Bachelor Thesis

Czech
Technical
University
in Prague

F3 Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Foveated Rendering

Ilia Timofeev

Supervisor: prof. Ing. Jiří Bittner, Ph.D.
Field of study: Open Informatics
Subfield: Computer Games and Graphics
January 2026

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Timofeev Ilia** Personal ID number: **511296**

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

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Foveated rendering

Bachelor’s thesis title in Czech:

Adaptivní zobrazování podle směru pohledu

Name and workplace of bachelor’s thesis supervisor:

doc. Ing. Jiří Bittner, Ph.D. Department of Computer Graphics and Interaction

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

Date of bachelor’s thesis assignment: **09.09.2025** Deadline for bachelor thesis submission: **06.01.2026**

Assignment valid until: **14.02.2027**

Head of department’s signature

Vice-dean´s signature on behalf of the Dean

III. Assignment receipt

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

Date of assignment receipt

Timofeev Ilia

Student’s signature

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Timofeev Ilia** Personal ID number: **511296**

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

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Specialisation: **Computer Games and Graphics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Foveated rendering

Bachelor’s thesis title in Czech:

Adaptivní zobrazování podle směru pohledu

Guidelines:

Review existing methods that use foveated rendering and gaze tracking in the context of virtual reality and video games. Study the support of this technology in the Unity, Unreal, and Godot game engines. Based on the devices available in the virtual reality lab, select a test platform and create a test application that evaluates the features and benefits of this technology. The test application will allow studying the effect of foveated rendering on quality and performance. The application will be implemented in at least two of the above-mentioned game engines. Evaluate the application thoroughly in terms of both subjectively perceived display quality and computational demands.

Bibliography / sources:

[1] Wang, L., Shi, X. and Liu, Y., 2023. Foveated rendering: A state-of-the-art survey. Computational Visual Media, 9(2), pp.195-228.
[2] Patney, A., Salvi, M., Kim, J., Kaplanyan, A., Wyman, C., Benty, N., Luebke, D. and Lefohn, A., 2016. Towards foveated rendering for gaze-tracked virtual reality. ACM Transactions on Graphics (TOG), 35(6), pp.1-12.
[3] Mohanto, B., Islam, A.T., Gobbetti, E. and Staadt, O., 2022. An integrative view of foveated rendering. Computers & Graphics, 102, pp.474-501.
[4] Andersson, K. and Landén, E., 2023. The Impact of Foveated Rendering as Used for Head-Mounted Displays in Interactive Video Games.

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

DECLARATION

I, the undersigned

Student's surname, given name(s):	Timofeev Ilia
Personal number:	511296
Programme name:	Open Informatics

declare that I have elaborated the bachelor's thesis entitled

Foveated rendering

independently, and have cited all information sources used in accordance with the Methodological Instruction on the Observance of Ethical Principles in the Preparation of University Theses and with the Framework Rules for the Use of Artificial Intelligence at CTU for Academic and Pedagogical Purposes in Bachelor's and Continuing Master's Programmes.

I declare that I used artificial intelligence tools during the preparation and writing of this thesis. I verified the generated content. I hereby confirm that I am aware of the fact that I am fully responsible for the contents of the thesis.

In Prague on 22.12.2025

Ilia Timofeev

.....

student's signature

Acknowledgements

I would like to thank my supervisor, prof. Ing. Jiří Bittner, Ph.D., for all his help and advice during my thesis work. He always gave me clear guidance and useful feedback, even when things got difficult. I appreciate his patience and support, which helped me learn a lot and finish this thesis. I am sincerely grateful for his time and encouragement, which have been indispensable to the successful completion of this work.

I would also like to thank Ing. David Sedláček, Ph.D., for providing me with access to the laboratory where I could work, and for lending me the hardware required for this thesis. I am grateful for his helpful approach, kindness, and willingness to support me whenever I needed assistance.

I would like to express my sincere thanks to my family for their support and encouragement throughout my studies and during the preparation of this thesis.

Finally, I would like to thank Bc. Daria Mikhaylovskaya for being by my side during the entire time I was writing this work, and for her continuous support, patience, and motivation.

Declaration

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

I declare that I used AI tools to assist with translating specific sentences from my native language to English and to help rephrase my work in a style more suitable for academic work.

Prague, January 5, 2026

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

Prohlašuji, že jsem použil AI nástroje k pomoci s překladem konkrétních vět z mého rodného jazyka do angličtiny a k přeformulování mé práce do stylu vhodnějšího pro akademickou práci.

V Praze dne 5. ledna 2026

.....
Ilia Timofeev

Abstract

The aim of this thesis is to examine the benefits of foveated rendering in real-time applications and virtual reality. Custom implementations were developed in Unity and Unreal Engine using Variable Rate Shading for the PC platform, while the VR versions relied on official Meta plugins that provide built-in foveated rendering support. Performance was tested on both PC and VR, and a user study with 15 participants evaluated visual clarity as well as perceived performance during interactive tasks. The results indicate that foveated rendering substantially improves efficiency while preserving user-acceptable visual quality.

Keywords: foveated rendering, eye tracking, Gazepoint GP3, virtual reality, VRS, Unity, Unreal Engine, performance testing, user study

Supervisor: prof. Ing. Jiří Bittner, Ph.D.
Karlovo nám. 13,
121 35
Praha 2

Abstrakt

Cílem této práce je prozkoumat přínosy foveated renderingu v reálném čase a ve virtuální realitě. Pro platformu PC byly v Unity a Unreal Engine vytvořeny vlastní implementace využívající Variable Rate Shading, zatímco verze pro VR využívaly oficiální pluginy od společnosti Meta s integrovanou podporou foveated renderingu. Výkon byl testován jak na PC, tak ve VR a uživatelská studie s 15 účastníky hodnotila vizuální ostrost i vnímaný výkon během interaktivních úloh. Výsledky ukazují, že foveated rendering výrazně zlepšuje efektivitu při zachování přijatelné úrovně vizuální kvality.

Klíčová slova: foveated rendering, sledování očí, Gazepoint GP3, virtuální realita, VRS, Unity, Unreal Engine, testování výkonu, uživatelská studie

Překlad názvu: Adaptivní zobrazování podle směru pohledu

Contents

Figures

Tables

Chapter 1

Introduction

This chapter introduces the main motivation behind the thesis and explains why foveated rendering is a promising approach for real-time graphics. It formulates the research questions that guide the work and clarifies the scope of the evaluation across Unity and Unreal Engine on both PC and VR platforms. Finally, it outlines the overall structure of the thesis and briefly describes the content of the subsequent chapters.

1.1 Motivation

During my regular bus commute to the university, I noticed strong reflections on the bodywork of passing cars. When I looked directly at one reflection, fine highlights and contours were clearly visible; but as soon as I shifted my gaze to another vehicle, those details vanished in my peripheral vision [10]. This phenomenon aligns with established findings in vision science: the human visual system is capable of resolving fine, high-frequency structure only within the foveal region, while peripheral vision rapidly loses spatial detail [1].

This phenomenon raises a fundamental research question: if the visual system cannot resolve high-frequency detail beyond the fovea, why do conventional rendering pipelines dedicate equal computational effort to every pixel of each frame? Such uniform processing appears inefficient and suggests the potential for significant performance gains through selective image adaptation.

Accordingly, this thesis investigates **Foveated Rendering**, a framework in which image quality is dynamically adjusted according to the viewer's gaze. By maintaining high resolution within the foveal area and reducing sampling density in peripheral regions, it is possible to decrease the workload on the Graphics Processing Unit (GPU) without compromising perceived image fidelity. The objectives of this work are to investigate gaze-contingent quality-allocation strategies, quantify their impact on rendering performance, and evaluate their effectiveness under realistic viewing conditions.

1.2 Goals and Research Questions

The primary goal of this work is to systematically investigate the performance and perceptual impact of foveated rendering across two major real-time engines on both desktop and VR platforms.

This goal is supported by the following research questions:

- **RQ1**: How do different foveated-rendering configurations (FFR and ETFR) affect rendering performance across Unity and Unreal Engine on PC and VR systems?
- **RQ2**: What performance gains can be achieved using image-based VRS on desktop platforms, and how do they compare to integrated foveated-rendering solutions available on standalone VR devices?
- **RQ3**: How do users perceive visual quality under different foveation levels, and which configurations provide the best balance between clarity, stability, and frame rate?
- **RQ4**: What engineering differences exist between the Unity and Unreal Engine implementation pipelines, particularly regarding shading-rate integration and gaze-driven rendering?

These questions guide the experimental design and structure of the practical implementation.

1.3 Thesis Structure

This thesis is organized as follows:

- **Chapter 1 – Introduction**:
  Introduces the motivation, objectives, and structure of the thesis.
- **Chapter 2 – Related Work**:
  Reviews theoretical foundations and prior academic and industrial work on fixed and eye-tracked foveated rendering.
- **Chapter 3 – Methodology and Implementation**:
  Describes the implementation of foveated rendering techniques in Unity and Unreal Engine, including the used hardware and tools.
- **Chapter 4 – Experimental Methodology**:
  Presents the design of performance benchmarks and the user study.
- **Chapter 5 – Results**:
  Reports quantitative performance results and qualitative findings from the user evaluation.

- **Chapter 6 – Discussion**:
  Interprets the performance and user-study results, compares the observed trends across scenes and techniques, and discusses practical implications and limitations of the evaluation.
- **Chapter 7 – Conclusion and Future Work**:
  Summarizes the main contributions and key findings, answers the research question, and outlines directions for future improvements and further research.

Chapter 2
Related Work

This chapter reviews existing academic and industrial work related to foveated rendering. It begins by introducing the necessary background and theoretical context, including fundamental aspects of human visual perception, rendering pipelines, and the general principles behind foveated rendering techniques. The chapter then surveys prior research on fixed and eye-tracked foveated rendering across desktop and virtual reality platforms, followed by an overview of practical implementations in modern game engines and graphics APIs. Finally, the chapter summarizes the limitations of existing approaches and identifies research gaps that motivate the focus of this thesis.

2.1 Background and Theoretical Context

Foveated rendering is motivated by fundamental properties of the human visual system, in which visual acuity is highly non-uniform across the visual field (see Fig. 2.1). The region of highest acuity, the fovea, occupies only a small central portion of vision, while acuity rapidly decreases toward the parafoveal and peripheral regions (see Fig. 2.2). Numerous studies have shown that fine spatial detail is largely imperceptible outside the fovea, especially in the context of wide field-of-view displays such as head-mounted displays (HMDs) [1, 11]. This characteristic enables rendering systems to allocate computational resources non-uniformly without noticeable degradation in perceived image quality.

Traditional real-time rendering pipelines treat all pixels of a frame equally, regardless of their perceptual relevance. In rasterization-based pipelines, geometry processing, visibility determination, shading, and post-processing are typically executed at a uniform resolution across the entire image. In virtual reality, this approach becomes increasingly inefficient due to high display resolutions, stereo rendering, and strict latency requirements [12]. As a result, a significant portion of rendering effort is spent on regions that contribute little to perceived visual fidelity.

Foveated rendering techniques exploit the acuity fall-off of human vision by reducing rendering quality as a function of retinal eccentricity (see Fig. 2.3). These techniques can be broadly classified into fixed foveated rendering (FFR) and eye-tracked foveated rendering (ETFR). Fixed foveated rendering

Figure 2.1: Illustration of peripheral acuity loss: (a) original image; (b) image blurred to approximate loss of peripheral acuity; (c) exaggerated demonstration of peripheral blur, commonly used for illustrative purposes [1].

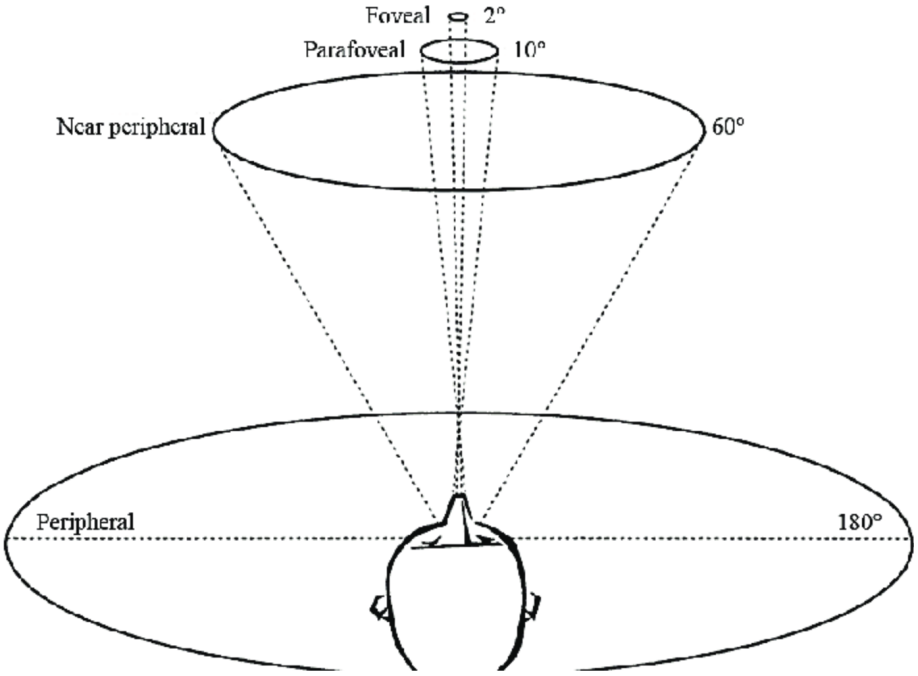


Figure 2.2: Schematic illustration of foveal and peripheral vision [2].

assumes that the user primarily looks toward the center of the display and applies a static foveation pattern, which is commonly used on standalone VR devices due to its simplicity and low hardware requirements [2]. However, this assumption limits perceptual accuracy when the user's gaze deviates from the center.

Eye-tracked foveated rendering dynamically adapts the high-quality region to the user's gaze, requiring real-time eye tracking with sufficiently low latency. Prior work has shown that latency, temporal stability, and contrast preservation are critical factors influencing the perceptual acceptability of gaze-contingent rendering [13]. In particular, peripheral regions are sensitive to temporal artifacts such as flickering and aliasing, which can significantly reduce immersion if not properly addressed.

From an implementation perspective, foveated rendering can be realized using image-based techniques or hardware-supported mechanisms. Image-

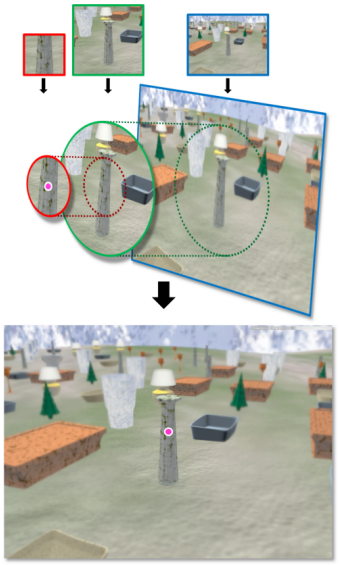


Figure 2.3: 3 stages of rendering quality. The red circle shows the high-quality part and the green circle is a transition layer where it is rendered in a bit lower quality. The blue part is the low-quality rendered zone [3].

based approaches typically operate as post-processing steps or custom shading strategies, while modern graphics APIs such as DirectX 12[14] and Vulkan[15] provide hardware-level support through Variable Rate Shading (VRS)[16]. VRS allows the shading rate to vary across the screen, enabling efficient foveation with minimal changes to existing rendering pipelines [5]. Due to its practical availability and engine support, VRS has become a dominant approach in contemporary real-time applications.

Overall, prior research establishes a strong perceptual and technical foundation for foveated rendering, while also highlighting challenges related to latency, temporal stability, and practical integration into real-world engines. These considerations form the theoretical basis for the comparative evaluation of foveated rendering techniques conducted in this thesis.

2.2 Academic Approaches to Foveated Rendering

Academic research on foveated rendering has primarily focused on exploiting the non-uniform resolution of the human visual system to reduce rendering cost while preserving perceived image quality. Early foundational work by Guenter et al. [12] introduced the concept of foveated graphics by dynamically varying rendering quality across the visual field, demonstrating that significant performance gains can be achieved without noticeable visual degradation when peripheral regions are rendered at lower fidelity.

