JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 38 (2026) 87-94
Received: 2 October 2025
Accepted: 17 November 2025

Analysis of latency reduction and performance improvement methods in selected VR applications

Mateusz Czapczyński*, Krzysztof Dziedzic

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
The aim of the work was to study the impact of two chosen optimization methods on the performance of selected VR applications. Four versions of two Unity applications have been tested, differing by the rendering mode used and the use of multithreaded rendering, and their performance metrics have been compared. Testing was performed using the Meta Quest Pro VR headset. The best overall metrics have been observed for the configuration combining multithreaded rendering with multipass rendering, which achieved a frame rate higher by over 5 frames per second compared to other configurations for one of the applications, and the lowest application GPU time for both applications. However, the use of multiview rendering led to a reduction in average CPU utilization of 6.83 to 9.32 percentage points.

Keywords: virtual reality; optimization; performance analysis; Unity engine

*Corresponding author
Email address: s95380@pollub.edu.pl (M. Czapczyński)

Published under Creative Common License (CC BY 4.0 Int.)

1. Introduction

In the modern world, new methods of utilizing computer technology are rapidly being developed due to constant technological advancement. One of the fields that have developed quickly during the recent decades is the field of virtual reality (VR), as the range of practical uses of this technology was significantly broadened by the growing efficiency of both hardware and software.

Applications utilizing virtual reality offer certain advantages over traditional computer applications, mainly due to the possibility of providing a significantly higher level of immersion, as well as creating new methods of interacting with the virtual world. VR technology can be used in education, in the training of specialists and athletes, as well as in rehabilitation and medicine [1-4]. Moreover, the use of virtual reality in entertainment is becoming more popular and accessible. Popular game engines such as Unreal Engine and Unity can be effectively used to create VR games.

However, more complex use cases tend to lead to higher hardware requirements, which can be more restrictive in the case of virtual reality applications compared to traditional software due to factors such as the need to display a separate image for each eye, as well as the relatively limited computing power of standalone VR headsets compared to personal computers. Additionally, in order to maintain user immersion and prevent the symptoms of cybersickness, a higher emphasis on smooth application performance and latency reduction is required in VR applications [5, 6]. VR systems are also commonly affected by certain practical problems, which are connected to hardware limitations. PCVR headsets that rely on a connection to a computer are not limited by their own computing power, which allows them to handle more demanding applications, but these kinds of solutions have a detrimental effect on convenience and ease of use. Highly performant VR devices also tend to be more expensive, which can be an obstacle for the widespread adoption of VR systems. For these reasons, methods of improving performance in VR applications are becoming increasingly important. Although a large part of the progress on this issue will be tied to improvements in hardware, an equally important avenue of improvement is the development of more performant software that will leverage the capabilities of that hardware in a more efficient manner.

The aim of this article was an analysis of optimization methods used to improve performance and reduce latency in VR applications. In particular, two optimization methods were evaluated: multithreaded rendering and multiview rendering. The following hypotheses were formulated for the purpose of the study:

- H1 – The usage of multiview rendering leads to lower CPU load compared to multipass rendering.
- H2 – Application versions utilizing multithreaded rendering achieve unambiguously better performance metrics compared to counterparts that do not use it.

2. Literature review

For the purposes of the article, a review of literature has been conducted concerning the topic of VR software and its optimization. In the article “The Past, Present, and Future of Virtual and Augmented Reality Research: A Network and Cluster Analysis of the Literature” a review of literature related to VR technology from the recent decades is carried out [7]. Apart from a general rise in interest over time, the increasingly interdisciplinary nature of studies concerning VR is emphasized, along with its broad use in areas such as medicine, rehabilitation and education.

Fariha Nusrat and others in their work “How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects” study the optimization fixes used by software developers in virtual reality projects on GitHub [8]. The authors gathered data related to optimizations in different projects and grouped them into categories, comparing their

results to data related to optimizations in projects not related to VR. Certain categories of optimization methods have been identified that are significantly more commonly used in VR applications compared to other projects. The authors also point out that optimizations in VR projects more commonly come at the cost of modifying other aspects of the project, for example through lowering the detail of certain graphical elements or reducing code readability.

