FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Armi Korhonen

AUTOMATED PERFORMANCE MEASUREMENT FRAMEWORK FOR SINGLE PLATFORM XR PARTICLE EFFECTS IN UNITY3D

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2024

Korhonen A. (2024) Automated Performance Measurement Framework for Single Platform XR Particle Effects in Unity3D. University of Oulu, Degree Programme in Computer Science and Engineering, 70 p.

ABSTRACT

This thesis investigates the optimization of particle effects in single platform VR and AR, collectively known as XR environments, using Unity3D. Previous work has highlighted the importance of maintaining high performance in XR applications especially to prevent user discomfort. Motivated by the need to find effective ways to optimize particle effects without compromising performance, this research aims to provide actionable insights for developers to enhance user experiences in VR and AR applications.

The constructive method was employed to design and implement a controlled virtual environment and develop an automated testing framework. This framework was used to simulate user interactions systematically and measure performance metrics. Data gathered from these tests were analyzed using polynomial regression to define optimized levels of particle effects for various XR modes.

The results reveal differences between the performance of particle effects in different modes, offering insights into the appropriate use of particle systems in XR environments. The automated testing framework not only proved effective for data gathering and optimization purposes but also provided a cost-effective alternative to the initially intended use of an expensive robot for testing. These findings contribute to a deeper understanding of particle system performance in XR environments and offer a robust methodology for optimizing visual effects, thereby enhancing the overall user experience in VR and AR applications.

Keywords: Extended Reality, Particle Systems, Unity3D, Visual Effects Graph, Automated Testing, Performance Optimization, Virtual Reality, Augmented Reality, Single Platform, VR, AR, XR, MR.

Korhonen A. (2024) XR-partikkeliefektien automaattinen suorituskyvyn mittauskehys yhtenäiselle alustalle Unity3D:llä. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 70 s.

TIIVISTELMÄ

Tämä diplomityö tutkii partikkeliefektien optimointia yksittäisellä alustalla toimivissa XR-ympäristöissä käyttäen Unity3D-pelimoottoria. Aiemmat tutkimukset ovat korostaneet korkean suorituskyvyn ylläpitämisen tärkeyttä XR-sovelluksissa erityisesti käyttäjien epämukavuuden ehkäisemiseksi. Diplomityö pyrkii tarjoamaan kehittäjille konkreettisia keinoja VR- ja AR-sovellusten käyttäjäkokemuksen parantamiseen suorituskykyä vaarantamatta.

Tutkimuksessa hyödynnettiin konstruktiivista menetelmää kontrolloidun virtuaaliympäristön suunnittelemiseksi ja toteuttamiseksi sekä automatisoidun testauskehyksen kehittämiseksi. Tätä kehystä käytettiin systemaattisesti käyttäjävuorovaikutusten simulointiin ja suorituskykymittareiden mittaamiseen. Testeistä kerätyt tiedot analysoitiin polynomiregressiolla määrittelemään eri partikkeliefektien optimoidut tasot sekä VR- että AR-tiloille.

Tulokset osoittavat eroja partikkeliefektien suorituskyvyssä eri tiloissa ja tarjoavat oivalluksia partikkelijärjestelmien asianmukaisesta käytöstä XR-ympäristöissä. Automaattinen testauskehys osoittautui tehokkaaksi paitsi tietojen keräämiseen ja optimointiin myös kustannustehokkaaksi vaihtoehdoksi alun perin suunnitellulle kalliille robottitestaukselle. Nämä löydökset edistävät syvällisempää ymmärrystä partikkelijärjestelmien suorituskyvystä XR-ympäristöissä ja tarjoavat vankan metodologian visuaalisten efektien optimointiin, mikä parantaa kokonaisvaltaisesti käyttäjäkokemusta VR- ja AR-sovelluksissa.

Avainsanat: Laajennettu todellisuus, Partikkelijärjestelmät, Unity3D, Visual Effects Graph, Automaattinen testaus, Suorituskyvyn optimointi, Virtuaalitodellisuus, Lisätty todellisuus, Yhtenäinen alusta, VR, AR, XR, MR.

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
TABLE OF CONTENTS
FOREWORD
LIST OF ABBREVIATIONS AND SYMBOLS
1. INTRODUCTION 8
    1.1. Research Questions and Method 8
    1.2. Author´s Contribution 9
    1.3. Key Concepts 10
    1.4. Structure of This Thesis 10
2. RELATED WORK 12
    2.1. History of Extended Reality 12
    2.2. Importance of Optimization in VR and AR 12
    2.3. Particle Effects in Unity3D 13
    2.4. Hardware Limitations 14
    2.5. Optimization Metrics 16
3. IMPLEMENTATION 18
    3.1. Unity3D Game Engine 18
    3.2. Assets 18
    3.3. Data Analysis Software 19
    3.4. Development and Testing Hardware 20
    3.5. Developing for Both VR and AR 21
    3.6. Particle Effects 22
        3.6.1. Built-In Particle Effect Configuration 22
        3.6.2. VFX Particle Effect Configuration 23
    3.7. Automated Testing 25
        3.7.1. Movement Recording 28
        3.7.2. Running the Test Sequences 28
    3.8. Dynamic Adjustment of Particle Spawn Rate 29
4. EVALUATION 32
    4.1. Data Gathering and Preprocessing 32
        4.1.1. Logcat Data 32
        4.1.2. CSV-Files 33
    4.2. Data Analysis 34
    4.3. Using Predictive Models to Optimize Particle Density 35
5. RESULTS 39
    5.1. Correlation between Different Performance Metrics 39
    5.2. Particle Count in Relation to Fps 44
    5.3. CPU and GPU Utilization 48
    5.4. Stale Frames 53
6. DISCUSSION 55
    6.1. Summary of Research 55
    6.2. Analysis of Research Questions 55

6.3. Key Findings and Recommendations 57
6.4. Limitations 57
6.5. Future Work 58
6.5.1. Enhancing the Predictive Models 58
6.5.2. Data Analysis within the Testing Application 59
7. CONCLUSION 61
8. REFERENCES 62
9. APPENDICES 66

FOREWORD

I would like to express my deepest gratitude to my supervisors, Dr. Paula Alavesa and Dr. Satu Tamminen for their excellent commentary and help during the creation and editing process of this work. Special thanks to Paula, in particular, who has been an unwavering source of support, assistance, and inspiration throughout the planning and execution of this work. I am also grateful to my mother, Terttu, as well as to my family and friends for their encouragement and support.

Additionally, I would like to extend my thanks to the University of Oulu and the UBICOMP research and teaching unit for providing me with the initial contact and education in the development of virtual and augmented reality.

Haluan esittää syvimmät kiitokseni ohjaajilleni Paula Alavesalle ja Satu Tammiselle erinomaisista kommenteistaan ja avustaan tämän työn luomis- ja editoimisprosessin aikana. Erityisesti Paula on ollut väsymätön tukija, auttaja ja innostaja työn suunnittelussa ja toteutuksessa. Olen myös kiitollinen äidilleni Tertulle, sekä perheelleni ja ystävilleni heidän kannustuksestaan ja tuestaan.

Lisäksi haluan kiittää Oulun Yliopistoa ja UBICOMP tutkimus- ja opetusyksikköä ensikosketuksesta ja koulutuksesta virtuaalisen ja lisätyn todellisuuden kehittämiseen.

Oulu, June 14th, 2024

Armi Korhonen

LIST OF ABBREVIATIONS AND SYMBOLS

3D	three dimensional
ADB	Android Debug Bridge
API	application programming interface
AR	augmented reality
CPU	central processing unit
FOV	field of view
FPS	frames per second
GPU	graphics processing unit
GUI	graphical user interface
HMD	head-mounted display
LOD	level of detail
MR	mixed reality
PCA	Principal Component Analysis
PCVR	PC-based Virtual Reality devices
RQ	research question
SDK	software development kit
VFX	visual effects
VR	virtual reality
XR	extended reality

1. INTRODUCTION

The evolution of mixed reality (MR) applications presents unique challenges and opportunities, especially in the realm of gaming. My interest in VR testing and optimization began when I founded a small company aimed at developing an XR puzzle game. In this game, users complete image puzzles that, upon completion, trigger environment-enhancing particle effects related to the finished image. This initiative highlighted the critical need for optimizing particle effects to ensure a smooth, immersive experience in both AR and VR modes, particularly when the games are played on the same device, i.e., platform, and a shift between AR and VR is possible.

Considering the performance limitations of mobile devices like the Meta Quest 2 [1] and Meta Quest 3 [2], which lack the processing power of standalone computers, optimizing these effects became an important aspect of development. The scarcity of comprehensive material on effective particle effect creation in mixed reality settings led me to choose this as the focus for my thesis. While the thesis does not center on the development of the game itself, this process provided valuable insights into particle effect optimization strategies for my upcoming game.

Initially, this thesis did not include the development of a testing framework. However, as the study progressed, it became clear that technical testing of the particle effects and their relationship with performance metrics was essential. Lacking access to robot testing and to avoid relying solely on user testing, which could introduce variability and less control over the test environment, an automated testing framework was designed and developed. This framework was designed to record and recreate real human interactions within the XR environment, offering an easy and streamlined way to run tests and gather data on performance metrics.

All the relevant code and data used in this thesis can be found in the GitHub repository for this thesis [3]. The repository includes the Unity3D script files for the automated testing framework, both raw and processed testing data, and the Jupyter Notebooks used in the data analysis. The code in the repository is licensed under the MIT License, ensuring it is freely available for modification and distribution. All images included in this thesis are licensed under the Creative Commons Attribution 4.0 International License, allowing for sharing and adaptation with proper attribution.

1.1. Research Questions and Method

This thesis explores the creation and performance implications of particle systems in Unity3D [4], focusing on two distinct methods: the built-in particle system [5] and the VFX graph [6]. Constructive method was used to build a research prototype application to be used as a construct to answer the research questions (RQs). [7]

This research was guided by the following RQs:

RQ1 How do different particle system creation methods in Unity3D compare in terms of performance and efficiency in XR environments?

RQ1.1 What are the performance differences between the built-in particle system and the Visual Effect Graph when used in VR and AR environments?

RQ2 How to implement testing method to consistently replicate user interaction and measure the impact of visual effect adjustments on performance metrics in AR and VR?

RQ2.1 What metrics most effectively measure and compare the performance impact of different particle system methods in Unity3D under AR and VR conditions?

RQ3 What recommendations can be made for developers choosing between particle system methods for their AR or VR projects?

One of the primary objectives of this research emerged with the decision to use technical testing instead of user testing; the development of an automated testing framework that allows for the consistent testing of performance metrics within an XR application. Moreover, this thesis aims to provide insights into the types of particle effects suitable for XR applications and their limitations.

Methodologically, the study involved creating a research prototype (a 3D scene in Unity3D), testing different particle effects, and simulating natural human movements during testing. Performance data was collected, processed, and analyzed to determine the optimal particle density that would not compromise performance.

1.2. Author´s Contribution

The work presented in this thesis was conducted solely by the author and encompasses the design, implementation, and analysis of various methodologies for optimizing particle effects in XR environments using Unity3D. The author's contributions are detailed as follows:

- The design and implementation of the virtual environments used in this study, along with the two different evaluated particle effects: the built-in particle system and the VFX graph. This involved conceptualizing the scene layouts, designing and configuring the particle effects, and ensuring that they functioned correctly across different XR platforms.
- An automated testing framework for Unity3D was developed from scratch, which was crucial for simulating user interactions and gathering consistent performance data. This framework allowed for the replication of human movements within the virtual environment, which was instrumental in assessing the impact of various particle densities on system performance.
- The collection, preprocessing, and analysis of all data obtained from the automated tests. This included setting up the data capture protocols, processing the raw data to extract meaningful metrics, and performing the statistical analyses that form the basis of the conclusions drawn in this thesis.

Throughout the project, numerous challenges were encountered and overcome, particularly in relation to the integration of different technologies and the precise measurement of performance impacts under varying conditions. Each phase of the research, from the initial design and implementation of the testing environment to the detailed analysis of the data, was conducted independently by the author, with a continuous focus on improving the reliability and validity of the results.

1.3. Key Concepts

In order to provide clarity and enhance the readability of this thesis, this chapter offers brief explanations of essential terms and technologies related to the study of particle system optimization in XR environments. Given the technical nature of this research, it is important to ensure that all readers, regardless of their familiarity with the specific subject matter, can fully understand the content and significance of the findings presented.

Table 1 describes key concepts such as Unity3D, particle effects, the Visual Effect Graph, frames per second (fps), extended reality (XR), and polynomial regression. By outlining these terms, this chapter aims to provide a foundational understanding that will support the reader's comprehension of the methodologies and results discussed in subsequent chapters.

1.4. Structure of This Thesis

This thesis is organized into several chapters that cover the research process. It begins with an Introduction that outlines the research questions and the structure of the thesis. This is followed by the Related Work chapter, which reviews existing literature and contextualizes the study within the field of extended reality. The Implementation chapter details the technical setup used for the experiments, including descriptions of the Unity3D game engine, assets, and development tools for both VR and AR environments. The Evaluation chapter describes the methodologies employed for testing the particle effects, and the Results chapter presents the findings from these evaluations. The Discussion chapter interprets the outcomes, addresses limitations, and incorporates a section on future work. The thesis concludes with a Conclusion chapter that summarizes the findings.

Table 1. Key concepts related to this thesis

Key Concept	**Description**
Unity3D	A cross-platform game engine used for developing games and applications. It provides a comprehensive suite of tools for designing, developing, and deploying interactive content, and it is widely used in the gaming industry as well as in various other fields such as simulations, training, and architectural visualization. [4]
Particle Effects	Graphical representations used in computer graphics to simulate fuzzy phenomena, such as fire, smoke, rain, explosions, and magical effects. These effects are generated by a large number of small images or particles, which are controlled by a particle system to create dynamic and complex visual effects. [8]
Visual Effect Graph (VFX Graph)	A tool within Unity3D that allows developers to create sophisticated particle effects using a node-based visual scripting interface. It leverages the power of the GPU to render complex effects efficiently, providing greater control and flexibility compared to the built-in particle system. [6]
Frames Per Second (fps)	A measure of how many frames (individual images) are displayed by a graphical system each second. Higher fps values indicate smoother and more responsive visual output, which is particularly important in XR applications to ensure a seamless user experience. [9]
Extended Reality (XR)	An umbrella term that encompasses virtual reality (VR), augmented reality (AR), and mixed reality (MR). XR technologies blend the physical and digital worlds, providing immersive experiences that can range from fully virtual environments to augmented views of the real world. [10]
Immersive VR Mode	In the context of this thesis, a mode where the user is completely surrounded by a digital 3D environment with no visual input from the real world.
Passthrough AR Mode	A mode in XR applications that allows users to see the real world with digital elements overlaid on top of it. The device's cameras are used to provide a live feed of the user's surroundings, blending augmented reality with the real environment. [11]
Polynomial Regression	A form of regression analysis in which the relationship between the independent variable and the dependent variable is modeled as an nth-degree polynomial. In this thesis, polynomial regression was used to optimize particle effects in XR environments, with fps as the dependent variable and the number of particles as the independent variable. [12]

2. RELATED WORK

This chapter explores the existing body of knowledge relevant to this thesis, examining the evolution and significance of XR technologies. It reviews the historical development of VR and AR, emphasizing the advancements that have shaped the current state of XR. The importance of optimization in VR and AR applications is discussed, highlighting key studies and industry recommendations. Further, the chapter examines the particle effects available in Unity3D, comparing the built-in particle system and the Visual Effect Graph. Hardware limitations and optimization metrics pertinent to XR development are also analyzed to provide a comprehensive understanding of the challenges and considerations in optimizing XR applications.

2.1. History of Extended Reality

Early VR systems, emerging from flight simulators and university laboratories, relied on primitive computer graphics and tracking technology. The advent of microprocessors and advances in display technologies in the late 20th century allowed for the creation of more sophisticated VR systems. With the smartphone revolution, key components such as gyroscopes, accelerometers, and high-resolution screens became widely available, inexpensive and miniaturized, facilitating the production of consumer-grade VR headsets. These technological advancements have propelled VR beyond simple visual displays, enabling fully immersive environments that engage multiple senses and track user interactions with high precision. [13]

In the evolution of input devices for VR, significant milestones include the development of lightweight gloves for hand movement monitoring, such as the Sayre Glove in 1976 and the Data Glove in 1987, which utilized fiber-optic wires and magnetic position tracking. Concurrently, in 1983, John Hilton pioneered 6DOF input devices with the creation of the Spaceball, enabling users to manipulate 3D objects along multiple axes. [14]

2.2. Importance of Optimization in VR and AR

A study by Wang, et al. [15] examined how different frame rates (60, 90, 120, and 180 fps) in VR HMDs affect user experience, performance, and simulator sickness. It found that 120 fps is a crucial threshold; beyond this, simulator sickness symptoms decrease without negatively impacting the user experience. Higher frame rates (120 and 180 fps) enhanced user performance compared to lower rates.

The study also found that lower frame rates in virtual reality, particularly at 60 fps, were associated with increased symptoms of simulator sickness. This included experiences of discomfort, nausea, and disorientation among participants. The study demonstrated that as the frame rate decreased, the likelihood and intensity of simulator sickness symptoms increased, underscoring the importance of higher frame rates for reducing these negative effects in VR environments. [15]

Based on the recommendations provided by Meta for the Meta Quest platform [16], developers should adhere to specific performance targets to ensure optimal user

experiences. For fps requirements, interactive applications must target a minimum of 72 fps, while media applications are allowed to target 60 fps. To manage draw calls effectively, developers should consider factors such as pipeline state switching, mesh sharing, and shader optimization. It is also recommended to create a non-VR version of the camera rig to facilitate scene spot-checking and profiling with third-party tools. Additionally, developers should pay attention to triangle counts, ensuring efficient memory access patterns and optimizing vertex attribute usage. [16]

In their study 'How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects', Nusrat et al. [17] emphasized that developers frequently utilize profiling and benchmarking tools throughout the development lifecycle to ensure optimal performance. Profiling tools are used to monitor application behavior in real-time, providing insights into resource usage, frame rate consistency, and potential bottlenecks. Benchmarking, on the other hand, is employed to measure the performance of the application against predefined standards or previous iterations, enabling developers to quantify the impact of their optimizations. [17]

Developers adopt several strategies for optimizing both code and assets within Unity-based VR applications. Code optimization often involves refining algorithms, reducing computational complexity, and minimizing API calls that are known to cause performance overhead. Asset optimization includes techniques such as implementing level of detail (LOD) systems, which adjust the complexity of 3D models based on their distance from the user, and asset culling, which prevents the rendering of objects not currently viewed by the user, thereby saving processing resources. [17]

2.3. Particle Effects in Unity3D

Unity3D offers two different ways to create particle systems [8]: the built-in particle system (also sometimes called the Shuriken particle system) and the Visual Effect Graph. Whilst both systems can be used to create particle effects, there are some differences, which might dictate which system is appropriate for each use case.

The built-in particle system is highly versatile and integrated deeply with Unity's native components. This system allows full scriptable control over particles, offering a detailed API that can access and modify particles at runtime. This makes it suitable for effects where you need direct control over particle behavior, and it supports interactions with Unity's physics system, allowing particles to interact with other physics-enabled objects in the scene. [5]

The Visual Effect Graph is designed to leverage the power of modern GPUs, creating elaborate visual effects by simulating large quantities of particles on the GPU. This makes it particularly useful for high-density effects like smoke, fire, or complex simulations of environmental phenomena. However, because it runs on the GPU and its simulation isn't part of the Unity Physics system, its particles do not inherently interact with Unity's physics system. [6]

Table 2. Comparison of Unity3D Particle Systems

Feature	**Built-in Particle System**	**Visual Effect Graph**
Render Pipeline Compatibility	All including Built-in	Only URP and HDRP
Scale of Particles	Supports thousands	Supports millions
Physics Interaction	Integrated with Unity's Physics Engine	Custom collision via scripting; no native physics interactions
Scripting and Control	Extensive C# scripting and API support	Node-based scripting with limited C# support
Frame Buffers Utilization	Does not use frame buffers	Uses frame buffers, especially in HDRP for advanced effects
Performance	Optimal for lower-end devices	High performance on modern hardware, leveraging GPU
Customization	Modular system via Inspector	Highly customizable through node-based design
Real-Time Modification	Possible via scripts	Real-time modifications through exposed parameters
Visual Fidelity	Good for general use	Excellent for high-density, complex simulations

2.4. Hardware Limitations

Figure 1 shows refresh rate and display resolution data from 121 different headsets released from 1989 to 2023. The headsets of interest in this thesis are marked in blue (Meta Quest 2) and red (Meta Quest 3). The data for the graph was gathered from the VRcompare website [18]. Since the resolution of the lenses vary so widely, total pixels of per-eye lenses was calculated by multiplying the vertical and horizontal pixels. Slight spreading between resolution total pixels and the refresh rate might suggest a trade-off between these two features.