From a technical standpoint, many academic approaches model foveation as a function of retinal eccentricity, defined as the angular distance between a pixel and the current gaze position. Let θ denote the retinal eccentricity.

Rendering quality or shading rate $q(\theta)$ is typically defined as a monotonically decreasing function of θ, for example:

$$q(\theta) = \begin{cases} q_{\text{high}}, & \theta \leq \theta_f \\ q_{\text{mid}}, & \theta_f < \theta \leq \theta_p \\ q_{\text{low}}, & \theta > \theta_p \end{cases}$$

where θ_f corresponds to the foveal region and θ_p defines the boundary of the parafoveal region. Such discrete region-based models are commonly used in fixed foveated rendering and early gaze-contingent systems.

Subsequent studies extended this idea toward perceptually driven rendering strategies. Patney et al. [11] proposed gaze-contingent foveated rendering for virtual reality systems and systematically analyzed perceptual constraints such as eye-tracking latency, transition smoothness, and contrast sensitivity. Their results showed that properly tuned eye-tracked foveation can remain perceptually indistinguishable from full-resolution rendering under realistic latency bounds, establishing key design guidelines that are still referenced in contemporary work.

In gaze-contingent approaches, the gaze position $\mathbf{g} = (x_g, y_g)$ is updated every frame, and retinal eccentricity for each pixel $\mathbf{p} = (x, y)$ is computed as:

$$\theta(\mathbf{p}) = \arctan\left(\frac{\|\mathbf{p} - \mathbf{g}\|}{d}\right),$$

where d denotes the eye-to-screen distance or effective viewing distance. To reduce visible artifacts at region boundaries, later work replaces hard thresholds with smooth falloff functions, typically implemented as spatial blending between adjacent eccentricity regions using Gaussian or sigmoid weighting functions.

Beyond rasterization-based pipelines, academic research has also explored foveated rendering in the context of ray tracing. Weier et al. [17] investigated foveated real-time ray tracing for head-mounted displays and demonstrated that users are less sensitive to rendering artifacts when their attention is focused on dynamic targets rather than static fixation points. Similar findings were reported by Roth et al. [18], who analyzed eye-tracking data and emphasized the role of visual tunneling effects during active viewing.

In ray tracing pipelines, foveation is commonly implemented by varying the number of rays per pixel (RPP) as a function of eccentricity:

$$\text{RPP}(\theta) = \text{RPP}_{\max} \cdot e^{-k\theta},$$

where $\text{RPP}_{\max}$ is the maximum sampling rate in the foveal region and k controls the rate at which sampling density decreases toward the periphery. This strategy directly reduces traversal and shading cost while maintaining high-quality illumination near the gaze point.

More recent work has focused on improving the practical applicability of foveated techniques. Meng et al. [19] introduced Kernel Foveated Rendering (KFR), which smoothly interpolates rendering quality across eccentricity

levels and achieves substantial performance gains in both rasterization and ray tracing pipelines. Comprehensive surveys further consolidate these findings, categorizing foveated rendering techniques by perceptual motivation, implementation strategy, and target platform [2].

Overall, academic approaches consistently demonstrate that foveated rendering can significantly reduce computational cost while maintaining visual quality, provided that perceptual constraints such as latency and temporal stability are respected. However, many studies focus on isolated rendering techniques or controlled experimental setups, leaving open questions regarding comparative evaluation across engines, platforms, and practical development workflows.

2.3 Fixed and Eye-Tracked Foveated Rendering

Foveated rendering techniques are commonly divided into fixed foveated rendering (FFR) and eye-tracked foveated rendering (ETFR), differing primarily in how the high-quality region is positioned within the image. Fixed foveated rendering applies a static quality distribution, typically centered on the display, under the assumption that users predominantly focus near the center of the visual field (see Fig. 2.4). This approach is widely used in standalone VR systems due to its simplicity and the absence of eye-tracking hardware requirements [3, 19].

From a formal perspective, the difference between FFR and ETFR can be expressed through the definition of the foveal center $\mathbf{c}$:

$$\mathbf{c} = \begin{cases} \mathbf{c}_0, & \text{FFR} \\ \mathbf{g}(t), & \text{ETFR} \end{cases}$$

where $\mathbf{c}_0$ denotes a fixed screen-centered position and $\mathbf{g}(t)$ represents the time-varying gaze position obtained from an eye tracker.

While FFR offers a favorable performance-to-complexity ratio, its static nature introduces perceptual limitations when the user's gaze deviates from the center. Several studies report that larger foveal regions are required to mask these deviations, which in turn reduces potential performance gains [2]. Nevertheless, user studies indicate that static foveation is often perceived as visually acceptable in dynamic gameplay scenarios, particularly when peripheral degradation coincides with lens distortions or natural attention focus [3].

This trade-off can be expressed by the required radius of the foveal region:

$$r_f^{\text{FFR}} = r_f^{\text{ETFR}} + \Delta r_{\text{gaze}},$$

where r_f denotes the foveal radius and Δr_{gaze} accounts for potential gaze deviations away from the screen center.

Eye-tracked foveated rendering dynamically aligns the high-quality region with the user's gaze, enabling a substantially smaller foveal area and greater computational savings (see Fig. 2.5). Academic work consistently

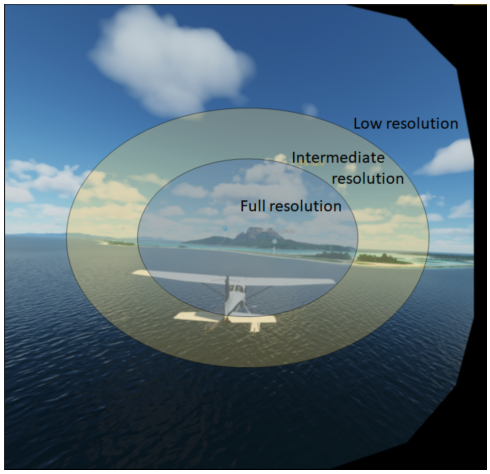


Figure 2.4: The image is divided into 3 regions, with the resolution decreasing the further we get from the center region [4].

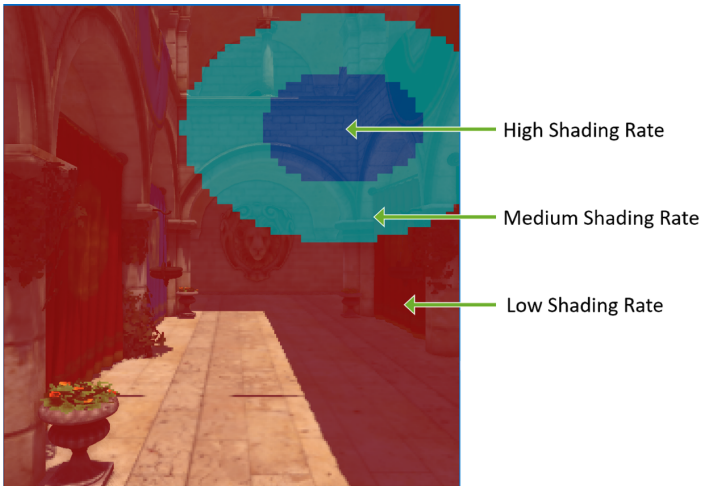


Figure 2.5: Shading rate regions dynamically aligned with the gaze position in eye-tracked foveated rendering [5].

demonstrates that ETFR can achieve superior quality–performance trade-offs compared to FFR, provided that system latency remains within perceptually acceptable bounds [11, 13]. However, ETFR introduces additional technical challenges, including eye-tracker accuracy, calibration stability, and temporal artifacts caused by delayed gaze updates.

Perceptual correctness of ETFR can be formulated as a latency constraint:

$$\Delta t_{\text{system}} \leq \Delta t_{\text{perceptual}},$$

where Δt_{system} represents the combined latency of eye tracking, rendering, and display, and $\Delta t_{\text{perceptual}}$ denotes the maximum latency tolerated by the human visual system without noticeable artifacts.

Comparative studies highlight that user sensitivity to ETFR artifacts strongly depends on task context and motion dynamics. When users ac-

tively track moving targets, awareness of foveation boundaries is significantly reduced due to attentional and perceptual effects such as visual tunneling [17, 18]. In contrast, static scenes or explicit artifact-search tasks tend to increase detectability, potentially biasing experimental outcomes.

Overall, prior work suggests that FFR remains a robust and accessible solution for current VR hardware, while ETFR offers higher theoretical efficiency at the cost of increased system complexity. The trade-offs between these approaches motivate further evaluation in practical engine-based workflows and real-world application scenarios.

2.4 Platform-Specific Studies (VR and Desktop)

Foveated rendering has been studied across a range of hardware platforms, with notable differences between standalone VR systems and desktop-based rendering environments. These platform-specific constraints strongly influence the choice of foveation technique, achievable performance gains, and perceptual outcomes.

Standalone VR devices, such as mobile-based head-mounted displays, are typically constrained by limited computational power, memory bandwidth, and thermal budgets. As a result, research on standalone platforms has largely focused on fixed foveated rendering, where a static quality distribution is applied to reduce fragment shading cost without requiring eye tracking [19]. Several studies report that this approach provides reliable performance improvements while remaining perceptually acceptable for most users, particularly when combined with lens-induced peripheral distortions that naturally mask quality loss [3].

In contrast, desktop-based systems offer significantly higher performance headroom and greater flexibility in rendering pipelines, enabling more advanced foveated techniques. Academic work on desktop platforms often explores eye-tracked foveated rendering, leveraging external or integrated eye trackers to dynamically adapt rendering quality based on gaze direction [12]. These studies emphasize the importance of low end-to-end latency and stable gaze prediction to prevent visible artifacts, especially during rapid eye movements.

Comparative research indicates that while desktop systems can achieve higher absolute performance gains with eye-tracked foveation, the perceptual benefits over fixed foveation are not always proportional to the added system complexity [17]. In interactive scenarios involving rapid head and eye movements, users may exhibit reduced sensitivity to peripheral artifacts regardless of platform, suggesting that task context plays a critical role in perceived quality.

Overall, prior work highlights that platform constraints significantly shape both the technical feasibility and perceptual effectiveness of foveated rendering. While standalone devices favor robust and low-overhead approaches such as fixed foveation, desktop systems enable more aggressive eye-tracked techniques at the cost of increased implementation complexity.

2.5 Industry and Engine-Level Techniques

In recent years, foveated rendering has transitioned from primarily academic research into practical deployment within commercial game engines and real-time rendering frameworks. This shift has been driven by increasing support for perceptually motivated rendering techniques at both the engine and graphics API levels, enabling developers to integrate foveation without redesigning entire rendering pipelines.

Modern game engines such as Unity [20] and Unreal Engine [21] provide built-in support for fixed and eye-tracked foveated rendering, particularly in virtual reality applications. These implementations typically rely on Variable Rate Shading (VRS)[16] or equivalent mechanisms to reduce fragment shading cost in peripheral regions while preserving full resolution in the foveal area. From an industry perspective, VRS has become a key enabling technology due to its compatibility with conventional rasterization pipelines and minimal impact on content authoring workflows.

Industry-oriented implementations often prioritize robustness and ease of integration over optimal perceptual tuning. Fixed foveated rendering is commonly favored as a default solution because it does not require eye tracking and behaves predictably across hardware configurations. Eye-tracked foveated rendering is increasingly supported in engines targeting high-end VR systems, but its adoption remains constrained by hardware availability, calibration requirements, and latency sensitivity, as highlighted in prior academic work [11].

Compared to academic prototypes, engine-level solutions expose limited control over foveation parameters such as region shape, transition smoothness, and temporal filtering. While this abstraction simplifies development, it also restricts experimentation with novel foveation strategies proposed in the literature, including perceptually optimized or kernel-based approaches [19]. As a result, many industry implementations represent practical compromises rather than direct realizations of state-of-the-art academic methods.

Overall, industry adoption of foveated rendering reflects a growing emphasis on deployable, hardware-supported techniques that balance performance gains with development cost and stability. This gap between theoretical optimality and practical usability motivates empirical evaluation of existing engine-level foveation solutions as pursued in this thesis.

2.6 Summary and Identified Research Gaps

Existing research on foveated rendering provides strong evidence that perceptually motivated quality reduction can significantly improve rendering efficiency while preserving visual quality. Academic studies have established foundational principles based on human visual perception, explored both fixed and eye-tracked approaches, and demonstrated their effectiveness across rasterization and ray tracing pipelines. User studies further indicate that

perceptual tolerance to foveation artifacts depends on task context, motion dynamics, and system latency.

Despite these advances, several gaps remain. Many academic approaches are evaluated in controlled or synthetic scenarios that do not fully reflect practical development constraints or real-world engine workflows. Comparative analyses between fixed and eye-tracked foveated rendering are often limited to specific implementations or isolated platforms, making it difficult to generalize conclusions across different hardware and rendering environments. Furthermore, while industry engines increasingly support foveated rendering through mechanisms such as Variable Rate Shading, the perceptual and performance implications of these engine-level solutions remain underexplored in a systematic manner.

In particular, there is a lack of empirical studies that evaluate existing, publicly available foveated rendering implementations across both desktop and virtual reality platforms using a unified methodology. This thesis addresses these gaps by focusing on a comparative evaluation of fixed and eye-tracked foveated rendering techniques as implemented in modern game engines, emphasizing practical performance characteristics and user perception rather than proposing new rendering algorithms.

Chapter 3

Methodology and Implementation

This chapter describes the methodology and technical implementation of foveated rendering across multiple real-time rendering engines and platforms. It outlines the selected hardware and software environment, the experimental scope, and the implementation details for both fixed and eye-tracked foveated rendering. The chapter focuses on practical realization in Unity and Unreal Engine on PC and VR platforms, mask acquisition, gaze integration, visualization, and pipeline-specific constraints. Finally, the capabilities and limitations of the Godot engine are discussed based on documentation analysis to provide contextual comparison.

3.1 Platforms and Tools

3.1.1 Software Scope

The scope includes:

- **Unity Engine**
  - **PC platform**: custom image-based VRS pipeline with manual selection of render passes for shading-rate injection. Unity version 6000.2.12f1 [22] was used because of support of VRS Tier 2.
  - **VR platform**: Meta XR SDK support for Fixed Foveated Rendering (FFR) and Eye-Tracked Foveated Rendering (ETFR), including mask extraction and gaze-based visualization. Unity version 2022.3.11f1 [23] was used.

- **Unreal Engine**
  - **PC platform**: implementation of image-based VRS in which the engine automatically determines appropriate render passes for shading-rate image (SRI) integration. Unreal Engine version 5.3.2 [24] was used.
  - **VR platform**: Meta XR plugin providing FFR and ETFR, including mask extraction and gaze-based visualization. Same Unreal Engine version was used.
- **Godot Engine**
  Godot does not currently support Foveated Rendering so no implementation work is performed. We only discuss the current limitations regarding Foveated Rendering. Evaluation is strictly based on publicly available documentation regarding OpenXR, gaze tracking, and potential VRS support.

3.1.2 Hardware Scope

All implementations and experiments were tested on the following hardware:

- **Desktop PC (Image-Based VRS experiments)**:

  GPU: NVIDIA GeForce RTX 4080
  CPU: Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz
  RAM: 128 GB
  OS: Windows 10

  This system was used for Unity PC and Unreal PC implementations of image-based VRS, where desktop-class GPUs allow explicit control over shading-rate images and render-pass injection.

- **Eye-Tracking Equipment**:

  Tobii GP3 (see Fig. 3.1) — an external desktop eye tracker capable of delivering high-frequency gaze data, allowing ETFR techniques to be evaluated on conventional monitors with desktop-class GPUs.

- **VR Headset (Unity VR / Unreal VR experiments)**:

  Meta Quest Pro (see Fig. 3.2) - a standalone headset with integrated infrared eye-tracking and vendor-supported ETFR APIs inside the Meta XR Software Development Kit (SDK) for Unity and Unreal Engine. This device provides hardware-level support for fixed foveation (FFR) and eye-tracked foveation (ETFR), enabling direct evaluation of gaze-contingent rendering in practical VR scenarios.

Figure 3.1: Tobii GP3 [6].

Figure 3.2: Meta Quest Pro [7].

