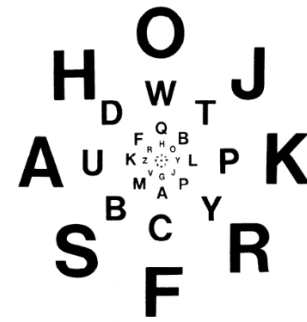


FIGURE 3 ALL THE LETTERS SHOULD BE EQUALLY READABLE (ANSTIS, 1974)

As seen in Figure 1, the retina has a small area that lacks photoreceptors called a “blind spot” (Durgin, et al., 1995). The brain recreates the “blind spot” as well as other missing parts of the vision by combining the information of both eyes and a person cannot perceive the lack of information in that region. If we can discard the portion of the screen that is hidden from the user’s perception and gradually decrease quality and resolution towards the parts of the screen that are not comprehended with the fovea we can increase the overall performance. In order to create such effect we need to know the location of the user gaze at real-time, therefore a robust eye tracking solution is required.

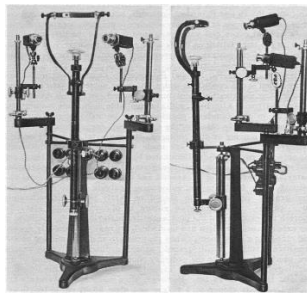


FIGURE 4 APARATUS FROM (YARBUS, 1960). IMAGE FROM (YARBUS EYETRACKER , 1960)



FIGURE 5 BLUEGAIN EOG (CRSLTD, 2015)



FIGURE 6 TOBII TX300 EYE TRACKER (TOBII TECHNOLOGY, 2011)

For the last half a century gaze tracking has been a widely researched field (Yarbus, 1960). The eye tracker hardware used to be bulky (Figure 4), expensive (Figure 6) and very intrusive for the user (Figure 5), but modern solutions solve

these issues and currently available eye tracking hardware is cheap, compact and versatile. Eye tracking hardware usually works in one of three ways.

The most intrusive method is the eye-attached tracking with a special contact lens that reflects infrared light (Chen & Kalinli, 2011). Eye tracking with contact lenses is not an applicable solution for a commercial product as it requires the user to apply eye lenses. This form of eye tracking provides accuracy and is used for research purposes.

Eye tracking can also be achieved with electrooculography (crsltd, 2015). This approach is good at measuring rapid eye saccade movements and is not an accurate solution when measuring slow eye movements such as tracking an object on the screen (Ebisawa & Suzu, 1994).

The last and most widely used approach is to use infrared light, cameras and image processing to determine the eye rotation and gaze position. The use of infrared light source is needed because the human eye can perceive light that falls in the visible spectrum and infrared light is mostly unperceivable (Palczewska, et al., 2014). One potential issue when using eye tracking is the rapid saccades a human eye makes. The eye tracking solution needs to work with fast frequency in order to track the eye while in a saccade. The human brain selectively discards most of the information received during rapid saccades in a process called saccadic masking (Burr, et al., 1994). It is therefore not imperative to track the eye location while in saccadic movement. As long as the eye tracking solution calculates the new gaze point fast enough after the saccade no delay will be perceivable by the user.

The use of eye tracking for distribution of visual fidelity, foveated rendering, has been researched extensively, with some very promising results (Guenter, et al., 2012), but such research leads to the conclusion that current generation of eye tracking solutions cannot be used commercially for foveated rendering as the required eye tracking hardware is too expensive or does not provide the required tracking speed. Recent commercial products have made eye tracking cheap and widely available and this project tries to review the use of eye tracking for foveated rendering with currently available and affordable eye tracking solutions. Foveated rendering has many applications, some of which include virtual reality, remote piloting and video conferencing (Baudisch, et al., 2003).

In foveated rendering eye tracking hardware is used to track the gaze point of the user eyes and multiple layers of degrading quality and increasing size are rendered. When foveated rendering is used successfully it could increase the performance and reduce the required bandwidth. In the context of real-time computer graphics we are not concerned with the bandwidth but are interested in the increase of performance. An estimated performance increase after a successful implementation of foveated rendering is a factor of five (Guenter, et al., 2012). This could lead to a breakthrough in the smart phone rendering and instead of devices with more processing power phones with low latency eye tracking and displays could be developed. The mobile market may not seem to be a suitable medium for foveated rendering due to the smaller screen size, but smart phones are front runners for the current revolution of virtual reality. As the devices feature both very good gyroscopes, high definition screens and their computational power continues to grow, it is possible to use them for wireless virtual reality headsets. Head-mounted displays are traditionally very bulky systems that are connected to a computer and sharing the virtual reality experience with other people is difficult due to their form factor.



FIGURE 7 GEAR VR (GEAR VR, 2015)



FIGURE 8 GOOGLE CARDBOARD (CARDBOARD, 2015)



FIGURE 9 ARCHOS VR (ARCHOS VR, 2015)

Virtual reality headsets specifically designed to work in conjunction with smart phones, such as Gear VR (Figure 7), Google Cardboard (Figure 8) and many other solutions (Figure 9) have emerged on the market. The two major reasons why these devices are not competitive with the tethered virtual reality experiences provided by Oculus Rift, HTC Vive and other similar HMDs is the lack of positional tracking for the head as well as the device performance. This project aims to solve the performance problem by proposing the use of foveated rendering in conjunction with eye tracking for smart phones.

The head position tracking problem could be solved in the near future with the addition of a variety of sensors. Technologies such as Project Tango (Google ATAP, 2015) feature depth sensors akin to the ones found in Kinect. There are also technologies that can not only scan the environment for 3D point cloud but also understand different hand gestures (Microsoft HoloLens, 2015).

Since the vast majority of the population already own smart phones and a gyroscope sensor is the only requirement for the phone to be used as a virtual reality headset, it is cheaper to use a phone as a head-mounted display than to invest in a tethered alternative that also requires a powerful computer. The wide adoption of sensors for depth and gestures is very important for the future of mobile virtual reality.



FIGURE 10 THE EYE TRIBE TRACKER FOR TABLET (THE EYE TRIBE, 2015)



FIGURE 11 FOVE EYE TRACKING HEADSET (FOVE INC, 2015)



FIGURE 12 STARVR HEADSET (STARBREEZE STUDIOS, 2015)

There are multiple eye tracking solutions available for mobile devices (Figure 10) but they usually target a form factor larger than a smart phone and are not suitable for use in a head-mounted display. A hypothetical eye tracking solution that works at a frequency fast enough to permit the use of foveated rendering for such device could lead to more robust widely available virtual reality devices that are not tethered to a PC and with the requirement of just a fraction of the computational power of current mobile virtual reality.

One relevant head-mounted display that was recently backed on Kickstarter is FOVE (Figure 11). It is tethered and therefore does not provide the benefits of smart phone virtual reality, but it features an eye tracking solution. FOVE will be the first commercially available head-mounted display with the ability to use foveated rendering.

Another future head-mounted display that will use eye tracking extensively is StarVR (Figure 12). It will feature two high definition screens and offer 210° field of view and 5K resolution.

Other custom eye tracking modifications of the development version of Oculus Rift are also available but are expensive (SensoMotoric Instruments, 2015) and mainly used for academic research. It is possible to create a very cheap custom eye tracking solution by using a PlayStation Eye camera (Sony , 2007) in a low resolution mode which works at 120Hz, but this solution is mainly used by enthusiasts and is not compact.

In order to eliminate the popping effect in foveated rendering perceived by the user a low latency system is required. A 60Hz display is not a viable solution and for successful implementation of foveated rendering both the display and the eye-tracker need to work at 90Hz or more (Thunström, 2014).

For the development and testing of the foveated rendering the Tobii EyeX (Tobii Technology, 2015) tracker was used. It is an affordable tracker targeting video games audience. The original frequency of the eye tracker is not fixed but it works at 30Hz or more. Custom build of the eye tracking engine was provided by Tobii for the foveated rendering testing. It increased the Tobii EyeX frequency to 90Hz, but the tracking accuracy deteriorated. Even with reduced accuracy the system works well for foveated rendering, but due to the lack of a high frequency monitor available for this research popping effects were still perceivable. The displays used for testing are 22-inch normal monitor, 42-inch automultiscopic monitor and a Commander 3D tablet with autostereoscopic capabilities.

Current computer systems are capable of rendering for multiple 4K monitors, but expensive graphics hardware is necessary. Existing game engines are capable of rendering realistic scenes, but with stereoscopy and multiscopy the hardware needs to render the scene from more than one view, which is computationally intensive. Foveated rendering enables a wide adoption of multiscopic displays for personal and commercial use as it reduces the required performance.

2.2 Dynamic Resolution

Since the dawn of 3D accelerated graphics researchers have been trying to increase rendering performance. Numerous rendering optimisations that increase visual fidelity or reduce processing power requirements have been produced. Some techniques worth mentioning include level of detail switching (Clark, 1976), tessellation (Microsoft Developer Network, 2013), occlusion culling (Oded Sudarsky, 1999). Most performance techniques provide performance increase in scenes that feature large amount of geometry by decreasing the overall quality of the geometry the further it is from the camera. This usually leads to perceptibly bad geometrical quality into the distance and object popping. Most performance optimisations do not take into account the current situation and are either always enabled or always disabled, but most of the games have a stable performance until a graphically intensive scene is reached. In these situations artefacts such as object popping could be avoided with smarter performance optimisation techniques that only optimise the rendering when it is needed. One such technique that adapts to performance to increase or decrease graphical fidelity is dynamic resolution. Dynamic resolution (Intel, 2011) features a real-time adaptive change of the internal resolution of the application. Traditionally the resolution in video games takes a lot of time to change and can only be set in the options menu. The difference between the traditional resolution settings and the dynamic resolution scaling is that the resolution change affects the graphics user interface of the game while the dynamic resolution changes the internal rendering resolution without any change to the user interface.

Dynamic resolution scaling is rarely used in console games when the targeted performance could not be met. Sometimes video games render the content at lower and upscale it to larger resolutions. In other times rendering is set to target a specific frame rate and the resolution is changed at run-time to accommodate the frame rate target. Some Xbox One game titles such as “The Witcher 3: Wild Hunt” (extremetech.com, 2015) and “Call of Duty: Advanced Warfare” (engadget.com, 2014) feature such resolution scaling system, but they render at pre-set width and only scale the height resolution.

Other games provide a more passive settings option to change the resolution scaling. One such game is Battlefield 4 (dualshockers.com, 2013) which features an option called “Resolution Scale” available from the options menu. This only applies to the PC version of the game and the value can be set between 25% and 200%. It is a separate option from the resolution of the game and it is effectively a multiplier of the internal rendering resolution. Changing this value does not change the user interface scaling. Figure 13 illustrates the difference of quality perceived by the user.



FIGURE 13. BATTLEFIELD 4 RESOLUTION SCALING (DUALSHOCKERS.COM, 2013)

Changing the “Resolution Scaling” option to a lower value greatly deteriorates the perceived quality but the amount of frames that can be rendered each second increases. Resolution Scaling does not provide the benefits of dynamic resolution but it is very useful for older rendering hardware.

Dynamic resolution is also used in some video editing software solutions (Adobe, 2015) to display a preview of the final footage. Some 3D renderers (VRay, 2011) are capable of showing a lower fidelity version of the final ray traced scene, but instead of changing the resolution of the view they render a smaller number of pixels for preview. One benefit of dynamic resolution is a smoother overall experience due the reduction of frame per second drops. This project is an opportunity to test dynamic resolution against other performance optimisation techniques. The expected result for dynamic resolution is to have a great performance benefit but to affects immersion in a negative manner. It needs to be capped so that the performance quality does not deteriorate more than a pre-set quality.

2.3 Depth Reuse

When rendering stereoscopic or multiscopic content the scene needs to be rendered multiple times, the number of which corresponds to the number of views. The human brain uses the difference of two views supplied from the eyes, called “stereopsis” (University of Cambridge, 2003), to determine the distance from objects (Figure 14).

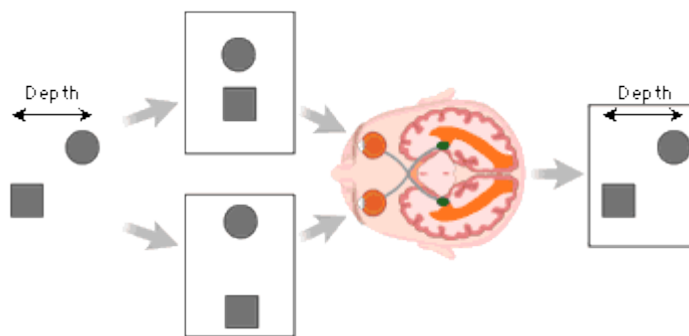


FIGURE 14 DEPTH PERCEPTION (BRAINHQ)

Stereopsis decreases for objects that are far away from the eye. When rendering virtual scenes that feature vast open areas some objects are far into the distance and even though they are rendered for each view they do not contribute to the stereopsis. It is possible to render the objects that are further than the distance of perceived depth and reuse the rendering for all of the views in stereoscopy and multiscopy. There are a number of ways in which this could be achieved. The depth buffer could be reused and combined with other depth buffers for each view or extensive use of “render textures” could be utilized. The “render texture” method was selected for the implementation and evaluation of this technique due to it being easier to implement and understand. It also works with existing image effects and provides the ability to change the refresh rate of the two “render textures” which could further be beneficial to performance. The first step is to render the distant objects in a “render texture”. The next step is to render the close objects for each view into another “render texture”. The final result for each view is achieved by combining the “render texture” of the view with the “render texture” of the depth.