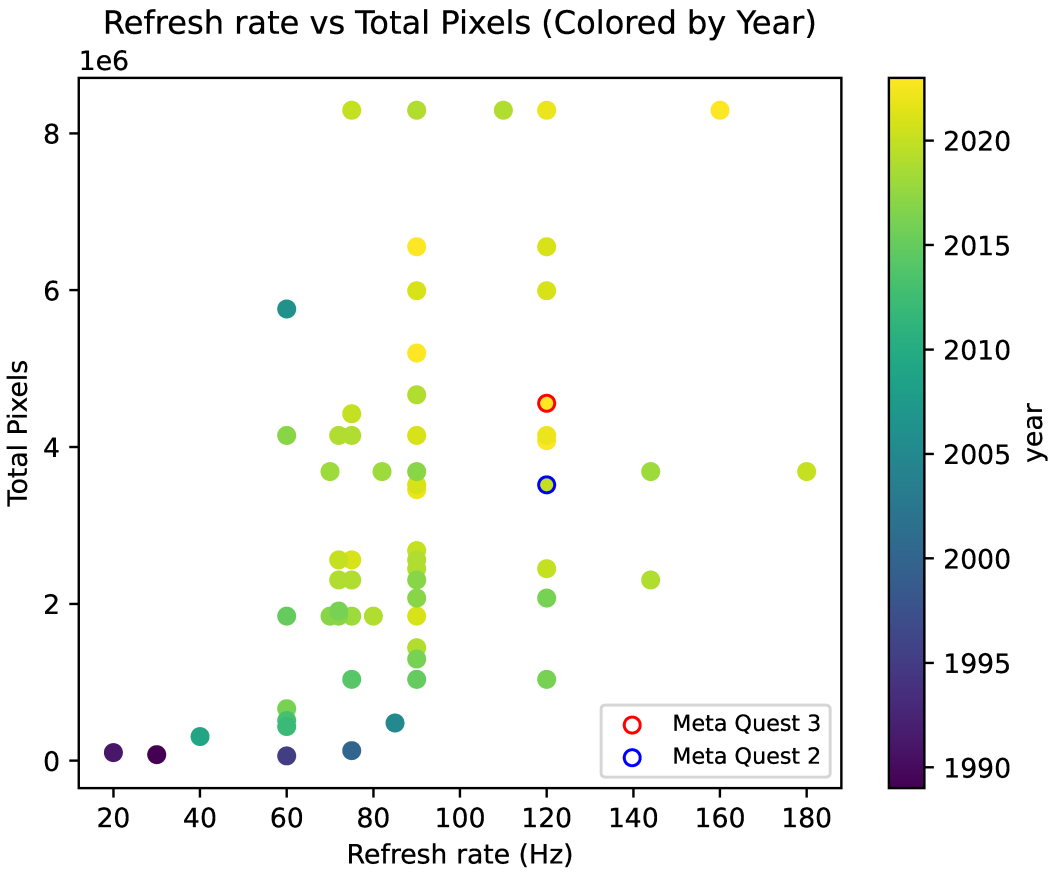

Figure 1. Refresh rate vs total pixels (millions), colored by year

Table 3 provides a detailed comparison of the hardware specifications for two of the most popular VR headsets on the market, the Meta Quest 3 and the Meta Quest 2, originally named the Oculus Quest 2. Despite both devices being intended for testing, due to certain issues, only the Quest 3 was used. Released three years apart, these headsets feature advancements in technology with the Meta Quest 3 showcasing superior specifications in several key areas. Both models share a 120 Hz refresh rate, but the Meta Quest 3 has higher resolution per eye, a wider field of view, and uses advanced pancake lenses compared to the Fresnel lenses of the Meta Quest 2. Additionally, the Meta Quest 3 is equipped with the newer Qualcomm Snapdragon XR2 Gen 2 chipset, more powerful GPU (Adreno 740), and greater memory capacity (8 GB), providing enhancements in processing power and graphical capabilities. The Quest 3 also features color passthrough cameras, which represent a significant upgrade over the grayscale passthrough cameras of the Meta Quest 2. These specifications underline the hardware limitations and capabilities of each headset, with the Meta Quest 3 offering a more advanced and immersive VR experience. [18]

Table 3. Hardware specifications of the Meta Quest 3 and the Meta Quest 2

	Meta Quest 3	**Meta Quest 2**
Release date	October 10, 2023	October 13, 2020
Optics	Pancake lenses	Fresnel lenses
Passthrough	Dual 18 PPD color passthrough cameras	Grayscale via tracking cameras
Display type	2 x LCD binocular	Single Fast switch LCD binocular
Resolution	2064x2208 per-eye	1832x1920 per-eye
Refresh rate	120 Hz	120 Hz
Visible FoV horizontal	110°	97°
Visible FoV vertical	96°	93°
Chipset	Qualcomm Snapdragon XR2 Gen 2	Qualcomm Snapdragon XR2
CPU	Octa-core Kryo(1x3.19 GHz, 4x2.8 GHz, 3x2.0 GHz)	Octa-core Kryo 585 (1x2.84 GHz, 3x2.42 GHz, 4x1.8 GHz)
GPU	Adreno 740	Adreno 650
Memory	8 GB	6 GB

2.5. Optimization Metrics

The hardware specifications of the XR devices used in this study were outlined in the previous sections to provide the reader with an understanding of their limitations. This context underscores the necessity for optimizing methods when building virtual environments for these devices, particularly for standalone devices such as the Meta Quest 2 and Meta Quest 3. Given the hardware constraints and the critical need to prevent user discomfort, it is essential to be aware of these limitations and the performance metrics that can be monitored and optimized. The table 4 summarizes some key performance metrics for optimizing virtual reality applications, each with a detailed description and the preferable values for achieving optimal performance. Each metric can provide insights into different aspects of application performance, from rendering speed to resource usage, helping developers fine-tune their applications for the best user experience in VR environments. [19]

Table 4. Optimization Metrics for VR Applications

Metric Name	**Description**	**Preferable Value**
Average Frames Per Second	Represents the average rate at which consecutive images (frames) are displayed in your app. This is directly indicative of the smoothness of the visual output. Matching or exceeding the display's refresh rate is generally desirable for a smooth user experience.	Match or exceed the display refresh rate
Stale Frames (per second)	Stale frames occur when the app-rendered frame is not ready for display and an older frame must be used instead. This metric is crucial for evaluating the user experience in VR, as it impacts how current the displayed frames are in relation to user actions. It is not a direct inverse of fps because an app can run at a high fps but still have many stale frames due to parallel CPU and GPU operations. Extra Latency Mode can be used to mitigate the impact of stale frames by allowing the compositor extra time to wait for new frames.	Lower numbers are better.
App GPU Time	Measures the time taken by the GPU to render a single frame. This metric helps in determining whether your application is GPU bound. If the GPU time is more than the time it takes for a frame at a given frame rate, the app is GPU bound, otherwise, it might be CPU bound. Monitoring this over time can indicate how changes (like shaders, textures, etc.) affect GPU load.	Less than one frame length
CPU and GPU Utilization	Indicates how much of the CPU and GPU resources your app is using. GPU Utilization is particularly important, reflecting the total work performed by the GPU. High GPU utilization can indicate that you are GPU bound. CPU Utilization shows the activity of the least performing core and might not always accurately represent thread distribution across cores.	Below 90% to avoid bottlenecks
CPU and GPU Level	Shows the current performance levels of CPU and GPU. Higher numbers generally indicate that the app is requiring more from the hardware, potentially due to not meeting frame rates or other performance benchmarks. This metric provides a good indication of whether you need to optimize more on the CPU or GPU.	Optimal levels for your app

3. IMPLEMENTATION

The contribution of this thesis is twofold: 1) the research prototype used as a construct and 2) the method used for performance testing. This chapter presents the first contribution, detailing the development of the research prototype, including the design and implementation of the virtual environments and particle systems in Unity3D. It also describes the implementation of the automated testing framework, which was essential for gathering performance data through simulated user interactions. This framework's setup and functionality are discussed to provide a foundation for the subsequent performance evaluation presented in Chapter 4.

3.1. Unity3D Game Engine

Unity3D was chosen [4] as the game engine for this work as it stands out for its compatibility with different target platforms, ease of use and extended support for both VR and AR. Extensive previous experience with Unity3D was also a key factor in why this particular game engine was chosen. The familiarity allowed for a smoother development process, as the engine's features could be utilized without the need for extensive learning. It helped in focusing more on the development rather than learning new software.

For the implementation part of this thesis, Unity version 2022.3.14f1 was used. Although newer versions were available, they posed dependency issues with several critical plug-ins. Consequently, an older, more stable version was chosen to ensure a smooth development process. Unity's Mixed Reality Template [20] was employed as the foundational framework for the project. This template significantly expedited the initial setup by configuring the project environment and pre-installing essential packages. It provided a convenient starting point that enabled the use of augmented reality passthrough very easily. The XR Interaction Toolkit [21] provided by Unity was used to simplify the development of mixed reality applications. It ensures compatibility with OpenXR, which is beneficial for future adaptations to other headsets. To facilitate deployment and testing with the Meta Quest 3, OpenXR with Meta Quest feature group was integrated into the project's XR plugin management.

3.2. Assets

Midjourney [22] was used for creating the image which is used as a placeholder for the puzzle image. Adobe's Substance 3D Assets [23] were used for the leaf particle effect. Low Poly Ultimate Pack by polyperfect [24] as well as Nebula Skyboxes [25] by Road Turtle Games from the Unity Asset store [26], was used for the immersive VR scene, shown in Figure 2.

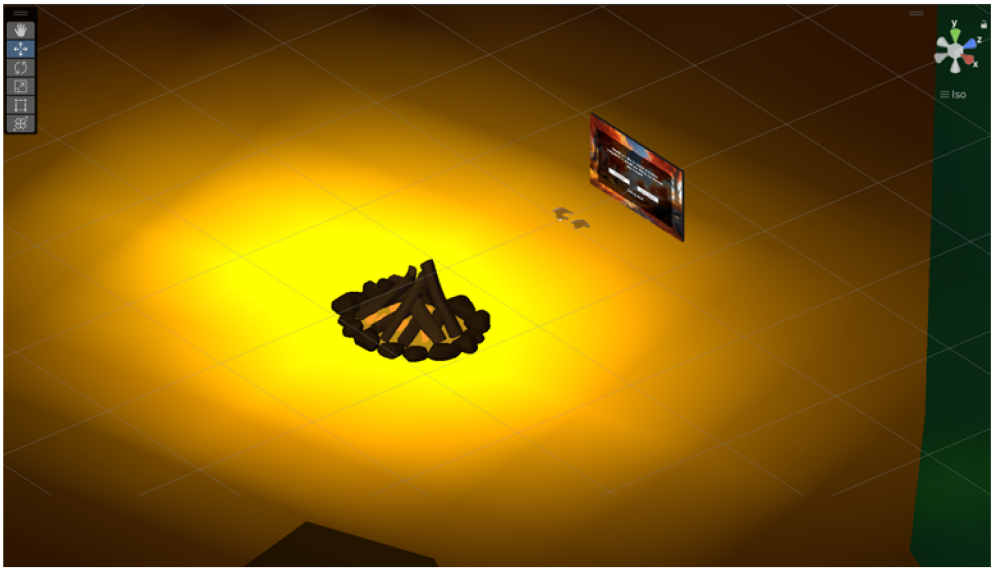

Figure 2. VR scene design in Unity3D

3.3. Data Analysis Software

Jupyter Notebooks [27] were used for the data analysis, as this was found to be the most flexible and easiest way to iteratively process and visualize the data. Copies of the existing notebooks were made and reused as new data were gathered from the automated tests. Table 5 lists all the tools and libraries used in the data analysis part of this thesis.

Table 5. Overview of Libraries and Tools Used in Thesis

Library/Tool	**Purpose**	**Use Case in Thesis**
Jupyter Notebook [27]	Interactive computing environment	Platform for executing Python code, data analysis, and visualization.
pandas [28]	Data manipulation and analysis	Data reading, cleaning, and transformation.
numpy [29]	Numerical computing	Support for large, multi-dimensional arrays and matrices, mathematical functions.
scikit-learn [30]	Machine learning	Polynomial regression modeling and data pipeline construction.
scipy.optimize [31]	Optimization and root finding	Used fsolve for solving equations to find optimal particle counts.
matplotlib [32]	Data visualization	Creating all the plots used in the thesis.
seaborn [33]	Advanced visualization	Creating correlation matrices.
glob [34]	File pattern matching	Handling datasets across multiple files.
os [35]	Miscellaneous operating system interfaces	File and directory manipulation.
re [36]	Regular expressions	Data cleaning and text manipulation.
itertools.groupby [37]	Iterative operations	Grouping data entries for detailed analysis.

3.4. Development and Testing Hardware

Standalone VR headsets, including the Meta Quest 2 and Meta Quest 3, incorporate all necessary computing hardware within the headset itself. This architecture offers significant benefits in terms of portability and ease of use, eliminating the need for external cables and heavy computer hardware. Users can engage with VR content anywhere, bringing immersive experiences into a variety of new settings beyond the confines of a room equipped with a high-performance PC. However, the convenience

of standalone VR systems comes with inherent limitations, primarily related to processing power. The mobile processors found in devices like the Meta Quest 2 and Meta Quest 3, while increasingly powerful, do not match the capabilities of the high-end CPUs and GPUs typical of PCs. This disparity poses considerable challenges for developers, particularly in the realms of complex VR and AR applications, where the demand for real-time graphics processing and data computation is intensive.

Meta Quest 3 [2] was selected as the primary testing platform due to its affordability, mobility, and advanced mixed reality capabilities. Some issues were encountered with the older Oculus Quest 2 device [1], when trying to get it to work in developer mode, which unfortunately meant not being able to run the tests on that device. The choice of Meta Quest 3 as the primary platform for testing was not arbitrary but rather strategic, given its status as standalone VR headset. Unlike PC-based VR systems (PCVR), which rely on the processing power of connected computers, standalone VR devices such as the Meta Quest 3 operate independently with built-in processors and graphics capabilities, making optimization even more important for these platforms. The Meta Quest 3's support for color passthrough technology also significantly enhances the mixed reality experience, offering a more immersive and realistic interaction between digital and physical worlds.

Originally, Optofidelity's Buddy [38] was intended to be used for testing. However, due to unforeseen circumstances, this was not possible. This challenge, however, presented an opportunity to innovate a new type of testing framework. An automated testing framework was designed and developed to allow for the recording and programmatically replaying human movements. This framework not only compensated for the inability to use Optofidelity's Buddy but also enhanced the scope of this thesis by adding a unique tool to the mixed reality development process.

3.5. Developing for Both VR and AR

In the thesis, a strategic design decision was made to ensure compatibility with both VR and AR. This choice was influenced by prevailing trends and future projections within the headset technology market, alongside considerations of user experience. As the mixed reality headset market increasingly embraces AR capabilities, it has become evident that leading devices like the Meta Quest 3 and the Apple Vision Pro [39] are focusing heavily on enhancing the quality of AR experiences. By developing an application compatible with both AR and VR, the work in this thesis was aligned with these technological advancements.

User experience, particularly comfort, was another significant consideration in the decision to develop for both VR and AR. AR is generally known to induce less nausea compared to VR, making it a more comfortable option for many users. By integrating an AR mode, applications can be made more accessible to a broader audience, including those who might experience discomfort or motion sickness in a fully immersive VR environment. In developing the virtual environment for the mixed reality experience, significant inspiration was drawn from Apple's best practices for VisionOS [40]. These guidelines, which prioritize user comfort and engagement, were instrumental in shaping a virtual space that is immersive yet non-intrusive.

In the VR mode, a virtual background was created to cater to users who prefer full immersion within a digital environment. This mode is intended for those who seek an all-encompassing virtual experience. The AR mode was designed for users who favor a less immersive experience or wish to remain connected to their real-world surroundings. This mode overlays digital elements onto the real world, enabling interactions with virtual content while maintaining a presence in the physical environment.

3.6. Particle Effects

In Unity3D, developers have the option to use two distinct particle effect systems: the built-in particle system and the Visual Effect Graph. Both systems offer unique capabilities and are essential for creating dynamic visual effects in virtual environments. To ensure a fair and effective comparison of these systems under testing conditions, it was important that the particle effects implemented in both systems correspond as closely as possible in terms of their visual and functional characteristics. There were, however, some issues with using the same color and texture with the two different particle systems. This discrepancy is covered in Chapter 6.

Given that the built-in particle system is the older of the two and may not support all the features available in the newer Visual Effect Graph, a decision was made to design the particle effects initially using the built-in system. This approach allowed for the establishment of a baseline in terms of particle behavior and performance metrics. Subsequently, the same particle effect was recreated in the Visual Effect Graph, adhering as closely as possible to the original specifications. This method ensured that any observed differences in performance between the two systems could be attributed to the systems themselves rather than variations in the particle effects.

3.6.1. Built-In Particle Effect Configuration

The particle system was designed with a fixed duration of 5 seconds, looping continuously to simulate ongoing particle generation. Each particle was initialized with a lifetime of 5 seconds, ensuring all particles had uniform longevity and would disappear before moving through the ground in the immersive environment. In order to get test data of how the amount of particles affected the performance, particles were emitted at an increasing rate over time. The system was capped at 1000 particles at any given time. Particles were emitted with an initial velocity of -0.2 units. In Unity3D, one unit equals one meter, meaning the particles were moving downwards at a rate of 20 cm per second relative to the emitter's position.

The particles were emitted from a box-shaped volume. This specific shape allowed for a controlled yet diverse distribution of particles within defined spatial boundaries. The randomness in both the direction and position of the particles (each set to 0.2) enhanced the naturalistic dispersion within the given volume. Each particle was programmed to rotate independently about all three axes (X, Y, and Z) at a consistent rate of 45 degrees, contributing to a dynamic and visually engaging particle motion. The particles were rendered using a mesh, specifically a leaf mesh shown in Figure

3, which allowed for realistic and detailed particle representation suitable for visual simulations involving natural phenomena such as leaves.

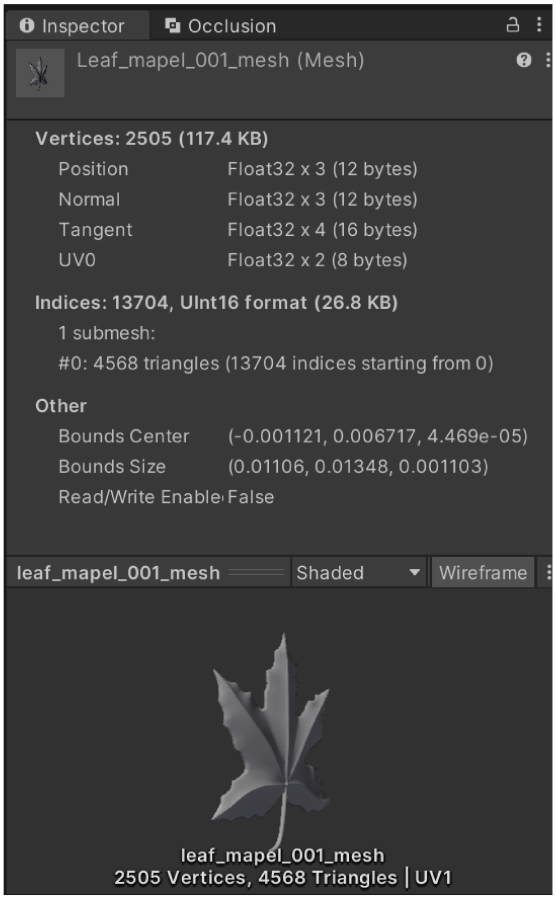

Figure 3. 3D mesh of a leaf used in the particle effects

3.6.2. VFX Particle Effect Configuration

The particle effect created with the visual effects graph was configured as shown in Figure 4.