In the article "Analyzing Performance Issues of Virtual Reality Applications" Jason Hogan and others present an analysis of issues related to performance in VR applications created in Unreal Engine [9]. Based on Unreal Engine documentation, as well as fixes introduced in GitHub projects, the authors studied common issues negatively impacting VR application performance. Most of the optimizations found were tied to Unreal Engine project settings stored in configuration files, usually related to display and rendering, although optimizations of C++ source code were also found. The settings most often modified while attempting to increase performance concerned elements such as occlusion culling, ray tracing, pixel density and minimum desired framerate.

The effect of various values of FPS (frames per second) on user experience and performance in VR applications is studied in the article "Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality" [10]. Participants were tasked with completing tasks in two specifically prepared games running at different FPS values: 60, 90, 120 and 180. Data were gathered using objective metrics such as accuracy and reaction time, as well as subjective metrics such as experienced symptoms of cybersickness. The results of the study pointed at significant differences in subjective metrics between different FPS values, where lower values achieved worse results. Data related to objective metrics were less conclusive, but also pointed to a relationship between FPS and metrics such as accuracy. The authors of the study suggest that software developers should aim to create applications running at 120 FPS or more.

In an article by the title "Performance Optimization for Standalone Virtual Reality Headsets", the use of optimization techniques for improving the performance of a specific VR application is described [11]. The authors use an application called "PlayBrics" designed to mimic the functionality of lego bricks in a VR environment. They describe methods of solving the issue of the application slowing down when attempting to build a structure using a large number of blocks. An instanced static mesh is used to significantly reduce the computational cost of creating many identical objects. Additionally, the use of method calls on every object on each frame was replaced with a system of events and delegates in order to reduce CPU load when a large number of objects is present. The final optimization consisted of reducing the detail of models of spheres which contained a high number of polygons, allowing for a reduction in GPU load. As a result of using the described optimizations, the number of blocks that could be simultaneously used in the application without causing a drop in frame rates rose from 120 to 12,100 blocks.

Jan-Philipp Stauffert, Florian Niebling and Marc Erich Latoschik, in their article "Latency and Cybersickness: Impact, Causes, and Measures. A Review" study the existing literature on the topic of the role of latency in VR applications and its effects on the presence of symptoms of cybersickness [12]. Research results showing negative effects of latency on user experience are presented and various methods of measuring latency are discussed. The authors draw attention to the fact, that different methods of measuring latency are used by different studies, and that not all methods are usable with all types of hardware, which makes comparing results difficult. Moreover, latency values are often volatile and depend on hardware configuration and the movement patterns of the user, and are often measured in conditions which do not reflect realistic scenarios of using VR applications. The authors also point out that studies researching the use of VR systems usually do not contain information about the amount of latency present in the studied system, even though their results could be affected by it.

The article "PC VR vs Standalone VR Fully-Immersive Applications: History, Technical Aspects and Performance" describes the history of development of virtual reality tools as well as their categorization, emphasizing the distinction between headsets connected to personal computers (PCVR) and standalone VR headsets (SVR) [13]. The authors point to the performance advantage of PCVR headsets resulting from the ability to use the resources of the computer. However, it is noted that the parameters of SVR headsets are improving as new versions are being developed, which, in combination with their greater ease of use, leads to an increase in their relevance. An example of optimization is also presented by the authors, which allowed an application designed for PCVR equipment to be used on an SVR headset. This was achieved using methods such as static batching, simplifying models and replacing dynamic lighting with static lighting.

In the article "Measuring motion-to-photon latency for sensorimotor experiments with virtual reality systems" the use of VR technology in studying sensorimotor behaviours is discussed [14]. In the context of these kinds of studies, "motion-to-photon" type latency, which consists of delays between the movement of the user and the displayed image, may lead to distortions of the results. Methods of measuring this type of latency often do not account for the impact of motion prediction algorithms, which in the opinion of the authors, limits the possibility of generalizing the results of these kinds of measurements. For the purposes of the article, a method of latency measurement was prepared, which relies on cameras and was designed to work independently of the system being used. Performed experiments have shown that movement prediction leads to significant latency reductions, but its usefulness falls in conditions of sudden acceleration or change of direction, which leads to varying levels of latency in different phases of the movement.