It is expected that this will be beneficial for vast scenes with a lot of geometry, especially for multiscopic rendering where the depth will be reused multiple times. It is very important to choose a distance from the camera that does not jeopardize the stereopsis. For Oculus Rift a distance at which depth cues between the two eyes are smaller than one pixel is 52 meters (SlowRiot, 2013). Such depth optimisation is not viable for use in traditional rendering and is expected to decrease the overall performance, but its benefits should scale up with the number of different views rendered. Even at more than eight views it is possible to combine different views in pairs and render depth only once for each pair. This is expected to provide a large increase of performance and could work well in conjunction with other performance optimisations discussed in this project. As this performance optimisation is limited to multi-view rendering it is not widely researched or evaluated and this project will feature one of the first implementations of such system. To the best of the author's knowledge no similar system has been implemented or evaluated in an academic or commercial environment.

3. Artefact Design, Development and Testing

This chapter describes the use of the artefact as well as the development process.

3.1 Development and research methodology

The Scrum development methodology was used throughout the development of the project. Each new feature was implemented in a different sprint. The length of each sprint was between a few days and two weeks. Smaller features that are relevant to each other were combined and implemented in one sprint. In some occasions such as the dynamic resolution optimisation specifications changed after the original sprint and a shorter sprint was needed to accommodate the changes.

Testing was done mostly at the end of the development cycle. Manual integration testing was utilized to make sure that the components work and the user cannot provide awry values and break them. Additional care was taken to make sure that all of the custom cameras could work with a variety of field of view and culling settings. The Unity profiler was used to confirm that there are no memory leaks and unusual bottlenecks. After the testing period all the detected problems were fixed and the final performance tests were executed.

Git was used for version control and the private repository hosting was provided by Bitbucket. Microsoft Word was used for the writing of this thesis and Dropbox was used for backup and synchronisation.

Quantitative research methodology was utilized in the performance evaluation as it is more suitable for the purposes of this project than a qualitative approach. Performance was measured and evaluated in the thesis and used to determine the best use cases for each performance optimisation technique.

3.2 Use of the artefact

The artefact is a tool for evaluation of different rendering optimisations for traditional, stereoscopic and multiscopic rendering. It also consists of the implementation of all the optimisations and is set up to be as simple to implement to existing Unity projects as possible. As the artefact is also a plug-in, in order to reduce its size there will be no example scenes included. The user has to import the plug-in to an existing Unity project in order to properly utilize and measure the performance optimisations.

The artefact will be both shared as a Unity project repository on Github as well as downloadable “.unitypackage” file that can be imported and used in an existing project.

The rendering optimisations are split in three categories: traditional, stereoscopic (interlaced) and multiscopic rendering.

For each category there are four prefabs:

- Default rendering
- Foveated Rendering
- Depth Optimisation
- Dynamic resolution

The prefabs act as cameras when added to the scene. The user can edit different options for the rendering optimisations using the “inspector” tab in Unity.

The artefact also features a system that can track, measure and save the performance information into a file that can be opened with Microsoft Excel. The evaluation graphs in this thesis are generated using the information from those files.

3.3 Design and implementation of the performance optimisations

This subsection provides additional information on the implementation and use of the performance optimisation techniques.

3.3.1 Foveated rendering system

Each foveated rendering view requires at least two cameras. One of the camera is readjusting based on the user gaze point and renders a small portion of the screen. The other camera renders the whole screen, but with a reduced resolution. To further increase performance it is possible to have two level of detail versions of every model. The foveated camera should render the high-poly version of the models while the camera that renders the whole screen should only use the low-poly objects. The cameras render to “render textures” and the “render textures” are then displayed to the screen with their proper location and order. This enables us to change the rendering resolution at will. The user can provide the parameters for the rendering resolutions for both of the cameras, as well as additional rendering settings.

The parameters that can be adjusted include:

- Change resolution multiplier for each region. The resolution of the scene is multiplied to the resolution multiplier. In order for a region to have a resolution of half the normal resolution the multiplier should be set to 0.5. In order to use super sampling

anti-aliasing the user can choose a resolution multiplier of 2 or more.

- Antialiasing level of the render texture. This value should be either 1, 2, 4 or 8. It is available for every region.

- The location of the foveated rendering on screen. The foveated rendering can be used with both gaze location, mouse location or it could be placed in the centre of the screen. In order for gaze location to be used the user needs to implement an eye tracking solution and provide the location to the foveated rendering.

- The user can apply image processing effects for each region. The foveated rendering can work in conjunction with other full screen image processing effects.
- The user can change the drawing order of the render textures or even use the output render textures for the needs of the application.
- The user can change the rendering layer of the camera. When each LOD level is in a different layer the user can specify the LOD layer for each region.
- The resulting foveated region is a square.

The foveated rendering system is applied in a different way depending on whether the rendering is in 2D, stereoscopic or multiscopic.

The traditional (2D) foveated rendering camera consists of a prefab game object with three game objects as children (Figure 15).

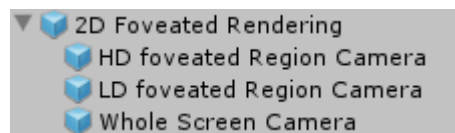


FIGURE 15 2D RENDERING CAMERA PREFAB

The prefab is called “2D Foveated Rendering” and has only one component called “RenderTexturesDrawingOrder” (Figure 16).

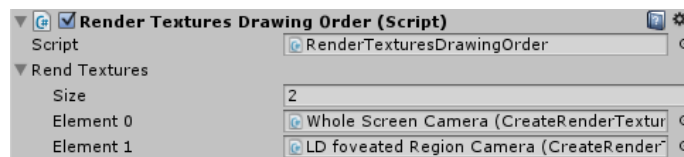


FIGURE 16 DRAWING ORDER OF THE RENDER TEXTURES

As the name implies the script is used to display the “render textures” of the foveated regions as well as the whole screen before them. An array of “CreateRenderTexture” objects is used to provide connection to the “render textures” to be rendered. The order of the objects in the array is used to determine the rendering order of the render textures. The current implementation of 2D foveated rendering is the only implementation of foveated rendering in the artefact that provides the possibility of having more than one foveated region. The stereoscopic and multiscopic foveated renderings are implemented with only

one foveated region in mind due to limitations in the maximum amount of textures in a shader.

The “RenderTexturesDrawingOrder.cs” script draws the render texture on the screen based on the order of the “Rend Texture” and the “full screen” toggle for each ‘CreateRenderTexture’ object. The first texture to be displayed before the foveated regions should always be a full screen “render texture”.

The foveated regions are never full screen textures. They are displayed at the appropriate location on the screen using the focus point, which could be the gaze-point, mouse position or other position supplied to the “UpdateViewport” component.

Every game objects that is a child of the “2D Foveated Rendering” is a camera.

The user can manually add or remove cameras as each camera is another region of the foveated rendering. The prefab is by default set to draw the “render textures” of two cameras. There is only one foveated region and the other camera renders the whole screen. Another camera is available, called “HD foveated Region Camera”, but it is disabled by default. The purpose of this camera is to demonstrate that additional cameras could be added and it was originally used to render a foveated region with super-sampled resolution in a configuration of two foveated regions. In order to activate this foveated region the user has to enable the “HD foveated Region Camera” game object and place it as the last element of the “rendTextures” array in the “RenderTexturesDrawingOrder” component.

Every camera has its own components that determine multiple rendering attributes.

“Whole Screen Camera” renders the whole screen before the foveated rendering is applied on top.

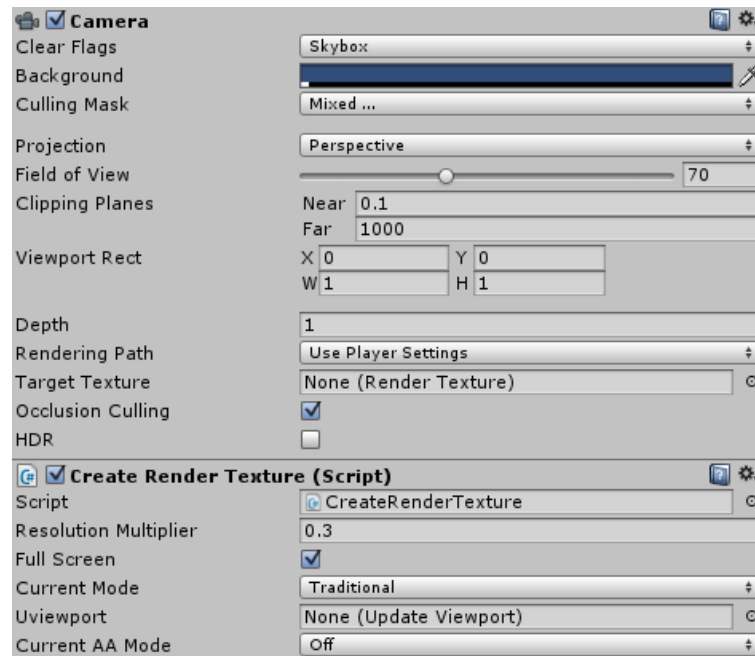


FIGURE 17 WHOLE SCREEN CAMERA FOR FOVEATED RENDERING

The “Whole Screen Camera” game object has two components attached (Figure 17). The first component is a standard Unity camera component. The culling mask field is important as it provides additional performance increase. It is possible to set the “Whole Screen Camera” to render the low resolution version LOD level while the cameras that render the foveated regions could render the high resolution LOD level.

The field of view value of the “Whole Screen Camera” is used as the basis for the calculation of the field of view of the foveated regions. All of the performance optimisations are designed and implemented in a way to accommodate the use of any field of view and aspect ratio.

The “CreateRenderTexture” component has multiple fields. Each field does as follows:

- “Rend Tex” is exposed in the inspector for debugging purposes. The render texture is going to be created on start.
- “Resolution multiplier” is a float variable that is going to determine the resolution of the render texture.
- Full screen toggle specifies whether the render texture is full screen or not.
- The “Current mode” field can be set to “Traditional”, “Stereoscopic” or “Multiscopic”
- “Uviewport” is a field that should be given a reference to the “Update Viewport” component only if the camera is used for a foveated region. If the camera renders the whole screen this reference will not be used and can be ignored.
- “Anti-Aliasing Power of Two” is used to specify the antialiasing value of the render texture. It can hold a value of 1, 2, 4 or 8. This will be ignored if there are image processing effects added to the camera or the camera has enabled high dynamic range.

The current mode is used to determine the resolution of the created render texture. If the mode is set to stereoscopic the width of the render texture will be half of the width of the screen. Each mode requires render textures with different sizes.

Every camera used in a foveated region has “UpdateViewport” component attached (Figure 18). The “UpdateViewport” component is used to update the viewport according to the focus position value.

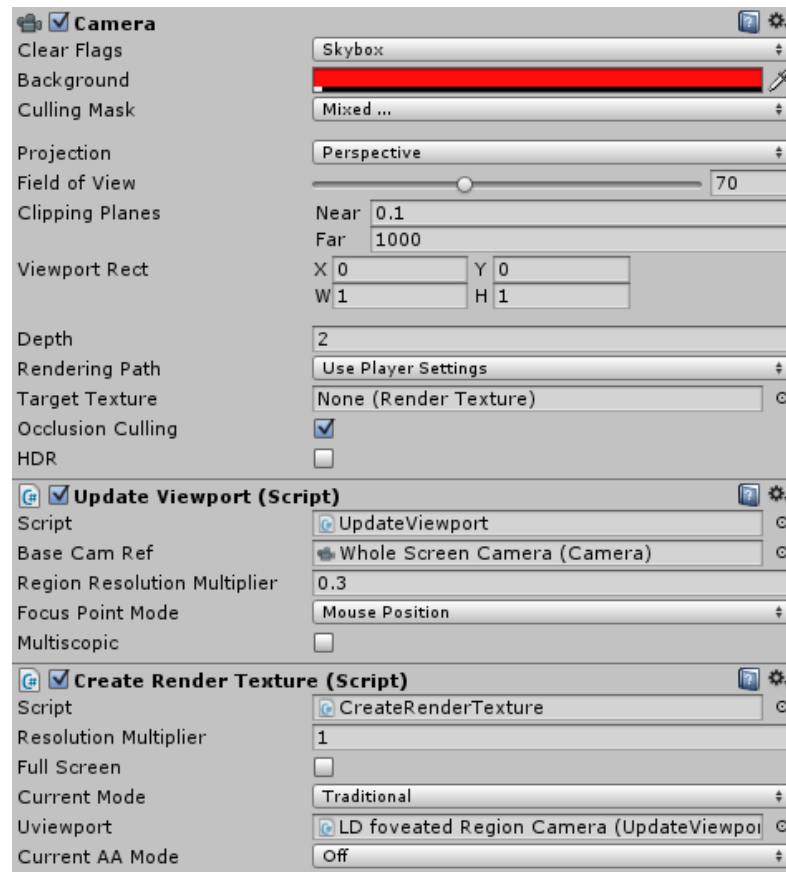


FIGURE 18 FOVEATED REGION

The exposed fields of the “UpdateViewport” are as follows:

- “Base Cam Ref” is a reference to the “base camera”, which is the camera that renders the whole screen. In this example that camera is called “Whole Screen Camera”. The field of view of this camera determines the field of view of the foveated cameras.
- “Region Resolution Multiplier” determines the size on screen of the foveated region. It is a float variable multiplied to the screen height resolution to determine the size of the foveated region. A 0.25 “Region Resolution Multiplier” would result in a foveated region that is 25% the screen height.
- “Multiscopic” is a Boolean toggle and should be enabled if the camera is used for multiscopic foveated rendering.
- The “Region Resolution Multiplier” should be changed according to the eye tracking latency, the display latency and the screen size and distance from

the user. The longer the delay the bigger the “Region Resolution Multiplier” should be. If the screen is large and is far from the observer the “Region Resolution Multiplier” should be large. Head-mounted displays can have a very small foveated region.