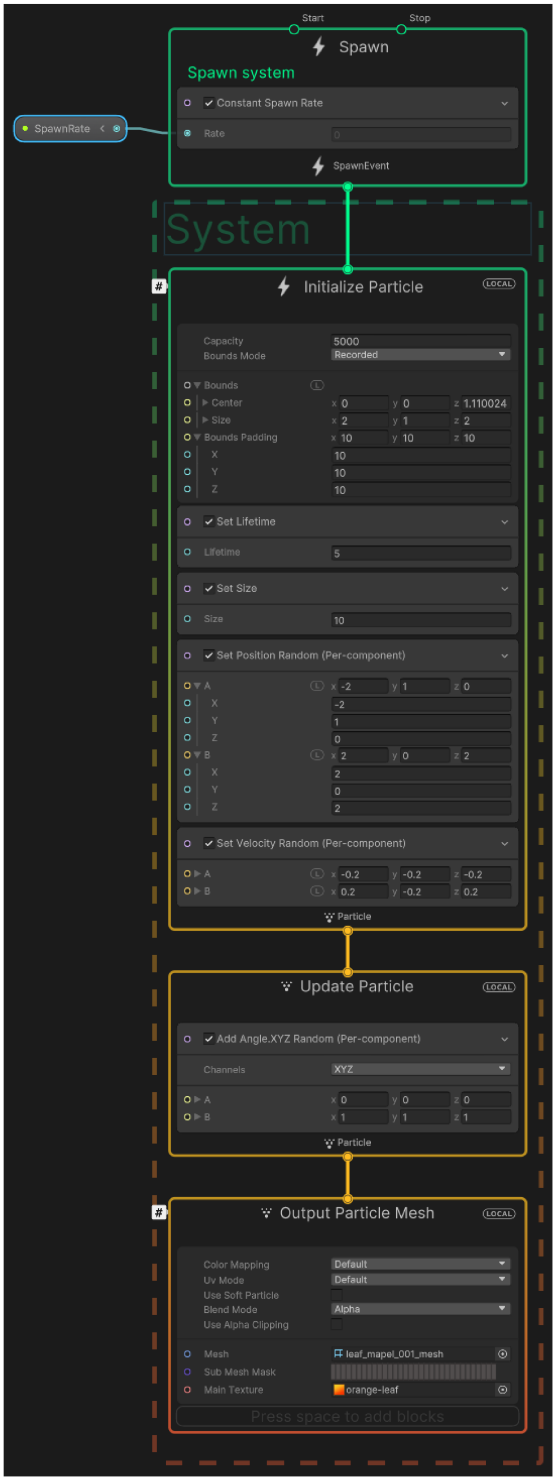

Figure 4. Screenshot of the visual effects graph used for creating the particle system

As an initial test for how much the particle effect would affect the frame rate, an additional toggle button was added to the user interface, where the user could turn on the particle effect during runtime. Performance metrics such as fps was accessed from the profiler [41] at runtime and shown in the user interface, as depicted in Figure 5.

Figure 5. Screenshot of the first versions of the particle effect in Meta Quest 3 in augmented reality mode

When in immersive virtual reality mode, but without the particle effects turned on, the frame rate in the initial tests was at a constant 90 fps, with very minor deviance. However, when the particle effect was turned on, the frames per second started to drop consistently down to 45 fps as more and more leaves were drawn on the screen. When the user moved their head, the frames per second dropped as low as 30 fps. These initial tests proved that the optimization of the particle effects was indeed an important task, as without it, the user experience would've been severely crippled.

3.7. Automated Testing

The first step in the automated testing process involved directly connecting the VR device to a development computer. This connection was typically achieved via a USB-C cable, which facilitated both data transfer and device charging. Once connected, the device was ready to receive the application builds directly from the development environment. Testing on the actual device was critical for several reasons. Firstly, it allowed for an understanding of how the application would perform under the specific hardware constraints of the device, including processing power, memory capacity, and graphical capabilities, all of which could significantly affect the application's performance. Secondly, on-device testing helped in identifying real-world issues such

as thermal performance, battery drain, and interface responsiveness, which might not be apparent in a simulated environment.

To install the VR application onto the device, Unity3D's "Build and run" dialog was used. Unity3D compiled the application and automatically deployed it to the connected device, streamlining the development and testing process. The entire process is detailed in the flowchart shown in Figure 6.

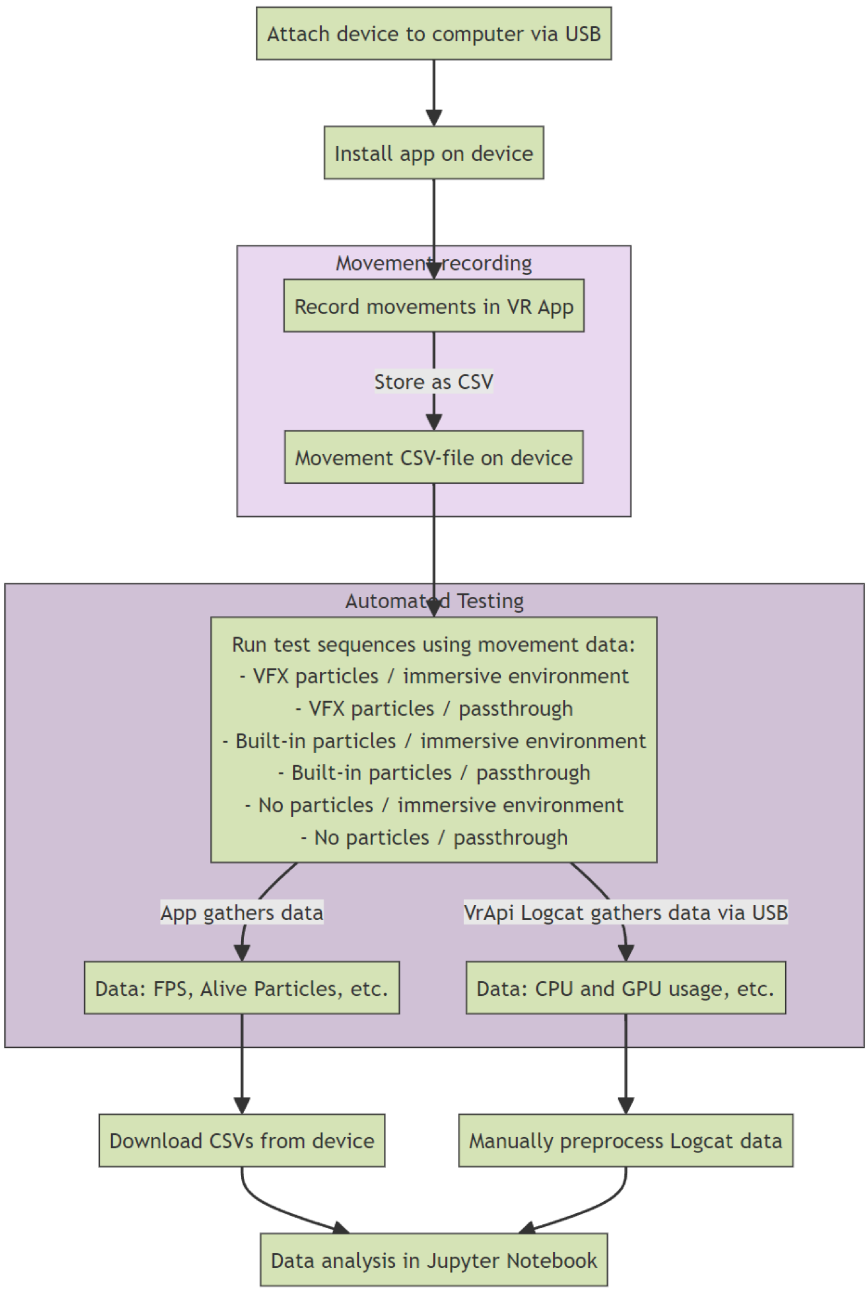

Figure 6. Flowchart of the automated testing procedure

To accurately assess how users interact with particle effects in virtual reality, an initial movement recording was performed. The testing application's starting screen, which provides options to either record movements or run tests, is depicted in Figure 7. During the movement recording process, a particle effect was displayed within the VR environment, allowing the user to naturally engage with elements that attracted

their attention. As the user moved their head to explore the scene, all corresponding movements were tracked and recorded. This data was captured and stored in memory. Subsequently, this data was saved into a CSV file on the device, providing a structured dataset that could be used for further analysis and testing.

Figure 7. Screenshot of the starting screen in the testing application

The movement data stored in the CSV file was then leveraged to recreate the exact sequence of user head movements observed during the initial recording. The tests for each mode (immersive VR or passthrough AR and built-in, VFX or no particles) were performed three consecutive times. The consistent replication of movement ensured that each test run followed the same parameters, allowing for reliable comparisons between different test results. Consequently, this method provided a standardized basis for evaluating the performance impacts of various changes in the visual effects and optimizations, ensuring that any observed differences in performance metrics were due to changes in the application itself rather than variations in user behavior.

During the automated testing sequence, data collection was executed through two distinct methods to ensure comprehensive performance analysis. Firstly, the application itself was configured to automatically gather and record key performance indicators, including the current fps and the count of active particles within the scene. This direct data capture from the application provided insights into the immediate graphical performance and the computational demands of the particle effects. Secondly, additional system-level information, such as CPU and GPU usage, was extracted from the VrApi Logcat [42]. Once the automated testing was

completed, the CSV files generated by the application were downloaded from the device. Concurrently, the data extracted from the VrApi Logcat underwent a manual preprocessing step.

3.7.1. Movement Recording

To ensure the accuracy and reproducibility of user interactions during testing, a reference point was added within the VR scene. This reference point was critical because of the nature of tracking the VR headset: it ensured that all movements were recorded and recreated in relation to this fixed point in the virtual environment. If a user had moved to a different position in real life between the recording and the testing phases, without this reference point, the recreated movements would have been offset by this physical displacement. This could have led to significant discrepancies in how the movements were executed during testing compared to the original recording. Anchoring the recorded data to a reference point within the virtual environment ensured that the testing conditions remained consistent, regardless of the user's actual physical location in the real world.

The app had a button where the movement recording could be started and stopped, as shown in Figure 8. The camera movements as well as the timestamp was recorded in every frame. This also meant that depending on the frame rate there might've been more or less data recorded. The camera's position and orientation were recorded to capture the movement trajectory, which could later be used for replaying the scenario or for further analysis.

A VFX particle effect was used during the movement recording, however, the particle effect used during the movement recording had a modest spawn rate of 20 particles per second. This was done to not overwhelm the device whilst the movement data was being recorded. Any disruptions in the user experience could've also affected the way the user reacts the stimuli in the environment, thus causing unnatural movement to be used as data in the automated testing portion. The movement data recorded and used in the tests can be seen in Figure 41 in the Appendix 1.

3.7.2. Running the Test Sequences

The automated tests were orchestrated through a series of coroutine-driven sequences that manipulated the environment and recorded data. The latest movement recording saved on the device was used for each test. The tests were initialized by disabling manual tracking and setting initial conditions. For each test, the system set specific conditions such as visual effects (VFX, built-in particles, or none) and passthrough options (enabled or disabled). Having tests with no particle effects was a vital benchmark for assessing the additional impacts when particle systems were introduced, allowing for a focused analysis on how particle effects modify performance beyond the base requirements.

Environmental setups were adjusted accordingly, such as toggling the 3D environment visibility and setting camera background properties based on passthrough settings. Each test was run three consecutive times (e.g., immersive mode with VFX

Figure 8. Screenshot of the recording screen in the testing application

particles was run three times in a row) to allow for averaging the data and mitigating any anomalies that might be present during a single test run.

During each test scenario, the framework recorded key performance metrics at predefined intervals. These metrics included fps, memory usage, GPU usage, and particle counts from the active visual effects. After each test, the system performed necessary cleanup, reset the environment, and prepared for the next test cycle. There was also an additional wait time of five seconds to give the system time to store the recorded data in a file on the device. Once all test scenarios were executed, the system re-enabled manual tracking and finalized the testing session, ensuring all resources were properly released and recorded data was saved securely. Screenshots from the test scenarios can be seen in Figures 42, 43, 44 and 45 in the Appendix 2.

3.8. Dynamic Adjustment of Particle Spawn Rate

In the exploration of optimization methods for particle effects in XR environments, experiments were conducted with dynamically adjusting the particle spawn rate based on real-time performance metrics. The feasibility of autonomously regulating the particle amount was investigated, aiming to enable the particle effects to adapt independently of the scene's complexity or hardware constraints.

A system was developed that adjusted the number of particles in response to fluctuations in fps. This system was designed to incrementally add particles until a noticeable drop in fps occurred, with the objective of maximizing particle density without exceeding the hardware's rendering capabilities. The feedback loop implemented in the system continuously increased particle spawn rates during periods of stable fps and decreased them when a decline in performance was detected. Each particle was assigned a lifetime of five seconds, which caused a delay in the system's response to changing performance conditions. This delay often resulted in adjustments being made based on outdated performance data, as the visual impact of added or removed particles only became apparent several seconds after the modification.

The results from this experiment were enlightening but ultimately highlighted significant drawbacks, as shown in Figure 9. The system tended to increase particle counts when the user looked at less complex parts of the scene. This often led to performance struggles when the focus shifted to areas with naturally higher particle density, resulting in performance drops at critical times.

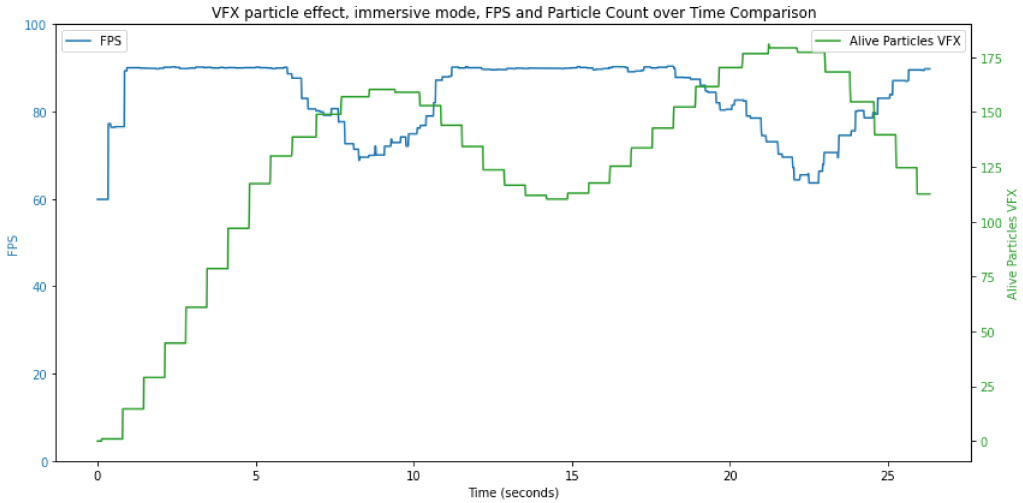

Figure 9. Dynamic VFX particle spawn rate adjustment's effects on fps over time

Figure 10 further illustrates how the lag caused by the particles' lifetime meant that the system was always reacting to past conditions, rendering it ineffective for real-time adjustments. These challenges led to the conclusion that, at least with this approach, dynamic optimization was not suitable for this application. This experience prompted the exploration of other predictive methods, such as polynomial regression, which offered a more stable and reliable framework for predicting appropriate particle levels without significant drops in performance metrics.

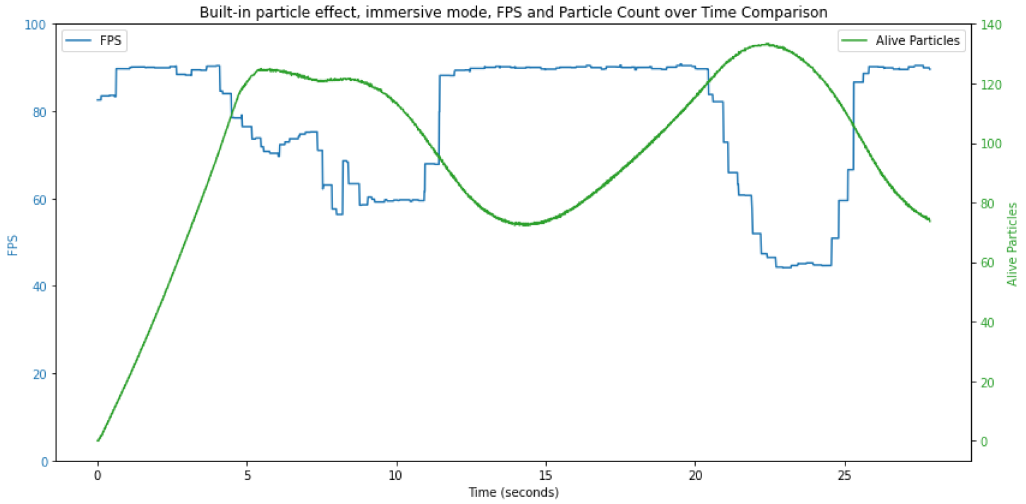

Figure 10. Dynamic built-in particle spawn rate adjustment's effects on fps over time

4. EVALUATION

This chapter focuses on the second major contribution of this thesis: the method used for performance testing of particle systems in XR environments. It details the process of data gathering from the automated tests and explains the use of polynomial regression to define the optimized levels of particles for each mode. The evaluation specifically analyzes the relationship between fps and the number of particles. The results of these evaluations are used for the final particle optimization, which is further discussed in the Chapter 5.

4.1. Data Gathering and Preprocessing

This section describes the methodology used for collecting and preparing the data necessary for evaluating the performance of particle systems in XR environments. It outlines the automated testing procedures employed to gather the performance metrics. The preprocessing steps, including data segmentation, formatting, and standardization, are detailed to ensure that the data is suitable for subsequent analysis. These steps transform raw log data into a structured format that can be effectively used for polynomial regression and performance optimization.

4.1.1. Logcat Data

Logcat [42] is a command-line utility included in the Android SDK [43], used extensively to capture real-time log messages from both the Android OS [44] and applications running on the device. For developers working with Meta Quest devices, Logcat can be used for monitoring and diagnosing behaviors of both the operating system and applications in real time. It captures a wide range of information, including system and Android-related messages, Meta Quest OS logs, and specific application performance data. Developers can leverage Logcat to track events such as application start and stop, headset interactions, and changes in CPU and GPU levels, as well as to log custom messages using the Android Log class. This makes Logcat a critical tool for identifying performance bottlenecks and determining the causes of crashes. [42]

In the context of this thesis, Logcat was used to collect detailed logs from the testing application to analyze the application's performance and operational status under various conditions. By connecting the Meta Quest device via USB or Wi-Fi and utilizing the Android Debug Bridge (ADB) [45] with commands like *adb logcat -s VrApi,PerformanceManager_ZSF,Unity > logcat-01.txt*, relevant log messages were filtered and saved into text files. These files were then parsed to extract meaningful performance metrics and operational data, which were crucial for the subsequent data analysis. The ability to filter logs by tags such as 'VrApi' and 'PerformanceManager_ZSF' significantly reduced the volume of data to a manageable and relevant subset, focusing specifically on the aspects critical to the study's objectives. This method provided a low-overhead, engine-agnostic way to gather insights into the app's performance, directly influencing the development and optimization of the XR application.

In the initial stage of the data preparation process, the raw log file was parsed to extract distinct recording sessions based on predefined start ("Recording starting") and end ("Recording stopped") markers found within the text, added by the automated testing application. This parsing facilitated the segmentation of the continuous log data into discrete sections, each corresponding to a specific test, which were then stored separately for subsequent analysis.

To efficiently handle and analyze the segmented data, each section was converted into a structured format using the Pandas library in Python. Column names extracted from the logs were standardized by stripping extraneous whitespace and renaming them for consistency and clarity. Key columns, such as timestamps, were converted from strings to datetime objects to enable time-series analysis. For columns containing numerical data embedded within strings, regular expressions were employed to extract and convert these values to appropriate numeric types, facilitating quantitative analysis. Each section was then transformed into a separate DataFrame, a two-dimensional data structure with labeled axes, to support data manipulation and analysis.