3.1.3 Experimental Scope

The thesis includes:

- Controlled performance benchmarks on both PC and VR hardware;
- Comparison of foveation levels (Off, Low, Medium, High);
- User perception tests in a VR shooting-range scenario evaluating perceived visual quality, comfort, and mask preference.

3.2 Unity Engine

3.2.1 Foveated Rendering in Unity VR

The project targets Meta Quest Pro [7] using the Meta All-in-One SDK [25] with OpenXR [26]. At runtime, eye-tracking data are read each frame via 'OVRPlugin'. If both eyes are tracked, their orientations are blended by spherical interpolation to produce a stable world-space gaze vector; if only

one eye is tracked, it is used. When eye tracking is unavailable, the system falls back to the camera's forward direction.

Two rendering modes are supported. Fixed Foveated Rendering (FFR) reduces peripheral resolution using discrete levels ('Off/Low/Medium/High') controlled by 'OVRManager'. Eye-Tracked Foveated Rendering (ETFR) enables a dynamic sharp region that follows the user's gaze. These modes can be toggled at runtime for comparison.

To visualize and debug gaze alignment, the scene includes (i) a "gaze point" marker placed by a physics raycast from the center eye, and (ii) a "fovea mask" UI image held at a fixed depth along the gaze direction to approximate the high-quality region.

Level-of-Detail (LOD) selection is gaze-contingent. If the gaze ray hits an object's collider, LOD0 is forced. Otherwise, the angle between the gaze vector and the object direction drives thresholds (tighter for ETFR, wider for FFR) to choose lower LODs. When foveation is disabled, the script releases control, restoring Unity's distance-based LOD.

Validation used the Meta XR Developer Hub and in-headset performance overlays to (i) test performance under different FFR/ETFR settings, (ii) stream the headset view to the PC for live inspection and recording, and (iii) create and debug foveation masks, verifying size and alignment with gaze.

Mask Acquisition

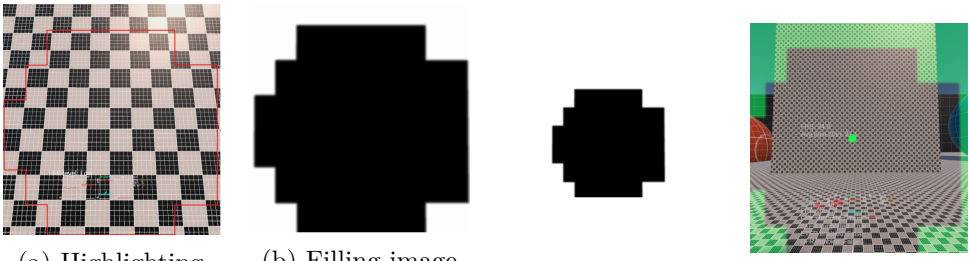
Unity's XR renderer does not expose any on-screen overlay that reveals the variable-rate-shading (VRS) pattern applied by Fixed- or Eye-Tracked Foveated Rendering. To analyse those patterns I therefore implemented a workflow:

1. **API preparation** – The project was migrated to the OpenXR back-end with the Oculus feature plug-ins enabled.
2. **Frame capture** – For each FFR and ETFR level (Low, Medium, High) I recorded full-resolution screenshots directly from the headset stream.
3. **Mask extraction** – In an external image editor I manually highlighted borders that mark the transitions between 1×, 2×, 4×, 8×, and 16× rate regions based on visual blending of a reference pattern.

To illustrate the acquisition procedure, consider the Fixed Foveated Rendering (FFR) setting "Low". The workflow for capturing its corresponding shading-rate mask proceeds as shown in Fig. 3.3.

Stereo Consistency Analysis

To examine the spatial symmetry of the foveated-rendering masks, we hypothesized that the per-eye shading patterns should be mirror images of one another; that is, the left-eye mask would be the horizontal reflection of the



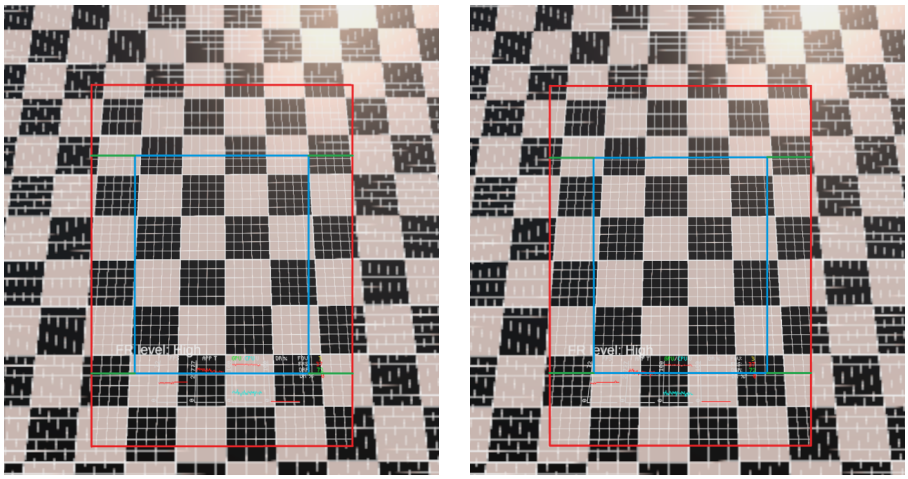
(a) Highlighting rendering layers based on screenshot.

(b) Filling image with colors corresponding to rendering factor.

(c) Creating 512×512 pixels image.

(d) Setting image as transparent mask.

Figure 3.3: Workflow for generating and applying a rendering-layer mask in Unity.



(a) left-eye mask.

(b) right-eye mask.

Figure 3.4: Comparison of left- and right-eye foveal masks at the Eye-Tracked Foveated Rendering (ETFR) "High" setting. The only difference is a small horizontal displacement. The overlaid red, green, and blue guide lines indicate the boundaries between regions rendered at different shading rates (see Fig. 3.5).

right-eye mask. I therefore captured framebuffer snapshots for both eyes at each foveation level and performed a tile-by-tile comparison (see Fig. 3.4).

Visual inspection confirms that the two masks are pixel-wise identical; the only difference is a small horizontal displacement. That offset is expected because each eye is rendered in its own off-axis viewport before lens-distortion correction, but it does not constitute a mirror inversion. Thus Unity applies one and the same shading-rate pattern to both eyes, rather than reflecting the mask about the vertical axis.

The use of identical, non-mirrored foveation masks for both eyes is a deliberate design choice in modern VR foveated rendering systems. Rather than defining foveation independently in each eye's image space, the foveation region is typically specified in a shared head-centric or cyclopean coordinate system and then projected into the left and right eye views using their

respective off-axis projection matrices. This approach preserves binocular correspondence of high-acuity regions and avoids depth inconsistencies that could arise from independently mirrored masks. As noted by Patney et al. [11], defining foveation in a common perceptual space is essential to maintain stereo fusion and visual comfort in head-mounted displays, since the human visual system integrates information binocularly rather than processing each eye in isolation.

Mask Visualization

Based on the extracted masks, a combined diagram illustrating Meta's runtime allocation of shading-rate regions for both Fixed Foveated Rendering (FFR) and Eye-Tracked Foveated Rendering (ETFR) on Quest-class devices was created (see Fig. 3.5).

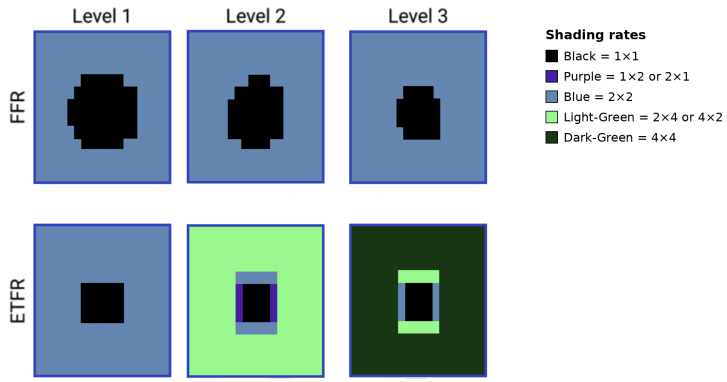


Figure 3.5: Unity VR shading rate allocation: FFR vs ETFR.

Gaze-Point Visualization

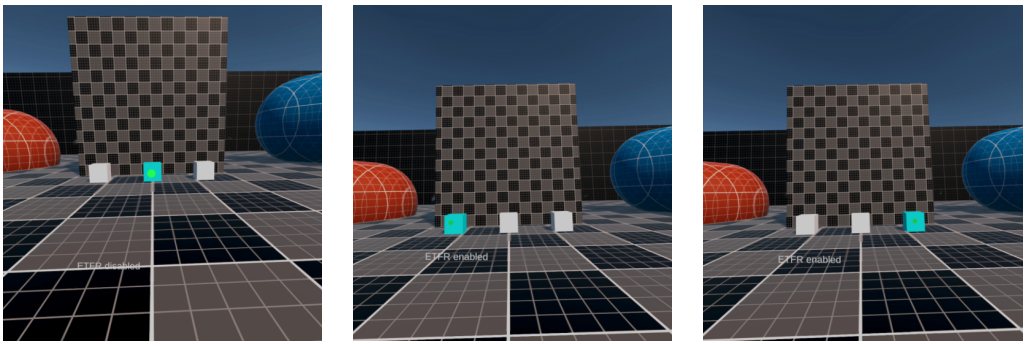
The gaze-point indicator is realised as a sphere, its position is updated every frame from hardware eye-tracking data (if it is available, otherwise it takes the forward vector of the main camera as gaze direction) (see Fig. 3.6).

1. **Data acquisition** – Binocular pose samples are retrieved in the Render step. If both eyes are valid, the quaternions are spherically interpolated; if only one eye is valid, its orientation is used directly.
2. **World-space conversion** – The resulting forward vector is rotated by the current head-rig transform, producing a world-space gaze direction D.
3. **Hit testing** – A single physics ray-cast is launched from the cyclopean eye along D.
   - On hit, the impact point is set on the surface.
   - On miss, the marker is projected onto an auxiliary plane at a fixed depth in front of the camera.

4. **Temporal filtering** – The marker's position is smoothed to suppress saccadic jitter.

5. **Visibility toggle** – Visibility can be toggled interactively with a controller button, permitting on-device validation without altering scene state.

The procedure falls back to camera-forward placement whenever eye-tracking support is unavailable, ensuring graceful degradation on non-ETFR devices.



(a) the central object observed at a fixed gaze depth.

(b) the left-hand object observed at dynamically updated gaze depth.

(c) the right-hand object observed at dynamically updated gaze depth.

Figure 3.6: Illustration of the gaze marker shown in green with gaze-interactive objects under two depth configurations.

Gaze-Contingent Level-of-Detail Selection

Every mesh that participates in gaze-adaptive rendering holds a Unity `LODGroup` and a lightweight LOD component.

1. **Angular metric** – For object centre p the angle θ is computed with the same vector d_g derived for the gaze point:

$$\theta \;=\; \cos^{-1}\!\Big(d_g \;\cdot\; \frac{p - p_{\text{eye}}}{\| p - p_{\text{eye}} \|}\Big).$$

Here, d_g denotes the normalized gaze direction vector obtained from the eye tracker, p_{eye} represents the eye (camera) position in world space, and p corresponds to the center of the rendered object. The expression computes the angular deviation θ between the user's gaze direction and the direction towards the object. Small values of θ indicate that the object lies close to the current point of fixation and should therefore be rendered with a higher level of detail, whereas larger angular offsets correspond to peripheral vision and allow for more aggressive level-of-detail reduction.

2. **Mode-aware thresholds** – Four angular thresholds $\theta_0, \ldots, \theta_3$ are exposed for each fixed-foveation level (Off / Low / Medium / High) and for eye-tracked foveation (ETFR). They are tuned empirically (5°–50°) to trade polygon count against perceptual fidelity.

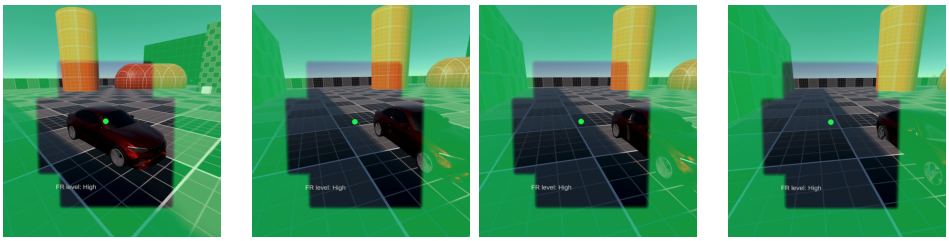
3. **Ray confirmation** – If the primary gaze ray intersects the object's mesh collider, the component immediately forces LOD 0, guaranteeing maximum detail for the user's explicit fixation.

To illustrate the level-of-detail (LOD) transition, I employed a custom 3D model (see Fig. 3.7). Minor imperfections in its normal map intentionally accentuate the differences between successive LODs, making the transitions more apparent.

(a) LOD0 (31372). (b) LOD1 (25096). (c) LOD2 (15686). (d) LOD3 (3349).

Figure 3.7: All four levels of detail (with their polygon counts).

See also an illustration of LOD changing based on gaze position on Fig. 3.8.



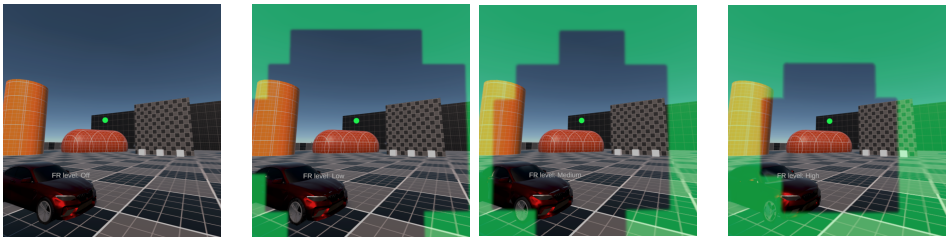
(a) LOD0. (b) LOD1. (c) LOD2. (d) LOD3.

Figure 3.8: Gaze-dependent progression through the model's four levels of detail (LODs).

To assess how fixed-foveated rendering (FFR) influences level-of-detail (LOD) selection, we empirically determined the effective radii for every FFR and ETFR tier. Calibrating these radii mitigates the otherwise abrupt transitions between successive LODs. The Fig. 3.9 applies these calibrated values to demonstrate that, even with a stationary gaze point, the active LOD changes as the FFR level increases.

3.2.2 Foveated Rendering in Unity PC

This subsection describes the implementation of foveated rendering on the PC platform using the Unity engine [20]. The chapter focuses on an image-based Variable Rate Shading (VRS) approach, where the shading rate is controlled explicitly by the application.



(a) LOD0. (b) LOD1. (c) LOD2. (d) LOD3.

Figure 3.9: Gaze-independent level of detail (LOD) shifts induced by progressively higher FFR settings.

First, the principle of image-based VRS and its applicability within Unity's Scriptable Render Pipeline is introduced. Next, the need for manual selection of compatible render passes is discussed, highlighting the differences from VR runtime-driven foveation. The chapter then describes the acquisition of gaze data using the Gazepoint GP3 eye tracker [6] and its API, including the transformation of normalized gaze coordinates for use in Unity.

Subsequently, the process of generating a shading rate image from the gaze data and integrating it into the rendering pipeline is explained. Finally, methods for visualizing both the gaze point and the shading rate mask are presented, followed by a comparison between the implemented PC-based approach and typical VR rendering pipelines.

Image-based Variable Rate Shading

Image-based Variable Rate Shading (VRS)[16] is a GPU-supported technique that enables spatially varying fragment shading rates across the render target. Instead of executing fragment shaders uniformly for all pixels, a shading rate image specifies the shading frequency for individual screen regions, typically 8×8 or 16×16 (depends on API and hardware), allowing computational effort to be concentrated in perceptually important areas.

On PC platforms, VRS is available on modern GPUs via DirectX 12 [14] or Vulkan [15]. In Unity, this functionality can be accessed through the Universal Render Pipeline (URP)[27]. In this work, image-based VRS is used to implement foveated rendering by dynamically generating a shading rate image driven by the user's gaze position.