As the name implies the “UpdateViewport” changes the viewport of the cameras every frame.

In order for foveated rendering to work at different field of views the field of view of the foveated region camera needs to be calculated from the field of view of the “whole screen” camera. A standard frustum size at distance formula (Unity Technologies, 2015) is used to calculate the field of view of the foveated region camera.

Other variables calculated are the “helperValX” and “helperValY”. They are used for properly shifting the vanishing point of the cameras.

If the multiscopic toggle is enabled the code is almost the same with the only difference that the camera aspect ratio is explicitly set to “1” and another variable helper “multiscopicHelper” is calculated. The code is fully commented and provides additional information regarding the implementation of each system.

The “SetFocusPoint” function is called every time the foveated region needs to move. The “focusPoint” variable is used to determine the new position of the foveated region. A new vanishing point (Unity Technologies, 2015) for the camera is set using the new “focusPoint” and the helper variables. This only happens if the “focusPoint” is somewhere in the application window.

The camera prefabs for Stereoscopic and Multiscopic foveated rendering work in a different way from the traditional foveated rendering with one view. Instead of displaying the render texture on the screen, shaders are needed to combine the views appropriately.

The “Multiscopic Foveated Rendering” and the “Stereoscopy Interlaced Foveated Rendering” prefabs consist of only one game object each. This is due to the fact

that they use additional helper prefabs for each view. The helper prefabs are instantiated in the right locations and each one represents a different view.

The “Stereoscopy Interlaced Foveated Rendering” prefab consists of two components. A standard camera and a custom “InterlacedFoveatedCameraSetup” component (Figure 19).

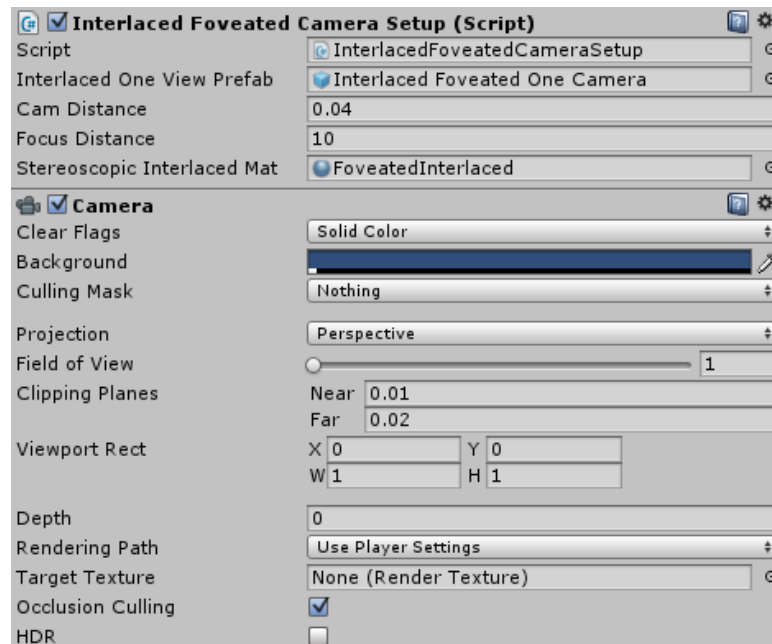


FIGURE 19 STEREOSCOPY INTERLACED FOVEATED RENDERING

- “Interlaced One View Prefab” is a helper prefab with a camera set for foveated rendering that will be used for each of the two views.
- “Cam Distance” determines the distance between the two cameras.
- “Focus Distance” - the cameras will point at a location that is at this distance in front of this game object.
- “Foveated Interlaced Mat” is the material with the shader set for foveated interlaced rendering.

The “Interlaced Foveated One Camera” helper prefab consists of only one camera and is a copy of the “2D Foveated Rendering” prefab with small changes. The “RenderTexturesDrawingOrder” component is disabled and a new component called “Ref Helper” is attached to it. This component provides references to the two render textures.

The “InterlacedFoveatedCameraSetup” component is used to set up the stereoscopic foveated rendering cameras. In the “Start” function two camera systems for stereoscopy are instantiated and set up accordingly. In the “SetMatProperties” the material that will interlace the textures is provided. Each of its properties will be set-up in order for the shader to interlace all of the render textures into stereoscopic foveated rendering. The final function of the “InterlacedFoveatedCameraSetup” component displays the shader pass on the screen.

As previously mentioned the “Foveated Interlaced Mat” is a reference to the material with a shader that can combine the two foveated views into one. The shader is called “FoveatedInterlaced” and requires four render textures. Each foveated view has two textures and there are two views for stereoscopy. As shaders do not work with pixels the screen and resolution also need to be provided to the shader. The fragment segment of the shader determines the pixel location of the current texel and returns the appropriate texture.

In order to support other types of stereoscopic rendering such as “side by side” the shader needs to be edited accordingly. For the artefact of this project the only targeted stereoscopic implementation is horizontally interlaced with each pixel row being either perceived by the left eye or the right. Later a more powerful system could be implemented that enables the user to edit the interlaced type between horizontal and vertical as well as side by side stereoscopy.

The “Multiscopic Foveated Rendering” prefab (Figure 20) works in a very similar manner.

A helper prefab is used for each view of the multiscopic foveated rendering. The helper prefab is very similar to the helper prefab for foveated stereoscopic rendering but all of the modes in its “CreateRenderTexture” component are set to multiscopic.

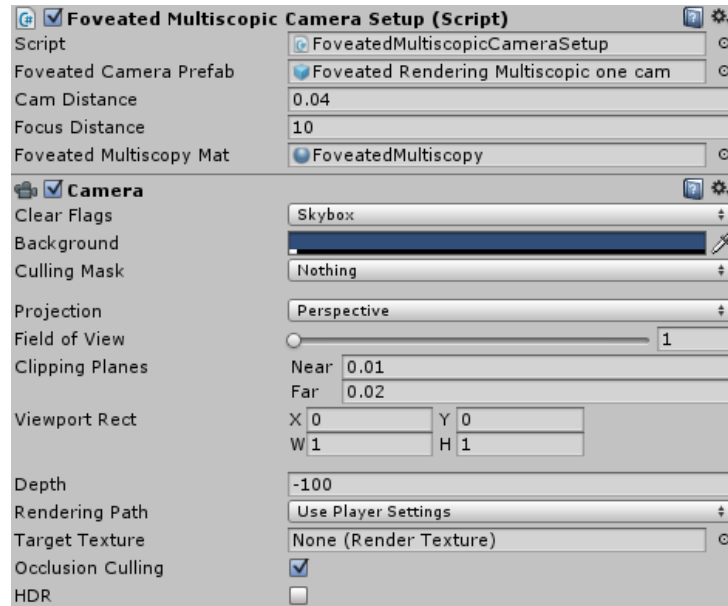


FIGURE 20 MULTISCOPIC FOVEATED RENDERING

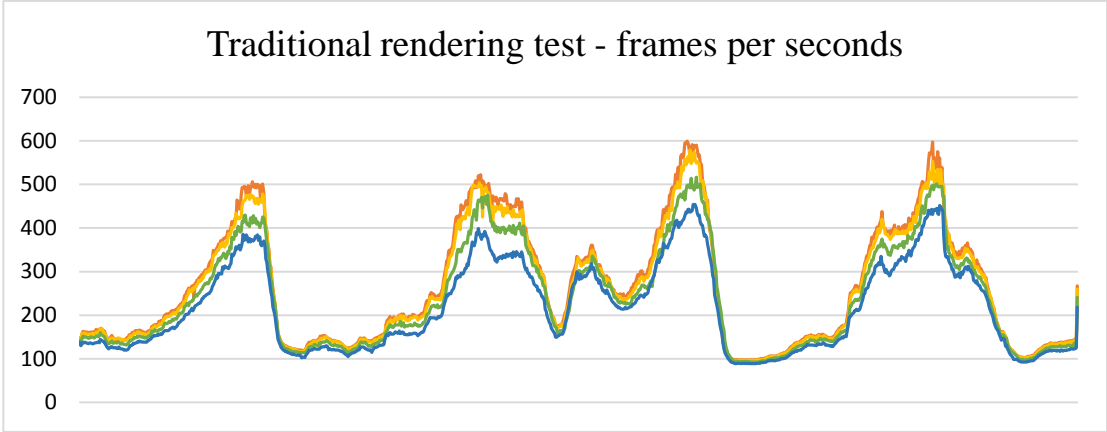
The shader used to combine the views is called “FoveatedMultiscopy” and requires sixteen textures, two for each of the eight views. It first determines the pixel and then the views of every subpixel. It also determines if the pixel is part of the foveated region or not and returns a texel accordingly. Since the Unity texture limit per shader is sixteen the implementation of foveated rendering with more than one region is not possible with the current implementation. Other implementations with multiple shaders may enable the use of many foveated regions.

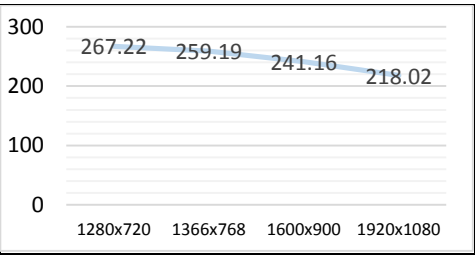
3.3.2 Dynamic resolution system

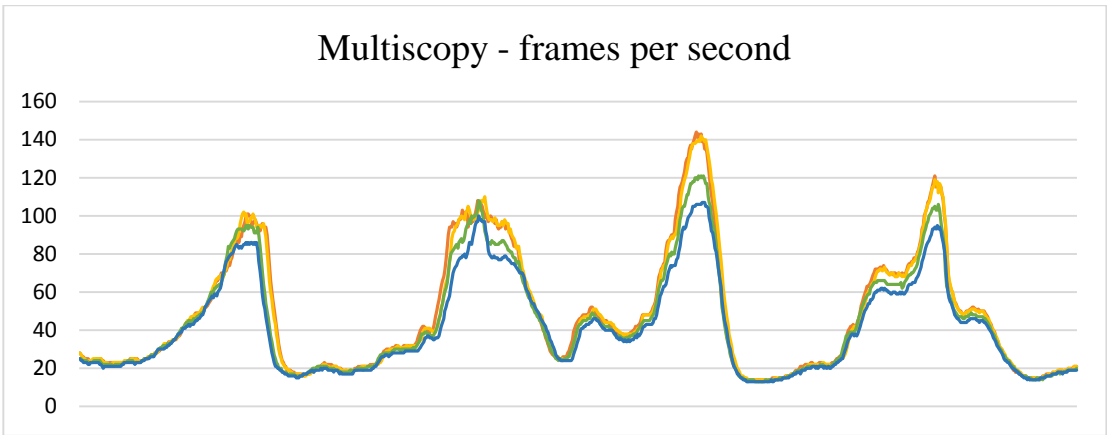
Dynamic resolution is rarely used in video games as it generally decreases the overall experience quality of the user. It is questionable whether dynamic resolution is very beneficial for the performance and the first part of this subsection will try to answer this question by evaluating data gathered from conducting an experiment.

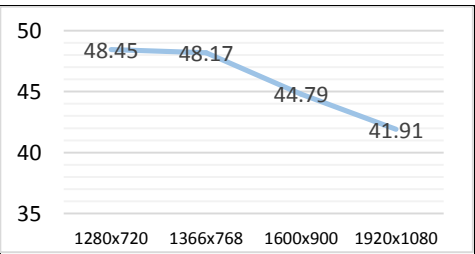
One of the test scenes called “Viking Village” was used to conduct the experiment and it consists of running the same walk-through scene at different resolutions. The expected result is a sufficient increase of performance with resolution reduction to justify implementation of the system. Another expected result is scalability with

very predictable increase factor when the resolution is reduced. The experiment was conducted for both traditional rendering as well as multiscopic rendering and the resolutions evaluated were four of the most used resolutions.