To refine the DataFrames, specific columns of interest were selected, reducing the dataset to a manageable size by excluding irrelevant data. This step focused the analysis on variables critical to the study's objectives. For further analysis, the DataFrames were grouped by their test configurations. This grouping was achieved by organizing DataFrames into groups based on their test configuration names, excluding the run numbers to aggregate data from similar test setups.

To calculate the average DataFrame for each configuration, DataFrames belonging to the same configuration were concatenated along the vertical axis to form a single, comprehensive DataFrame. Non-numeric columns were filtered out to ensure that only numeric data was included in the averaging process, by attempting to convert each column to a numeric type and excluding those that could not be converted. The remaining numeric columns were then averaged by grouping the concatenated DataFrame by its index, computing the mean of each numeric column across all runs for the same configuration.

Using the Matplotlib library, multiple plots were organized in a grid layout to allow comparative analysis across different sections, providing visual insights into the trends and anomalies within the data. Throughout the data preparation process, error handling mechanisms were integrated to manage issues such as missing values, incorrect data formats, and parsing errors. This approach ensured the robustness and reliability of the data transformation process.

4.1.2. CSV-Files

In the preprocessing stage of the data analysis, the Python libraries pandas, matplotlib.pyplot, glob, and os were imported. The dataset consisted of several metrics including fps, number of triangles, draw calls, vertices, memory usage, GPU usage, and particle counts. The raw data files, however, lacked column headers, necessitating the preliminary step of defining appropriate column labels to ensure the data arrays were accurately understood and manipulated.

Utilizing the glob module, the script programmatically located all CSV files within a specified directory, allowing for a dynamic adaptation to any number of files, i.e.

tests, located in this directory. This dynamic importing and handling of the files also makes the testing procedure faster, as new files can be added to the folder for analysis without any need to change filename references. For each file identified by the glob function, the data was read into a pandas DataFrame without pre-assigned headers. After reading, headers were assigned to the DataFrame based on a predefined list. Each DataFrame was then stored in a dictionary, keyed by a unique identifier derived from the filename. This approach allowed for efficient storage and retrieval of individual datasets, supporting distinct analyses per data set.

4.2. Data Analysis

The data was first segmented by test configuration, with each configuration representing different modes (such as passthrough and immersive) and types of particles (VFX or built-in). For each configuration, relevant performance metrics such as fps, number of triangles, draw calls, vertices, memory usage, and the count of alive particles (VFX and built-in, as applicable) were extracted from the averaged datasets. Any metrics consistently reporting zero values, such as GPU Usage, were excluded to streamline the analysis.

In the automated tests, the calculation of fps was done by tracking the number of frames that render over a specific period. Each time the game updates, which happens once per frame, a frame counter increases by one, and the time taken to complete each frame is added together. This continues until half a second has passed. At this point, the script calculates the average fps by dividing the total number of frames by the total time elapsed in that half-second period. This gives a measure of how many frames, on average, are being rendered each second during that interval. After the calculation, the counters for both the number of frames and the elapsed time reset, allowing the measurement process to start fresh for the next half-second period. This ongoing tracking helps in assessing the performance of the application in terms of how smoothly it is running.

The automated tests were run again, this time using the suggested maximum particle counts from the predictive models shown in Table 6. Since the maximum allowed particles could not be dynamically changed at runtime, the maximum spawn rate was calculated by dividing the suggested maximum particle count with the lifetime of the particles, which in this case was 5 seconds. Like in the previous tests, the particle spawn rate was increased over time, but this time, instead of using the maximum spawn rate of 50, an individual maximum count was used for each particle effect and mode. The results from these tests are detailed in Chapter 5.

Table 6. Estimated maximum particle count and particles spawn rate (particles created per second) for different particle effects in different modes

Particle Effect and Mode	**Estimated Maximum Particle Count**	**Calculated Spawn Rate**
VFX particles, immersive mode	130	26
VFX particles, passthrough mode	152	30.4
Built-in particles, immersive mode	80	16
Built-in particles, passthrough mode	83	16.6

4.3. Using Predictive Models to Optimize Particle Density

In this study, a predictive modeling approach was used to optimize the performance of particle effects using the data gathered from the automated tests. To achieve this, data were gathered on the relationship between the particle count and the resultant fps, with the aim of maintaining a high user experience by ensuring a stable fps. The analysis was commenced by isolating the subset of data where the fps first reached and sustained a minimum of 90 fps, to prevent any interference from any initial anomalies caused by the starting phase of the tests.

A polynomial regression model was then constructed to capture the non-linear relationship between the particle count and fps. The model was instantiated as a second-degree polynomial, a decision based on preliminary analyses suggesting that a quadratic relationship sufficiently encapsulated the complexity of the interaction between the variables. Utilizing the PolynomialFeatures [46] module from scikit-learn [30], the model was designed to include not only the linear term but also the square of the particle count, thus allowing for a curved fit that better matched the observed data.

The predictive power of the model was quantitatively assessed through the coefficient of determination, R^2, which provided insight into the proportion of variance in fps that could be explained by the polynomial model. The R^2 value, calculated post model fitting, served as a metric to evaluate the effectiveness of the particle count in predicting fps outcomes. Table 7 shows the R^2 values calculated for each particle effect and mode, suggesting the varying degrees of fit achieved by the polynomial regression models. The R^2 values range from 0.68 in passthrough mode for VFX particles, indicating a moderate fit, to 0.98 for built-in particles in both immersive and passthrough modes, which suggests an excellent fit.

Table 7. R^2-value for different particle effects in different modes

Particle Effect and Mode	R^2
VFX particles, immersive mode	0.94
VFX particles, passthrough mode	0.68
Built-in particles, immersive mode	0.98
Built-in particles, passthrough mode	0.98

To determine the maximum number of particles that could be sustained while maintaining an fps of at least 90, fsolve from scipy.optimize [31] was used. Since fps was the dependent variable predicted by the model, and the goal was to find the corresponding number of particles for a desired fps threshold, fsolve was employed to invert the model. This numerical solver allowed for the estimation of maximum number of particles that would achieve the targeted fps, providing actionable guidance on optimizing particle effects without compromising performance.

Visual representations of the model's fit and predictions were generated using matplotlib [32], illustrating both the actual fps data points and the regression line predicted by the model. These visualizations not only served to validate the model visually but also helped in communicating complex relationships in an accessible format, supporting the interpretability and transparency of the analytical process.

Figure 11 displays the relationship between particle count and fps for a VFX graph particles in immersive mode, using polynomial regression to predict performance. The red line represents the predicted fps based on the polynomial regression model, while the blue dots signify the actual fps observed at various particle counts. The graph demonstrates that as the particle count increases, the actual fps generally decreases, following the trend predicted by the model. Notably, the actual fps remains relatively stable until the particle count exceeds approximately 150, after which it begins to decline more steeply, dramatically dropping as the count approaches 250.

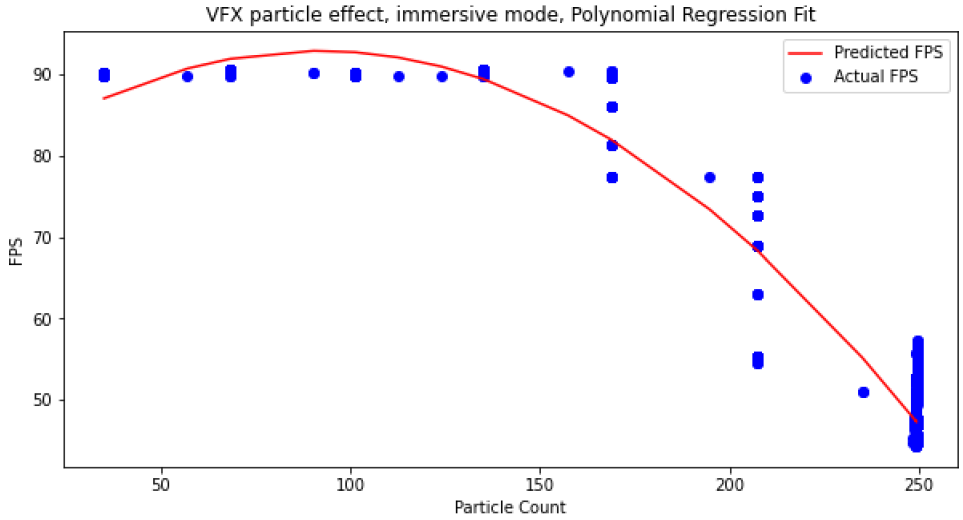

Figure 11. Polynomial regression fit for VFX particles in immersive mode

Similar to the previous plot, Figure 12 illustrates the relationship between particle count and fps for VFX graph particles, this time in passthrough AR mode. As with the immersive VR mode, the polynomial regression model predicts a decline in fps with increasing particle counts. The actual fps depicted by the blue dots remains relatively stable until surpassing 150 particles, where it begins to drop sharply, especially as it approaches 250 particles, mirroring the pattern observed in the immersive mode but with a slightly steeper decline towards the higher counts.

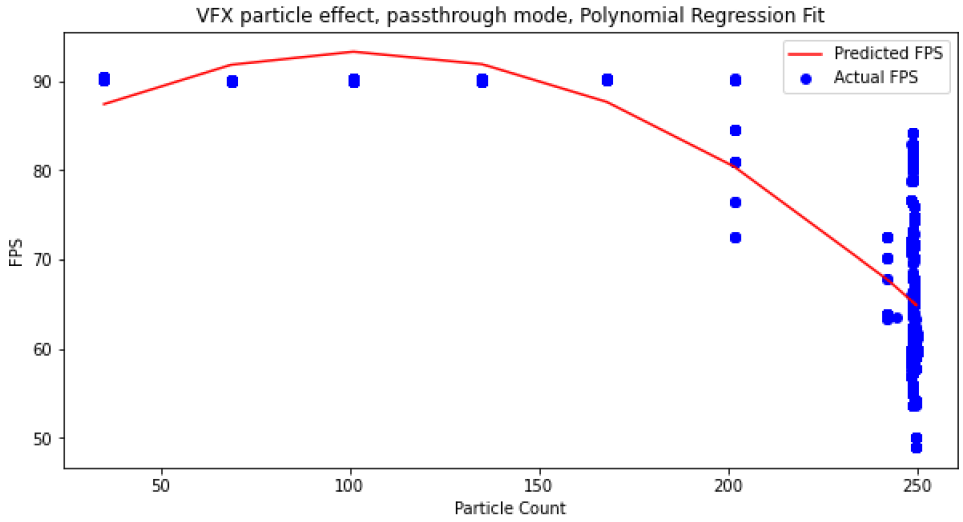

Figure 12. Polynomial regression fit for VFX particles in passthrough AR mode

Figure 13 extends the analysis to the built-in particle effect in immersive mode, displaying the relationship between particle count and fps. Similar to the previous plots, it shows a decline in actual fps with increasing particle counts. The red line represents the predicted fps based on a polynomial regression model, which in this case could potentially benefit from a third-degree polynomial for a closer fit to the actual data. The actual fps, depicted by the blue dots, shows a more pronounced decrease compared to the VFX particle effects, beginning a significant drop as early as 150 particles and continuing to decline steeply towards 250 particles.

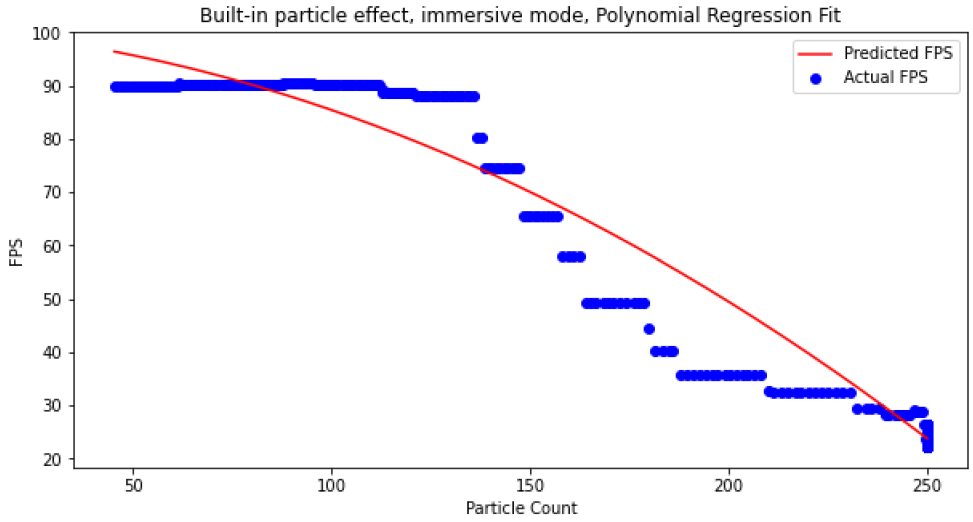

Figure 13. Polynomial regression fit for built-in particles in immersive mode

Figure 14 illustrates the performance of the built-in particle effect in passthrough mode, analyzed through polynomial regression to predict fps. As with the previous plots, the decline in actual fps with increasing particle counts is evident, depicted by the blue dots. The predicted fps, represented by the red line, suggests a second-degree polynomial fit, though considering a third-degree polynomial might offer a closer approximation given the curvature and steeper decline observed beyond 150 particles. The disparity between predicted and actual fps widens as particle counts approach 250, underscoring the potential for further refinement in the regression model to more accurately mirror real-world performance.

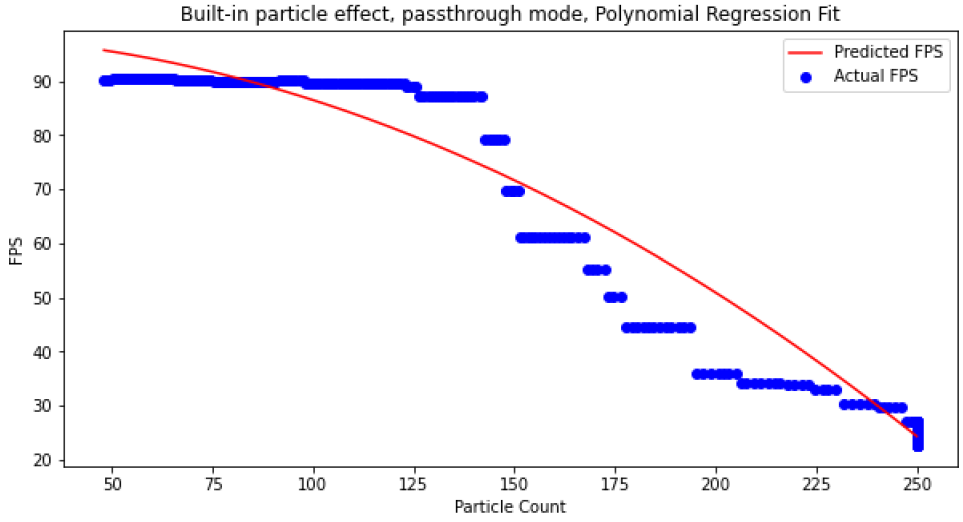

Figure 14. Polynomial regression fit for built-in particles in passthrough mode

5. RESULTS

In this chapter, a comparative analysis is presented that visually delineates the performance implications of initially utilizing an almost unlimited number of particles versus the improved performance after implementing restrictions on the maximum number of particles. These restrictions were derived from a polynomial regression analysis conducted on data obtained from initial tests. The 'before' images or plots represent the baseline performance metrics—such as GPU and CPU utilization, App GPU Time, and the occurrence of stale frames—captured under conditions where the particle effects were maximized without constraints. These initial conditions were intended to stress the system and establish a performance benchmark at peak load. The performance metrics discussed in this chapter have been further explained in Table 4 in Chapter 2.

The 'after' images or plots reflect the subsequent performance following the application of a calculated maximum particle threshold, which was determined to optimize the rendering efficiency and overall application responsiveness. This methodical approach allowed for a controlled analysis of how specific adjustments to the number of particles influence key performance metrics. By presenting the data in a 'before and after' comparison, the narrative clearly demonstrates the direct impact of particle optimization on enhancing the XR experience. This comparative presentation underscores the effectiveness of the optimizations and provides a clear, visual representation of the potential for fine-tuning performance through targeted adjustments based on analytical findings.

To further ensure the validity and comparability of the results presented in this thesis, an automated testing framework was developed and utilized. This framework controlled the camera movement consistently across all tests, standardizing the test conditions and eliminating variations that could arise from manual testing. The automation of these tests streamlined the data collection process and provided a robust foundation for comparing the 'before' and 'after' scenarios, ensuring that the performance improvements documented are directly attributable to the implemented optimizations and not external variables.

Additionally, correlation matrices were used to examine the relationships between various performance metrics. These matrices provided insights into how different variables interact and influence each other, highlighting the differences between passthrough AR and immersive VR modes. This analysis clarified the question of whether there is a performance difference between these modes and how they could be further optimized. While the optimization in this thesis was limited to the number of particles, the results from the correlation matrices suggest potential areas for further optimization beyond number of particle.

5.1. Correlation between Different Performance Metrics

The correlation matrices presented here were generated using Python to show the relationships between various performance metrics and the particle systems. The Seaborn library [33] was then used to compute and visually represent the correlation

coefficients in a heatmap format, providing a color-coded depiction of how strongly each pair of metrics is related.

The correlation matrix for built-in particles in immersive mode (Figure 15) highlights several notable relationships. There is a strong negative correlation (-0.88) between fps and the number of triangles and vertices, indicating that higher complexity in these aspects leads to a decrease in fps. This suggests that as the complexity of particle effects increases, the performance of the application in immersive mode tends to degrade. Similarly, there is a negative correlation between alive particles (i.e., the number of particles in the scene at that moment) and fps, which underscores the performance challenges when managing large volumes of particles.

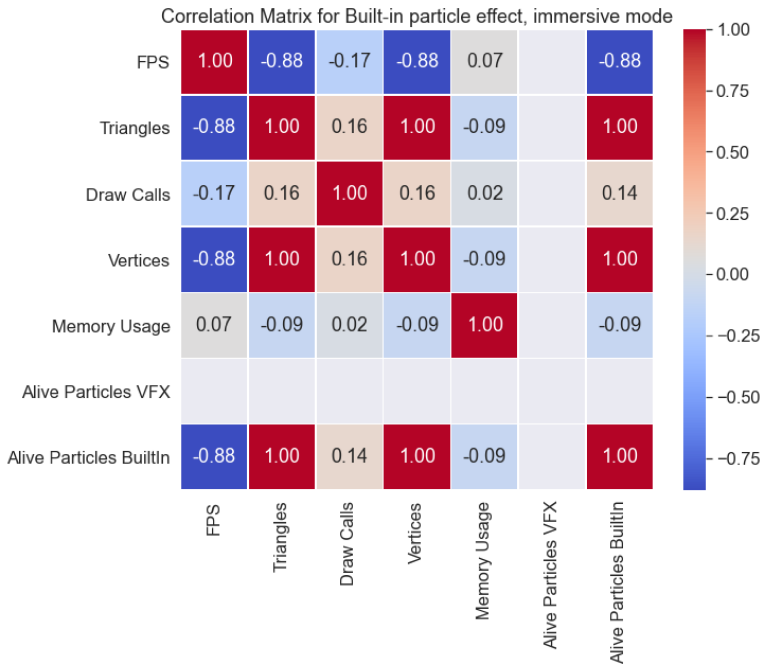

Figure 15. Correlation matrix for built-in particle effects in immersive mode

In passthrough mode, the correlation matrix in Figure 16 shows similar strong negative correlations between fps and key graphical elements like triangles and vertices. The negative correlations here are slightly less severe compared to immersive mode, suggesting that passthrough mode, while still affected by increased complexity, handles it somewhat more efficiently. Noteworthy is the consistent strong negative impact of alive built-in particles on fps across both modes, reinforcing the need for careful management of particle counts to maintain optimal performance.