The range of shading rates used in this implementation was limited to a small set of discrete values that are commonly available in image-based VRS on PC platforms. The baseline rate was 1×1 for the foveal region, while lower rates such as 2×2 were used in peripheral regions. If the hardware and the selected graphics API supported additional coarse rates, 4×4 was also used for the far periphery; otherwise, the system fell back to the closest supported rate (e.g., 2×2).

Shading Rate Image Generation

The concrete shading rate values were determined by mapping each pixel of the shading rate image to a discrete shading rate supported by the graphics API. For each rendered frame, the distance between a given pixel and the current gaze-centered foveal position was evaluated and used to classify the pixel into one of several concentric regions.

Each region was assigned a predefined shading rate, such as 1×1 for the foveal region and 2×2 or 4×4 for peripheral regions. These shading rates were encoded into the shading rate image using pixel values, where each channel represented a specific shading rate identifier according to the API specification.

The resulting shading rate image therefore acted as a lookup texture, in which the GPU interpreted the stored values not as visual colors, but as control data defining the fragment shading frequency for the corresponding screen regions.

Manual Selection of Render Passes

Unlike VR runtimes, where foveated rendering is often applied implicitly at the driver or compositor level, Unity on PC requires explicit integration of VRS into the rendering pipeline. The developer must manually specify the render passes to which the shading rate image is applied.

In the Universal Render Pipeline (URP), this is achieved using custom render passes implemented via the Render Graph API. The shading rate image can only be attached to passes that perform fragment shading (e.g., opaque or transparent geometry passes). Passes such as post-processing, UI rendering, or compute-only passes are excluded, as they do not support variable shading rates.

Figure 3.10 illustrates a simplified rendering scheme showing where the shading rate image is applied within the URP rendering pipeline.

Gaze Data Acquisition

The gaze position of the user was obtained using the *Gazepoint GP3* eye tracker [6]. The device provides eye-tracking data through its proprietary API, which continuously outputs the gaze position as normalized screen-space coordinates (x, y) in the range $\langle 0, 1 \rangle$.

According to the Gazepoint API documentation, the coordinate system uses the top-left corner of the screen as the origin, with the y axis increasing downwards. In contrast, Unity textures and render targets use a bottom-left origin with the y axis increasing upwards. Therefore, the y coordinate had to be inverted before further processing, using the transformation:

$$y_{\text{unity}} = 1 - y_{\text{gp3}}.$$

The resulting normalized coordinates were then used directly to position the foveal region within the shading rate image.

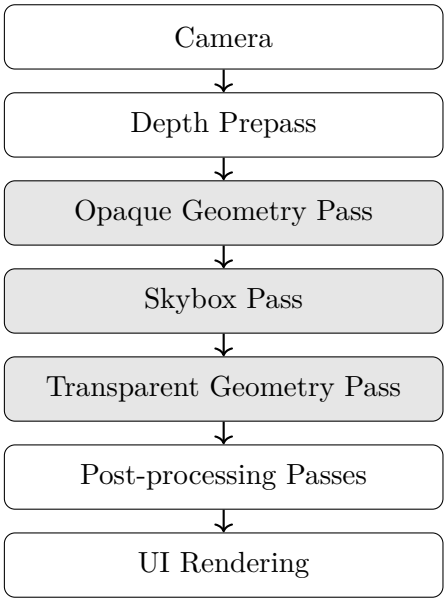


Figure 3.10: Simplified Universal Render Pipeline. Passes highlighted in gray apply foveated rendering by attaching the shading rate image to the render pass.

Visualization of the Gaze Point

To verify correct gaze data acquisition and coordinate transformation, the gaze position was visualized during development. This visualization was implemented as a simple screen-space marker rendered at the computed gaze location, similar to the VR project.

The visualization enabled verification of correct alignment between the reported gaze position and the rendered image, as well as detection of potential issues such as coordinate inversion, latency, or noise in the eye-tracking signal.

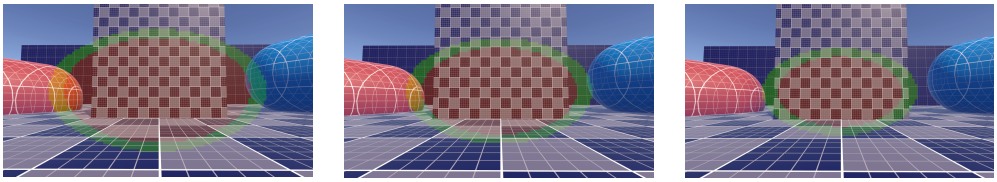
Visualization of the Shading Rate Mask

The shading rate image was also visualized for debugging purposes. This was achieved by rendering the mask as a color-coded overlay, where individual colors correspond to different shading rates applied by the GPU.

This visualization made it possible to verify that the foveal region correctly followed the gaze position, that peripheral regions were assigned lower shading rates, and that the shading rate image was updated consistently at runtime. An example of such visualization, showing three distinct foveation levels on Unity PC, is provided in Figure 3.11.

Differences from the VR Rendering Pipeline

Foveated rendering implemented in Unity on PC differs significantly from VR-specific rendering pipelines. In many VR systems, foveation is handled



(a) Low level. (b) Medium level. (c) High level.

Figure 3.11: Example visualization of a shading rate mask with three foveation levels in Unity PC. Red color represents a 1×1 shading rate, green represents 2×2, and blue represents 4×4.

internally by the runtime or hardware driver and requires minimal application-level control.

In contrast, the Unity PC approach requires explicit gaze integration, manual shading rate image generation, and careful selection of compatible render passes. While this increases implementation complexity, it provides full control over the foveation strategy and makes the approach well suited for experimental and research-oriented applications.

3.3 Unreal Engine

3.3.1 Foveated Rendering in Unreal VR

This section describes the implementation of foveated rendering (FR) in Unreal Engine for virtual reality applications. The goal of this implementation is to evaluate the practical feasibility of fixed and gaze-contingent foveated rendering within the constraints of the Unreal Engine VR rendering pipeline and to enable a direct comparison with the Unity-based implementation described in the previous section.

The implementation is based on Unreal Engine 5 and relies on the OpenXR standard for VR device integration, extended by the Meta XR plugin to support eye-tracking and platform-specific foveation features on Meta Quest devices. Both fixed foveated rendering (FFR) and eye-tracked foveated rendering (ETFR) are considered, depending on hardware capabilities and runtime configuration.

Unlike Unity, Unreal Engine provides limited direct access to low-level shading rate control and foveation masks. As a result, the FR implementation in Unreal Engine is primarily mediated by engine-level abstractions and plugin-provided interfaces rather than explicit user-defined shading rate images. This significantly influences how foveation regions are defined, updated, and visualized.

The chapter is structured to reflect the key components required for foveated rendering in VR. First, the acquisition of the foveation mask is described, including the distinction between fixed and eye-tracked approaches. This is followed by an analysis of stereo consistency to ensure correct alignment of foveated regions across both eyes. Subsequent sections focus on the

visualization of the foveation mask and the eye gaze point, which are essential for debugging, validation, and experimental analysis. Finally, the differences between the Unreal Engine and Unity VR rendering pipelines are discussed to highlight engine-specific constraints and their implications for foveated rendering research.

This structured approach allows the Unreal Engine implementation to be analyzed not only as a functional system, but also as a case study demonstrating the practical limitations and trade-offs of foveated rendering in a modern, production-oriented game engine.

Mask Acquisition

Similarly to Unity VR, Unreal Engine does not expose the foveated rendering or variable-rate shading pattern applied at runtime as an explicit on-screen overlay. Consequently, an indirect acquisition procedure was employed.

The mask acquisition workflow in Unreal Engine mirrors the approach used in Unity VR and consists of the following steps:

1. **API preparation** – The project was configured to use the OpenXR runtime with the Meta XR plugin enabled, providing access to platform-level foveated rendering and eye-tracking functionality.
2. **Frame capture** – For each supported foveation mode (FFR and ETFR) and quality level (Low, Medium, High), full-resolution per-eye framebuffers were captured directly from the VR output.
3. **Mask extraction** – The captured images were processed externally to identify transitions between shading-rate regions corresponding to different effective pixel densities.

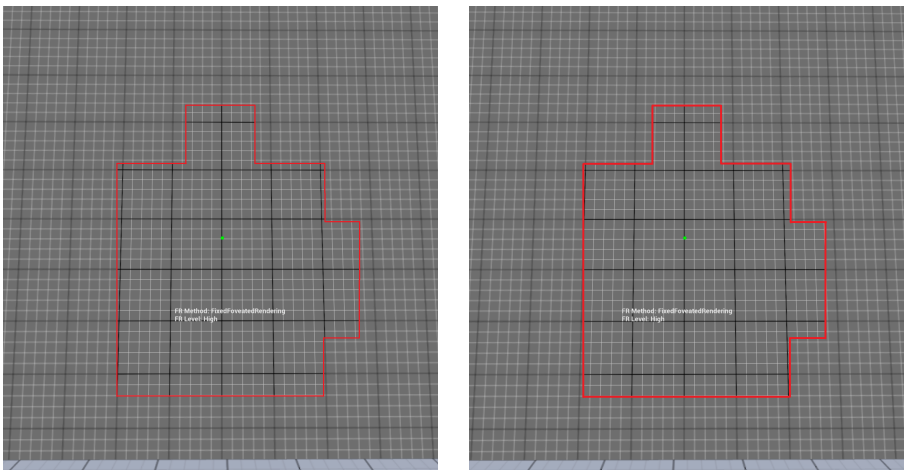
As in the Unity VR case, the extracted masks represent an approximation of the internal shading-rate layout rather than an exact GPU-level representation. Nevertheless, this approach enables qualitative and comparative analysis of foveation behavior across engines.

Stereo Consistency Analysis

To evaluate stereo consistency in Unreal Engine, the same hypothesis as in the Unity VR implementation was adopted: the foveated rendering masks applied to the left and right eyes should be spatially consistent and perceptually aligned.

Framebuffer snapshots were captured separately for the left and right eye for each foveation mode and quality level. These images were then compared visually to assess symmetry, alignment, and potential discrepancies between eyes. A representative comparison of the left- and right-eye framebuffers is shown in Fig. 3.12.

The analysis confirms that Unreal Engine applies a consistent foveation pattern to both eyes. As observed in Unity VR, the masks are not mirror



(a) left-eye mask. (b) right-eye mask.

Figure 3.12: Comparison of left- and right-eye foveal masks at the Fixed Foveated Rendering "High" setting. The only difference is a small horizontal displacement.

images of each other. Instead, a small horizontal offset is present between the left-eye and right-eye masks. This displacement arises from the use of per-eye off-axis projection and separate viewports prior to lens-distortion correction and does not indicate a violation of stereo consistency.

Thus, Unreal Engine, like Unity, applies a single logical foveation layout across both eyes, ensuring binocular coherence while accounting for headset-specific rendering geometry.

Mask Visualization

Based on the extracted masks, a visual representation (see Fig. 3.13) of Unreal Engine's foveated rendering layout was created to illustrate the spatial distribution of shading-rate regions for both FFR and ETFR.

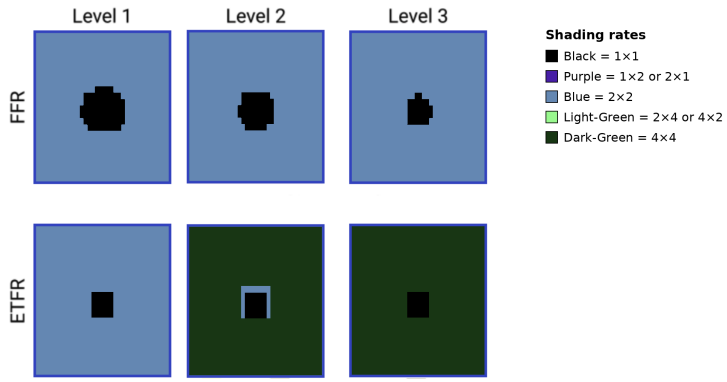


Figure 3.13: Unreal VR shading rate allocation: FFR vs ETFR.

The visualization approach closely follows the Unity VR implementation, with one important distinction. In the case of fixed foveated rendering, separate visualization masks were applied for the left and right eyes with a small horizontal offset, in Unity VR only one mask for both eyes was used. This adjustment was necessary to ensure that the visualized mask contours precisely matched the perceived boundaries of resolution change in the rendered image.

For eye-tracked foveated rendering, a single shared mask was used for both eyes. Due to the dynamic nature of ETFR, where the foveation region continuously follows the user's gaze, the mask boundaries are not perceptually stable and are difficult to observe directly. As a result, per-eye offsets are visually indistinguishable in this mode and unnecessary for qualitative analysis.

The visualization uses a color-coded scheme corresponding to different effective pixel-density reductions, consistent with the Unity VR diagrams, allowing direct comparison between engines.

Gaze-Point Visualization

To validate eye-tracking behavior and indirectly confirm gaze-contingent foveation, the gaze point was visualized directly within the Unreal Engine scene.

The gaze direction provided by the OpenXR eye-tracking interface was used to cast a ray from the headset origin into the virtual environment. At the intersection point, a spherical marker was rendered. This marker employed a custom unlit material with depth testing disabled, ensuring constant visibility regardless of scene lighting or occlusion.

This visualization technique enables real-time inspection of gaze stability, latency, and alignment with the perceived high-resolution region. While it does not directly expose the foveation mask used by the renderer, it provides strong qualitative evidence that the foveation region follows the user's gaze as expected in ETFR mode.

Differences from the Unity VR Rendering Pipeline

From a conceptual perspective, the Unreal Engine VR rendering pipeline does not differ substantially from Unity VR with respect to the application-level behavior of foveated rendering. In both engines, the foveation process is largely abstracted away and handled by the VR runtime and platform-specific plugins.

As a result, the methodologies used for mask acquisition, stereo consistency analysis, and visualization remain largely identical across both engines. This consistency supports a fair comparative evaluation of foveated rendering behavior and reinforces the validity of cross-engine analysis.

3.3.2 Foveated Rendering in Unreal PC

This chapter describes the implementation of foveated rendering on PC using Unreal Engine. The goal of this chapter is to explain how image-based Variable Rate Shading is used, how gaze data are integrated through custom logic, and how this approach differs from the Unity PC implementation. The chapter focuses on technical aspects of the rendering pipeline, shading rate image generation, and visualization methods used for validation.

Overview of Foveated Rendering in Unreal Engine

Unreal Engine supports foveated rendering on PC through image-based Variable Rate Shading, which is available on compatible GPUs using DirectX 12[16]. Unlike Unity, Unreal Engine manages most rendering passes internally, which limits direct manual control over individual stages of the rendering pipeline.

Image-based Variable Rate Shading

Image-based Variable Rate Shading allows the shading rate to vary across the screen using a shading rate image (SRI). Each element of the SRI defines the shading rate for a corresponding screen region, typically covering blocks of 8×8 or 16×16 pixels in the final rendered image. Supported shading rates include full rate 1×1 as well as reduced rates such as 1×2, 2×1, and 2×2, which are available on all Variable Rate Shading tiers.

Some devices additionally support coarser shading rates, including 2×4, 4×2, and 4×4. By applying lower shading rates in peripheral regions and higher rates in the foveal region, image-based VRS significantly reduces fragment shader workload while preserving visual quality in perceptually important areas.

Shading Rate Image Generation

The shading rate image is generated dynamically based on the current gaze position. The image typically has a lower resolution than the final rendered frame to reduce overhead. A radial mask is applied, where the highest shading rate is assigned to the foveal region, and progressively lower shading rates are used toward the periphery. The SRI is updated each frame to follow the user's gaze position. Similar to Unity PC.

Automatic Render Pass Selection

In Unreal Engine, the selection of render passes affected by VRS is performed automatically by the engine. The developer does not explicitly choose which passes use reduced shading rates. This behavior differs from Unity, where specific render passes must be manually configured. The Unreal approach simplifies integration but reduces experimental flexibility when analyzing the impact of foveated rendering on individual rendering stages.

Gaze Data Integration

Unreal Engine does not provide native support for gaze-driven foveated rendering on PC; therefore, custom integration logic was implemented.