Performance test – Viking Village – traditional		
Resolution:	Average fps	
1280x720	267.22	
1366x768	259.19	
1600x900	241.16	
1920x1080	218.02	



Performance test – Viking Village – Multiscopic		
Resolution:	Average fps	
1280x720	48.45	
1366x768	48.17	
1600x900	44.79	
1920x1080	41.91	

As seen in the comparison chart rendering at reduced resolution increases performance and the performance increase is very predictable. A reduction of the resolution from 1080p to 720p results in an increased performance by 20.27% for traditional rendering and 14.47% for multiscopic rendering. It is therefore a viable performance optimisation technique and it is going to be evaluated in this project.

The dynamic resolution (Figure 21) system does not track performance but it needs the performance information to properly function. The “PerformanceManager” component attached to prefab with the same name measures the performance. It has some additional functionality and can display the performance information on either the left or the right corner of the screen. The “PerformanceManager.cs” script is also a singleton and only one instance should be enabled in the scene at any given moment.

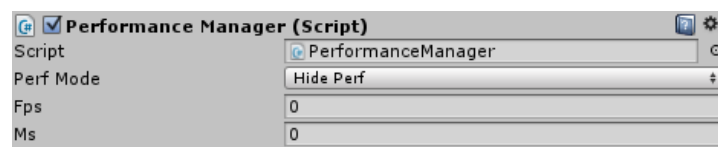


FIGURE 21 PERFORMANCE MANAGER

The overall performance of the scene is tracked by tracking two variables. The “Fps” is the frames per second of the application. Bigger “Fps” numbers mean smoother gameplay experience. The “Ms” is the milliseconds each frame takes to render. One of the variables can easily be calculated from the other, but both of them are displayed in the Unity inspector for the convenience of the user.

The Performance Manager was compared to other performance tracking solutions such as “Fraps” and “GeForce Experience” and proved to be a robust performance tracking solution with fast response.

As with the other performance optimisations, adding any of the dynamic resolution camera prefabs would result in a proper implementation of the camera in the scene, but all of them still need an instance of the “PerformanceManager” prefab to function properly. If there is no instance of the “PerformanceManager” in the scene it will be instantiated when needed.

The dynamic resolution prefab for traditional rendering is called “2D Dynamic Resolution” and it has two components attached. One of the components is a standard camera and the other component is the “DynamicResolutionManager” script.

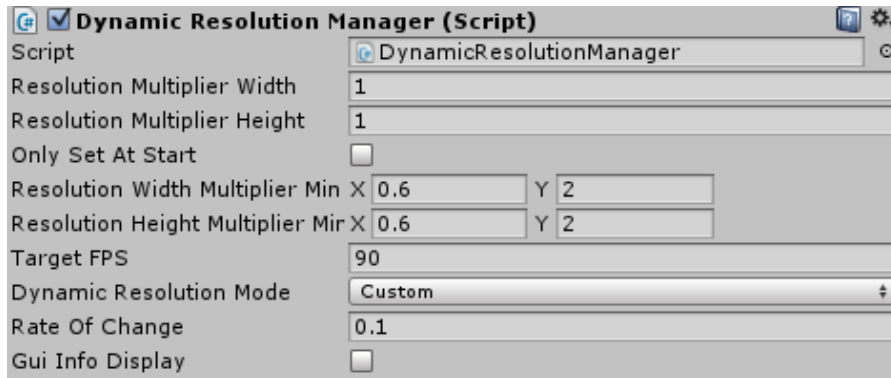


FIGURE 22 DYNAMIC RESOLUTION MANAGER

The user has a large variety of options to edit:

- “Resolution Multiplier Width” and “Resolution Multiplier Height” are the starting resolution multipliers for the width and height.
- If “Only Set at Start” toggle is enabled the initial resolution multipliers will be used at the start to set the resolution and the resolution will not be changed based of performance.
- “Resolution Width Multiplier Min Max” and “Resolution Height Multiplier Min Max” are used to specify the minimum and maximum values of the resolution multiplier.
- “Target FPS” is the currently targeted frames per second the dynamic resolution system will try to achieve.
- “Dynamic Resolution Mode” selects the dynamic resolution mode. There are two different versions of dynamic resolution implemented and this field changes between them.
- “Rate of Change” specifies the “wait time” of the coroutine that calls the functions that changes the resolution dynamically.
- When “Gui Info Display” toggle is enabled a gui text appears at the top right corner of the screen. The text shows some information regarding the dynamic resolution system.

With dynamic resolution the user specifies a target frame rate and the application changes resolution at run time in order to meet the frame rate target. The difference between current frame rate and targeted frame rate is evaluated once in a while and the resolution multiplier variable is changed based on that. The resolution multiplier variable is multiplied by the screen resolution and the resulting value is the new resolution. When performance is needed the resolution could be very low and when the frame rate exceeds targeted frame rate the resolution multiplier variable is increased and an anti-aliasing super sampling is achieved. Sometimes in video games there are scenes in which a major event happens with a lot of geometry and effects displayed on the screen at the same time. This leads to reduced performance and lagging.

With a dynamic resolution system such event would reduce the rendering resolution in order to keep the application playable and the user experience smooth. In order to only benefit from super-sampling the dynamic resolution could be set up to only increase the resolution and never decrease it beneath the monitor screen resolution. Instead of trading quality for performance this is a trade of frames per seconds for super sampled resolution. At rare performance intensive scenery the super sampling effect would be disabled in order to increase performance as needed.

As the change of the resolution multipliers leads to generating new render texture it is performance intensive in itself and should not happen every frame. A coroutine called “AdaptResolution” is utilized. Its rate of change value could be set from the “Rate of Change” field. Different applications may require different coroutine execution times and therefore experimentation is advisable.

The dynamic resolution system has two different modes implemented. The first mode, called “Custom”, is designed to reduce the amount of times the resolution changes.

The user experience deteriorates when the resolution changes frequently and if this happens only when needed it could be very beneficial for the overall immersion. The “Custom” mode only adapts the resolution when the current frame rate and the target frame rate have a difference greater than two frames. The “Custom” mode

features a novel implementation that was designed and created for this project and its purpose is to provide an alternative to the widely adapted “Intel” method (Intel, 2011). The “Intel” mode features a standard implementation of dynamic resolution first proposed by Intel.

The dynamic resolution for stereoscopic and multiscopic rendering have very similar implementations but they also have additional fields exposed in the Unity “inspector”. These fields are intended for setting up the stereoscopic and multiscopic rendering. The fields of the components that control the dynamic resolution system are the same for traditional, stereoscopic and multiscopic rendering.

3.3.3 Depth reuse system

The camera frustum of the depth reuse system is made out of two different frustums from two cameras. One of the cameras renders the closer part of the frustum and the other one renders the further part. The camera that renders the closer frustum has its “Clear flags” set to “Solid Colour” and the background colour alpha channel set to 0. The alpha channel is used in the shader to stitch the two textures together.

The goal of the depth reuse system is to render the distant objects of the scene only once and reuse them in all of the views for stereoscopy and multiscopy. The resolution of the depth texture can be changed to further increase performance. This performance optimisation is not expected to increase the performance of traditional rendering of one view unless the render texture resolution multiple of the far camera is set to a low value.

Having two frustums for close and far objects also gives us the possibility to change their rendering rate. Even though this is not implemented in the artefact it is possible to set the far camera to render at thirty or fifteen frames per second and the closer camera to render at sixty. Just like in real world the further an object is from the camera the slower its movement is perceived. In some cases high frame rates for distant objects is not needed.

The implementation of depth optimisation in the artefact can render two LOD levels with the two cameras to further increase performance.

The frustum culling for the two cameras is set to 0.1 to 30 for the camera that renders near objects and 27 to 1000 for the camera that renders far objects. The two frustums should overlap, otherwise artefacts appear. Occlusion culling should also be disabled for the camera that renders far objects for the proper functioning of the system.

In order to add a depth optimised camera to an existing screen the user has to add the appropriate prefab to the scene. Depth optimisation camera prefabs exist for traditional, stereoscopic and multiscopic rendering.

The “2D Depth Optimisation” prefab provides depth optimised camera for traditional rendering of one view. The game object has three child objects.

- “Near” is the camera that renders game geometry that is close to the camera.
- “Far” is the camera that renders far objects.
- “ScriptHolderCam” is another camera that does not render anything, but is required for the combination process.

The “ScriptHolderCam” has a component called “Combine Depth” (Figure 23).

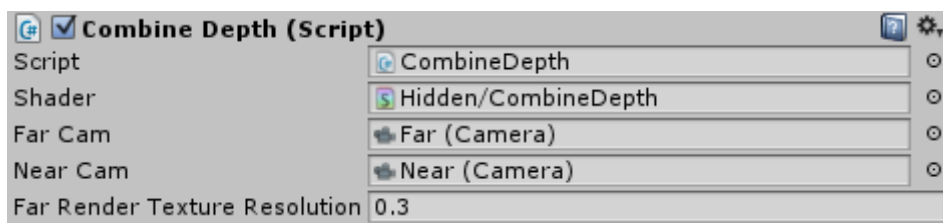


FIGURE 23 SCRIPT HOLDER CAM GAME OBJECT

The fields exposed to the user are as follows:

- “Shader” is a reference to the shader used for combining the render textures.
- “Far Cam” is a reference to the camera that renders far geometry
- “Near Cam” is a reference to the camera that renders near geometry
- “Far Render Texture resolution” is float value multiplier used to determine the resolution of the output render texture of the far camera.

The “CombineDepth.cs” derives from “ImageEffectBaseCustom.cs” which is a modification of the “ImageEffectBase.cs” script from Unity.

The shader that combines the two render textures tests if the alpha colour value of each texel is smaller than “1”. If the value is smaller the output is set to the far texture and if the value is “1” the output is set to the near texture.

The stereoscopic and multiscopic implementations are similar to the traditional implementation, but instead of having two cameras for one viewport there are two cameras for each.

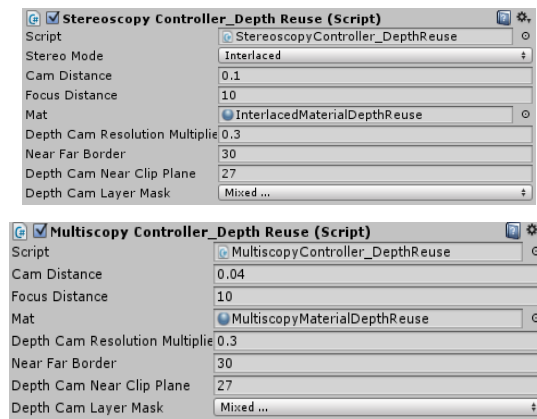


FIGURE 24 STEREOSCOPIC AND MULTISCOPIC IMPLEMENTATIONS DEPTH OPTIMISATION

The scripts from stereoscopic and multiscopic depth optimisation both derive from their appropriate base scripts and therefore reuse the code. The difference from the traditional depth optimisation implementation is that instead of a shader a material reference is required. The shaders are attached to the materials and have very similar implementation to the “CombineDepth” shader, but for multiple views. In order to provide more realistic results the depth render texture is shifted horizontally for each view. Different scenarios require different shifting value and therefore an experimentation is required until a good result is reached.

3.4 Design and implementation of additional systems

This section examines the other systems and techniques utilized in the artefact.

3.4.1 Automultiscopic rendering

Multiscopic rendering plug-in for Unity is not available and the creation of a custom solution for the purposes of this project was required.

After extensive research in multiscopic rendering, shaders and the workings of the Alioscopy monitor that was available the end product was a small and robust plug-in that lets the user combine multiple views into an output texture suitable for lenticular automultiscopic screens.

The Alioscopy monitor works by combining subpixels of different views. Each new pixel is a combination of three subpixels from three different views. This is a trade-off between width and height resolution.

Subpixel rendering is not a new concept and has been used as an anti-aliasing effect for text when rendered on screens that have separated red, green and blue subpixels. When using subpixel rendering it increases the width resolution by three times and decreases the height resolution by three times.

The lenticular sheet of the automultiscopic Alioscopy monitor is placed at an angle in which every subpixel in a row of pixels is a part of a different view. Each pixel consists of three different views. Each subpixel of the original pixel becomes a part of another pixel. Different subpixel configurations, varying lenticules per inch and different rotations of the lenticular sheet can lead to different amounts of views displayed on the screen. The Alioscopy monitor that was used for testing consists of eight different views and a resolution of 1920 by 1080 pixels. After the subpixels are combined the new resolution becomes 5760 by 360. Since there are 8 views the resolution for each view is 720 by 360.

This is a resolution that is not viable for commercial or home use, but with the creation of displays with higher resolution this technology will become more viable.