Another exploration of latency in VR can be found in the work "Overview of Motion-to-Photon Latency Reduction for Mitigating VR Sickness", in which the authors emphasize the contribution of "motion-to-photon" type latency to the risk of cybersickness [15]. They also carry out a review of existing methods of measuring latency, dividing methods of measurement into methods working on the level of hardware and software. Several methods of latency reduction are also described, focusing mainly on improvements in data transfer in the context of VR videos, which consist of sending only the data that is currently required by the user considering his current field of view.

In the article "Optimizing Immersion: Analyzing Graphics and Performance Considerations in Unity3D VR Development" a literature review focusing on the development of VR applications using Unity engine is carried out [16]. The importance of balancing visual quality with application performance is highlighted and several methods of optimizing graphics and performance in Unity applications are described. The discussed strategies include utilizing dynamic resolution, as well as level of detail (LOD) and occlusion culling. Object batching and GPU instancing are also recommended as methods of reducing the number of draw calls handled by the application, and the use of texture compression is discussed as a means of optimizing video memory utilization.

Another review of methods relevant to optimizing graphics and performance can be found in the article "Advanced techniques and high-performance computing optimization for real-time rendering" [17]. The work discusses the capabilities of Unity and Unreal Engine in the context of creating performant applications with high quality graphics. Technologies used for enhancing visual detail are discussed, along with various methods of improving rendering performance, such as level of detail, occlusion culling, shader optimization and parallel processing.

3. Materials and methods

The study was carried out using an experimental method and consisted of running and testing selected VR applications while utilizing different optimization methods. Two applications were tested in several configurations of project settings, and performance metrics for each configuration were measured. The gathered data for different application versions were compared in order to determine the impact of each setting on performance.

3.1. Metrics

Data related to the following metrics were collected during the experiment:

- Percentage CPU Utilization
- Percentage GPU Utilization
- Application GPU Time – the time spent by the application on rendering a single frame.
- Stale Frame Count – the number of frames that have not been delivered on time, forcing the application to use an older frame.
- Average FPS – number of frames displayed per second.
- CPU Level – value controlling the processor clock speed. It is set dynamically depending on the current needs of the application in order to balance performance and power usage, which means that it can be useful for identifying CPU bottlenecks. Assumes values from 1 to 5.
- GPU Level – value analogous to CPU level, but applying to GPU clock speed.
- Unique Set Size (USS) – the amount of private memory used by the application, which means memory not shared with other applications.
- Proportional Set Size (PSS) – the total amount of private memory and proportional shared memory. The total amount of memory shared by the application is divided by the number of processes using it, and then added to the amount of private memory used.

The number of frames per second and the number of stale frames are the simplest metrics for evaluating application performance since a low or unstable number of displayed frames directly impacts user experience. Stale frames may also lead to latency. The CPU and GPU utilization metrics as well as application GPU time point at the load caused by the application and can be useful for identifying the sources of performance issues by pointing to which aspects of the application may be causing the relevant performance bottlenecks. Tracking the current CPU level allows for a broader insight into CPU performance compared to only tracking CPU utilization. USS and PSS metrics are useful for tracking the actual amount of memory used by the application, but differ in the exact method of calculating the used memory. Both of these metrics were included in order to provide a clearer view of the application memory usage.

3.2. Studied methods of improving performance

During the experiment, the influence of the following two settings available in the Unity engine for VR projects was tested: multithreaded rendering and rendering mode.

Multithreaded rendering allows the application to move the handling of CPU processes related to graphics rendering to a separate thread, which lightens the burden on the main CPU thread.

Rendering mode determines the method of rendering graphics in a VR application. The first of the available modes is multipass, which involves rendering the image in two separate draw calls, one for each eye. This method is simple and does not cause issues with compatibility, but may lead to higher CPU and GPU load. The second of the available modes is multiview rendering, which renders the image for both eyes using a single draw call through the use of instancing. Due to reducing the workload associated with rendering, particularly for the CPU, it may improve performance, but tends to cause more issues related to compatibility.

The described settings can be found in the Unity project settings. The option for multithreaded rendering can be found in the "Other Settings" section of the "Player" tab, while the rendering mode options can be found under

the “XR Plug-in Management” tab in the “OpenXR” section.