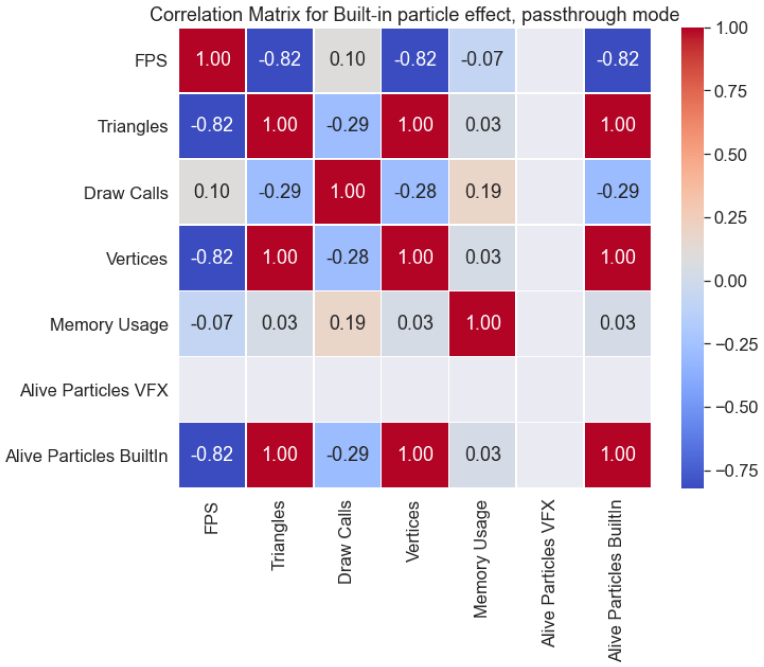

Figure 16. Correlation matrix for built-in particle effects in passthrough mode

For VFX graph, in immersive mode, as depicted in Figure 17, there is a moderate negative correlation between the number of alive VFX particles and fps (-0.70), which is less pronounced compared to the strong negative impacts observed with built-in particles. This suggests that VFX particles may be less taxing on fps, potentially due to more efficient handling or lower complexity in their rendering processes. Additionally, the matrix shows strong positive correlations among triangles, draw calls, and vertices, indicating that increases in scene complexity significantly elevate resource demands.

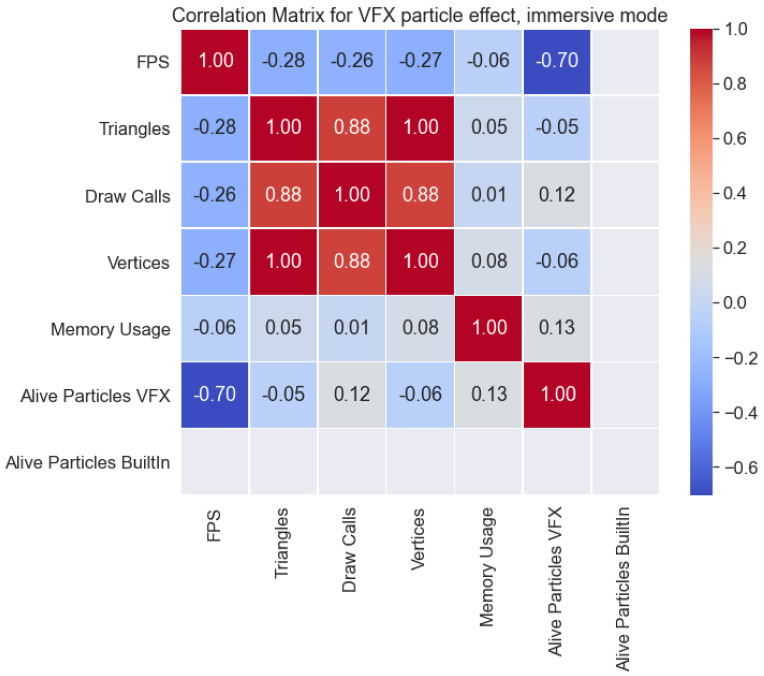

Figure 17. Correlation matrix for VFX particle effects in immersive mode

In passthrough mode, as shown in Figure 18, the negative correlation between alive VFX particles and fps is slightly stronger (-0.71) compared to immersive mode. Like in immersive mode, there is a strong interconnectedness between triangles, draw calls, and vertices, consistent with the demands imposed by increased scene complexity.

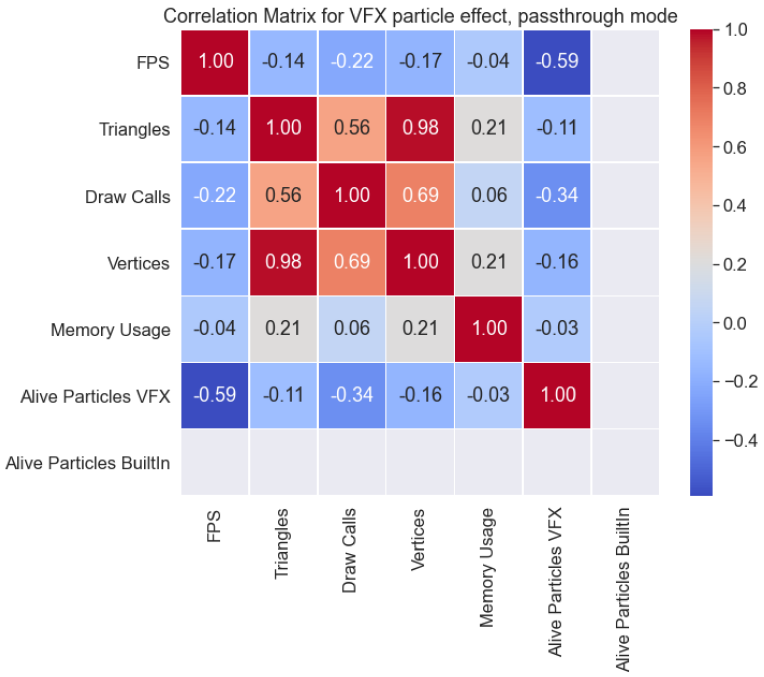

Figure 18. Correlation matrix for VFX particle effects in passthrough mode

In immersive mode with no particle effects, as shown in Figure 19, there is a very strong positive correlation between triangles, draw calls, and vertices, with draw calls exhibiting almost perfect correlations with triangles and vertices (0.94 and 1.00 respectively). This demonstrates that increased geometric complexity in scenes directly escalates rendering demands, reflecting fundamental aspects of graphical rendering in VR environments.

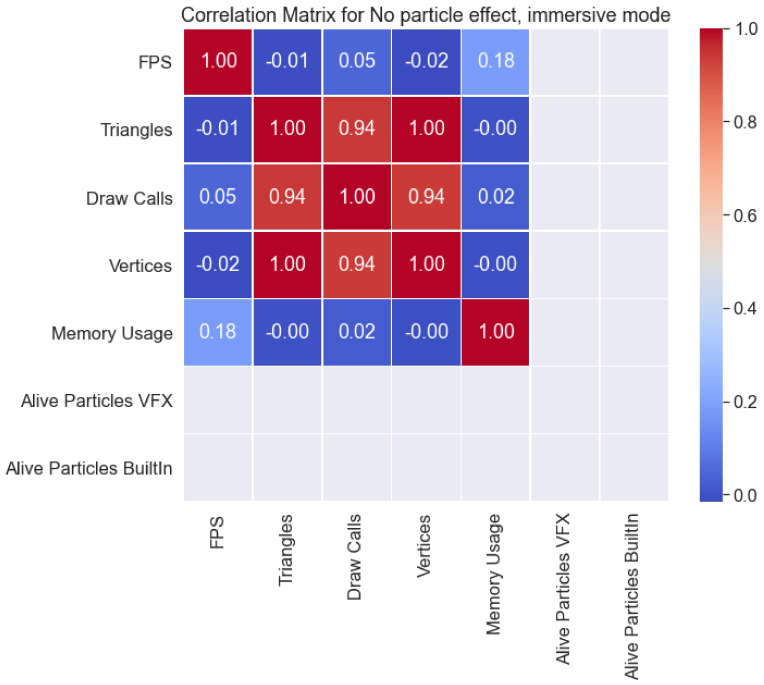

Figure 19. Correlation matrix for no particle effects in immersive mode

In passthrough mode with no particles, as illustrated in Figure 20, the correlation matrix reveals similar strong relationships between triangles, draw calls, and vertices. Although slightly lower than in immersive mode, the significant correlations (draw calls with triangles at 0.98 and with vertices at 1.00) still highlight that the fundamental complexities of scene geometry continue to play a crucial role in determining rendering demands, even in the absence of particle effects.

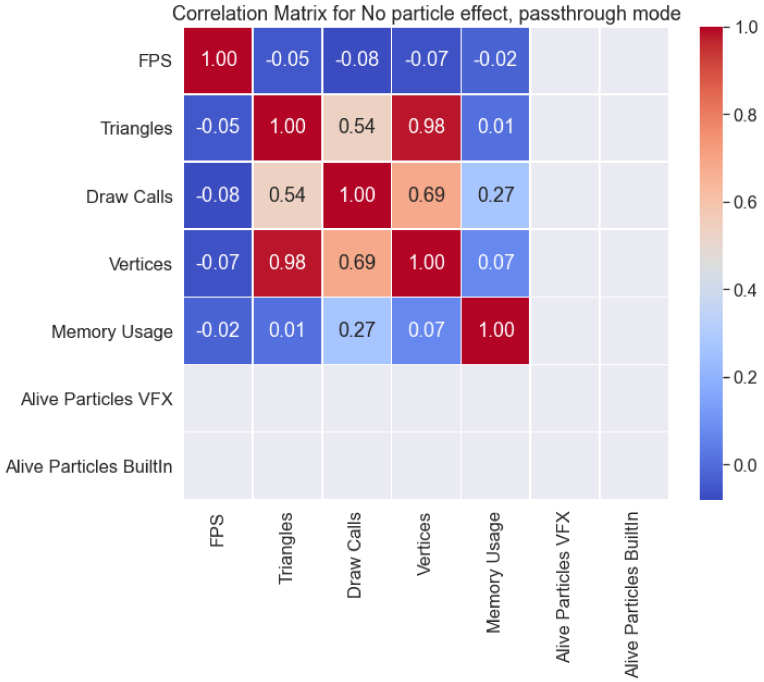

Figure 20. Correlation matrix for no particle effects in passthrough mode

5.2. Particle Count in Relation to Fps

This section examines the impact of particle count on fps before and after optimization. The goal of the optimization was to achieve or approach a maximum of 90 fps, enhancing both the visual quality and responsiveness of the VR experience.

Initially, as depicted in Figure 21, the fps significantly fluctuates as the particle count increases over time. This plot captures the starting conditions where the system was stressed with a high number of particles without any optimization. The initial data shows that as particle count ramps up, the fps correspondingly dips, struggling to maintain a consistent performance level.

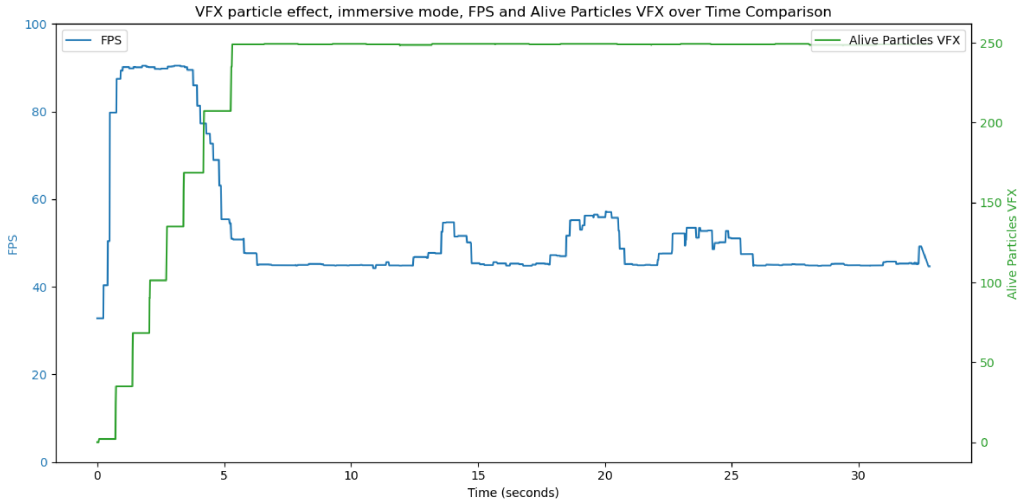

Figure 21. VFX particles and it's relation to fps in immersive mode, before optimization

Figure 22 shows the effects of the optimization. Here, the fps is maintained closer to the 90 fps target despite the gradual increase in particle count, indicating successful optimization efforts.

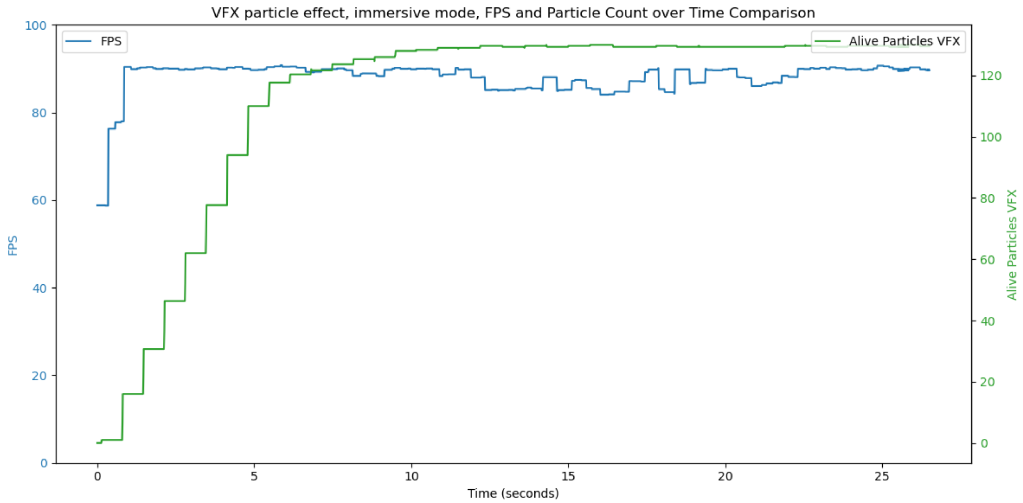

Figure 22. VFX particles and it's relation to fps in immersive mode, after optimization

Slight deviation from the maximum could still be seen in these results, which can be attributed to test number 2, as shown in Figure 23. It is not quite clear what caused this, but it could be considered an anomaly.

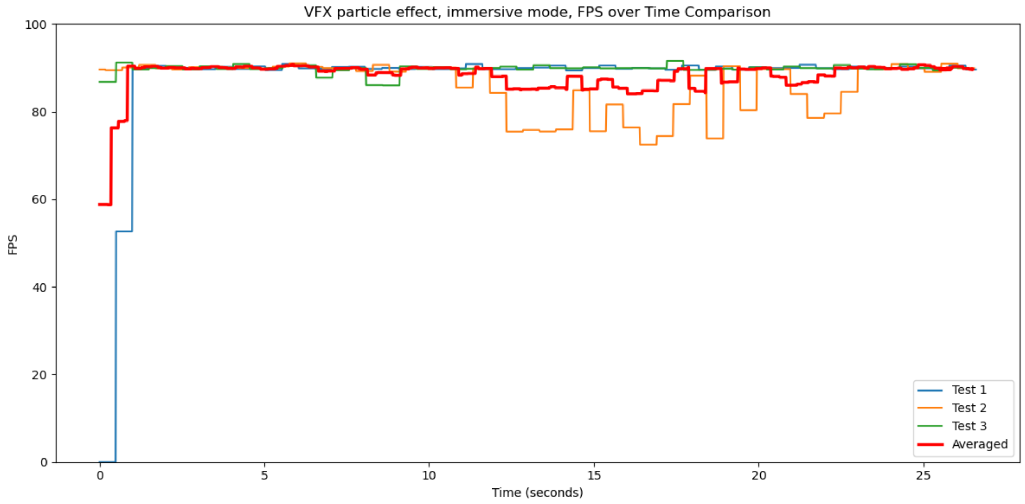

Figure 23. VFX particles and it's relation to fps in immersive mode, after optimization, showing all the tests and their averages

The initial performance in passthrough mode for VFX graph particles, as illustrated in Figure 24, reveals substantial fluctuations in fps as the particle count increases over time.

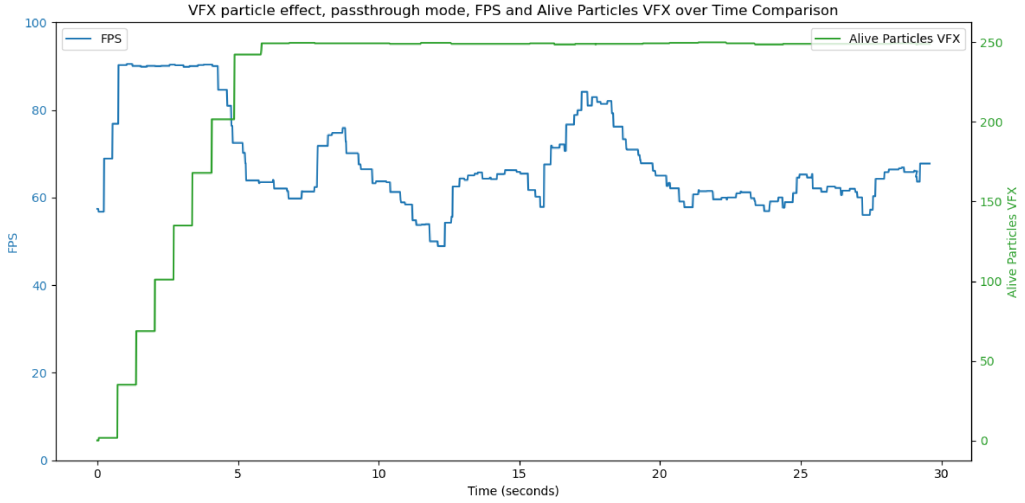

Figure 24. VFX particles and it's relation to fps in passthrough mode, before optimization

Following the application of optimization derived from polynomial regression analysis, Figure 25 shows a marked improvement in maintaining stable fps even as particle counts increase. By setting a maximum threshold for particles based on the regression analysis, the system is able to sustain nearly optimal fps throughout the duration of the test, effectively demonstrating the benefits of the optimization.

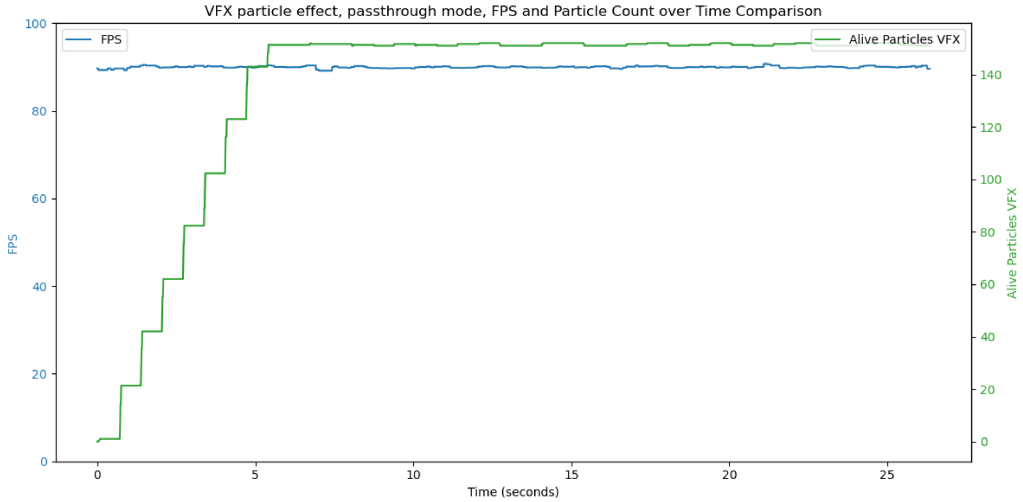

Figure 25. VFX particles and it's relation to fps in passthrough mode, after optimization

The initial performance captured in Figure 26 shows the dynamic relationship between fps and the count of built-in particles over time in immersive VR mode. Initially, as the particle count rapidly increases, there is a significant and sharp drop in fps, which illustrates the system's struggle to maintain performance under the stress of high particle densities. This steep decline in fps as particle counts rise highlights the challenges faced without optimization, where the rendering of built-in particles heavily taxes the system.