As of 2015 4K “Ultra High Definition” screens are widely available with a standard resolution of 3840 by 2160. If such screen is used for eight view automultiscopic monitor the resolution of each view would be 1440 by 720 pixels.

Future technologies such as the 11K Samsung display announced to be released in 2018 (Hardawar, 2015) would have a resolution of 11264 by 6336 pixels and if used for eight view multiscopic rendering each view would be with a resolution of 4224 by 2112 pixels.

Traditional Resolution:	Subpixel rendering resolution:	Resolution per view (8 views)	Total amount of pixels:
1280x720	3840x240	480x240	921600
1920x1080	5760x360	720x360	2073600
3840x2160	11520x720	1440x720	8847360
11264x6336	33792x2112	4224x2112	71368704

When a rendering output for multiscopic screen is viewed on a traditional screen the image appears blurred. Only after the image is displayed on the specialised monitor with the appropriate pixel distribution, lenticules per inch and angle of lenticular lens the automultiscopy is perceived. Figure 25 demonstrates the difference between rendering for a traditional screen and the output of combining eight views for automultiscopic screen.

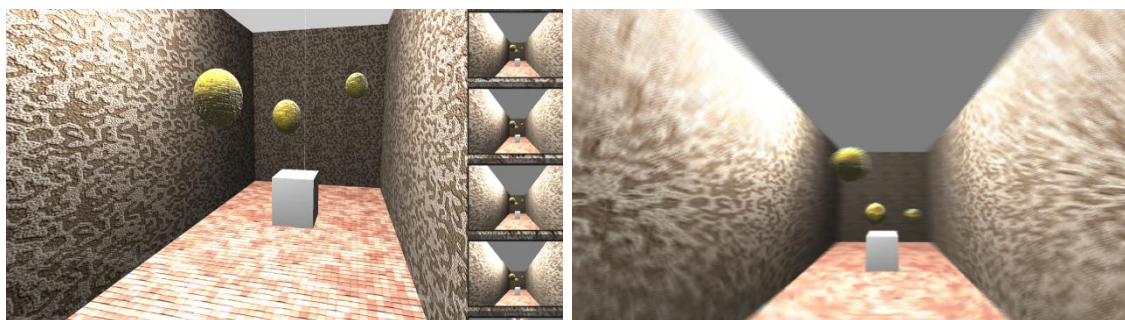


FIGURE 25 TRADITIONAL AND MULTISCOPIC RENDERING

Currently available multiscopic displays have a small amount of views and even though they provide more freedom of movement then autostereoscopic screens, the user cannot move more than a few centimetres horizontally. One solution for this problem is the use of head tracking to move the virtual cameras based of the user head position, but it would also require a more active parallax barrier.

3.4.2 Performance measuring system

When performance was measured the frames per second and the milliseconds per frame values were saved into an excel file. The values were used for the creation of numerous graphs that can be seen in the “Evaluation” section of this thesis.

The “PerformanceManager” can only track performance but does not save it to a file. In order for the system to save the performance information into a file a “SavePerfDataToFile” component needs to be added to the scene. There are two types of “SavePerfDataToFile” components. The base version is the one that saves the frames per second and the milliseconds per frame to file (Figure 26). The second component that also inherits all the features of the “SavePerfDataToFile_Base” is called “SavePerfDataToFile_DynamicResolution” and can save additional information for the dynamic resolution current values to the file.

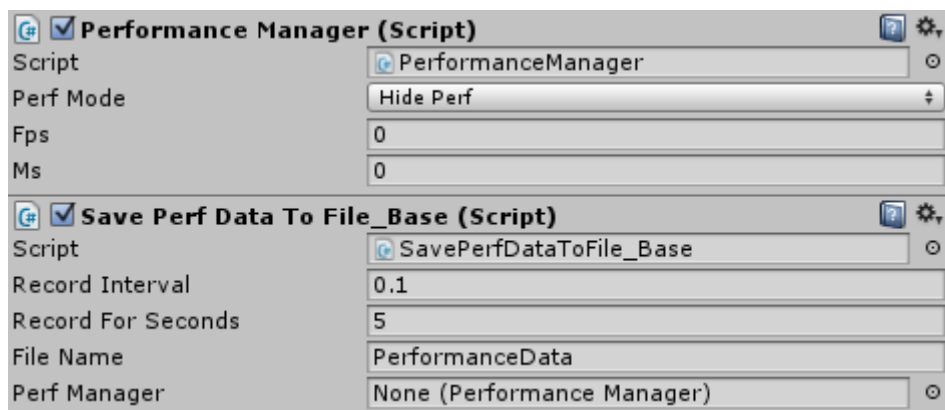


FIGURE 26 PERFORMANCE MANAGER SET TO SAVE DATA TO FILE

The “SavePerfDataToFile_Base” component has multiple fields:

- “Record Interval” is the interval at which performance information will be recorded.
- “Record for Seconds” is the length of the recording. The information is recorded for this amount of time and at the end of the recording the performance information file is generated.
- “File name” is a string of letters that are used as the beginning of the recording file name.

- “Perf Manager” is a reference to the performance manager. As multiple “SavePerfDataToFile” instances can possibly be added to the same scene they could be attached to game objects different to the “Performance manager” game object, but they all need a reference to the “Performance manager” game object to record performance.

In order to properly evaluate different optimisation techniques all of the cameras have to follow the same movement path. A spline editor (Schoen, 2013) was used for each performance experiment. The spline system is not a part of the artefact and is not available in the plug-in.

3.5 Problems encountered

The first performance optimisation implemented was foveated rendering. Dynamic resolution and depth optimisation followed afterwards. Through the development process there were multiple problems encountered. Some of the most persistent ones are as follows:

- The first problem was the proper implementation of subpixel rendering. This proved to be a hard task and required custom shader that combines eight views into one. Each view is rendered at its proper resolution and therefore no pixels are lost in the conversion. This complicates the task even furthered and therefore the first version implemented rendered each view at the resolution of the screen. Since no open source solution for multiscopic rendering was available there was no starting point and extensive research in stereoscopy, multiscopy and shaders was required.
- Properly shifting the foveated rendering camera matrix was an issue. In the original implementation the foveated region was rendered with the resolution of the screen due to shifting of the camera viewport in Unity. With the new implementation that relies on render textures such shifting is not needed and the render textures could be at the exact size required, which increases performance.

- Since the size of the available alioscopy screen exceeds the maximum monitor size for the eye tracking hardware the only way to test foveated rendering on the multiscopic screen is to use mouse location instead of the user gaze. Even though foveated rendering is currently not applicable for large multiscopic screens future high resolution displays for smart phones could be used with eye tracking for foveated rendering in head-mounted displays. The artefact is intended to be a starting point for future developments of foveated rendering for such head-mounted displays.
- The multiscopic foveated rendering could now work well with aspect ratios other than 16 by 9 and field of view other than 70. This issue was later solved by incorporating the field of view and aspect ratio values to the calculation of the vanishing point shift.
- It was not possible to fully test the capabilities of foveated rendering. In order to have a system with unperceivable latency a screen with refresh rate of 120Hz is required and such screen was not available for the development of the artefact. The eye tracker also needs to be with a very low latency and such eye trackers are very expensive and not commercially available.

A custom build of the eye tracking engine was provided by Tobii. It increased the frequency of the eye tracker to 90Hz which makes it applicable for commercial foveated rendering use, but this mode is still being testing and may never be released for the consumers as it reduces the tracking quality.

- The dynamic resolution performance optimisation had a very persistent bug that disabled the shadows of the scenes after a while. It also restarted the computer after prolonged use. The bug was a memory leak due to the generation of the new render texture without destroying the old one.
- The current method of combining views in the depth optimisation does not work well with scenes that have transparent objects. White or pink lines could appear on transparent objects or water. This bug is the result of the current implementation of the depth reuse shader.

3.6 Testing

Testing was conducted shortly after the development of the artefact. Each of the components is thoroughly tested with different settings to make sure that the user cannot break any of the performance optimisation cameras. Multiple bugs were detected and fixed and the overall stability of the system increased.

The Unity profiler was used to check for unexpected bottlenecks and the windows task manager was used to test for memory leaks.

After the testing period the detected problems were fixed. It was important to fix the issues and polish the code before the evaluation of each performance optimisation was conducted as the data gathering process takes a lot of time and small changes in the code would require to repeat the process additional times.

4. Evaluation

In order to evaluate the performance optimisation techniques appropriately multiple experiments were conducted. A custom performance tracking script “SavePerfDataToFile.cs” was implemented. The script has access to the “PerformanceManager.cs” script and logs the current frame rate as well as the milliseconds per frame value. At the end of the performance tracking period the logs are combined into an .xml file that is compatible with Microsoft Excel. The experiments are conducted for the performance optimisations as well as the “Default Rendering” for comparison.

The main goal of the experiments is to illustrate when each performance optimisation increases performance enough to be a viable solution.

The experiments were executed on the same machine and the Unity scenes had the same settings applied in order to minimize the differences between them. The only difference was the scene geometry and the amount of time it took to execute the test. In order to further minimise any differences a waypoint system was used to create a path for the camera. For the foveated rendering performance optimisation the foveated region was locked at the centre of the screen and no eye tracking was used.

The settings used in Unity for the experiments are as follows:

- All cameras have a field of view set to 70.
- The clipping planes of the cameras are set to: 0.1-1000.
- The cameras are set to render the appropriate LOD layers.
- Forward rendering was used.
- No anti-aliasing. Texture quality set to “Full Resolution”.
- The performance was logged once every 0.1 seconds. “The Viking Village” scene walk-through takes 101 seconds. The Museum scene walk-through takes 22 seconds. The Blacksmith scene walk-through takes 65 seconds.
- Due to Unity bug another “dummy” camera was placed. That camera does not render any layer and is solely used to fix a Unity bug. Without this fix

the custom cameras fail to start rendering when the executable is built and played.

Performance optimisation specific settings for the Foveated rendering experiments:

- Low Definition camera set to 0.3 (0.3 of the resolution).
- Foveated region was set to 0.3 (0.3 of the height of the screen).

Performance optimisation specific settings for the Depth rendering experiments:

- Near camera clipping set to: 0.1-30
- 2D Far camera clipping set to: 29-1000
- 3D Far camera clipping set to: 27-1000

For the multiscopic experiments:

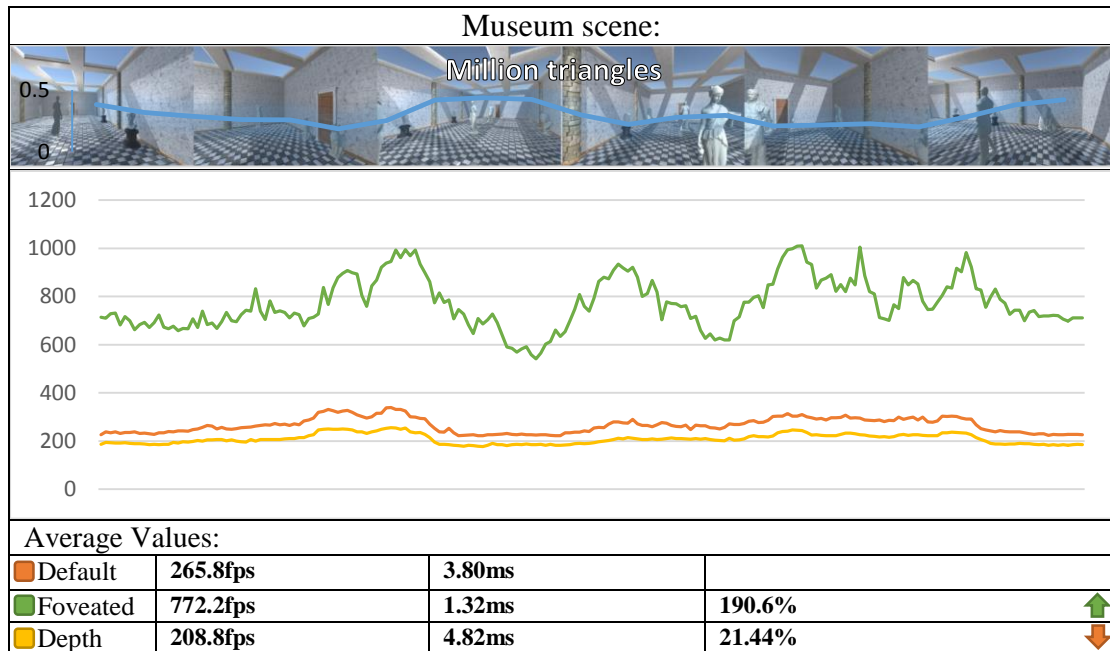
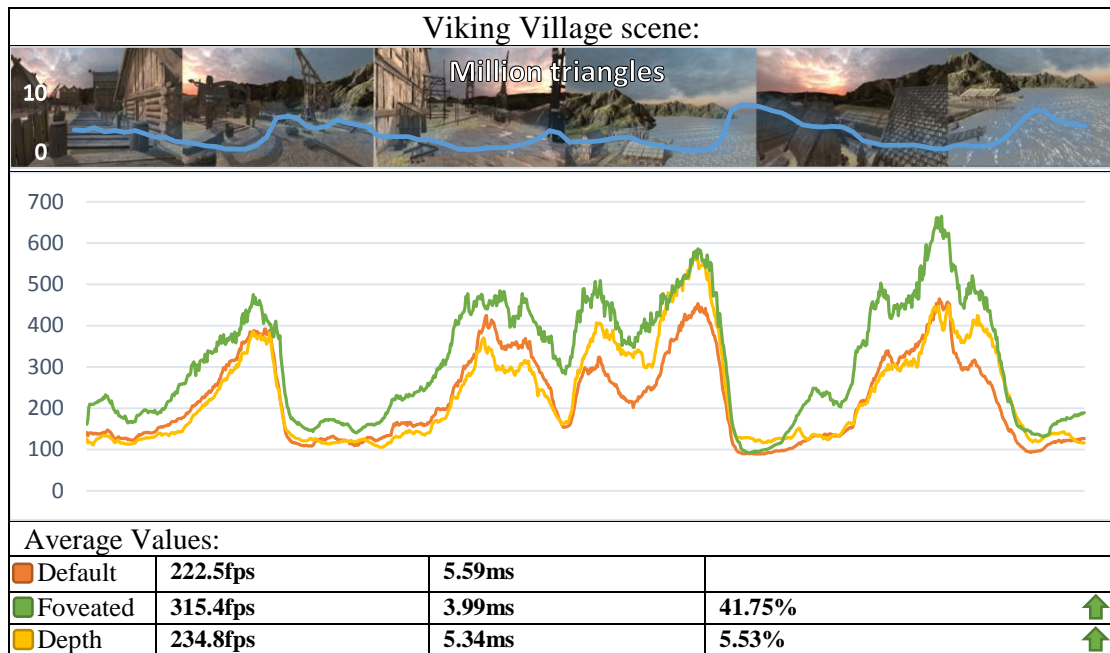
- Distance between the cameras: 0.04
- Focus distance: 10