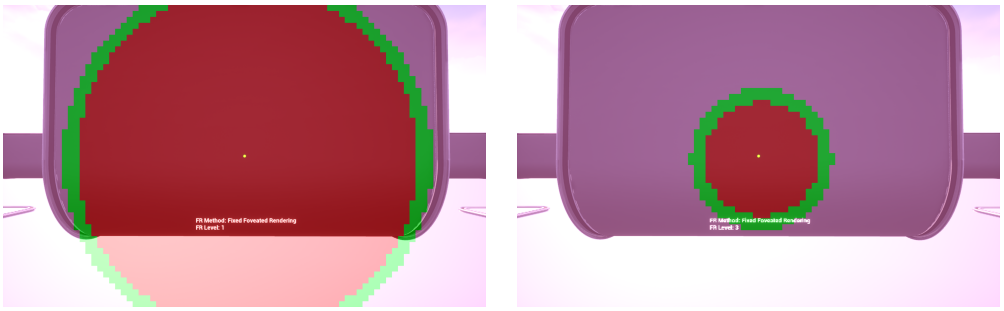
Eye-tracking data are acquired externally using the Gazepoint GP3 eye tracker [6] and its official application programming interface (API), and the raw gaze data are processed and transformed into normalized screen-space coordinates.

These coordinates define the center of the foveal region and are used as input for shading rate image generation, enabling gaze-contingent foveated rendering while remaining independent of engine-specific eye-tracking APIs.

Visualization and Debugging

To verify the correctness of the implementation, visualization methods were used. These include rendering the gaze point as an overlay and displaying the shading rate mask directly on the screen. Such visualizations are essential for validating gaze alignment, region boundaries, and temporal stability of the foveated rendering system.

An example of this visualization is shown in Fig. 3.14, where both the gaze point and the corresponding shading rate masks are presented.



(a) Low level. (b) High level.

Figure 3.14: Example visualization of a shading rate masks and gaze point in Unreal PC. Red color represents a 1×1 shading rate, green represents 2×2, and pink represents 4×4. Green dot in the center is gaze point.

Comparison with Unity PC Implementation

Compared to the Unity PC implementation, Unreal Engine provides less low-level control over the rendering pipeline. Unity allows explicit selection of render passes and more flexible scripting access. However, Unreal Engine offers a more automated and robust integration. These differences affect experimentation speed, debugging complexity.

3.4 Godot Engine

3.4.1 Foveated Rendering in Godot

Godot is an open-source game engine that emphasizes a lightweight design, transparent architecture, and community-driven development. Recent versions of the engine introduced a Vulkan-based rendering backend, which significantly improves its graphical capabilities compared to earlier releases [28]. Despite these improvements, support for advanced XR-specific rendering optimizations remains limited.

OpenXR Support

Godot provides experimental support for OpenXR through its official XR plugin. This integration enables basic virtual reality functionality, including head tracking, stereoscopic rendering, and controller input [29]. However, the OpenXR feature set is not yet as comprehensive as in commercial engines, and support for advanced OpenXR extensions is limited.

Eye Tracking Support

At the time of writing, Godot does not provide native eye tracking support via OpenXR. The engine documentation does not expose access to gaze direction vectors or eye tracking data required for gaze-contingent rendering techniques [30]. As a result, eye-tracked foveated rendering (ETFR) cannot be implemented without substantial engine-level modifications.

Foveated Rendering and Variable Rate Shading

Godot does not include built-in support for fixed foveated rendering (FFR) in XR applications. Furthermore, hardware-level Variable Rate Shading (VRS) is not currently exposed through the engine's rendering API [31]. Image-based VRS techniques, which rely on shading rate images to control pixel shading density, are also not supported. These limitations prevent the implementation of both fixed and eye-tracked foveated rendering within the standard Godot workflow.

Comparison with Unity and Unreal Engine

In contrast to Godot, Unity and Unreal Engine provide mature and production-ready support for foveated rendering. Unity supports fixed and eye-tracked foveated rendering through XR plugins and platform-specific SDKs, including OpenXR-based workflows [32]. Unreal Engine offers native support for hardware Variable Rate Shading and image-based VRS, which can be combined with eye tracking data for gaze-contingent rendering [33]. These features allow researchers and developers to experiment with foveated rendering techniques without modifying the engine source code.

Limitations for Practical ETFR Implementation

Due to the absence of native eye tracking access, lack of VRS support, and limited control over XR rendering stages, Godot is currently not suitable for a full experimental implementation of eye-tracked foveated rendering. Implementing ETFR would require extensive low-level changes to the engine, which is beyond the scope of this thesis.

Summary

Godot represents a promising open-source engine for general game development and basic VR applications. However, its current XR and rendering feature set does not yet support advanced foveated rendering techniques. For this reason, Godot is included in this work only as a theoretical reference and is excluded from the practical evaluation, which focuses on Unity and Unreal Engine.

Chapter 4

Experimental Methodology

This chapter combines objective performance measurements with a controlled user study to evaluate both technical and perceptual aspects of foveated rendering.

4.1 Performance Testing Methodology

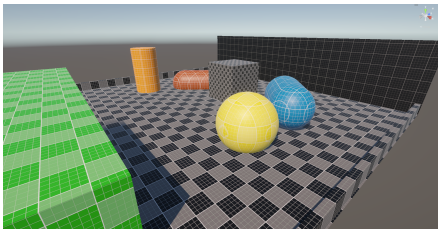
Test Scenes

For each performance test, two separate scenes were used to target different rendering bottlenecks (see Fig. 4.1):

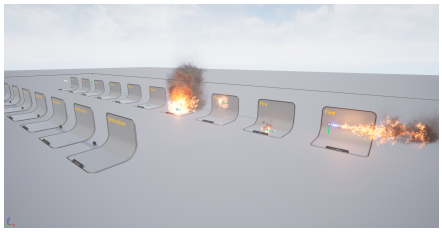
- **Shader-driven scene** – a scene composed primarily of shader-based objects with relatively simple geometry. It is designed to stress fragment shading and material complexity, enabling evaluation of foveated rendering under shading-limited conditions.

- **Geometry-intensive scene** – a heavy scene with high polygon density and complex meshes. Without foveated rendering, this scene achieves a baseline performance of approximately 20–30 FPS and is used to assess foveation effectiveness in geometry-limited scenarios.

The scenes were not identical across all tests or engines. However, they were designed to be as similar as possible in terms of visual appearance, scene layout, and performance characteristics to ensure fair and comparable evaluation results across platforms and environments.

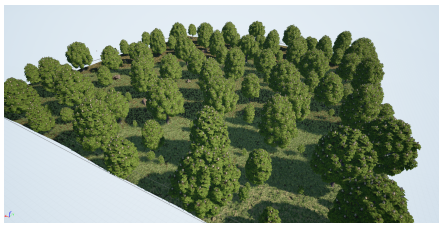


(a) Shader Unity (15k) [34, 35].



(b) Shader Unreal (100k) [36].

(c) Geometry Unity (5mil) [37].



(d) Geometry Unreal (1.3mil) [38].

Figure 4.1: Scenes used for performance tests with rough number of polygons in brackets.

Foveated Rendering Levels

Two foveated rendering approaches are evaluated using matched quality levels:

- **Fixed Foveated Rendering (FFR):** Off, Low, Medium, High
- **Eye-Tracked Foveated Rendering (ETFR):** Off, Low, Medium, High

This design enables a direct comparison between FFR and ETFR under equivalent foveation strengths.

Measured Metric

Performance comparison is based exclusively on:

- Frames per second (FPS)

FPS is selected as the primary metric due to its direct relevance to real-time interactivity and user comfort in VR scenarios.

Test Environments

All performance measurements are conducted across four distinct runtime configurations:

- Unity Engine VR
- Unreal Engine VR
- Unity Engine PC
- Unreal Engine PC

This setup allows engine-level and platform-level effects to be observed independently.

4.2 User Study Methodology

Goal of the User Study

The user study focuses on evaluating perceived visual quality, performance differences, and overall comfort across multiple foveated rendering presets in a VR environment.

Hardware Setup

The experiment is conducted using a Meta Quest Pro [7] headset with integrated eye tracking, enabling real-time gaze-contingent rendering.

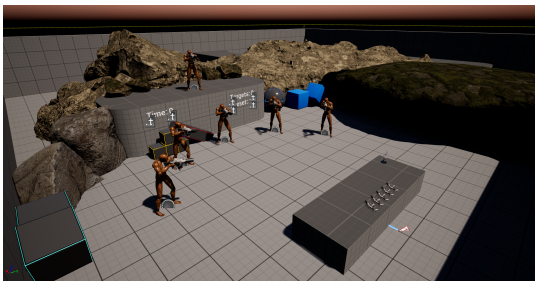
Shooting Range Scenario

A custom VR shooting-range scene is used as the experimental task, specifically designed to induce frequent gaze shifts and sustained visual attention. An overview of the scene layout and visual complexity is shown in Fig. 4.2, which also serves as a reference for the spatial distribution of targets and background elements.

Rendering Presets

Five predefined rendering presets are evaluated throughout the experiments:

- **Preset A** – Normal rendering without foveated rendering.
- **Preset B** – FFR Low.
- **Preset C** – FFR High.
- **Preset D** – ETFR Low.
- **Preset E** – ETFR High.



(a) Scene overview.

(b) During test.

Figure 4.2: User study setup.

Procedure

The experiment follows a structured and repeatable procedure (see Fig. 4.3):

1. Before entering the scene, eye tracking is individually calibrated for each participant using the headset's built-in calibration routine.
2. The participant then enters the shooting-range scene and is given sufficient time to freely look around and adapt to the initial rendering preset.
3. A total of five rendering presets are evaluated. Upon entering the scene, the participant always starts with the non-foveation preset.
4. When the participant indicates readiness, a shooting round begins with a randomly selected preset. The non-foveation preset is also treated randomly.
5. Each shooting round lasts exactly one minute, during which the participant actively engages with the scene.
6. After the round ends, the participant is again allowed to explore the scene freely before proceeding.
7. The participant's task is to notice and assess differences in both performance and visual quality between presets.

Questionnaires

The questionnaire process is divided into three stages (see Appendix B):

- **Pre-test questionnaire:** collects general participant information prior to testing.

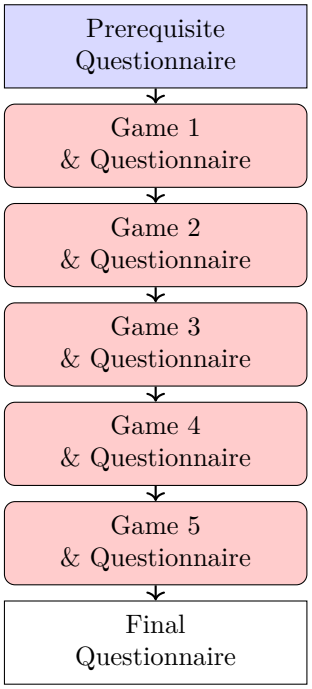


Figure 4.3: Overview of the user study procedure and questionnaire flow.

- **Per-preset questionnaire:** after each preset, participants answer the same set of questions related to perceived visual quality and performance.
- **Post-test questionnaire:** at the end of the session, participants indicate which preset they preferred most and least in terms of visual quality and performance.

Performance Context

The baseline preset without foveated rendering achieves approximately 30–35 FPS. The most aggressive configuration (Preset E) reaches the system-imposed maximum of 72 FPS, providing a clear contrast in both performance and perceptual experience.

Chapter 5

Results

5.1 Performance Evaluation

Performance results are reported separately for each engine and platform to highlight implementation- and hardware-specific behavior described in section 3.1.

5.1.1 VR

Unity Engine

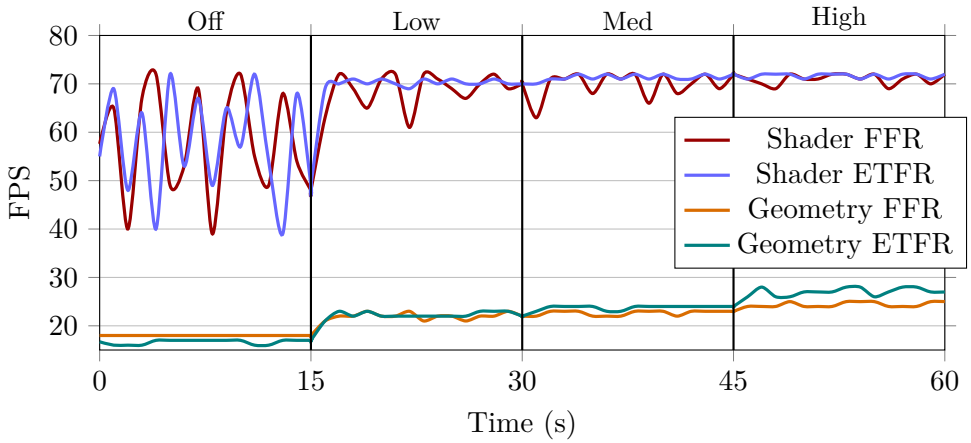


Figure 5.1: FPS over time for the Unity VR tests, with the rendering preset switched every 15 seconds (Off → Low → Med → High).

As shown in Fig. 5.1, the Shader scene benefits from foveated rendering mainly through higher average performance and improved stability. Shader FFR rises from 58 FPS (Off) to 69 (Low), 70 (Med), and 71 (High), which corresponds to **+20%**, **+21%**, and **+23%** compared to Off. Shader ETFR shows a similar average gain, increasing from 58 FPS (Off) to 70 (Low) and 72 FPS (Med and High), i.e., **+22%** in Low and **+24%** in both Med and High. When comparing both methods directly in the Shader scene, ETFR

achieves higher mean FPS than FFR in the same presets by **+1%** (Low), **+3%** (Med), and **+1%** (High).

In the Geometry scene (Fig. 5.1), the performance gain is also clear, and ETFR scales more aggressively than FFR. With Geometry FFR, the mean FPS increases from 18 (Off) to 22 (Low), 23 (Med), and 24 (High), corresponding to **+22%**, **+26%**, and **+36%** relative to Off. Geometry ETFR starts from a slightly lower baseline (17 FPS in Off) but reaches 22 (Low), 24 (Med), and 27 (High), i.e., **+34%**, **+43%**, and **+62%** compared to its Off segment. When comparing both methods directly in the Geometry scene, ETFR achieves higher mean FPS than FFR in the same presets by **+1%** (Low), **+5%** (Med), and **+11%** (High), suggesting that gaze-contingent allocation is more effective for this geometry-heavy workload.

Unreal Engine

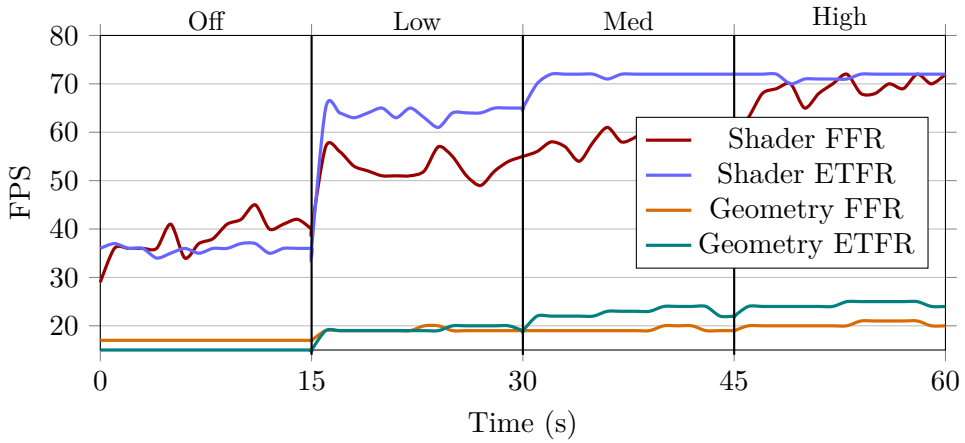


Figure 5.2: FPS over time for the Unreal VR tests, with the rendering preset switched every 15 seconds (Off → Low → Med → High).

As shown in Fig. 5.2, the Shader scene exhibits stronger performance scaling when foveated rendering is enabled. Shader FFR increases the mean FPS from 38 (Off) to 53 (Low), 58 (Med), and 68 (High), giving **+38%**, **+53%**, and **+80%** over Off. Shader ETFR achieves higher averages in the foveated presets, reaching 64 (Low), 71 (Med), and 72 (High) from an Off mean of 36, which corresponds to **+78%**, **+100%**, and **+100%**. Compared to Shader FFR at the same preset, Shader ETFR is higher by **+21%** (Low), **+22%** (Med), and **+4%** (High), indicating that eye-tracked foveation provides additional benefits beyond a fixed mask in this workload.