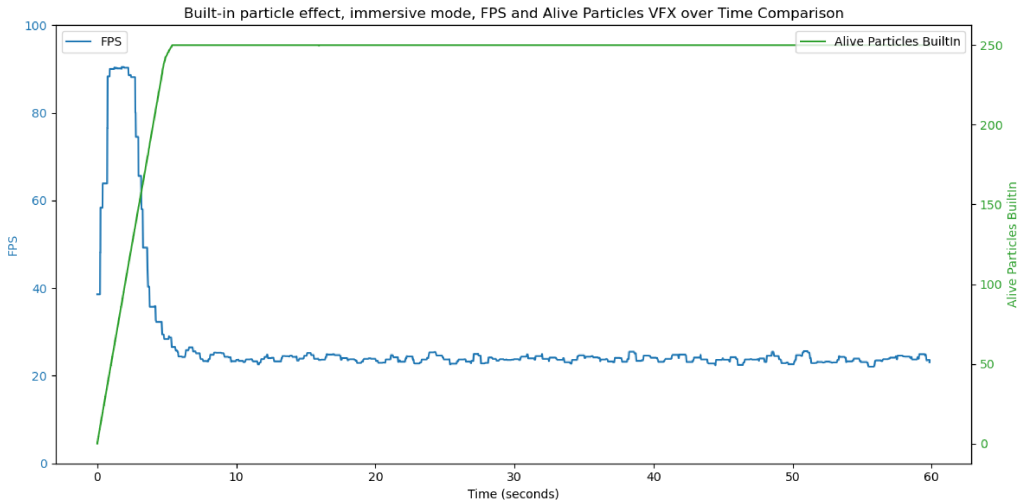

Figure 26. Built-in particles and it's relation to fps in immersive mode, before optimization

In contrast, Figure 27 demonstrates the effectiveness of the optimization strategies. After implementing a maximum particle count limit found through polynomial regression analysis, the fps remains stable and high as the particle count is being capped at a lower threshold. This optimized scenario reveals that the fps is maintained close to the ideal 90 fps, confirming that the adjustments made to the particle system

effectively mitigate the performance impact while still delivering a visually engaging experience.

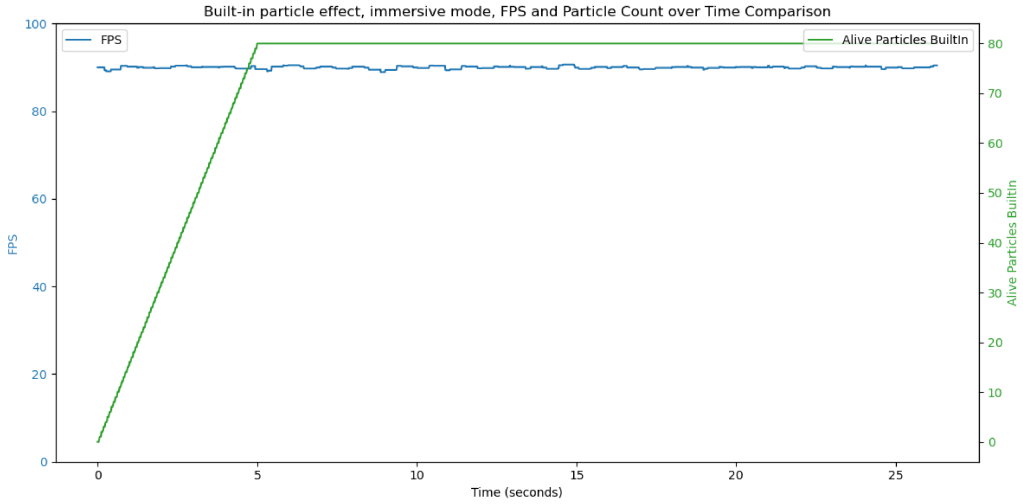

Figure 27. Built-in particles and it's relation to fps in immersive mode, after optimization

Figure 28 captures the initial performance metrics of built-in particle effects in passthrough AR mode. Initially, there is a steep decrease in fps as the number of alive particles increases, underscoring the substantial impact of high particle counts on system performance. The fps drops significantly from the maximum possible value, showing how the system struggles to handle the load before any optimizations are applied.

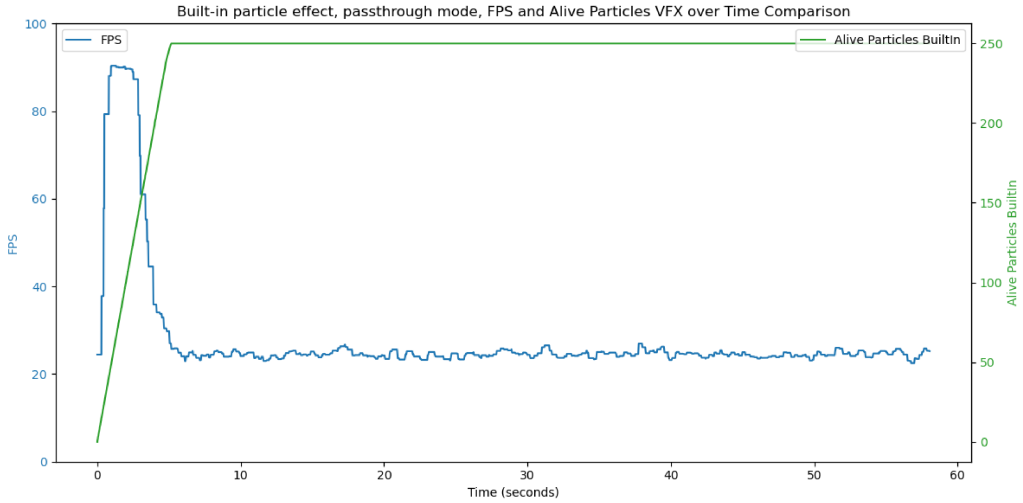

Figure 28. Built-in particles and it's relation to fps in passthrough mode, before optimization

Post-optimization, Figure 29 shows a stark contrast to the initial performance, with fps stabilizing at high levels even as the particle count increases. The optimization clearly demonstrates its effectiveness, maintaining fps close to the maximum throughout the test period, reflecting successful adjustments to manage particle effects without sacrificing performance.

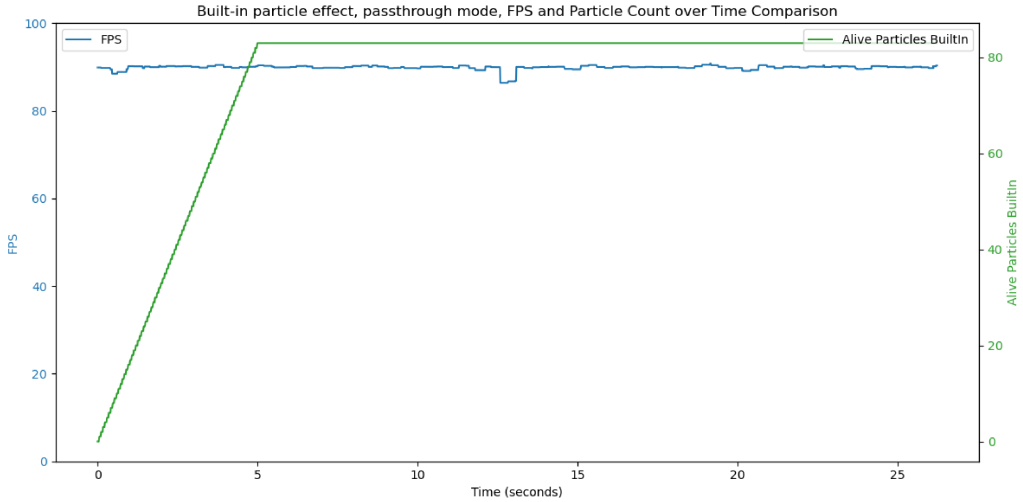

Figure 29. Built-in particles and it's relation to fps in passthrough mode, after optimization

Similarly to the VFX particle effect in immersive mode, there was also a slight performance issue seen in the built-in particle effect in passthrough mode. As seen in Figure 30 this was also caused by test number 2. The cause is unclear, but could be considered an anomaly.

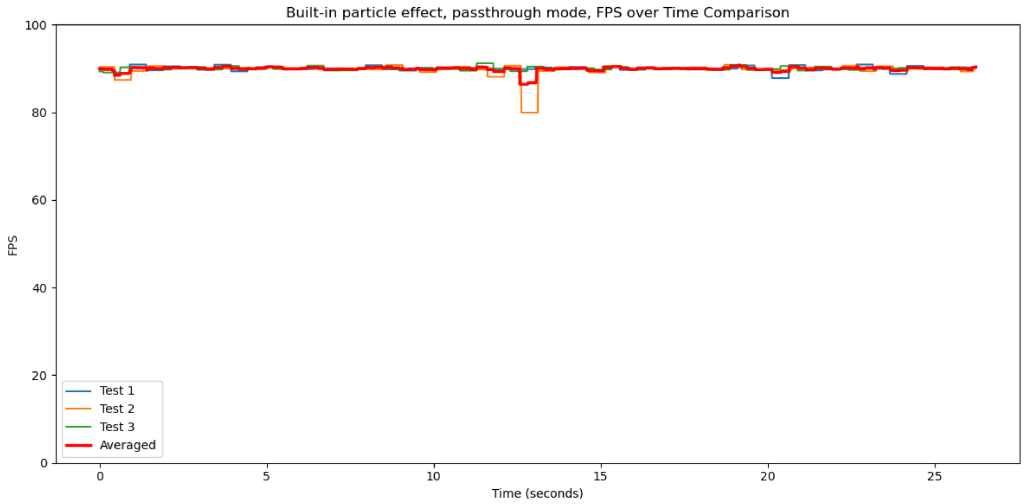

Figure 30. Built-in particles and it's relation to fps in passthrough mode, after optimization, showing all the tests and their averages

5.3. CPU and GPU Utilization

This section examines the CPU and GPU utilization before and after the optimizations. By analyzing these metrics, we can understand the impact of the optimizations on resource management and overall system performance. High CPU and GPU utilization can lead to scheduling issues and performance degradation. The figures presented below illustrate the improvements achieved through the optimizations.

Figure 31 shows the GPU utilization before optimization. The graph highlights that GPU usage frequently reached levels close to 90%, particularly in scenarios involving VFX particle effects. This high utilization suggests that the system was GPU-bound, struggling to maintain performance under heavy graphical loads. Such high levels of GPU utilization can lead to scheduling conflicts and performance bottlenecks, negatively impacting the frame rate and overall user experience.

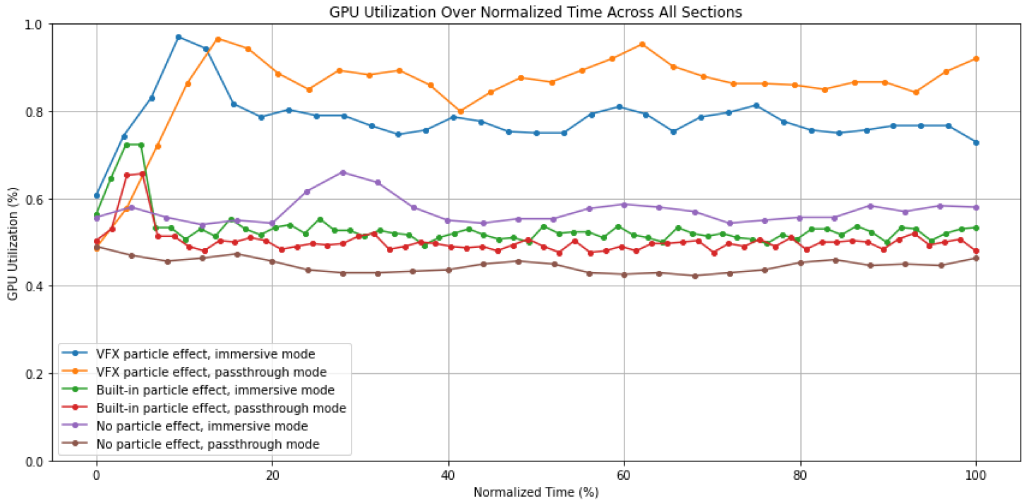

Figure 31. GPU Utilization before the optimization

Figure 32 illustrates the GPU utilization after optimizations were applied. The optimizations redistributed the workload more efficiently, reducing peak usage and ensuring that the GPU operates within a more stable and efficient range. This improvement helps maintain higher frame rates and a smoother visual output.

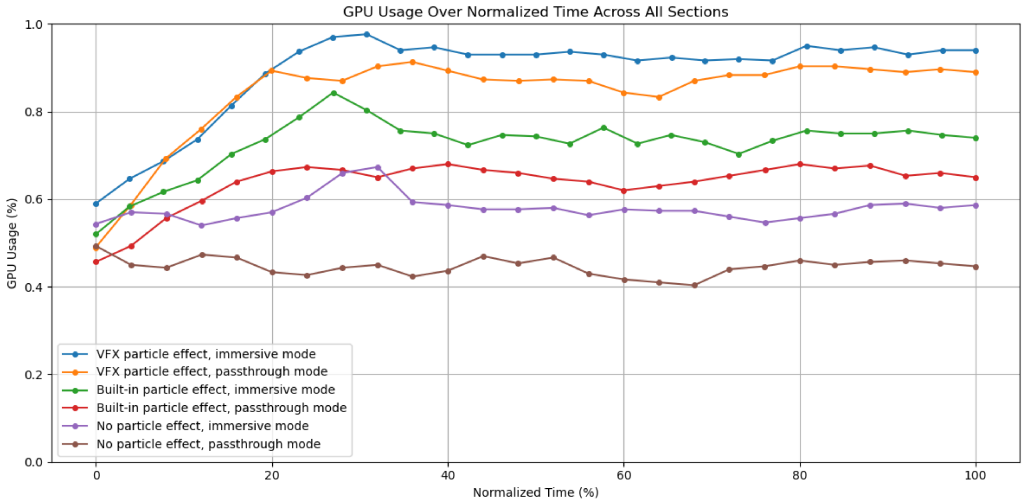

Figure 32. GPU Utilization after the optimization

Figure 33 depicts the CPU utilization before the optimizations. The data reveals that the CPU was heavily utilized, particularly during complex particle effect rendering. High CPU usage indicates intensive computational demands, which can result in increased latency and reduced application responsiveness. Managing CPU load is crucial for maintaining smooth operation and preventing performance degradation.

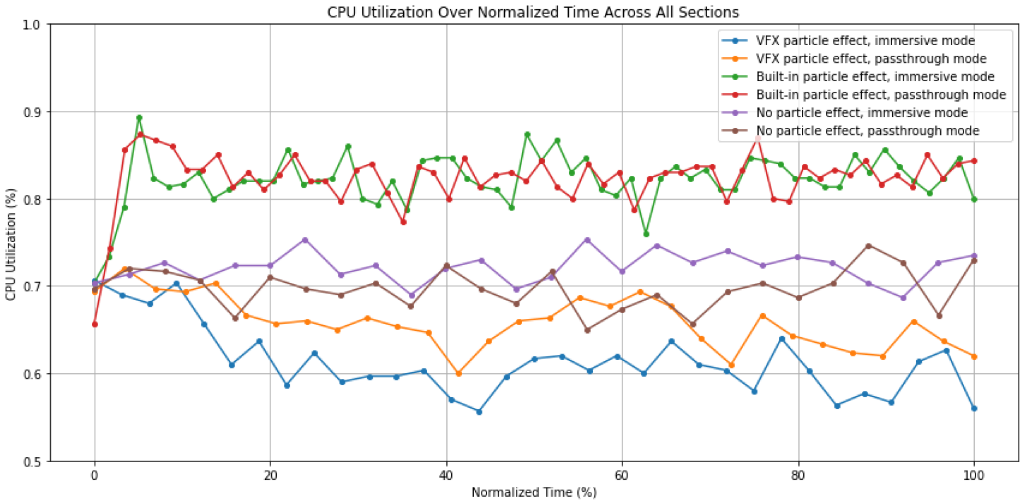

Figure 33. CPU Utilization before the optimization

Figure 34 presents the CPU utilization post-optimization. There is a decrease in CPU load across all scenarios, demonstrating the effectiveness of the optimizations. By optimizing the particle effects and balancing the computational demands, the CPU usage was also more stable, enhancing the overall performance and responsiveness of the application.

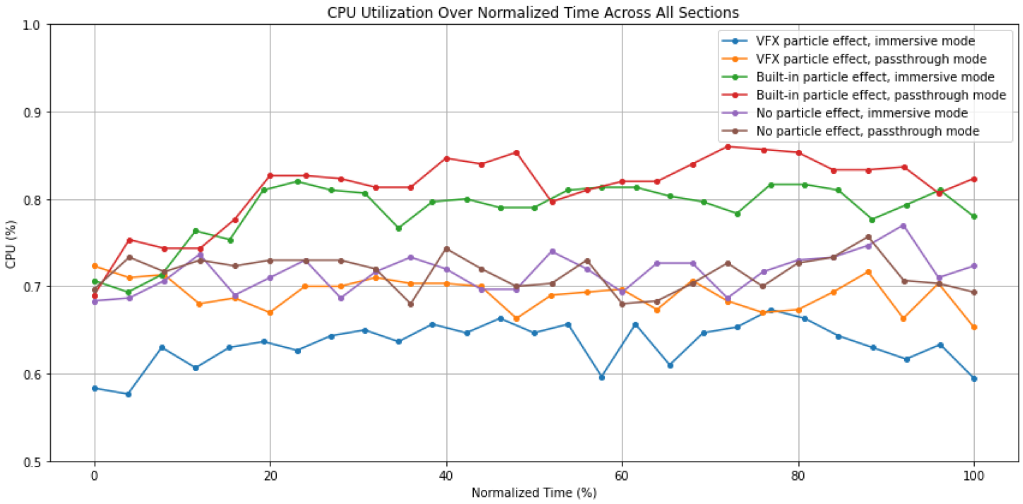

Figure 34. CPU Utilization after the optimization

Figure 35 shows the combined CPU and GPU utilization before optimizations. The graph provides a comprehensive view of the system's resource usage, indicating high utilization rates for both CPU and GPU. This combined high utilization highlights the system's struggle to manage the computational demands, underscoring the need for effective optimization strategies.

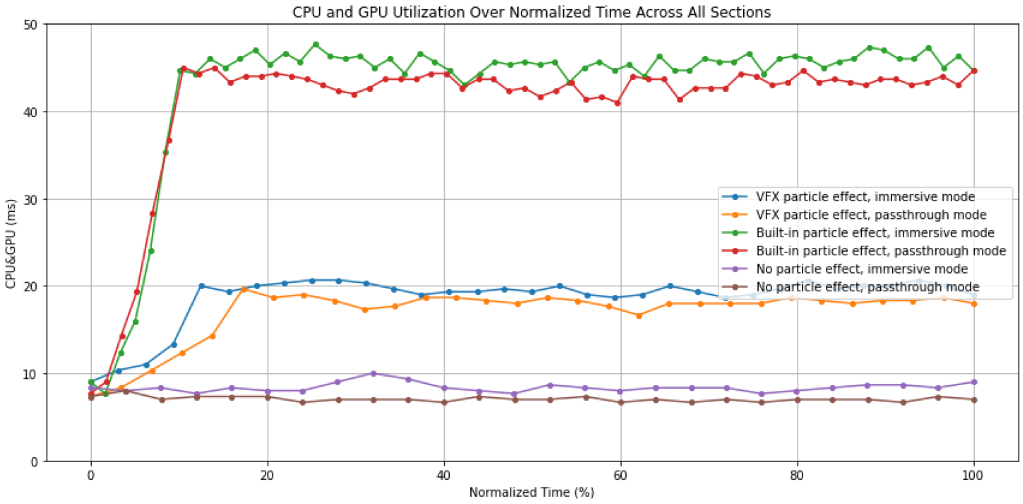

Figure 35. CPU and GPU Utilization in milliseconds before the optimization

Figure 36 illustrates the combined CPU and GPU utilization after the optimizations. The data shows a significant reduction in both CPU and GPU usage, especially for the built-in particle systems, indicating that the optimizations successfully alleviated the computational load. This improvement in resource management ensures better performance stability and efficiency, providing a smoother user experience in XR environments.