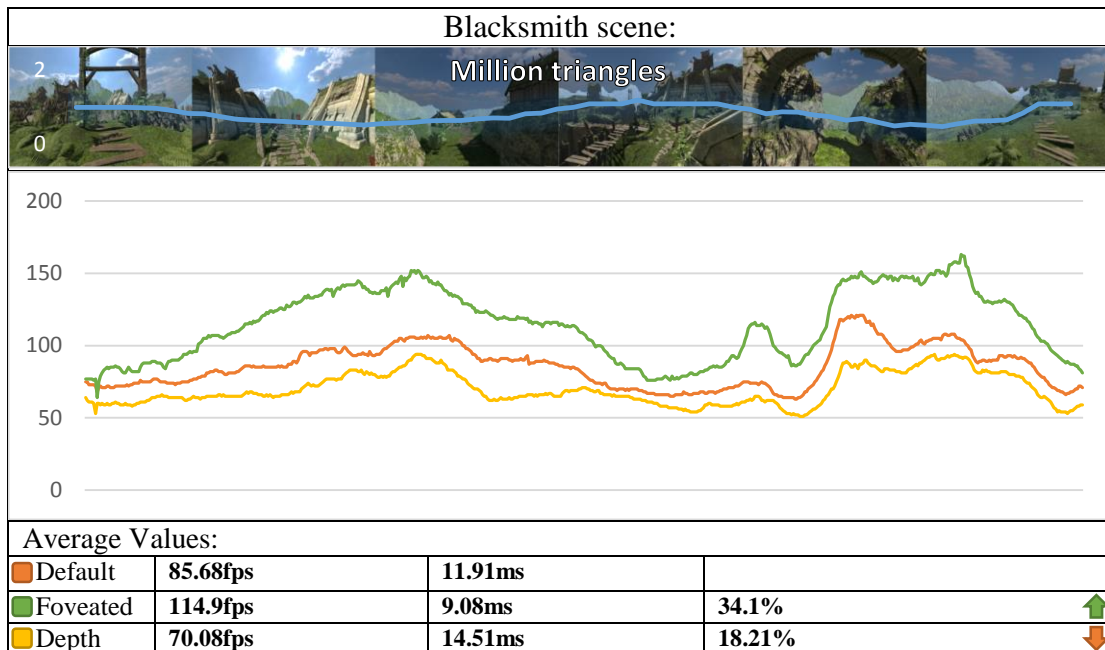
The machine used for the experiments:

GPU	GeForce GTX 750 Ti
CPU	Intel i7 – 3770 3.40GHz
Hard drive:	ATA Crucial CT240M50 SCSI
RAM	8.00GB
OS	Windows 7 Home Premium SP1 64-bit
Resolution	1920x1080
GPU Drivers	355.60 (Release date 13.08.2015)

The three scenes used for the evaluation experiments are chosen for their varying degree of complexity. The “Museum” scene is a closed environment while the “Viking Village” and the “Blacksmith” scenes are large vast areas.

4.1 Traditional rendering

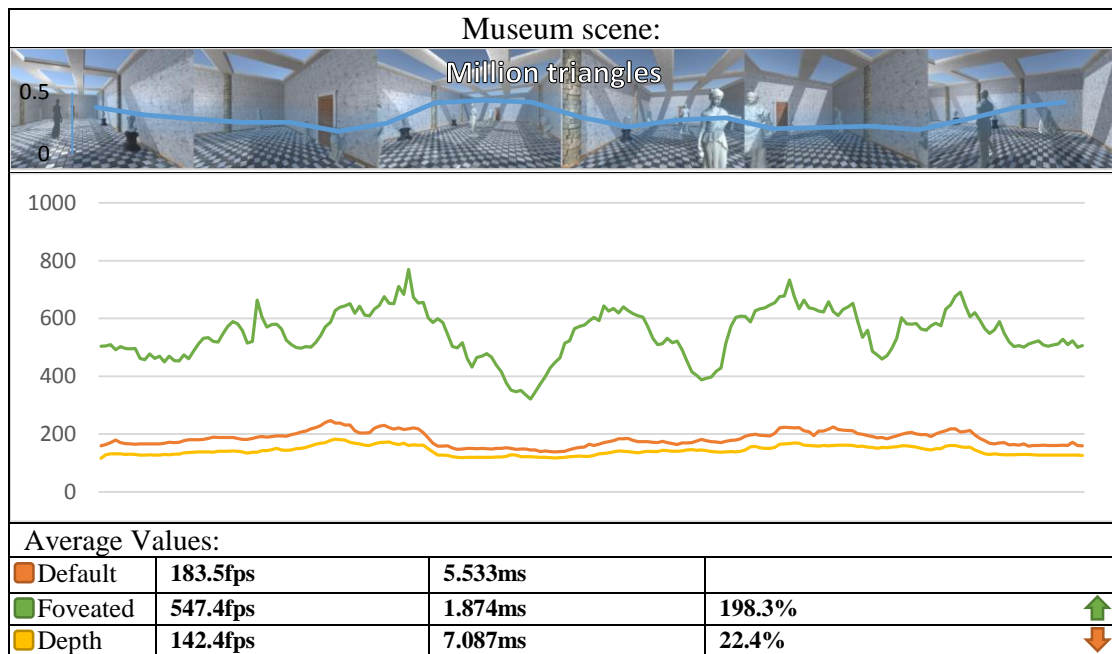
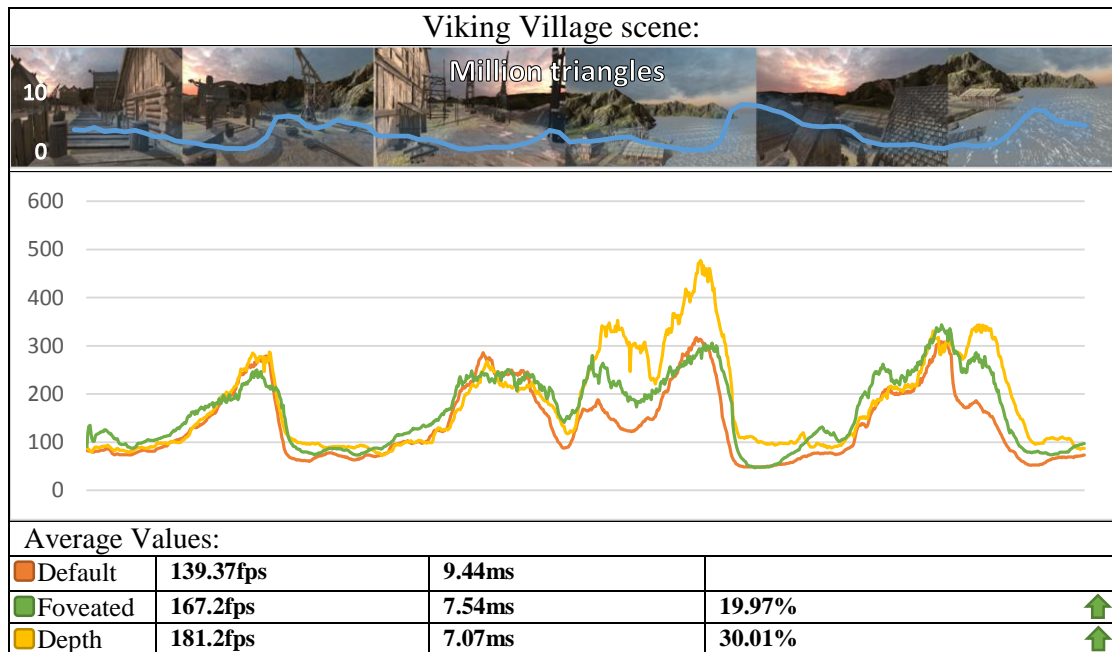


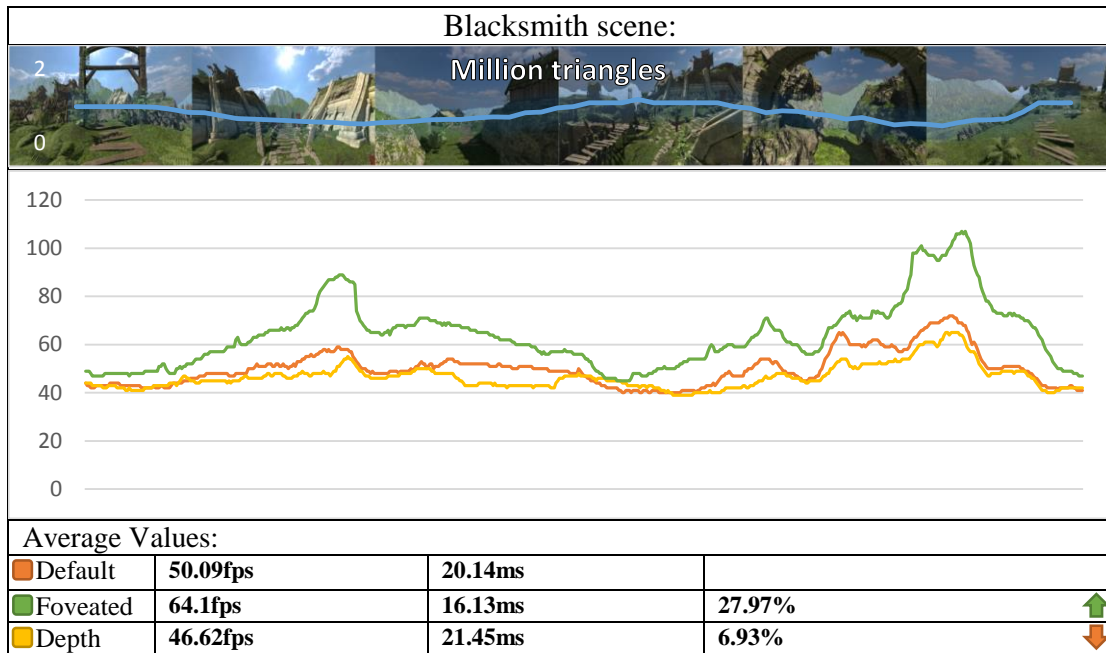


The foveated rendering optimisation proves to be very beneficial for traditional 2D rendering. It increases the average frame rate of the “Museum” scene by 190.6% and also provides a substantial boost in performance for both the “Blacksmith” and the “Viking Village” scenes. Both the “Viking Village” and the “Museum” scenes feature foveated rendering with different level of detail levels which greatly increases the overall performance. The “Blacksmith” scene does not have different level of detail levels implemented and therefore does not benefit from additional performance increase.

The depth optimisation is only beneficial for the “Viking Village” scene and it reduces the performance in the two other scenes. The main goal of the depth performance optimisation is to reuse the same render texture for multiple views and as the traditional rendering only requires one view no reuse is utilized.