In contrast, the Geometry scene in Fig. 5.2 shows a more gradual but still consistent FPS increase. With Geometry FFR, the mean frame rate rises from 17 (Off) to 19 (Low), 19 (Med), and 20 (High), which corresponds to **+13%**, **+13%**, and **+20%** compared to Off. Geometry ETFR provides a larger gain, increasing from 15 (Off) to 19 (Low), 23 (Med), and 25 FPS (High), i.e., **+29%**, **+52%**, and **+63%** relative to its Off baseline. When

comparing the two methods directly at the same preset, Geometry ETFR reaches **+1%** (Low), **+19%** (Med), and **+20%** (High) higher mean FPS than Geometry FFR.

Comparison between VR engines

Scene	Engine	Low	Med	High
Shader	Unity	+20%/+22%	+21%/+24%	+23%/+24%
	Unreal	+38%/+78%	+53%/+100%	+80%/+100%
Geometry	Unity	+22%/+34%	+26%/+43%	+36%/+62%
	Unreal	+13%/+29%	+13%/+52%	+20%/+63%

Table 5.1: Relative FPS change for each scene and engine in VR. Each cell shows FFR/ETFR gain.

Comparing the VR results in Fig. 5.1 and Fig. 5.2, both engines show the same general trend: foveated rendering increases FPS in both the Shader and Geometry workloads, and ETFR usually provides an additional gain over FFR at the same preset (Table 5.1).

For the Shader scene, Unreal exhibits a much stronger scaling with foveation than Unity. In Unity (Fig. 5.1), Shader FFR reaches 71 FPS at High (**+23%** over Off), and Shader ETFR reaches 72 FPS (**+24%** over Off), so the improvement is mainly limited by the upper FPS range. In Unreal (Fig. 5.2), Shader FFR reaches 68 FPS at High (**+80%** over Off), while Shader ETFR reaches 72 FPS (**+100%** over Off), indicating a much larger relative gain and a clearer benefit from ETFR in this workload. Overall, Unity shows stable but moderate VR improvements, while Unreal shows stronger relative scaling, especially for shader-heavy content and for ETFR at higher presets.

For the Geometry scene, Unity starts slightly higher (18 FPS vs. 17 FPS in Off), but Unreal shows a stronger ETFR scaling at Medium and High. In Unity, Geometry ETFR reaches 27 FPS at High, which is **+62%** over its Off baseline, while in Unreal it reaches 25 FPS at High, which is **+63%** over Off. However, the relative advantage of ETFR over FFR differs: in Unity the ETFR uplift over FFR is modest and grows with strength (**+1%** Low, **+5%** Med, **+11%** High), whereas in Unreal the ETFR advantage becomes much larger already from Medium (**+1%** Low, **+19%** Med, **+20%** High). This suggests that, for the geometry-heavy Geometry workload, Unreal benefits more from gaze-contingent allocation once the foveation strength increases.

5.1.2 Desktop

Unity Engine

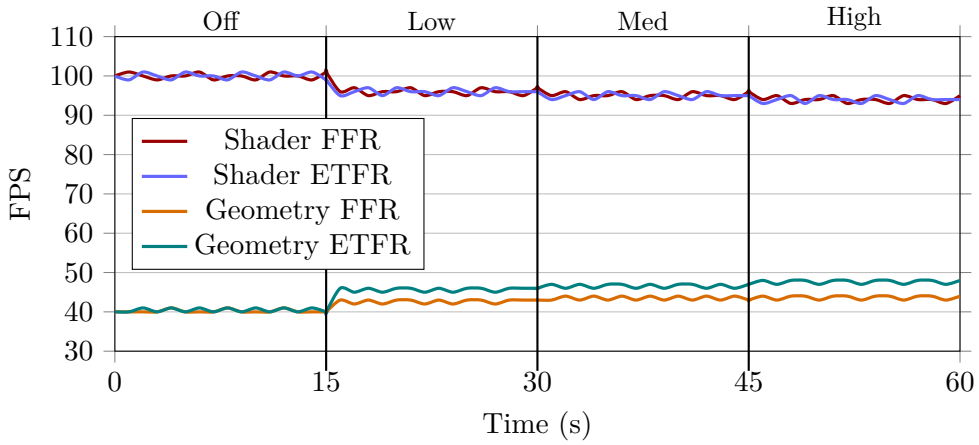


Figure 5.3: FPS over time for the Unity PC tests, with the rendering preset switched every 15 seconds (Off → Low → Med → High).

As shown in Fig. 5.3, the Shader scene does not benefit from foveated rendering in the PC tests. Shader FFR decreases from 100 FPS (Off) to 96 (Low), 95 (Med), and 94 (High), which corresponds to **-4%**, **-5%**, and **-6%** relative to Off. Shader ETFR follows the same trend with the same averages, therefore it does not outperform Shader FFR at any preset (**0%** difference in Low/Med/High). This drop is explained by the fact that the shader-heavy content in this scene does not use image-based Variable Rate Shading, so enabling foveation mainly adds overhead (mask logic and additional passes) without reducing the true shading workload.

In contrast, the Geometry scene in Fig. 5.3 shows a consistent FPS increase when foveation is enabled. Geometry FFR rises from 40 FPS (Off) to 43 (Low), 43 (Med), and 44 (High), giving **+8%**, **+8%**, and **+10%** over Off. Geometry ETFR increases from 40 FPS (Off) to 46 (Low), 47 (Med), and 48 (High), which corresponds to **+15%**, **+18%**, and **+20%** relative to Off. Compared to Geometry FFR at the same preset, Geometry ETFR is higher by **+7%** (Low), **+9%** (Med), and **+9%** (High), indicating that gaze-contingent allocation provides additional gains for the geometry-heavy workload.

Unreal Engine

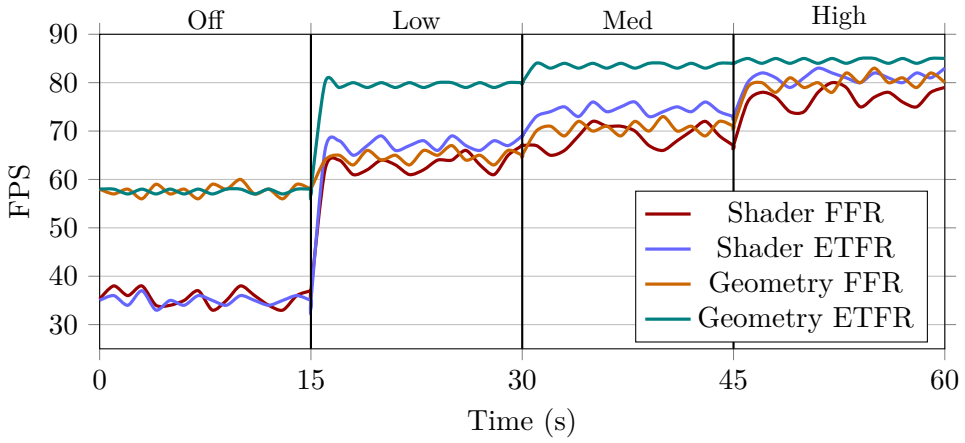


Figure 5.4: FPS over time for the Unreal PC tests, with the rendering preset switched every 15 seconds (Off → Low → Med → High).

As shown in Fig. 5.4, the Shader workload shows strong performance scaling when foveated rendering is enabled. Shader FFR increases the mean FPS from 36 (Off) to 63 (Low), 69 (Med), and 77 (High), which corresponds to **+75%**, **+92%**, and **+114%** relative to Off. Shader ETFR starts from a similar baseline (35 FPS in Off) but reaches higher averages in the foveated presets: 67 (Low), 74 (Med), and 81 (High), giving **+91%**, **+111%**, and **+131%** over Off. When comparing both methods at the same preset, Shader ETFR is higher than Shader FFR by **+6%** (Low), **+7%** (Med), and **+5%** (High).

In contrast, the Geometry workload in Fig. 5.4 improves more gradually. Geometry FFR increases the mean FPS from 58 (Off) to 65 (Low), 71 (Med), and 80 (High), which corresponds to **+12%**, **+22%**, and **+38%** compared to Off. Geometry ETFR keeps a similar Off baseline (58 FPS) but achieves 80 (Low), 84 (Med), and 85 (High), resulting in **+38%**, **+45%**, and **+47%** gains over Off. Compared to Geometry FFR at the same preset, Geometry ETFR is higher by **+23%** (Low), **+18%** (Med), and **+6%** (High). This trend is consistent with ETFR using a smaller full-resolution (1×1) region around the current gaze point and assigning the lowest shading rate (e.g., 1×16) over a larger part of the periphery than a fixed mask, which reduces the total shading cost more effectively.

Comparison between PC engines

Scene	Engine	Low	Med	High
Shader	Unity	-4%/-4%	-5%/-5%	-6%/-6%
	Unreal	+75%/+91%	+92%/+111%	+114%/+131%
Geometry	Unity	+8%/+15%	+8%/+18%	+10%/+20%
	Unreal	+12%/+38%	+22%/+45%	+38%/+47%

Table 5.2: Relative FPS change for each scene and engine in PC. Each cell shows FFR/ETFR gain.

A comparison between Unity and Unreal Engine reveals clear differences in how foveated rendering techniques affect performance across similar workloads Table 5.2.

For shader-dominated workloads, the engines behave fundamentally differently. In Unreal Engine, the shader test demonstrates very large gains from foveation, with FFR reaching up to **+114%** and ETFR up to **+131%** FPS improvement over the Off preset (Fig. 5.4). This suggests effective use of image-based Variable Rate Shading, allowing reduced shading cost in peripheral regions. In Unity, however, the Shader scene shows no performance gain from either FFR or ETFR. Instead, FPS decreases by about **-6%** at higher foveation levels (Fig. 5.3), due to the absence of image-based VRS for the tested shader objects and the additional overhead introduced by foveation logic.

In the Geometry scene, both engines benefit from foveated rendering; however, Unreal Engine shows consistently higher relative gains. As shown in Fig. 5.4, Unreal Geometry FFR achieves up to **+38%** FPS improvement in the High preset compared to Off, while Geometry ETFR reaches up to **+47%**. In contrast, Unity Geometry FFR increases performance by about **+10%**, and Geometry ETFR by **+20%**, as illustrated in Fig. 5.3. This indicates that Unreal's rendering pipeline benefits more strongly from both fixed and eye-tracked foveation in geometry-heavy scenes.

Overall, Unreal Engine demonstrates stronger and more consistent performance benefits from foveated rendering across both shader-heavy and geometry-heavy scenes, while Unity's results are more limited and strongly dependent on scene type and VRS compatibility. These results highlight that engine-level support for image-based VRS and efficient foveation integration plays a critical role in determining the practical effectiveness of FFR and ETFR.

5.2 User Study

The user study was conducted over ten days and collected 15 valid responses. Among the participants, three wore glasses; however, no participant reported difficulties evaluating visual quality, since the headset could be worn comfortably with glasses.

In total, ten participants were male and five were female. The age range was from 18 to 54 years: eight participants were 18–24, five were 25–34, and two were 45–54. Overall, participants were not familiar with VR gaming. Ten reported that they had never used VR or had only tried it a few times, four used VR occasionally, and one used VR regularly. In contrast, general gaming experience was higher: seven participants reported playing video games regularly, five played occasionally, and three did not play video games.

The frame rate during gameplay was observed to increase when foveated rendering was enabled. To evaluate both perceived quality and performance across different strength levels, five presets were used (see section 4.2).

All questionnaires used in the user study (pre-study, per-preset, and post-study) are provided in Appendix B.

Compared to Andersson and Landén [3], who evaluated three presets, this study expanded the design to five presets in order to examine whether participants could perceive differences between low and high foveation levels for both fixed and eye-tracked approaches. In addition, following the recommendations in [3], the scenario was adjusted to better reflect a practical VR game setting by using a more demanding scene and including a dynamic target to increase task engagement. The order of presets was randomized across participants, consistent with the randomized presentation used in [3], to reduce ordering and learning effects.

Under standard rendering the scene ran at approximately 30 FPS, while FFR Low reached around 40 FPS, FFR High around 50 FPS, ETFR Low around 60 FPS, and ETFR High around 72 FPS.

5.2.1 Game Specific Results

In this section, the participants' ratings of perceived visual quality and performance for each preset are presented. For the rating-based questions, the vertical axis shows the average score on a five-point scale (1–5). For the categorical questions (e.g., *Yes*, *No*, or *Not sure*), the results are shown as the number of participants selecting each answer.

Q1: How clear and sharp was the visual quality in fovea?

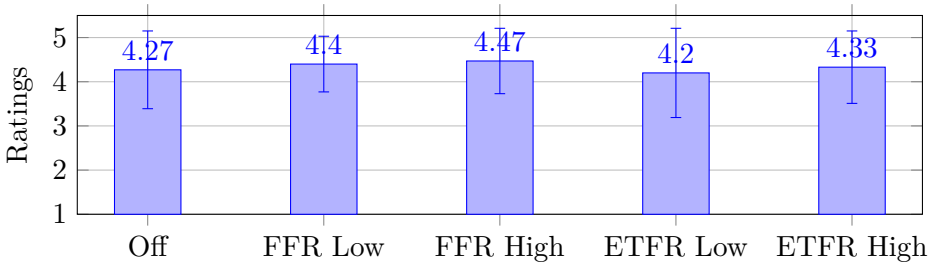


Figure 5.5: Average rating (1–5) of perceived sharpness in the foveal region (1 = not clear, 5 = very clear). Error bars indicate ± one standard deviation.

Q2: How clear and sharp was the visual quality in periphery?

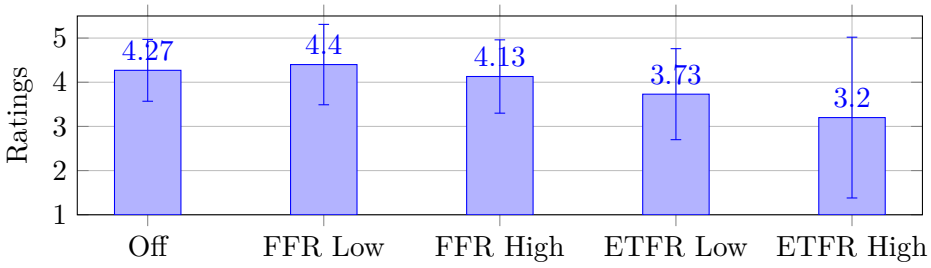


Figure 5.6: Average rating (1–5) of perceived sharpness in the peripheral region (1 = not clear, 5 = very clear). Error bars indicate ± one standard deviation.

Q3: How much blurriness or distortion did you notice during the gameplay?

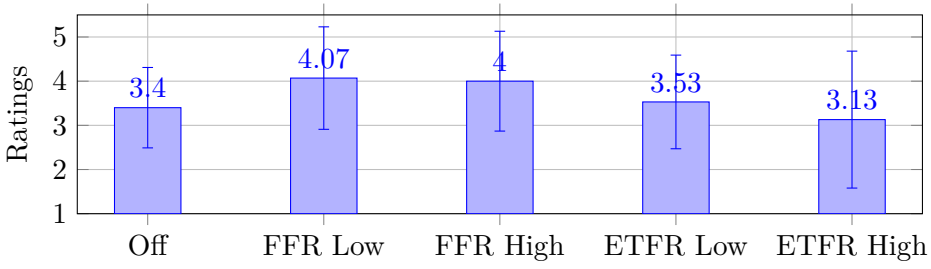


Figure 5.7: Average rating (1–5) of perceived blurriness or distortion during gameplay. Error bars indicate ± one standard deviation.

Q4: Did the blurriness affect your ability to play?