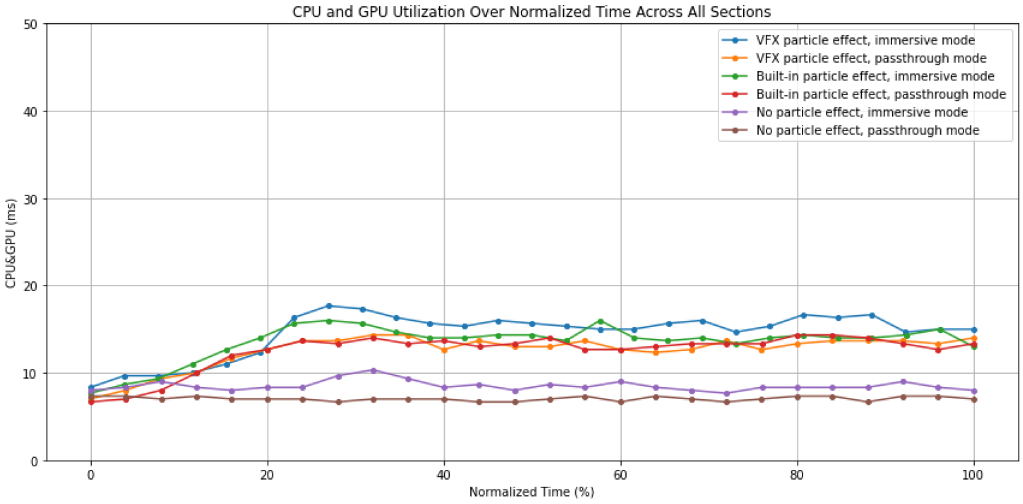

Figure 36. CPU and GPU Utilization in milliseconds after the optimization

App GPU Time indicates the amount of time spent rendering a single frame. Measured in milliseconds (ms), it helps determine whether your application is GPU-bound or CPU-bound. For a maximum frame rate of 90 fps, a single frame length should not exceed 11.11 ms. If App GPU Time is more than this threshold, it indicates the application is GPU-bound. Conversely, if it's less, the application is likely CPU-bound. Monitoring App GPU Time over time and during changes to shaders, textures, or meshes provides valuable insights into the performance impact of these elements. [19]

Figure 37 shows the App GPU Time before optimizations were applied. The data reveals that the time taken to render a single frame frequently exceeded the 11.11 ms

threshold, especially in scenarios involving VFX particle effects. This indicates that the application was GPU-bound, struggling to render the complex graphics efficiently. The high App GPU Time was particularly pronounced in immersive mode, where the visual demands are higher. Such high values suggest that the application was heavily reliant on the GPU, leading to performance bottlenecks and potentially lower frame rates.

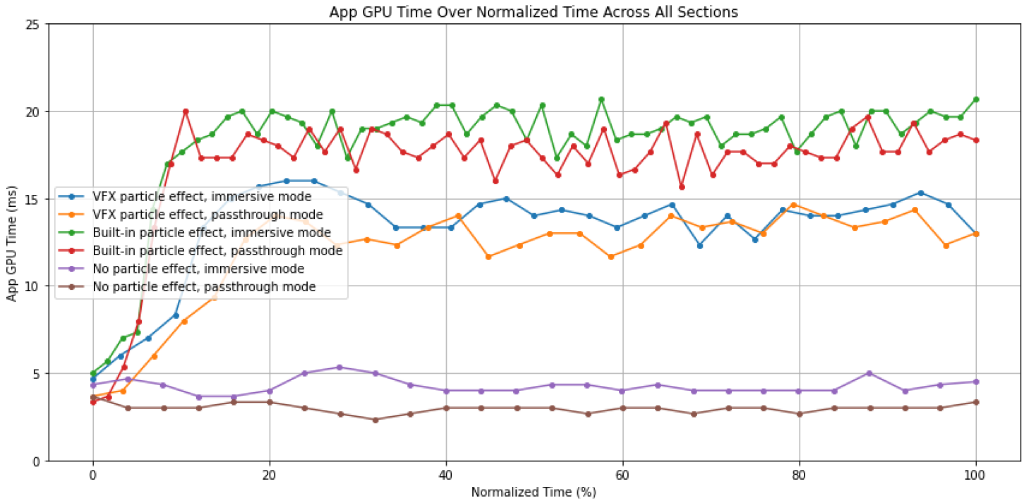

Figure 37. App GPU time in milliseconds before the optimization

Figure 38 illustrates the App GPU Time following the optimizations. There is a significant reduction in the GPU time across all configurations, with values largely stabilizing below the 11.11 ms threshold. This improvement indicates a shift away from being strictly GPU-bound, suggesting that the optimizations enhanced the application's ability to manage and render complex graphics more efficiently. Maintaining GPU time below the threshold is critical for high performance, ensuring a smooth user experience in virtual reality environments. This reduction in App GPU Time helps maintain higher frame rates, avoiding motion sickness and providing a more responsive experience for users.

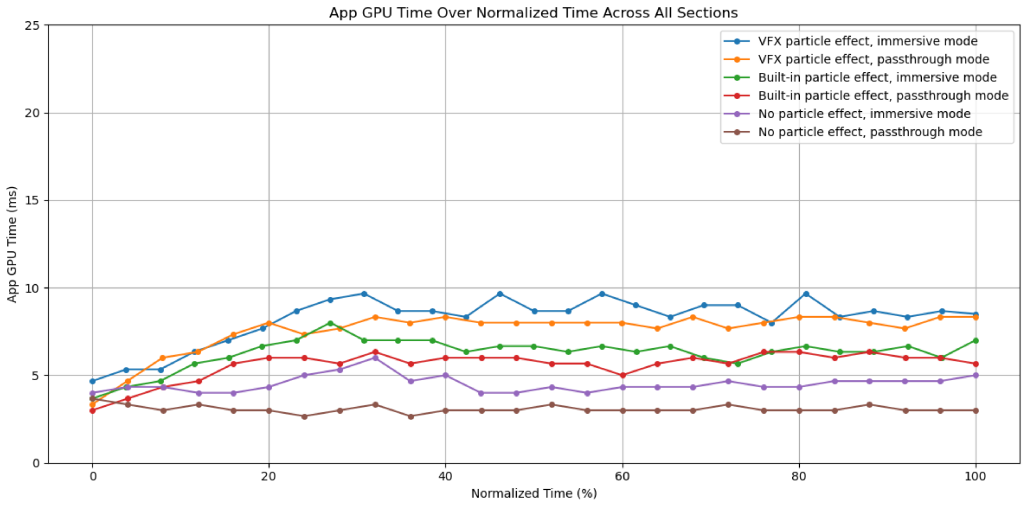

Figure 38. App GPU time in milliseconds after the optimization

5.4. Stale Frames

Stale frames are a metric for evaluating the user's quality of experience in virtual reality [19]. Because of the way XR applications work on the Meta Quest devices, the screen display refresh rate is not directly tied to the app's rendering of frames. Instead, there is an intermediate step called a compositor, also known as TimeWarp, which takes the last app-rendered frame, calculates orientation correction to account for user head movement, and presents it on the physical display. If the app-rendered frame is not ready in time, TimeWarp reuses the previous frame, which may be outdated or "stale." This can result in perceptible stutter or lag in visual rendering, detracting from the user experience.

As shown in Figure 39, certain configurations, particularly those with VFX particles in immersive mode, exhibited a sharp increase in stale frames. This increase indicates that frames were not being rendered in time, causing TimeWarp to reuse old frames. Despite maintaining an acceptable frame rate, the high number of stale frames resulted in noticeable stutter and lag in the visual output, significantly impacting the user experience. The prevalence of stale frames suggests inefficiencies in the frame rendering pipeline, leading to a higher latency between rendering and display.

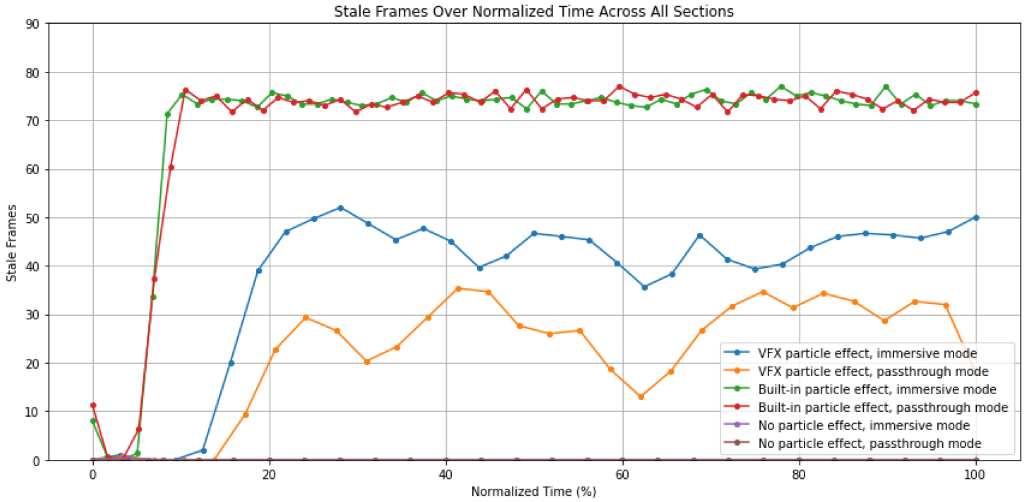

Figure 39. Stale frames before the optimization

After optimizations were applied, as shown in Figure 40, there was a drastic reduction in the occurrence of stale frames across all configurations. The frame presentation aligned more consistently, with fewer instances of frames being presented late. This improvement suggests that the frame rendering pipeline became more efficient, reducing the latency between frame rendering and display. The reduction in stale frames enhances the smoothness and responsiveness of the XR experience, which is crucial for maintaining visual coherence and minimizing latency. Such enhancements are vital for user comfort and immersion in XR applications.

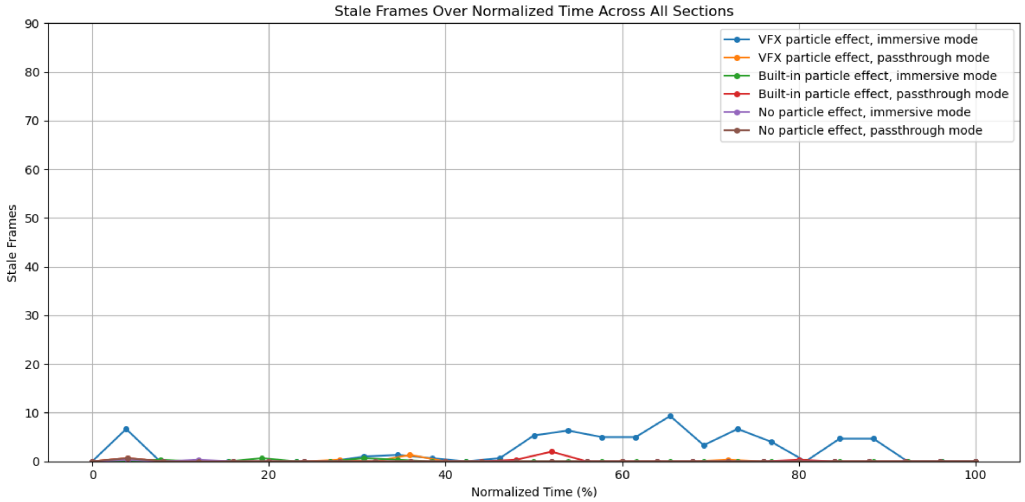

Figure 40. Stale frames before the optimization

6. DISCUSSION

This chapter provides further discussion of the research findings, addressing the research questions posed at the outset of the thesis. The results presented in the previous chapters are interpreted, exploring the implications of the performance differences observed between the built-in particle system and the Visual Effect Graph in XR environments. The impact of the automated testing framework and the polynomial regression analysis on optimizing particle effects is also considered. Furthermore, this chapter examines the broader significance of these findings for XR application development, offering recommendations for developers and highlighting potential areas for future research.

6.1. Summary of Research

This thesis investigated the creation and performance implications of particle systems in Unity3D, focusing on the built-in particle system and the VFX graph. A research prototype application was developed using Unity3D to explore these systems' performance in XR environments, including both immersive VR mode and passthrough AR mode. The study also involved the development of an automated testing framework to replicate user interactions and systematically measure performance metrics. Performance data was collected, processed, and analyzed to determine the optimal particle density that would not compromise the system's performance.

6.2. Analysis of Research Questions

RQ1: How do different particle system creation methods in Unity3D compare in terms of performance and efficiency in XR environments?

The comparative analysis of the built-in particle system and the VFX graph in Unity3D revealed distinct strengths and weaknesses for each system. The built-in particle system, while generally less demanding on processing power, showed limitations in handling high-density particle effects. Conversely, the VFX graph leveraged modern GPU capabilities more effectively, rendering complex particle effects with higher efficiency. For applications requiring high visual fidelity, the VFX graph proved to be superior due to its ability to handle more complex simulations without significantly compromising performance.

RQ1.1: What are the performance differences between the built-in particle system and the Visual Effect Graph when used in VR and AR environments?

The performance differences between the built-in particle system and the VFX graph were more pronounced in AR environments. In AR applications, where real-time interaction with the physical environment imposes additional computational loads, the VFX graph consistently maintained a higher fps compared to the built-in system. This finding underscores the VFX graph's superior handling of complex visual effects, especially in AR settings where computational demands are higher.

RQ2: How to implement testing methods to consistently replicate user interaction and measure the impact of visual effect adjustments on performance metrics in AR and VR?

The development of an automated testing framework using scripted interactions in Unity3D was a significant breakthrough in this research. This method allowed for precise control over the testing environment and facilitated robust comparisons of different particle systems under controlled conditions. By simulating natural human movements during testing, the framework provided a reliable dataset for assessing the impact of visual effects on system performance. The framework proved to be an efficient and cost-effective way to replicate user interactions consistently, ensuring that performance metrics were measured accurately and systematically.

RQ2.1: What metrics most effectively measure and compare the performance impact of different particle system methods in Unity3D under AR and VR conditions?

Several metrics emerged as particularly effective in evaluating the performance impacts of particle systems in XR environments. These included fps, GPU utilization, CPU utilization, and alive particles, i.e. the number of particles in the scene at that moment. Fps provided a direct measure of visual smoothness, while GPU and CPU utilization offered insights into the computational demands imposed by the particle systems. The number of alive particles was also a crucial indicator, particularly in understanding how particle density affected overall performance. Monitoring these metrics allowed for a comprehensive assessment of system performance in both immersive VR and passthrough AR settings.

RQ3: What recommendations can be made for developers choosing between particle system methods for their AR or VR projects?

Based on the collected and analyzed data, the recommendation would be to use the VFX graph for most XR projects. The built-in particle system, while less demanding on processing power, had limitations in handling high-density particle effects, which led to noticeable performance degradation. In contrast, the VFX graph utilized GPU resources more efficiently and supported a greater number of particles without significant performance issues. This makes the VFX graph a more suitable option for both AR and VR applications that require high visual fidelity and complex simulations.

However, it is important to consider specific project requirements. If there is a need for particles to interact with Unity3D's physics engine, the built-in particle system is the necessary choice, as the VFX graph does not support these physics interactions. For projects where physics interactions are not a primary concern, the VFX graph is recommended for its better performance and efficiency.

In conclusion, prioritizing the use of the VFX graph for most XR projects is suggested to leverage its enhanced performance capabilities. At the same time, careful optimization of the number of particles is essential to maintain performance within acceptable limits. Implementing adaptive systems that adjust particle density based on the the specific demands of AR or VR modes can help achieve this balance. By adopting these strategies, developers can enhance the visual quality of their applications while ensuring a smooth and immersive user experience in both VR and AR environments.

6.3. Key Findings and Recommendations

The automated testing framework developed in this thesis proved to be an invaluable tool for systematically evaluating the performance of particle systems. This framework provided consistent and reliable data, showcasing its potential for further refinement and adaptation in future research and development efforts. Importantly, using such an automated testing framework can be highly beneficial for students, independent game developers, and other individuals working on XR projects. The alternative—using robots to reproduce headset movements—is often prohibitively expensive. By offering an accessible and cost-effective solution, this framework can significantly aid those with limited resources in optimizing their XR applications.

This research provides actionable insights and practical recommendations for developers working with particle systems in XR environments. By understanding the performance implications of different particle system methods and utilizing effective testing frameworks, developers can optimize their applications to deliver high-quality user experiences in both VR and AR settings. The VFX graph, with its superior performance and efficiency, should be the preferred choice for all XR projects, with careful optimization to ensure the best possible performance.

6.4. Limitations

While the automated testing framework developed during this thesis advanced the ability to replicate human interaction consistently, it had some limitations that could have impacted the results. One limitation was not preventing interactions with the VR controllers during testing. Such interactions could inadvertently affect performance metrics, potentially introducing noise into the data that was not accounted for in the analysis. This should be fairly easy to remedy, by turning off tracking of the controllers and disabling them for the duration of the tests. Additionally, the testing environment was not completely controlled in terms of external conditions such as lighting, which might have influenced the results, especially in AR scenarios where passthrough mode is sensitive to environmental variables. The lack of control over these conditions introduces a variable that was not quantified in this study and could affect the reproducibility and reliability of the findings. It should also be taken into consideration, that recording the data during the automated tests would introduce some overhead in resource utilization. Some of it was remedied by making any disk operations before and after the actual test cycle.

Another limitation was the absence of visual recordings during the tests. There were instances where performance metrics indicated a drop, but without visual data, it was challenging to ascertain the exact cause of these drops. Incorporating video or photographic evidence during testing could have provided valuable insights into the specific environmental or operational factors affecting performance at those times, though it should be noted that this would also introduce further overhead in resource utilization.

There were some issues with using the same texture or color with both particle effects. In the initial tests, when ever a solid color was used with the VFX graph particle effect, a passthrough effect was observed in the leaves. This passthrough effect

can be seen in the Figure 46 in the Appendix 2. When the texture was used in the built-in particle effect, similar passthrough effect was observed. For this reason, the VFX particle effect used a texture, while the built-in particle effect used solid color for the particles.

The data analysis was conducted separately for logcat data and in-app gathered data. This separation might have limited the depth of insights that were achieved. Integrating these data sources could potentially reveal more comprehensive interactions between application performance and system operations, providing a richer understanding of the dynamics at play. The dataset used for optimizations also had its limitations. While the results were impressive, achieving a consistent 90 fps with the number of particle suggestions derived from the polynomial regression, it remains uncertain if the performance could have been further improved. The upper limits were reached in terms of particle density, but additional tests with increased particle counts could have provided more data, potentially yielding more accurate results. Since the automated testing framework facilitates easy testing and data gathering, conducting these additional tests would not pose a significant challenge. Additional tests were conducted using the same dataset to predict the number of particles that would result in 75 fps, a limit that is still acceptable to Meta for commercial interactive applications [16]. The results of these tests were less satisfactory, suggesting that the dataset might not be as effective for this type of prediction. This indicates that the dataset's limitations could influence the generalizability and precision of the optimization outcomes.

Addressing these limitations in future work could enhance the accuracy and applicability of the findings. It would also refine the automated testing process, making it more robust against external interferences and capable of providing more detailed diagnostic information. These improvements would be invaluable for developers looking to optimize particle effects in XR environments efficiently.

6.5. Future Work

There is significant potential for further development, particularly in the automated testing framework, which could be integrated into any XR application development and testing cycle to enhance continuous performance evaluation and optimization. By incorporating more advanced predictive models and data analysis techniques, the framework could also be improved to provide deeper insights into XR application performance. The following sections discuss specific ideas for enhancing the predictive models and data analysis within the testing application, offering a roadmap for future enhancements and research directions.

6.5.1. Enhancing the Predictive Models

In the pursuit of further enhancing the predictive accuracy and utility of performance optimization models for mixed reality applications, future research could expand the scope of the current model to incorporate multiple influencing factors. While the initial model focused primarily on the relationship between number of particles and

fps, the performance in VR and AR environments is likely influenced by a number of different factors including GPU load, CPU usage, memory usage, and network latency. A multivariate regression approach could provide a more holistic view of how these variables interactively affect XR application performance. It would allow for the analysis of how multiple predictors together influence the dependent variable, such as fps, which could lead to more robust and generalizable optimization strategies. For instance, understanding the relative impact of GPU and CPU utilization alongside particle effects could guide developers in balancing resource allocation, potentially leading to more effective performance optimizations.