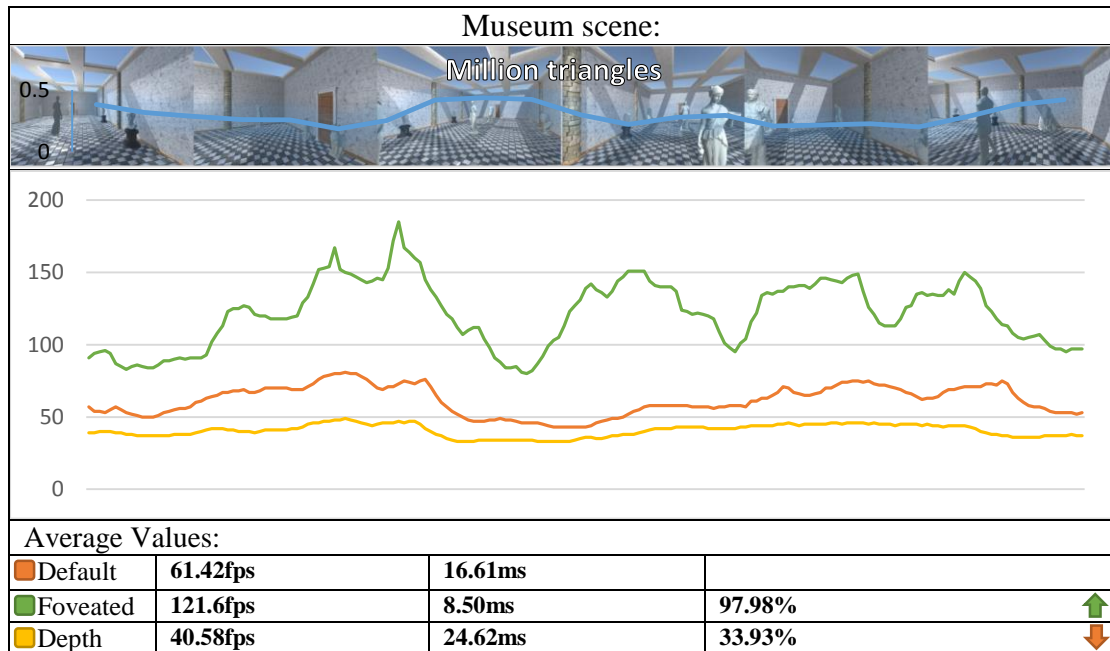
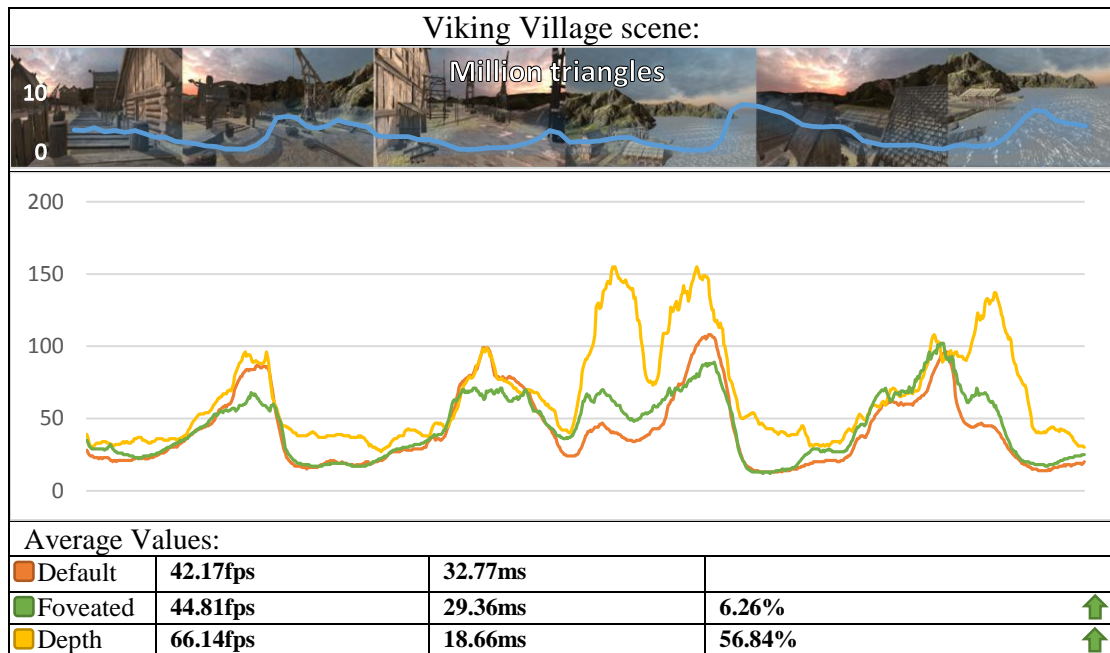
4.2 Stereoscopic rendering

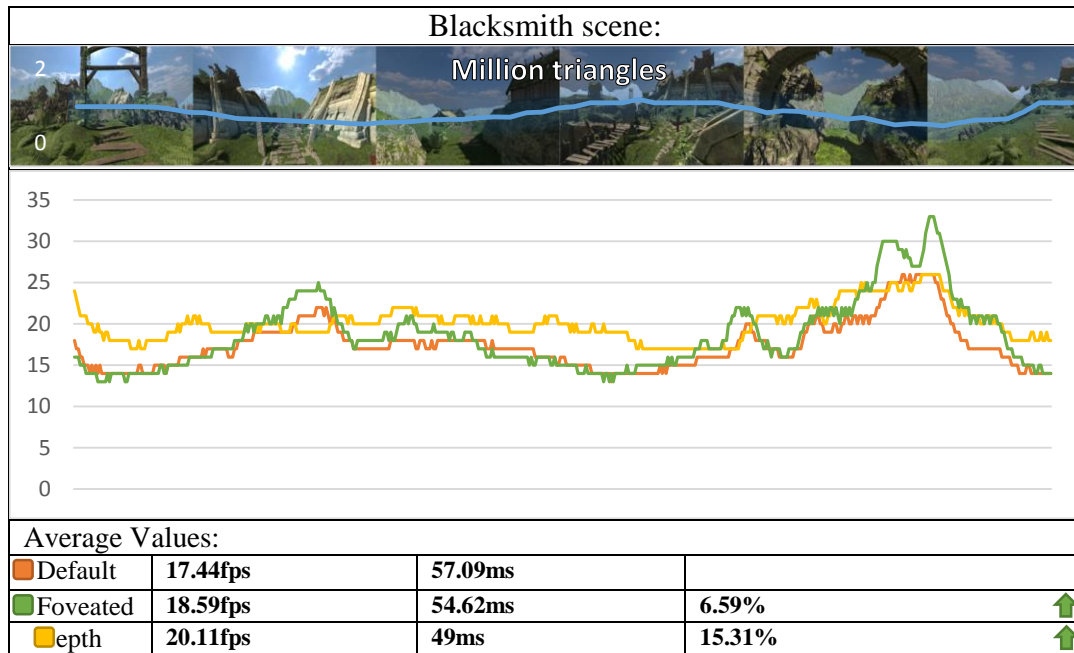




Foveated rendering proves to be a very beneficial performance optimisation for stereoscopy by increasing the overall performance of the “Museum” scene by 198%. The depth optimisation results are similar to the traditional rendering experiment. Depth optimisation should only be considered for stereoscopic rendering after careful consideration of the scene. Scenes with vast open areas could benefit from depth reuse, but only when the camera is rotated towards distant areas. Foveated rendering could be very useful for head-mounted displays and provide a large increase of performance. This experiment was conducted with relatively large foveated region and if such system is implemented for virtual reality the required region will be considerably smaller and the performance benefit will be greater.

4.3 Multiscopic rendering





With the increase of the amount of views the depth reuse technique proves to be a viable solution for performance increase. The “Museum” scene does not benefit from depth reuse because the scene features a small room and there are no objects far into the distance. If depth optimisation is used for commercial project it should only be enabled when there are objects far into the distance and it is beneficial for the performance. The foveated rendering does not provide large performance benefit for the “Blacksmith” and the “Viking Village” scenes with multiscopic rendering.

As gaze tracking in itself is very performance intensive it could reduce the overall performance for multiscopic rendering, especially if it’s not used in conjunction with different level of detail levels.

4.4 Dynamic resolution

The “Blacksmith” scene was used for the dynamic resolution experiments because it closely resembles an actual scene that could be found in a modern video game. The goal is to evaluate the default rendering against the two dynamic resolution modes implemented: Custom and Intel. The most important outcome of these experiments is the frames per second variable. The goal of the dynamic resolution is to keep the frames per second as close to the target frame rate and therefore the two implemented methods will be evaluated on their ability to keep the frame rate close to the set target.

The dynamic resolution settings are as follows:

- Minimum/Maximum multiplier values: 0.6 – 2
- Rate of change: 0.1
- Info GUI is disabled

The frames per second targets are set to be slightly larger numbers than the averages:

- 2D rendering targets 90 frames per second
- Stereoscopic rendering targets 60 frames per second
- Multiscopic rendering targets 20 frames per second

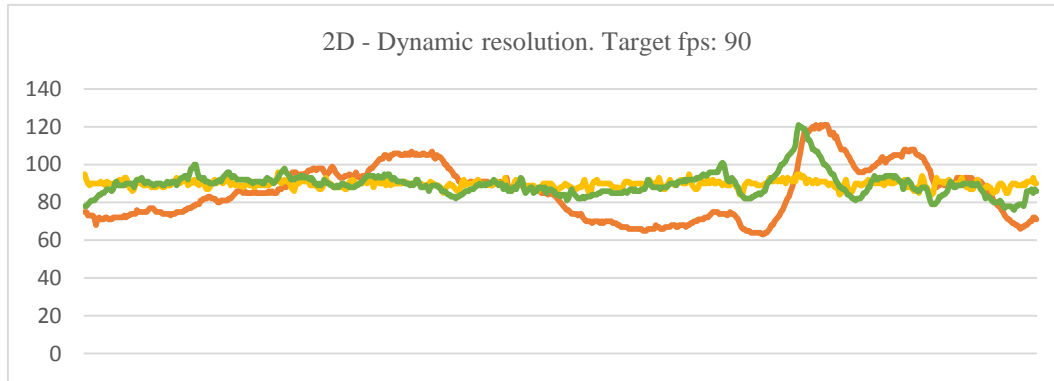


FIGURE 27 2D DYNAMIC RESOLUTION RENDERING

	Default	Custom	Intel
Average fps	85.68	89.65	89.61
Average width	1920	1751.233	1720.41
Average height	1080	985.06	967.73
Average multiplier	1	0.9120	0.8960

For traditional rendering the custom implementation is slightly better than the Intel implementation, but the improvement is negligible. Both custom and Intel implementations provide a stable adaptation to the targeted frame rate. As it can be seen in Figure 27 the Intel implementation has a few spikes while the Custom implementation is very stable and is therefore a better alternative for this particular scene.

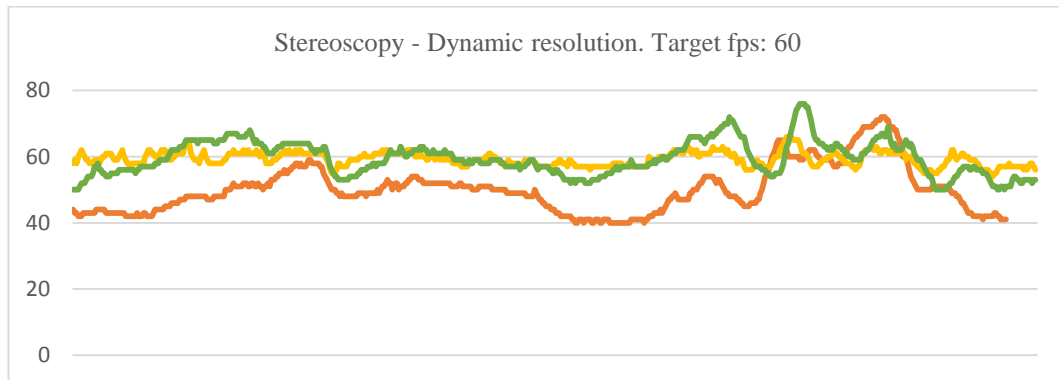


FIGURE 28 STEREOSCOPY - DYNAMIC RESOLUTION

	Default	Custom	Intel
Average fps	50.08	59.30	59.49
Average width	1920	1817.84	1766.88
Average height	1080	1022.53	999.49
Average multiplier	1	0.9467	0.9202

For stereoscopic rendering with dynamic resolution the Intel implementation provides average frame rate that is closer to the targeted frame rate than the custom implementation. Even though the Intel implementation has better average values the custom implementation does not have as many spikes (Figure 28) and it therefore would provide a smoother overall experience. Dynamic resolution should be considered for use in stereoscopic application when the desired frame rate cannot be reached or when there are spikes of geometric quantity as it will make sure that the application runs smooth. Head-mounted displays would greatly benefit from dynamic resolution as even small decrease of frame rate leads to very bad experiences in virtual reality and breaks the user immersion.