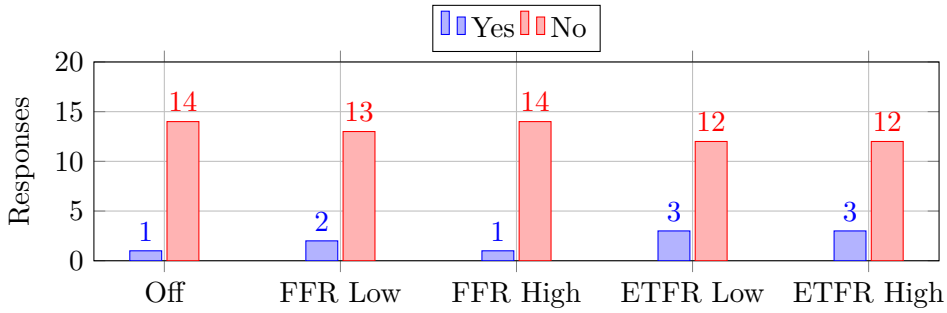


Figure 5.8: Number of participants reporting whether blurriness affected gameplay.

Q5: How much eye strain or fatigue did you feel?

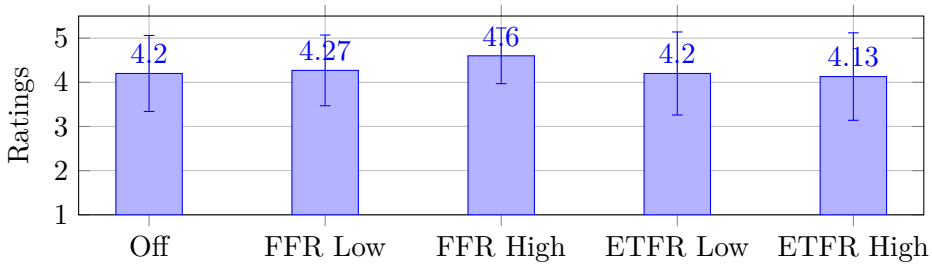


Figure 5.9: Average rating (1–5) of perceived eye strain or fatigue during gameplay for each rendering preset (1 = high, 5 = none). Error bars indicate ± one standard deviation.

Q6: Was any on-screen text (UI, score, instructions) easy to read?

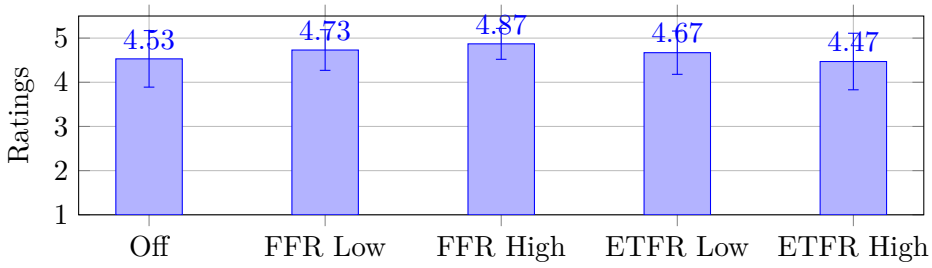


Figure 5.10: Average rating (1–5) of on-screen text readability, including UI elements such as score indicators, instructions, and timers (1 = difficult, 5 = easy). Error bars indicate ± one standard deviation.

Q7: How smooth was the gameplay (framerate perception)?

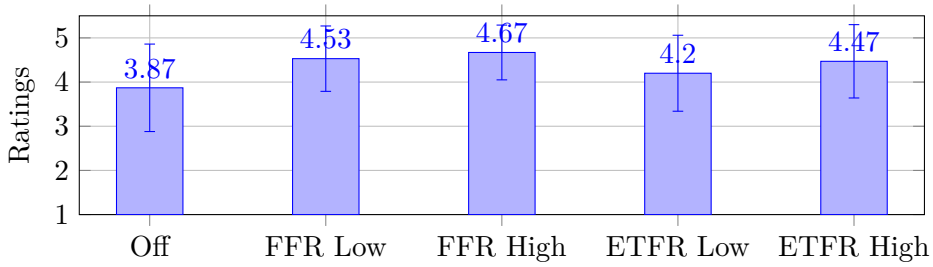


Figure 5.11: Average rating (1–5) of perceived gameplay smoothness across the evaluated techniques (1 = choppy, 5 = smooth). Error bars indicate ± one standard deviation.

Q8: Did you notice visual differences between central and peripheral vision?

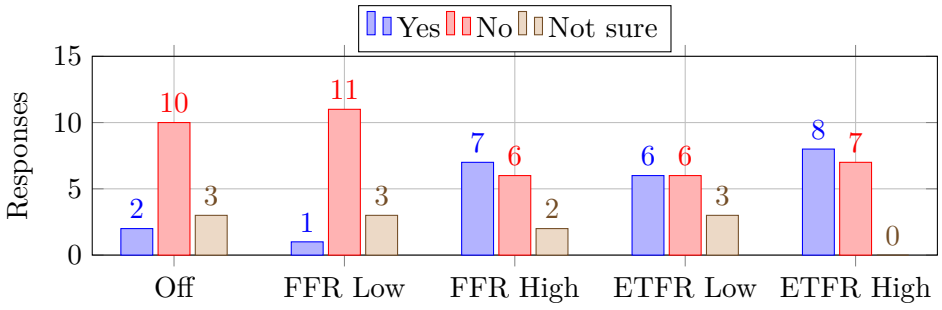


Figure 5.12: Distribution of responses on whether participants noticed a difference between central and peripheral visual quality.

Q9: Did you feel that you adapted to the visual characteristics during the minute of gameplay?

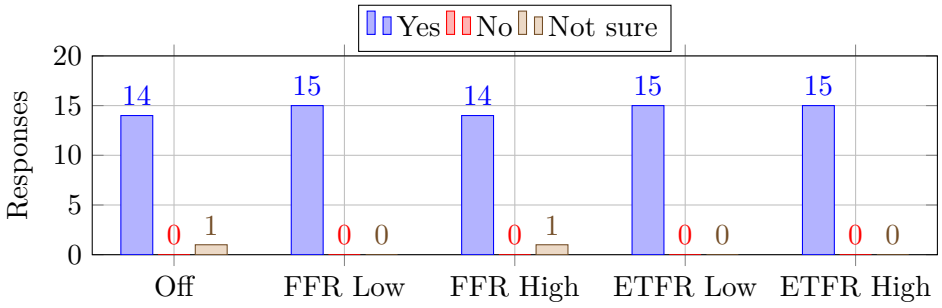


Figure 5.13: Distribution of responses describing how quickly participants adapted to the visual characteristics during one-minute gameplay.

Q10: What is your final rating for this preset?

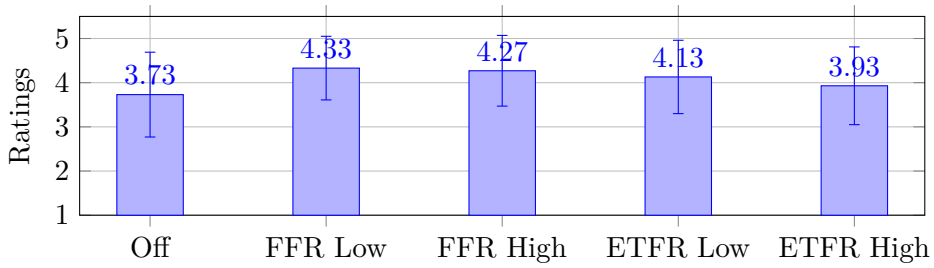


Figure 5.14: Average final rating (1–5) assigned to each evaluated technique (1 = bad, 5 = good). Error bars indicate ± one standard deviation.

Participants rated each preset on a 1–5 scale, where higher values indicated better perceived quality or comfort (e.g., higher sharpness, less blurriness, less eye strain, easier text readability, and smoother gameplay). Three questions were categorical and are therefore reported as response counts (Fig. 5.8, Fig. 5.12, Fig. 5.13). Overall, ratings for foveal sharpness were consistently high and very similar across presets (Fig. 5.5), suggesting that participants generally perceived the centre of view as clear even when foveation was enabled. Perceived peripheral sharpness shows a clearer separation between techniques: while fixed foveation remains close to the Off baseline, eye-tracked foveation decreases the peripheral sharpness score, with ETFR High receiving the lowest average value (Fig. 5.6). A similar pattern is visible for perceived blurriness: fixed foveation obtains higher (better) scores than Off, whereas ETFR High shows a lower score, indicating more noticeable blur or distortion in the scene (Fig. 5.7). Despite this, most participants reported that blurriness did not affect their ability to play across all presets (Fig. 5.8).

Comfort and usability ratings were also consistently positive. Eye strain scores are high for all presets and differ only slightly, with FFR High showing the best average result (Fig. 5.9). Text readability is rated very high across all techniques, again with only small differences between presets (Fig. 5.10). Perceived gameplay smoothness improves compared to Off for all foveated modes, with the strongest increase for fixed foveation (Fig. 5.11). Participants more often reported noticing a difference between central and peripheral quality in the stronger foveation presets (Fig. 5.12), but they also reported rapid adaptation: almost all responses indicate that participants adapted within the one-minute gameplay segment (Fig. 5.13). The final overall rating shows a small preference for fixed foveation, where FFR Low and FFR High score slightly above Off and both ETFR presets (Fig. 5.14), suggesting that participants tended to value performance improvements while remaining sensitive to peripheral quality changes in the stronger eye-tracked setting.

5.2.2 Overview of the Games

In this section, the participants' overall comparisons between the evaluated presets are presented using five pie charts. Each chart shows the distribution of responses in percentages, and the value in parentheses indicates the number of participants who selected a given option.

Q1: Did you notice differences between the presets overall?

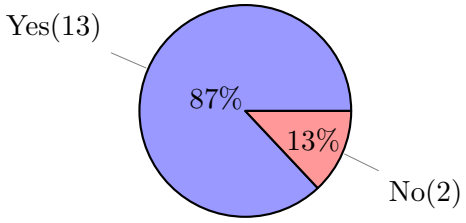


Figure 5.15: Distribution of responses on whether participants noticed differences between the evaluated presets.

Q2: Which preset had the best visual quality?

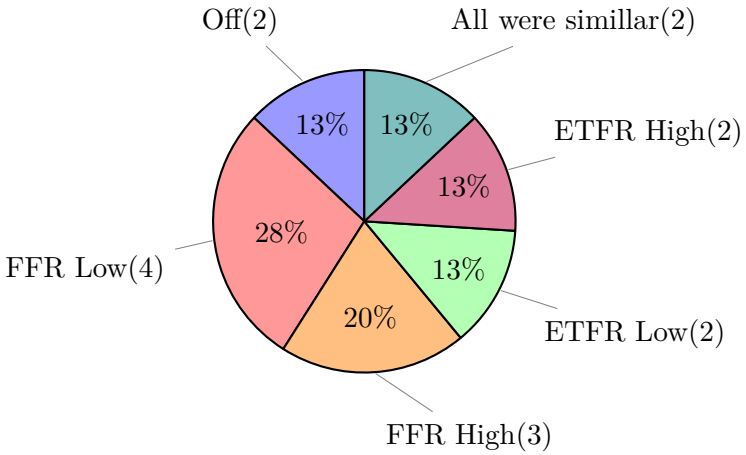


Figure 5.16: Distribution of responses on which preset was rated as the best visual quality.

Q3: Which preset had the best performance?

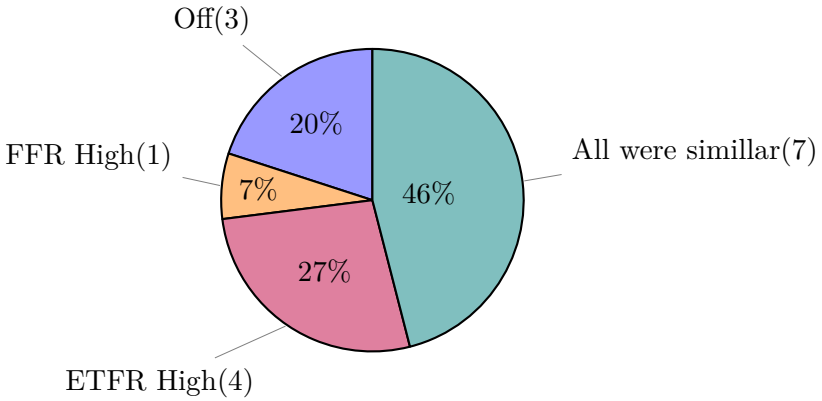


Figure 5.17: Distribution of responses on which preset was rated as the best performance.

Q4: Which preset had the worst visual quality?

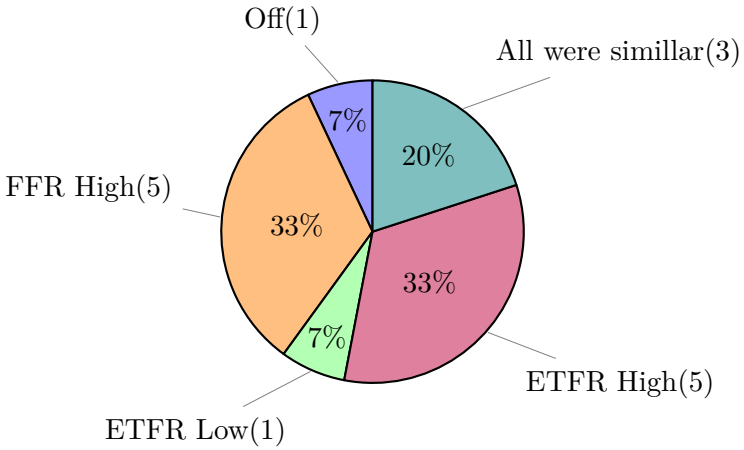


Figure 5.18: Distribution of responses on which preset was rated as the worst visual quality.

Q5: Which preset had the worst performance?

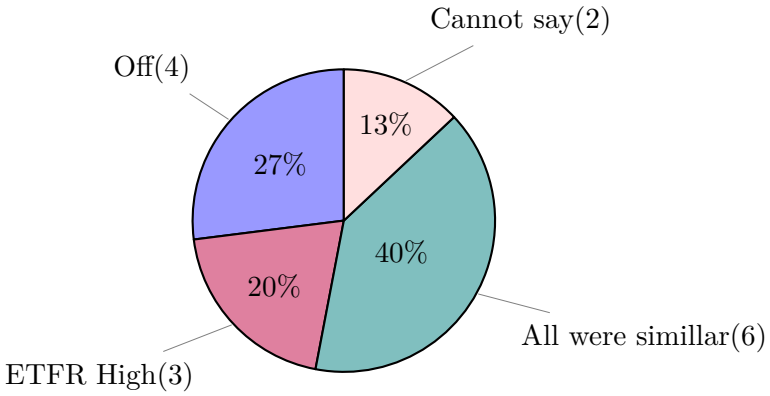


Figure 5.19: Distribution of responses on which preset was rated as the worst performance.

As shown in Fig. 5.15, most participants reported that they noticed differences between the evaluated presets overall (87% answered Yes). However, when participants were asked to select the best and worst presets, the answers suggest that these differences were often subtle and not always strong enough to clearly separate the presets.

For visual quality (Fig. 5.16), the most frequently selected option was FFR Low (28%), followed by FFR High (20%). The remaining options were chosen less often: Off, ETFR Low, ETFR High, and the answer "All were similar" each received 13%. For perceived performance (Fig. 5.17), almost half of the participants selected "All were similar" (46%), while ETFR High was the most common specific choice (27%). Off was chosen by 20%, and FFR High by 7%.

The worst visual quality results (Fig. 5.18) show that the strongest foveation presets were most often associated with lower quality: FFR High and ETFR High both received 33% of the responses. "All were similar" accounted for 20%, while Off and ETFR Low were each selected by 7%. Finally, for worst performance (Fig. 5.19), the most common answer was again "All were similar" (40%). Off was selected by 27%, ETFR High by 20%, and 13% of participants stated that they could not decide.

Chapter 6

Discussion

6.1 Performance impact on PC tests

The PC benchmarks show that the performance benefit of foveated rendering depends strongly on both the workload and the engine integration. In Unreal Engine, foveation produces large FPS gains in the shader-heavy test (Fig. 5.4), while the Geometry scene improves more moderately. This suggests that in Unreal, the foveation presets effectively reduce shading work in the periphery, which is especially important when the workload is GPU-limited by pixel shading.

In Unity, the Geometry scene shows a clear improvement when foveation is enabled (Fig. 5.3), but the shader-oriented scene behaves differently: the average FPS decreases when switching from Off to foveated presets. This indicates that, for the tested shader content, the foveation setup did not reduce the actual shading cost (e.g., missing image-based VRS usage), while it still introduced overhead from the foveation logic. As a result, the benefit is scene-dependent, and Unity gains are most visible in the geometry-heavy workload rather than in the shader test.

