What are the main hurdles to the adoption of foveated rendering in HMDs?

Joe Down

April 11, 2024

1 Background

Humans direct their gaze towards an object by rotating their eyes such that light reflected by the object viewed falls on the eye's fovea[4]. This is a small region at the centre of the retina with increased cone density[10]: the photoreceptor cells of the eye which send signals to brain when stimulated by light[2]. Additionally, the brain is structured to process this information with a space-variant resolution dedicating most resources to the fovea region[19]. These conditions mean that spatial acuity is strongest for objects viewed within this area[4].

In virtual reality systems, it has been found that delay in frame delivery results in motion sickness, discomfort[17], and decreased performance in certain motor tasks[11]. Since rendering requirements continue to increase as screen reolutions and scene complexities increase, it remains difficult for modern computing devices to render complex VR scenes with low visual latency[18]. Foveated rendering has frequently been proposed as a technique to reduce frame render times and help mitigate this problem. It aims to, in one of a few ways, reduce render quality for parts of a VR scene in the periphery and/or increase render quality within the foveal view[20], exploiting the eye's space-variant resolution. Mohanto et al. [7] specify a range of subcategories for render quality reductions, all falling within 4 main subcategories: adaptive resolution (i.e. render the periphery with lower resolution or some other form of reduced sampling), geometric simplification (i.e. render objects with a less detailed model in the periphery), shading simplification/chromatic degeneration (i.e. simplify lighting simulation for pixels in the periphery), and spatio-temporal deterioration (i.e. partially reusing parts of the frame in the periphery to avoid having to re-render the entire view at each refresh).

To understand existing limitations, a high-level structure of a foveated rendering pipeline needs to be defined. There are two main types of foveated rendering pipeline with slightly different structures: static, and dynamic. Dynamic foveated rendering requires some form of eye tracking capability

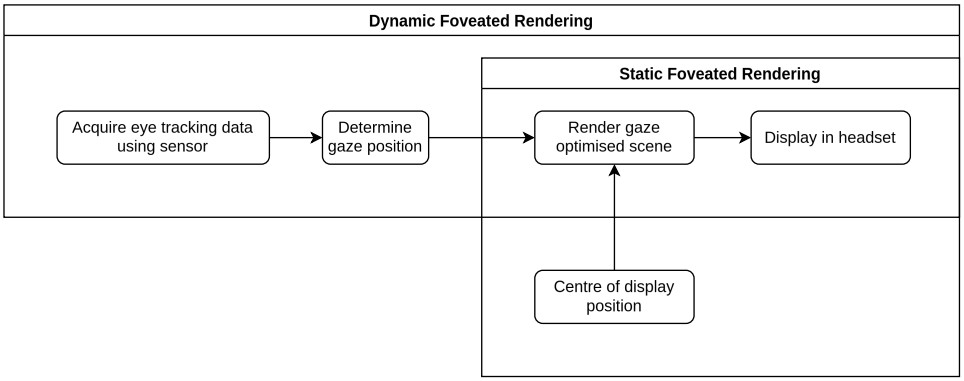

Figure 1: High level foveated rendering pipeline.

within the HMD to determine which part of the stereoscopic displays the corresponding eye is directed towards. In software, frames must be received from this eye tracking device, and processed using some algorithm to determine where the eye is directed within a scene and/or on the physical display. The program being used must then be able to take this information and use it to in some way change how the scene is rendered. The rendered image must then be transmitted back to the headset and displayed, all within a short enough timeframe to avoid noticeable visual latency. Static foveated rendering is mostly the same regarding the software implementation, however rather than requiring eye tracking hardware and processing, it simply assumes the fovea region is in the centre of each display and performs the rendering optimisations accordingly. This is not a realistic model, but will lead to performance gains at the cost of likely noticeable quality loss when the user averts their gaze from the centre. A summary of this pipeline can be viewed in fig. 1.

Static foveated rendering has been used by certain applications with non specialised hardware for years now. Dynamic foveated rendering has only started to become more common within the past few years, as increasingly mainstream HMDs such as PSVR2[**<empty citation>**] and Apple Vision Pro[**<empty citation>**] have been released with eye tracking functionality. The following sections aim to address why dynamic foveated rendering adoption is not yet ubiquitous across all HMDs, and identify shortcomings with existing implementations of both types of foveated rendering.