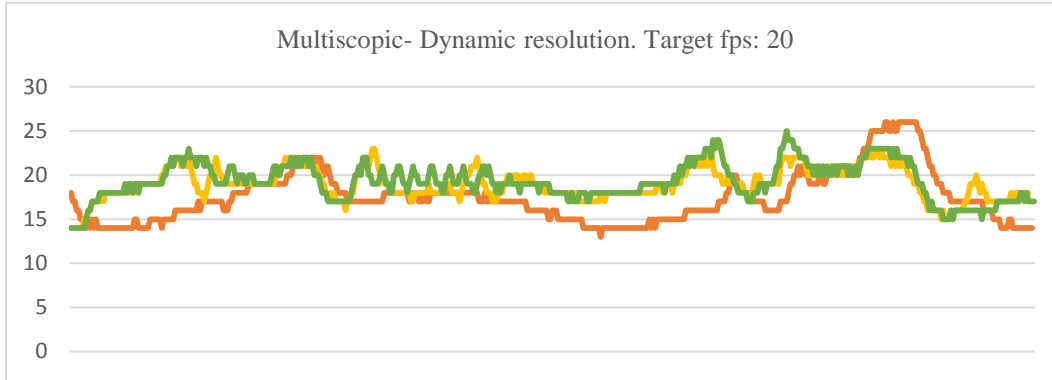


FIGURE 29 MULTISCOPIC- DYNAMIC RESOLUTION

	Default	Custom	Intel
Average fps	17.43	19.03	19.28
Average width	1920	1505.15	1464.76
Average height	1080	846.64	835.17
Average multiplier	1	0.7839	0.7628

For multiscopic rendering the average value of Intel dynamic resolution implementation is closer to the targeted frame rate. As seen in (Figure 29) both of the implementations produce a lot of spikes, but as the Intel implementation provides closer frame rate to the target it is a better solution for multiscopic rendering.

Dynamic resolution is very easy and cheap to implement for multiscopic rendering. It does not provide a lot of performance increase but the super sampling anti-aliasing that could be achieved with it really helps the visual fidelity of the virtual world when viewed on an automultiscopic screen. It should be considered for situations where the current frame rate exceeds the targeted frame rate as the perceived quality would greatly benefit from super sampling.

5. Conclusion and future work

This section outlines the findings of this thesis and provides suggestions for future work.

5.1 Conclusion

Both foveated rendering and dynamic resolution performance optimisations proved to be viable solutions for traditional, stereoscopic and multiscopic rendering.

The dynamic resolution method proposed in this project works just as well as other implementations and can even provide a smoother experience in some occasions.

The depth reuse optimisation technique could potentially be used to increase the performance of stereoscopic and multiscopic rendering but only with special care to enable it when needed. Foveated rendering provides great performance increase and should be utilized for future head-mounted displays. It could be used to substantially increase the visual fidelity of future mobile virtual reality headsets and is important for real-time multiscopic rendering.

All of the performance optimisations discussed in this project can be implemented together and provide great performance benefits for multiscopic and stereoscopic rendering. They could also enable the future use of mobile virtual reality devices for performance intensive tasks.

This project fulfils its aim and objectives and provides additional functionality. It is a good starting point for future developments of the performance optimisation techniques and will be useful for both commercial and academic purposes.

5.2 Future work

Future implementations of foveated rendering could use multiple shaders to combine the render textures. This will enable the use of more than one foveated region for multiscopic rendering as well as proper masking of the region. A circular mask could be specified with additional texture which would make the foveated region harder to locate and differentiate and provide better immersion.

A custom blurring solution could be implemented for the low resolution region that falls outside of the foveated regions. This could help with the blending between the regions.

Depth reuse should only be enabled when it could be beneficial for the performance and therefore a custom system that tracks the visible geometry amount can be implemented. Only when the geometry in the distance exceeds a predetermined amount would the depth optimisation system be enabled.

Different rendering rates could be implemented for the different parts of the depth reuse system. The near render texture could have a refresh rate of 60 frames per second while the far render texture has a refresh rate of 30 frames per second. This would greatly increase the performance by reducing the amount of geometry rendered every frame.

A combination of multiple performance optimisations discussed in this project could be implemented in conjunction to greatly increase frame rate. The dynamic resolution could be used for the far render texture of the depth reuse system. Foveated rendering could be implemented with numerous foveated regions, each one with dynamic resolution and a different target frame rate.

References

- Adobe, 2015. *Previewing/Fast Previews*. [Online]
Available at: https://helpx.adobe.com/after-effects/using/previewing.html#id_66718
[Accessed 02 05 2015].
- Anstis, S. M., 1974. A chart demonstrating variations in acuity with retinal position. *Vision Research*, Issue 14, pp. 589-592.
- Archos VR, 2015. *Archos VR headset makes any phone virtual reality before your very eyes*. [Online]
Available at: <http://www.cnet.com/uk/news/archos-vr-headset-turns-any-phone-virtual-reality-before-your-very-eyes/>
[Accessed 2015].
- Baudisch, P., DeCarlo, D., Duchowski, A. . T. & Geisler, W. S., 2003. Focusing on the essential: considering attention in display design. *Communications of the ACM*, March, 46(3), pp. 60-66 .
- brainhq, n.d. *How Vision Works*. [Online]
Available at: <http://www.brainhq.com/brain-resources/brain-facts-myths/how-vision-works>
[Accessed 07 09 2015].
- Burr, D. C., Morrone, M. C. & Ross, J., 1994. Selective suppression of the magnocellular visual pathway during saccadic eye movements. *Nature*, Volume 371(6497), pp. 511-51.
- Cardboard, 2015. *Android VR und Cardboard: Mehr soll es zur Google I/O geben*. [Online]
Available at: <http://www.smartdroid.de/android-vr-und-cardboard-mehr-soll-es-zur-google-io-geben/>
[Accessed 2015].
- Chen, R. & Kalinli, O., 2011. *Interface using eye tracking contact lenses*. United States, Patent No. US20120281181 A1.
- Clark, J. H., 1976. Hierarchical geometric models for visible surface algorithms. *Association for Computing Machinery*, Volume 19, pp. 547 - 554.
- crsltd, 2015. *BlueGain EOG Biosignal Amplifier*. [Online]
Available at: <http://www.crsLtd.com/tools-for-vision-science/eye-tracking/bluegain-eog-biosignal-amplifier/>
[Accessed 07 08 2015].
- Curico, C. A., Sloan, K. R., Kalina, R. E. & Hendrickson, A. E., 1990. Human Photoreceptor Topography. *THE JOURNAL OF COMPARATIVE NEUROLOGY*, Issue 292, pp. 497-523.
- dualshockers.com, 2013. *www.dualshockers.com*. [Online]
Available at: <http://www.dualshockers.com/2013/10/03/battlefield-4-is-the-game-with->

the-most-scalable-graphics-in-history/
[Accessed 06 09 2015].

Dubuc, B., 2002. *THE RETINA*. [Online]
Available at: http://thebrain.mcgill.ca/flash/i/i_02/i_02_cl/i_02_cl_vis/i_02_cl_vis.html
[Accessed 2015].

Durgin, F. H., Tripathy, S. P. & Levi, D. M., 1995. On the Filling in of the Visual Blind Spot: Some Rules of Thumb. *Perception*, Volume 24, pp. 827-840.

Ebisawa, Y. & Suzu, K., 1994. *Dynamics of saccades occurring during smooth pursuit eye movement*. Baltimore, Maryland, Institute of Electrical and Electronics Engineers, pp. 420 - 421.

engadget.com, M. S. -, 2014. *Call of Duty: Advanced Warfare scales up to 1080p on Xbox One*. [Online]
Available at: <http://www.engadget.com/2014/11/03/call-of-duty-advanced-warfare-scales-up-to-1080p-on-xbox-one/>
[Accessed 02 09 2015].

extremetech.com, G. B. -, 2015. *Witcher 3 uses dynamic resolution scaling on Xbox One to hit 1080p*. [Online]
Available at: <http://www.extremetech.com/gaming/205487-witcher-3-uses-dynamic-resolution-scaling-on-xbox-one-to-hit-1080p>
[Accessed 01 09 2015].

FOVE Inc, 2015. *The World's First Eye Tracking Virtual Reality Headset*. [Online]
Available at: <http://www.getfove.com/>
[Accessed 10 08 2015].

Gear VR, 2015. *Gear VR*. [Online]
Available at: <http://www.lakento.com/products/es/gafas-de-realidad-virtual-samsung-gear-vr-oculus.html>
[Accessed 2015].

Google ATAP, 2015. *A mobile device that can see how we see*. [Online]
Available at: <https://www.google.com/atap/project-tango/>
[Accessed 07 07 2015].

Gunter, B. et al., 2012. *Foveated 3D Graphics*. s.l., ACM SIGGRAPH Asia.

Hardawar, D., 2015. *Samsung is building an 11K mobile display that can mimic 3D*. [Online]
Available at: <http://www.engadget.com/2015/07/10/samsung-11k-display/>
[Accessed 13 09 2015].

Intel, D. B., 2011. *Dynamic Resolution Rendering Article*. [Online]
Available at: <https://software.intel.com/en-us/articles/dynamic-resolution-rendering-article>
[Accessed 17 07 2015].

Microsoft Developer Network, 2013. *Tessellation Overview*. [Online]
Available at: <https://msdn.microsoft.com/en-us/library/ff476340>
[Accessed 12 08 2015].

Microsoft HoloLens, 2015. *Holographic computing is here..* [Online]
Available at: www.microsoft.com/microsoft-hololens
[Accessed 18 08 2015].

NASA , 1964. *SP-3006 Bioastronautics Data Book*, Washington D. C.: s.n.

Oded Sudarsky, C. G., 1999. Dynamic Scene Occlusion Culling. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), pp. 13-29.

Palczewska, G. et al., 2014. *Human infrared vision is triggered by two-photon chromophore isomerization*. Berkeley, California, Proceedings of the National Academy of Sciences of the United States of America.

Schoen, M., 2013. *Spline Controller*. [Online]
Available at: http://wiki.unity3d.com/index.php?title=Spline_Controller
[Accessed 16 06 2015].

SensoMotoric Instruments, 2015. *Eye tracking HMD upgrade package for Oculus Rift DK2*. [Online]
Available at: <http://www.smivision.com/en/gaze-and-eye-tracking-systems/products/eye-tracking-hmd-upgrade.html>
[Accessed 15 06 2015].

SlowRiot, 2013. *When there's no need to render twice*. [Online]
Available at: <https://forums.oculus.com/viewtopic.php?t=4155>
[Accessed 15 09 2015].

Sony , 2007. *PlayStation Eye*. [Online]
Available at:
<http://uk.playstation.com/ps3/accessories/detail/item78698/PlayStation%C2%AEEye/>
[Accessed 06 02 2015].

Starbreeze Studios, 2015. *Starbreeze in collaboration with Tobii to integrate its world-leading eye tracking technology into the unique 210-degree, 5K resolution StarVR HMD*. [Online]
Available at: <http://starbreeze.com/2015/09/starbreeze-in-collaboration-with-tobii-to-integrate-its-world-leading-eye-tracking-technology-into-the-unique-210-degree-5k-resolution-starvr-hmd/>
[Accessed 23 09 2015].

The Eye Tribe, 2015. *The Eye Tribe Tracker*. [Online]
Available at: <https://theeyetribe.com/products/>
[Accessed 01 08 2015].

Thunström, R., 2014. *Passive gaze-contingent techniques relation to system latency*. Karlskrona: Blekinge Institute of Technology.

Tobii Technology, 2011. *Tobii TX300 Eye Tracker for reading studies*. [Online]
Available at: http://www.mynewsdesk.com/us/tobii_technology/images/tobii-tx300-eye-tracker-for-reading-studies-76565
[Accessed 12 09 2015].

Tobii Technology, 2015. *Seeing is doing with Tobii EyeX*. [Online]
Available at: <http://www.tobii.com/en/eye-experience/eyex/>
[Accessed 10 09 2015].

Unity Technologies, 2015. *OffsetVanishingPoint*. [Online]
Available at: <http://wiki.unity3d.com/index.php?title=OffsetVanishingPoint>
[Accessed 12 03 2015].

Unity Technologies, 2015. *The Size of the Frustum at a Given Distance from the Camera*. [Online]
Available at: <http://docs.unity3d.com/Manual/FrustumSizeAtDistance.html>
[Accessed 16 02 2015].

University of Cambridge, 2003. *Principles of three-dimensional vision*. [Online]
Available at: <http://www-g.eng.cam.ac.uk/3d-displays/princip.htm>
[Accessed 02 08 2015].

VRay, 2011. *Vray Quick-Render*. [Online]
Available at: <https://www.youtube.com/watch?v=hGqoVh5rfRk>
[Accessed 19 09 2015].

Yarbus eyetracker , 1960. *Yarbus eyetracker photograph*. [Online]
Available at:
https://upload.wikimedia.org/wikipedia/commons/thumb/a/a8/Yarbus_eye_tracker.jpg/626px-Yarbus_eye_tracker.jpg
[Accessed 10 08 2015].

Yarbus, A. L., 1960. *Eye Movement and Vision*. Moscow (later translated and published in New York): Plenum Press.