Furthermore, techniques such as Principal Component Analysis (PCA) [47] could be employed to identify which variables contribute most significantly to the variance in fps. PCA is a powerful statistical technique that transforms a large set of variables into a smaller one that still contains most of the information in the large set. By applying PCA in this context, it would be possible to reduce the dimensionality of the problem, simplifying the model without significant loss of information. This could be particularly useful in distinguishing primary from secondary influences on fps, thereby directing optimization efforts more efficiently.

Such advanced analytical approaches would not only refine the current predictive models but also enhance the decision-making process in the development and optimization of mixed reality applications. By systematically exploring the interplay of multiple variables, developers can gain a more nuanced understanding of performance bottlenecks and opportunities for enhancement, ultimately leading to smoother and more immersive user experiences in VR and AR settings.

6.5.2. Data Analysis within the Testing Application

The integration of data analysis and model fitting directly within the Unity3D testing application could significantly advance the development pipeline for XR applications. Currently, data collection is already implemented within the installed application; extending this to include in-app data analysis and model fitting could vastly improve efficiency by enabling real-time adjustments and optimizations.

Incorporating machine learning libraries compatible with Unity3D, such as ML-Agents [48] or external C# compatible libraries, could allow developers to perform data analysis directly within the application. This approach would facilitate the immediate use of collected data to fit models that predict fps based on various parameters like particle count. By automating this process, the application could dynamically adjust settings in real-time based on predictive models, optimizing performance without the need to first download the data into an external program for processing and analysis.

Further, integrating an automated testing and refitting routine within the testing application would allow continuous improvement of the predictive models. As new data are collected under varying conditions and with different user interactions, the model could be retrained or refined to better predict fps outcomes, adapting to new insights or changes in application dynamics. This could be achieved by setting up a feedback loop within the application where performance data is constantly evaluated against model predictions, and discrepancies (where actual fps deviates significantly from predicted fps) trigger a retraining of the model.

This type of automated, in-app analytic capability would not only speed up the optimization process but also enhance the adaptability of the application. Developers could implement a system where, based on ongoing user interactions and system performance feedback, the application self-adjusts to maintain optimal performance. This approach aligns with the principles of adaptive systems in software engineering, where applications dynamically adjust to the environment and usage patterns to improve user experience and system efficiency.

7. CONCLUSION

The findings of this study reveal significant differences in the performance capabilities of the built-in particle system and the VFX graph. The VFX graph demonstrated superior efficiency in handling complex particle effects, especially in scenarios demanding high graphical processing power. These results underscore the importance of choosing the appropriate particle system based on the specific requirements of XR projects, as the VFX graph offers considerable advantages in terms of performance and visual fidelity.

The study highlights the effectiveness of the automated testing framework developed for this thesis. This framework proved essential in replicating user interactions and consistently measuring the impact of visual effects adjustments on performance metrics. By automating the testing process, the framework provided a cost-effective and accessible alternative to traditional user testing methods, which is particularly relevant for independent developers and small companies with limited resources. The significant cost associated with using robots for automated testing of VR headsets further emphasizes the value of this framework. It streamlined the performance evaluation process in XR application development, allowing faster iterations in the testing and development cycles.

Moreover, the application of polynomial regression for performance optimization was validated through this study. The predictive models developed using polynomial regression provided a robust method for determining the maximum particle density that could be sustained while maintaining a high fps, crucial for ensuring a smooth and immersive user experience. The results offer practical guidance for developers on optimizing particle effects in XR environments, emphasizing the importance of balancing visual complexity with system performance.

In conclusion, this study provides valuable insights into the performance implications of different particle system methods in XR environments and offers a practical solution for evaluating and optimizing performance through automated testing. These findings can inform developers' decision-making processes and contribute to the advancement of XR application development practices. The methodologies and tools developed in this thesis, including the automated testing framework and the polynomial regression model, present significant potential for further research and application in the field of XR, paving the way for more efficient and immersive virtual and augmented reality experiences. Addressing the limitations identified in this study and expanding the dataset for optimization purposes could yield even more refined results, further enhancing the capabilities of XR applications.

8. REFERENCES

[1] Meta, Meta Quest 2: Immersive all-in-one VR headset | Meta Store. URL: https://www.meta.com/fi/en/quest/products/quest-2/.

[2] Meta, Meta Quest 3: New mixed reality VR headset – Shop now | Meta Store. URL: https://www.meta.com/fi/en/quest/quest-3/.

[3] Korhonen A. (2024), ArmiKorhonen/AutomatedPerformanceMeasurement Framework. URL: https://github.com/ArmiKorhonen/AutomatedPerformanceMeasurementFramework.

[4] Unity Technologies, Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. URL: https://unity.com/.

[5] Unity Technologies, Unity - Manual: Using the Built-in Particle System. URL: https://docs.unity3d.com/Manual/PartSysUsage.html.

[6] Unity Technologies, VFX Graph | Unity. URL: https://unity.com/visual-effect-graph.

[7] Bang A.L., Krogh P., Ludvigsen M. & Markussen T. (2012) The role of hypothesis in constructive design research. In: Proceedings of the art of research iv.

[8] Unity Technologies, Unity - Manual: Particle systems. URL: https://docs.unity3d.com/Manual/ParticleSystems.html.

[9] Cord Technologies, Inc., Frames Per Second (FPS) Definition | Encord. URL: https://encord.com/glossary/frames-per-second-fps-definition/.

[10] Adil M., Song H., Khan M.K., Farouk A. & Jin Z. (2024) 5G/6G-enabled metaverse technologies: Taxonomy, applications, and open security challenges with future research directions. Journal of Network and Computer Applications 223, p. 103828. URL: https://www.sciencedirect.com/science/article/pii/S1084804524000055.

[11] Meta, Use Passthrough on Meta Quest | Meta Store. URL: https://www.meta.com/en-gb/help/quest/articles/in-vr-experiences/oculus-features/passthrough/.

[12] Qamar U. & Raza M.S. (2023) Regression analysis. In: Data science concepts and techniques with applications, Springer International Publishing, Cham, pp. 313–352. URL: https://doi.org/10.1007/978-3-031-17442-1_10.

[13] LaValle S.M. (2023) Virtual Reality. Cambridge University Press, 1 ed. URL: https://www.cambridge.org/core/product/identifier/9781108182874/type/book.

[14] Berkman M.I. (2024) History of virtual reality. In: N. Lee (ed.) Encyclopedia of computer graphics and games, Springer International Publishing, Cham, pp. 873–881. URL: https://doi.org/10.1007/978-3-031-23161-2_169.

[15] Wang J., Shi R., Zheng W., Xie W., Kao D. & Liang H.N. (2023) Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality. IEEE Transactions on Visualization and Computer Graphics 29, pp. 2478–2488. URL: https://ieeexplore.ieee.org/document/10049694/.

[16] Testing and Performance Analysis | Oculus Developers. URL: https://developer.oculus.com/documentation/unity/unity-perf/.

[17] Nusrat F., Hassan F., Zhong H. & Wang X. (2021) How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, Madrid, ES, pp. 473–485. URL: https://ieeexplore.ieee.org/document/9402052/.

[18] Brown R., Compare all VR Headsets. URL: https://vr-compare.com/vr.

[19] Dasch T., OVR Metrics Tool + VrApi: What Do These Metrics Mean? URL: https://developer.oculus.com/blog/ovr-metrics-tool-vrapi-what-do-these-metrics-mean.

[20] Unity Technologies, Mixed Reality Template Quick Start Guide | Mixed Reality | 1.0.1. URL: https://docs.unity3d.com/Packages/com.unity.template.mixed-reality@1.0/manual/index.html.

[21] Unity Technologies, XR Interaction Toolkit | XR Interaction Toolkit | 3.0.3. URL: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@3.0/manual/index.html.

[22] Midjourney, Midjourney. URL: https://www.midjourney.com/home?callbackUrl=%2Fexplore.

[23] Adobe, Substance 3D Assets Homepage. URL: https://substance3d.adobe.com/assets.

[24] polyperfect, Low Poly Ultimate Pack | 3D Props | Unity Asset Store. URL: https://assetstore.unity.com/packages/3d/props/low-poly-ultimate-pack-54733.

[25] Road Turtle Games, Nebula Skyboxes | 2D Sky | Unity Asset Store. URL: https://assetstore.unity.com/packages/2d/textures-materials/sky/nebula-skyboxes-219924.

[26] Unity Technologies, Unity Asset Store - The Best Assets for Game Making. URL: https://assetstore.unity.com/.

[27] Kluyver T., Ragan-Kelley B., Pérez F., Granger B., Bussonnier M., Frederic J., Kelley K., Hamrick J., Grout J., Corlay S., Ivanov P., Avila D., Abdalla S. & Willing C. (2016) Jupyter Notebooks – a publishing format for reproducible computational workflows. In: F. Loizides & B. Schmidt (eds.) Positioning and power in academic publishing: Players, agents and agendas, IOS Press, pp. 87 – 90.

[28] team T.p.d. (2024), pandas-dev/pandas: Pandas. URL: https://doi.org/10.5281/zenodo.10537285.

[29] Harris C.R., Millman K.J., van der Walt S.J., Gommers R., Virtanen P., Cournapeau D., Wieser E., Taylor J., Berg S., Smith N.J., Kern R., Picus M., Hoyer S., van Kerkwijk M.H., Brett M., Haldane A., del Río J.F., Wiebe M., Peterson P., Gérard-Marchant P., Sheppard K., Reddy T., Weckesser W., Abbasi H., Gohlke C. & Oliphant T.E. (2020) Array programming with NumPy. Nature 585, pp. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2, publisher: Springer Science and Business Media LLC.

[30] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M. & Duchesnay E. (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, pp. 2825–2830.

[31] The SciPy community, Optimization and root finding (scipy.optimize) — SciPy v1.13.1 Manual. URL: https://docs.scipy.org/doc/scipy/reference/optimize.html.

[32] Hunter J.D. (2007) Matplotlib: A 2D graphics environment. Computing in Science & Engineering 9, pp. 90–95. Publisher: IEEE COMPUTER SOC.

[33] Waskom M. (2021) seaborn: statistical data visualization. Journal of Open Source Software 6, p. 3021. URL: https://joss.theoj.org/papers/10.21105/joss.03021.

[34] Python Software Foundation, glob — Unix style pathname pattern expansion. URL: https://docs.python.org/3/library/glob.html.

[35] Python Software Foundation, os — Miscellaneous operating system interfaces. URL: https://docs.python.org/3/library/os.html.

[36] Python Software Foundation, re — Regular expression operations. URL: https://docs.python.org/3/library/re.html.

[37] Python Software Foundation, itertools — Functions creating iterators for efficient looping. URL: https://docs.python.org/3/library/itertools.html.

[38] Optofidelity, Automate AR/VR HMD testing with OptoFidelity BUDDY | OptoFidelity. URL: https://www.optofidelity.com/en/products/buddy.

[39] Apple Inc., Apple Vision Pro. URL: https://www.apple.com/apple-vision-pro/.

[40] Apple, Designing for visionOS. URL: https://developer.apple.com/design/human-interface-guidelines/designing-for-visionos/.

[41] Unity Technologies, Unity - Scripting API: Profiler. URL: https://docs.unity3d.com/ScriptReference/Profiling.Profiler.html.

[42] Meta, Collect Logs with Logcat: Native/android | Oculus Developers. URL: https://developer.oculus.com/documentation/native/android/ts-logcat/.

[43] Google, Download Android Studio & App Tools. URL: https://developer.android.com/studio.

[44] Google, Android | Do More With Google on Android Phones & Devices. URL: https://www.android.com/.

[45] Google, Android Debug Bridge (adb) | Android Studio. URL: https://developer.android.com/tools/adb.

[46] scikit-learn, PolynomialFeatures. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html.

[47] Jolliffe I. (2011) Principal component analysis. In: M. Lovric (ed.) International encyclopedia of statistical science, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1094–1096. URL: https://doi.org/10.1007/978-3-642-04898-2_455.

[48] Juliani A., Berges V.P., Teng E., Cohen A., Harper J., Elion C., Goy C., Gao Y., Henry H., Mattar M. & Lange D. (2020) Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627 URL: https://arxiv.org/pdf/1809.02627.pdf.

9. APPENDICES

Appendix 1 Plots and graphs
Appendix 2 Screenshots and images

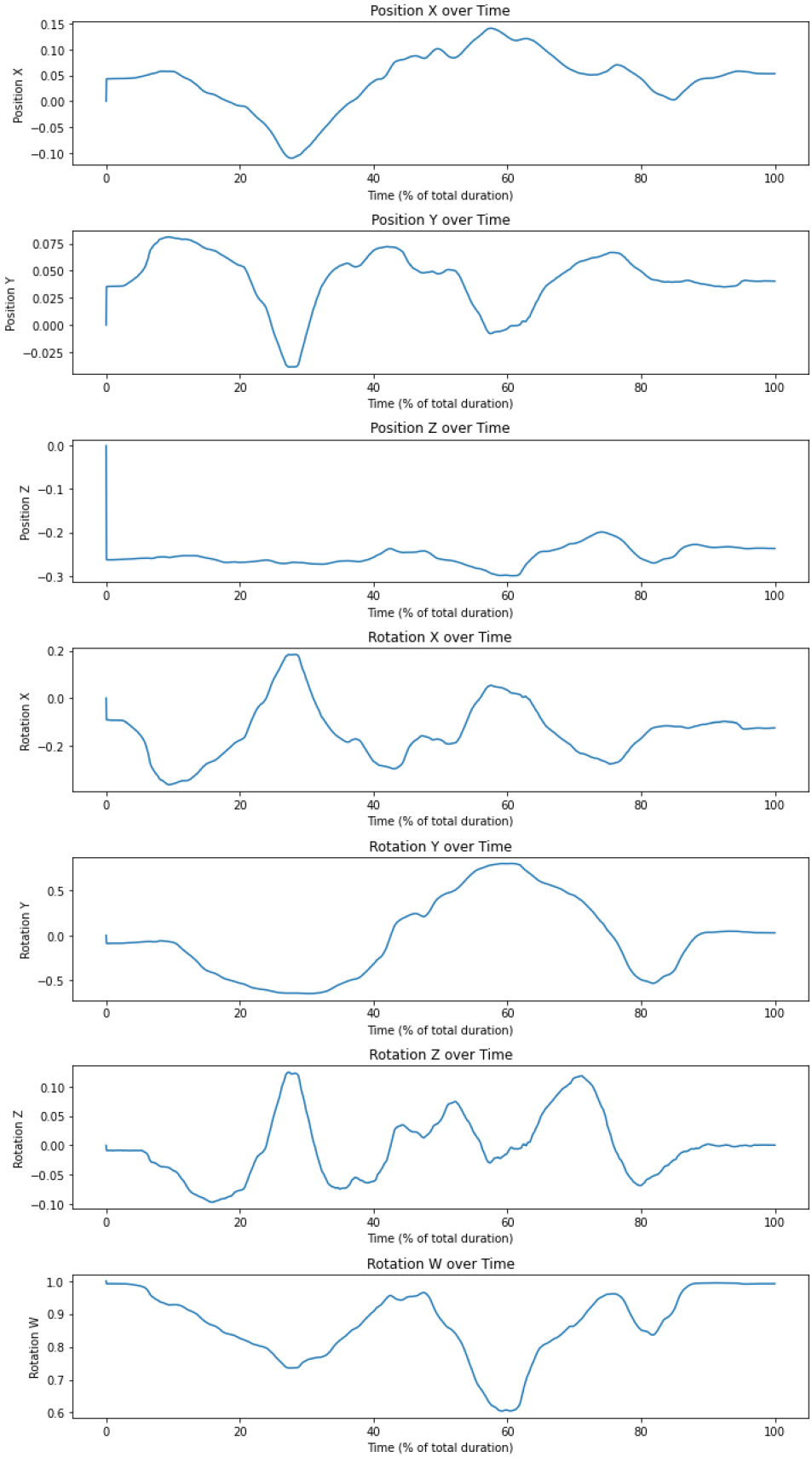

Figure 41. Movement data used in the automated tests

Figure 42. Screenshot from the testing application

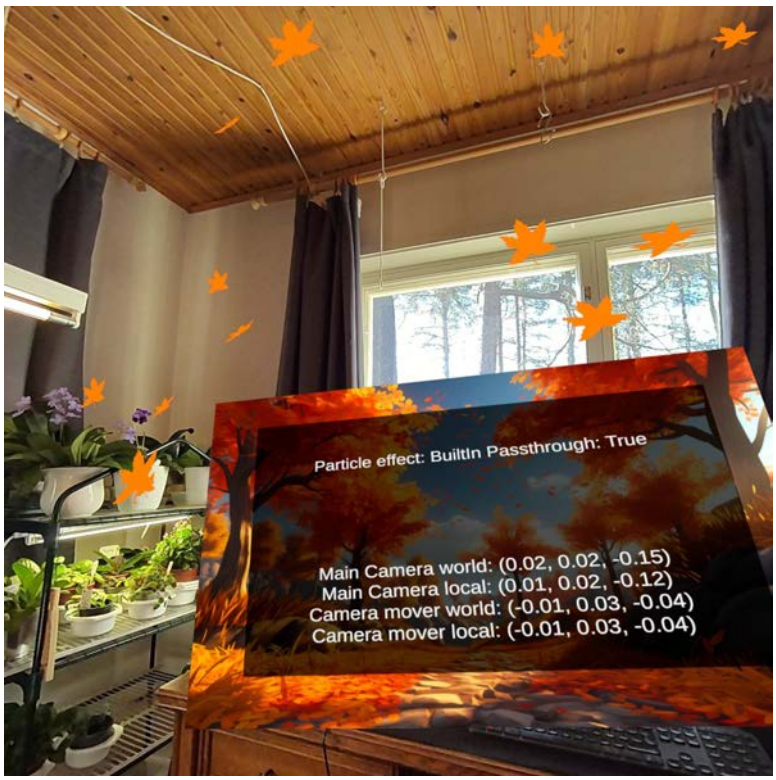

Figure 43. Screenshot from the testing application

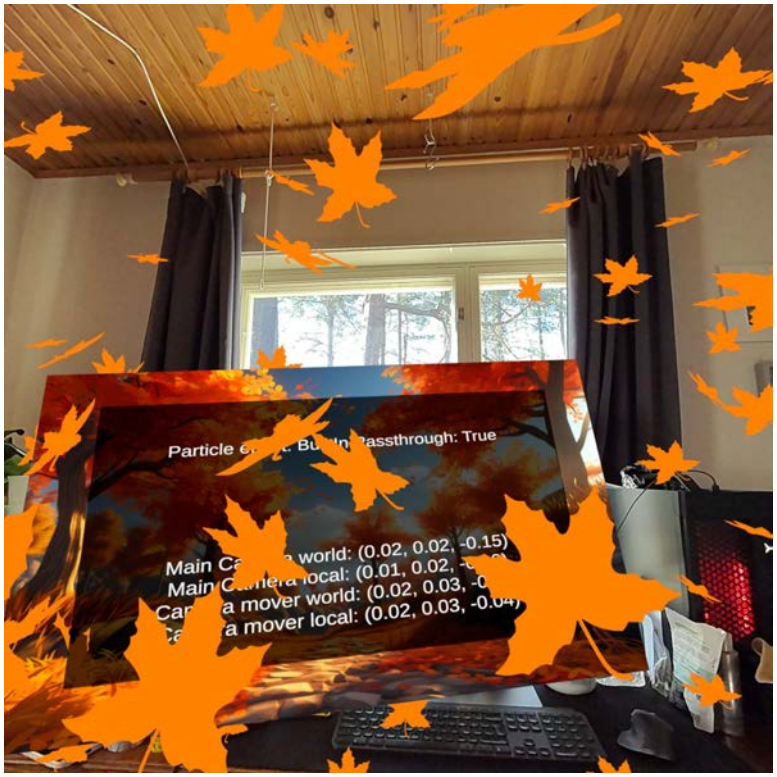

Figure 44. Screenshot from the testing application

Figure 45. Screenshot from the testing application

Figure 46. Passthrough effect in the VFX graph particles