2 Time Limitations

Most limitations relate to the time taken to perform all of the foveated rendering process and optimisations. If this cannot be done in a shorter amount of time than just rendering an unchanged frame, then the entire

process is redundant as visual latency has been worsened. Additionally, with dynamic foveated rendering, if the foveated region cannot be updated with low enough latency, the issue shown in fig. 2 will occur where the user noticeably looks into the lower quality region of the scene after large, rapid eye movements (saccades). Albert et al. [1] suggest gaze tracking latency for foveation of any level is unnoticeable with 20-40ms latency, becomes difficult to notice with less than 50-70ms of latency, while less aggressive foveation could be tolerated with latencies up to 150ms. Li et al. [5] similarly suggest an upper bound of 50ms latecy, while no paper on the topic seems to aim for a latency below 14ms[3]. In this section, issues will be broken down in the order of the dynamic foveated rendering pipeline as defined in fig. 1. Difficulties faced will be similar for static foveated rendering, ignoring parts relating to eye tracking.

2.1 Eye Tracking Sensor

The first time-based (hardware) consideration is the time taken to acquire sensor data for eye tracking. Stein et al. [12] performed experiments to determine eye tracking delays for three headsets with eye tracking capabilities: FOVE-0, Varjo VR-1, and the HTC Vive Pro Eye. They found that the delay before the eye movement data became available ranged from between 15-52ms. Given the previously described latency requirements for unnoticeable gaze tracking, this leaves little time to process, render, and display within the 20-40ms available. It does leave some room however for less aggressive foveation possible with 50-70ms latency. This suggests that perhaps foveated rendering has not yet become ubiquitous due to current hardware latency difficulties in existing, expensive, headsets.

Stein et al. [12] also test the EyeLink 1000 eye tracker, an external webcam-like device capable of sampling at 1000HZ. From the same experimental setup this had a measurable delay of 0ms, suggesting significant room for improvement if this technology can be miniaturised. FINISH THIS

2.2 Eye Tracking Processing

The second time-based (software) consideration is regarding the processing of eye tracking data from a sensor to determine gaze position. This is often performed using a trained machine learning model. TODO FINISH

2.3 Scene Rendering

The third time-based (based) consideration is regarding the gaze optimised rendering of the scene. TODO FINISH THIS BIT

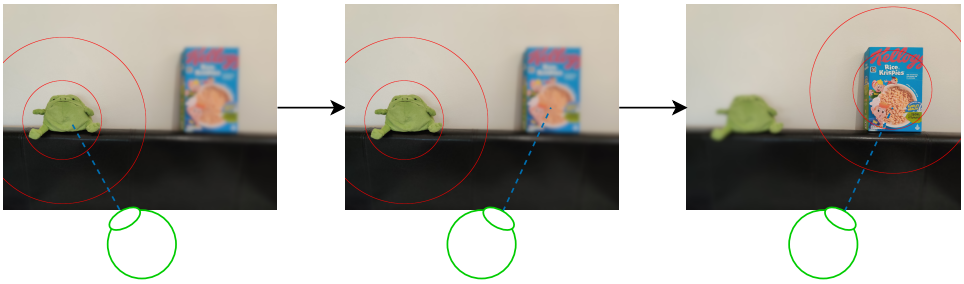

Figure 2: Large eye movements (saccades) result in the eye looking into the lower quality region before the frame updates. If eye tracking or image display is of high latency, this will be noticeable. Based on a similar diagram by Albert et al. [1].

2.4 Displaying in Headset

The final time-based (based) consideration is regarding displaying the image in the headset. Since this must be performed even when not performing foveated rendering it presumably is not a source of problems. I was unable to find any literature referencing foveated rendering specific difficulties with this step. WHAT IS A NORMAL LATENCY

3 Other Software Limitations

A significant hardware limitation is simply adoption. Since only very few headsets currently implement eye tracking of any sort, there is little motivation for developers to implement foveated rendering support if there is not a significant enough user base.