6.2 Performance impact on VR tests

The VR benchmarks follow the same general trend, but the absolute FPS is lower due to the higher rendering cost of stereoscopic VR. In Unity VR, both Shader and Geometry scenes gain performance with foveation (Fig. 5.1), and ETFR scales more strongly than FFR, especially in the Geometry workload. In Unreal VR, the Geometry scene shows smaller improvements for FFR but a larger gain for ETFR (Fig. 5.2). The Shader scene in Unreal VR scales strongly for both FFR and ETFR, which is consistent with foveation reducing per-pixel workload when shading cost is a dominant factor.

Overall, the VR results indicate that ETFR can provide additional performance benefits beyond FFR, but these benefits are more visible when the rendering pipeline can convert peripheral quality reduction into real GPU savings.

6.3 Perceived visual quality and comfort in the user study

The rating-based results show that participants generally reported high visual quality across presets. Ratings for foveal sharpness remain close across all techniques (Fig. 5.5), while peripheral sharpness decreases for stronger ETFR settings (Fig. 5.6). The blurriness scores follow a similar pattern (Fig. 5.7), but most participants still reported that blurriness did not affect gameplay (Fig. 5.8). Eye strain scores are also similar across presets (Fig. 5.9), and on-screen text remained easy to read in all cases (Fig. 5.10). Perceived smoothness improves when foveation is enabled (Fig. 5.11), which matches the measured FPS gains in the VR benchmarks.

The overall preference questions show mixed outcomes. Many participants reported noticing differences between presets (Fig. 5.15), but responses about the best and worst presets are distributed across multiple answers, and several participants selected "All were similar" (Fig. 5.16–Fig. 5.19). This suggests that even when participants perceived changes, they often did not form a strong and consistent preference for a single technique.

6.4 FFR versus ETFR in practical use

Across the performance tests, ETFR often achieves higher FPS than FFR at the same preset level (e.g., Table 5.1 and Table 5.2). A reason is that ETFR can keep the full-resolution (1×1) region tighter around the current gaze point, while allowing a larger part of the periphery to use lower shading rates such as 1×16. This reduces the overall shading cost more effectively than a fixed mask, especially in scenes where pixel shading is expensive.

However, the user study results also show a trade-off: stronger ETFR settings reduce peripheral sharpness more than FFR (Fig. 5.6). Even so, most participants reported that this did not prevent them from playing (Fig. 5.8), and the final preset ratings remain close (Fig. 5.14). In practice, this means that ETFR can provide extra performance, but it must be tuned carefully to avoid noticeable peripheral degradation.

6.5 Limitations and validity considerations

This work has several limitations that should be considered when interpreting the results. First, the user study involved 15 participants and relatively short play sessions per preset, which limits statistical power and may hide small differences in perception. Second, the evaluation focuses on a specific FPS-style scenario and a limited set of scenes, so results may not generalize to other genres or visual styles. Third, the measured performance depends on the specific engine configurations and hardware, meaning that changes in VRS support, eye-tracking quality, or rendering settings could change the

observed gains.

To reduce ordering effects, the preset order was randomized in the study, similarly to the approach used by Andersson and Landén [3]. In addition, the study design was extended compared to their work by using a more complex scene and a dynamic target, and by evaluating five presets instead of three. Even with these improvements, future studies with larger and more diverse participant groups, longer exposure times, and additional game scenarios would provide stronger evidence about perceptual thresholds and practical deployment guidelines.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis evaluated fixed foveated rendering (FFR) and eye-tracked foveated rendering (ETFR) using two complementary approaches: (i) performance benchmarks on PC and VR, and (ii) a user study in a fast-paced shooting-range scenario with five presets (Off, FFR Low, FFR High, ETFR Low, ETFR High), presented in randomized order.

The performance measurements show that foveated rendering can provide clear frame-rate improvements, but the magnitude depends on both the engine and the workload type. Geometry-heavy scenes generally gained moderate performance improvements, while shader-dominated workloads (Shader) could benefit much more when the rendering pipeline effectively reduces peripheral shading cost (e.g., via image-based VRS). At the same time, the Unity PC shader experiment demonstrates an important limitation: when image-based VRS is not applied to the tested content, enabling foveation may introduce overhead without reducing shading work, which can reduce FPS instead of improving it.

The user study results indicate that, under the tested conditions, most participants did not reliably distinguish between the presets, and the overall subjective ratings remained close across the techniques. When participants reported noticing differences, the preferences were not strong, and the results suggest that moderate foveation can be deployed without a clear negative impact on perceived quality for this type of gameplay. Together, these findings support the practical use of foveated rendering as a performance tool in VR, while highlighting that its effectiveness depends on correct integration with the engine-level rendering pipeline and the specific scene content.

7.2 Future Work

If further research on foveated rendering is performed, an important direction is to explore different mask configurations and transition strategies. In this thesis, the ETFR mask used a fixed-sized high-quality region and a transition layer; however, these parameters can be varied systematically. For

example, the full-resolution area could be reduced, the transition layer could be removed or split into multiple steps, and the peripheral rate could be made more aggressive. This could increase performance further, but it should be tested carefully to find the smallest acceptable mask that does not introduce distracting artifacts. Another related topic is to study the impact of gaze latency and prediction, since ETFR quality depends on how quickly the mask follows the user's eyes, especially during rapid saccades.

A second direction is to increase the gameplay difficulty and visual complexity to better represent real VR games. Although this thesis already introduced a moving target and a more populated scene compared to the baseline approach in related work, future studies could use more dynamic and stressful situations, such as targets moving towards the player at different speeds, more frequent head movement, and stronger time pressure. It would also be useful to test different game genres (e.g., puzzle, exploration, or racing), because each genre has different visual attention patterns and different tolerance to peripheral blur.

Finally, future work should include a larger and more diverse participant group and longer play sessions. More participants with regular VR experience could provide more reliable results, and longer exposure could reveal effects that may not appear in short trials, such as fatigue, adaptation, or delayed discomfort. It would also be beneficial to combine subjective questionnaires with additional objective measures, such as logging gaze stability, head motion, and frame-time statistics, and to test more hardware configurations to check how well the findings generalize across headsets and eye-tracking systems.

Bibliography

[1] R. Rosenholtz, "Capabilities and limitations of peripheral vision," *Annual review of vision science*, vol. 2, no. 1, pp. 437–457, 2016.

[2] L. Wang, X. Shi, and Y. Liu, "Foveated rendering: A state-of-the-art survey," *Computational visual media*, vol. 9, no. 2, pp. 195–228, 2023.

[3] K. Andersson and E. Landén, "The impact of foveated rendering as used for head-mounted displays in interactive video games," 2023.

[4] "Foveated Rendering - OpenXR Toolkit," https://mbucchia.github.io/OpenXR-Toolkit/fr.html, OpenXR Toolkit, accessed: January 5, 2025.

[5] NVIDIA, "Easy VRS Integration with Eye Tracking," https://developer.nvidia.com/blog/vrs-wrapper/, May 2019, accessed: January 5, 2025.

[6] "Gazepoint GP3," https://www.gazept.com/product/gazepoint-gp3-eye-tracker/, Gazepoint, accessed: January 5, 2025.

[7] "Meta Quest Pro," https://bryle-pro-virtualni-realitu.heureka.cz/meta-quest-pro-256gb/, Meta, accessed: January 5, 2025.

[8] "Oculus Pro controllers," https://www.meta.com/quest/accessories/quest-touch-pro-controllers-and-charging-dock/?srsltid=AfmBOoqzFjIIGvhCa0_YI1xyFMAP2X37Uf9A7Yc0amQ9TxVwb8JYu0dQ, Meta, accessed: January 5, 2025.

[9] "Keyboard and mouse," https://www.fab.com/listings/d35f1b79-cc72-485c-947c-fbe524273254, Open World Development, accessed: January 5, 2025.

[10] H. Strasburger, I. Rentschler, and M. Jüttner, "Peripheral vision and pattern recognition: A review," *Journal of vision*, vol. 11, no. 5, pp. 13–13, 2011.

[11] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn, "Towards foveated rendering for gaze-tracked virtual reality," *ACM Transactions On Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016.

[12] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, "Foveated 3D graphics," *ACM transactions on Graphics (TOG)*, vol. 31, no. 6, pp. 1–10, 2012.

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

[14] Microsoft Corporation, "What is directx 12?" 2021, accessed: January 5, 2025. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/direct3d12/what-is-directx-12-

[15] Blanco, Victor, "What is vulkan?" 2020, accessed: January 5, 2025. [Online]. Available: https://vkguide.dev/docs/introduction/vulkan_overview/

[16] "Variable-rate shading," https://learn.microsoft.com/en-us/windows/win32/direct3d12/vrs, Microsoft, 2023, accessed: January 5, 2025.

[17] M. Weier, T. Roth, E. Kruijff, A. Hinkenjann, A. Pérard-Gayot, P. Slusallek, and Y. Li, "Foveated real-time ray tracing for head-mounted displays," in *Computer Graphics Forum*, vol. 35, no. 7. Wiley Online Library, 2016, pp. 289–298.

[18] T. Roth, M. Weier, A. Hinkenjann, Y. Li, and P. Slusallek, "An analysis of eye-tracking data in foveated ray tracing," in *2016 IEEE second workshop on eye tracking and visualization (ETVIS)*. IEEE, 2016, pp. 69–73.

[19] X. Meng, R. Du, M. Zwicker, and A. Varshney, "Kernel foveated rendering," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 1, no. 1, pp. 1–20, 2018.

[20] "Official homepage of Unity," https://unity.com/, Unity, accessed: January 5, 2025.

[21] "Official homepage of Unreal Engine," https://www.unrealengine.com/, Unreal Engine, accessed: January 5, 2025.

[22] "Unity version 6000.2.12f1," https://unity.com/releases/editor/whats-new/6000.2.12f1, Unity Technologies, accessed: January 5, 2025.

[23] "Unity version 2022.3.11f1," https://unity.com/releases/editor/whats-new/2022.3.11f1, Unity Technologies, accessed: January 5, 2025.

[24] "Unreal Engine version 5.3.2," https://www.unrealengine.com/en-US/blog/unreal-engine-5-3-is-now-available, Epic Games, Inc., accessed: January 5, 2025.

[25] Meta, "Meta XR All-in-One SDK," https://assetstore.unity.com/packages/tools/integration/meta-xr-all-in-one-sdk-269657, 2025, accessed: January 5, 2025.

[26] Khronos, "Unity OpenXR Plugin," https://docs.unity3d.com/Packages/com.unity.xr.openxr@1.16/manual/index.html, 2025, accessed: January 5, 2025.

[27] Unity, "Universal Render Pipeline," https://docs.unity3d.com/Manual/urp/urp-introduction.html, accessed: January 5, 2025.

[28] Godot, "Rendering in Godot," https://docs.godotengine.org/en/stable/tutorials/rendering/index.html, accessed: January 5, 2025.

[29] ——, "OpenXR Settings in Godot," https://docs.godotengine.org/en/4.4/tutorials/xr/openxr_settings.html, accessed: January 5, 2025.

[30] ——, "XR Support in Godot," https://docs.godotengine.org/en/stable/tutorials/xr/index.html, accessed: January 5, 2025.

[31] ——, "Engine Feature List in Godot," https://docs.godotengine.org/en/stable/about/list_of_features.html, accessed: January 5, 2025.

[32] Unity, "Foveated Rendering in Unity," https://docs.unity3d.com/Manual/xr-foveated-rendering.html, accessed: January 5, 2025.

[33] Epic Games, "XR Performance Features in Unreal Engine," https://dev.epicgames.com/documentation/en-us/unreal-engine/xr-performance-features-in-unreal-engine, accessed: January 5, 2025.

[34] "Customize Prototype Grid Shader (HDRP, URP) ," https://assetstore.unity.com/packages/vfx/shaders/customize-prototype-grid-shader-hdrp-urp-194586, Evgeny Onyanov, accessed: January 5, 2025.

[35] "Trails VFX - URP," https://assetstore.unity.com/packages/vfx/trails-vfx-urp-242574, Vefects, accessed: January 5, 2025.

[36] "Realistic Starter VFX Pack Vol 2 ," https://www.fab.com/listings/ac2818b3-7d35-4cf5-a1af-cbf8ff5c61c1, FX Cat UA, accessed: January 5, 2025.

[37] "Unity Terrain - URP Demo Scene," https://assetstore.unity.com/packages/3d/environments/unity-terrain-urp-demo-scene-213197, Unity Technologies, accessed: January 5, 2025.

[38] "Chestnuts Pack (Aesculus hippocastanum)," https://www.fab.com/listings/d35f1b79-cc72-485c-947c-fbe524273254, Open World Development, accessed: October 22, 2025.

Appendix A
Controls

A.1 VR

On Fig. A.1 :

- X - Disable ETFR mode (press)
- Y - Enable ETFR mode (press)
- B - Raise FR mode (press)
- A - Lower FR mode (press)
- Oculus Button - Recenter headset origin position (hold)
- Menu Button - Start new preset in user study (press)
- Left Thumbstick - (En/Dis)able dynamic depth for gaze point (press)
- Right Thumbstick - (En/Dis)able UI overlay for gaze point and mask visualization (press)

Figure A.1: Picture of the VR controllers [8].

A.2 Desktop

On Fig. A.2 :

- WASD - Movement (push)
- F - (En/Dis)sable ETFR mode (press)
- E - Raise FR mode (press)
- Q - Lower FR mode (press)
- Enter - Start new preset in user study (press)
- R - (En/Dis)able UI overlay for mask visualization (press)
- T - (En/Dis)able UI overlay for gaze point visualization (press)

Figure A.2: Picture of the PC keyboard and mouse [9].

Appendix B
User Study Questions

B.1 Prerequisite Questions

Age:

- 18–24
- 25–34
- 35–44
- 45–54
- 55–64
- 65+
- Prefer not to say

Gender:

- Male
- Female
- Other
- Prefer not to say

Previous experience with VR:

- Never used VR
- Tried VR once or twice
- Use VR occasionally (1–2 times per month)
- Use VR regularly (weekly)
- Prefer not to say

Experience with playing video games:

- Never play video games
- Play occasionally
- Play regularly
- Prefer not to say

B.2 Game Questions

How clear and sharp was the visual quality in fovea?

- 1 – Not clear
- 2
- 3
- 4
- 5 – Very clear

How clear and sharp was the visual quality in periphery?

- 1 – Not clear
- 2
- 3
- 4
- 5 – Very clear

How much blurriness or distortion did you notice during the gameplay?

- 1 – A lot of blurriness
- 2
- 3
- 4
- 5 – No blurriness

Did the blurriness affect your ability to play?

- Yes
- No

How much eye strain or fatigue did you feel?

- 1 – A lot
- 2
- 3
- 4
- 5 – None at all

Was any on-screen text (UI, score, instructions) easy to read?

- 1 – Very difficult
- 2
- 3
- 4
- 5 – Very easy

How smooth was the gameplay (framerate perception)?

- 1 – Very choppy
- 2
- 3
- 4
- 5 – Very smooth

Did you notice visual differences between central and peripheral vision?

- Yes
- No
- Not sure

Did you feel that you adapted to the visual characteristics during the minute of gameplay?

- Yes, quickly
- Yes, slowly
- No
- Not sure

What is your final rating for this preset?

- 1 – Bad
- 2
- 3
- 4
- 5 – Good

Any further comments?

- Open-ended text response

B.3 Final Questions

Did you notice differences between the presets overall?

- Yes
- No
- Hard to say

Which preset had the best visual quality?

- Preset A
- Preset B
- Preset C
- Preset D
- Preset E
- All were similar
- Cannot say

Which preset had the best performance?

- Preset A
- Preset B
- Preset C
- Preset D
- Preset E
- All were similar
- Cannot say

Which preset had the worst visual quality?

- Preset A
- Preset B
- Preset C
- Preset D
- Preset E
- All were similar
- Cannot say

Which preset had the worst performance?

- Preset A
- Preset B
- Preset C
- Preset D
- Preset E
- All were similar
- Cannot say

Any final comments about the overall VR experience?

- Open-ended text response