Another potential issue is with conflicting implementation standards for gaze tracking. OpenXR provides extensions for gaze interaction extensions [9], which is implemented by the Unity engine in the XR Interaction Toolkit [13] and Unreal engine through the OpenXREyeTracker plugin[15]. This would seem to imply some sort of standardisation, however I was unable to find clear documentation for how these plugins can be used for foveated rendering. Unity provides the FoveatedRenderingMode render flag[14], however this is not well documented and I couldn't determine whether this is dynamic or static foveated rendering. Vendor specific dynamic foveated rendering Unity implementations seem to exist for Meta Quest Pro[6] and Vive Pro Eye[16], but these will work only for those specific headsets. Additionally, not all headsets support the OpenXR standard at all such as Apple Vision Pro and PSVR[8]. All of these obstacles would seem to make implementation of foveated rendering for a cross-platform application a significant undertaking.

4 Other Hardware Limitations

A significant hardware limitation is simply adoption. Since only very few headsets currently implement eye tracking of any sort, there is little motivation for developers to implement foveated rendering support if there is not a significant enough user base.

COST

EYE TRACKING ACCURACY

5 User Perception

6 Conclusion

References

[1] Rachel Albert et al. "Latency requirements for foveated rendering in virtual reality". In: *ACM Transactions on Applied Perception (TAP)* 14.4 (2017), pp. 1–13.

[2] Detlev Arendt. "Evolution of eyes and photoreceptor cell types." In: *International Journal of Developmental Biology* 47.7-8 (2003), pp. 563–571.

[3] Matias K Koskela et al. "Instantaneous foveated preview for progressive Monte Carlo rendering". In: *Computational Visual Media* 4 (2018), pp. 267–276.

[4] Marc Levoy and Ross Whitaker. "Gaze-directed volume rendering". In: *Proceedings of the 1990 symposium on interactive 3d graphics*. 1990, pp. 217–223.

[5] Richard Li et al. "Optical gaze tracking with spatially-sparse single-pixel detectors". In: *2020 IEEE international symposium on mixed and augmented reality (ISMAR)*. IEEE. 2020, pp. 117–126.

[6] *Meta Foveated Rendering in Unity*. URL: `https://developer.oculus.com/documentation/unity/unity-eye-tracked-foveated-rendering/`.

[7] Bipul Mohanto et al. "An integrative view of foveated rendering". In: *Computers & Graphics* 102 (2022), pp. 474–501.

[8] *OpenXR Conformant Products*. URL: `https://www.khronos.org/conformance/adopters/conformant-products/openxr`.

[9] *OpenXR Gaze Interaction*. URL: `https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_EXT_eye_gaze_interaction`.

[10] Richard J Pumphrey. "The theory of the fovea". In: *Journal of Experimental Biology* 25.3 (1948), pp. 299–312.

[11] Kjetil Raaen and Ivar Kjellmo. "Measuring latency in virtual reality systems". In: *Entertainment Computing-ICEC 2015: 14th International Conference, ICEC 2015, Trondheim, Norway, September 29-Ocotober 2, 2015, Proceedings 14*. Springer. 2015, pp. 457–462.

[12] Niklas Stein et al. "A comparison of eye tracking latencies among several commercial head-mounted displays". In: *i-Perception* 12.1 (2021), p. 2041669520983338.

[13] *Unity XRGazeInteractor*. URL: `https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.4/api/UnityEngine.XR.Interaction.Toolkit.XRGazeInteractor.html`.

[14] *Unreal FoveatedRenderingMode*. URL: `https://docs.unity3d.com/ScriptReference/Rendering.FoveatedRenderingMode.html`.

[15] *Unreal OpenXREyeTracker*. URL: `https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Plugins/OpenXREyeTracker?application_version=5.0`.

[16] *ViveFoveatedRendering*. URL: `https://github.com/ViveSoftware/ViveFoveatedRendering`.

[17] Thomas Waltemate et al. "The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality". In: *Proceedings of the 22nd ACM conference on virtual reality software and technology*. 2016, pp. 27–35.

[18] Lili Wang, Xuehuai Shi, and Yi Liu. "Foveated rendering: A state-of-the-art survey". In: *Computational Visual Media* 9.2 (2023), pp. 195–228.

[19] Cornelius Weber and Jochen Triesch. "Implementations and implications of foveated vision". In: *Recent Patents on Computer Science* 2.1 (2009), pp. 75–85.

[20] Martin Weier et al. "Foveated real-time ray tracing for head-mounted displays". In: *Computer Graphics Forum*. Vol. 35. 7. Wiley Online Library. 2016, pp. 289–298.