In order to test the selected settings, four versions of each of the tested applications have been prepared utilizing the following configurations:

- Version 1: multiview, multithreaded rendering.
- Version 2: multipass, multithreaded rendering.
- Version 3: multiview, no multithreaded rendering.
- Version 4: multipass, no multithreaded rendering.

The testing of these configurations allowed for evaluating the impact of individual optimizations, as well as the impact of using them concurrently. Apart from the described differences, the tested versions of the applications were identical. For each version, five tests were carried out, each lasting 2 to 3 minutes.

3.3. Research stand

For the purpose of the study, the standalone Meta Quest Pro headset was used, which allows for the use of VR applications without the necessity of connecting to a computer. The headset is characterized by the following parameters:

- Weight – 722 grams.
- RAM – 12 GB
- Storage memory – 256 GB
- Display resolution – 1800x1920 per eye
- Refresh rate – 90 Hz, with an available 72 Hz mode
- Field of view – 106° horizontal and 95.57° vertical
- Inbuilt eye-tracking
- Use of foveated rendering technology
- 6 degrees of freedom (DoF)
- 2 Meta Quest Touch Pro Controllers
- USB-C port
- Qualcomm Snapdragon XR2+ chipset
- Android operating system

The OVR Metrics Tool software was used in order to gather data related to the studied performance metrics during the experiment. OVR Metrics Tool allows for tracking the values of various performance metrics while using VR applications. Two modes are available when using the tool: report mode, which creates a report after the end of a given session, and performance HUD mode, which enables tracking and displaying the values of selected metrics in real time.

The applications tested for the purposes of the experiment were projects utilizing the Unity engine. To modify the applications, version 6000.0.27f1 the Unity platform has been used. Unity is a game engine which allows for the creation of 2D and 3D applications for various platforms, including VR applications.

The first of the tested applications was Polonez VR – a simple application created at the Lublin University of Technology using the Unity engine, intended to aid users in learning the traditional polonaise dance. The application contains information boards that describe the historical and cultural context of the dance as well as the move sequences used in it. The information boards are accompanied by three-dimensional visualizations of the described movements.

The second tested application was XR Interaction Toolkit Examples in version 3.0.7. It is a GitHub project which serves as an example of the use of the XR Interaction Toolkit for the handling of user interaction with objects and interface elements in extended reality (XR) applications created in the Unity engine. The project contains a complex scene containing example objects that react to the actions of the user accounting for physics, as well as complex user interface elements.

4. Results

The results of the performed experiment have been described in this chapter. From the results of all samples for each version of each application averages have been calculated and, in the case of the most relevant metrics, shown on line graphs in relation to time elapsed since the start of the measurement, in order to show the variability of the values over the duration of the experiment.

4.1. Results for the first application

Experiment results for the first of the tested applications – Polonez VR, have been presented on figures 1-3. The graph shown on figure 1 shows how the average CPU utilization changed over the time of the measurement for samples of each version of the application. It can be noticed that lower CPU utilization was maintained by versions using multiview rendering compared to counterparts using multipass rendering. CPU utilization was also lower for versions using multithreaded rendering compared to other versions, but the difference was less significant.

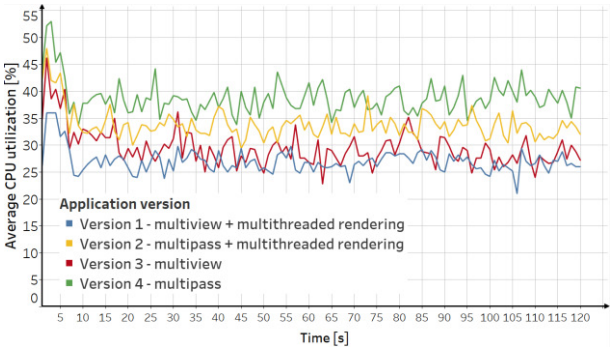

Figure 1: Average CPU utilization percentage over time for the Polonez VR application.

Figures 2 and 3 contain graphs showing respectively the average GPU utilization percentage and the average application GPU time measured in microseconds. Similar values of both metrics were maintained by most versions with the exception of Version 2, as its GPU utilization was around 10 percentage points lower than that of other versions and its GPU time was significantly lower. At the same time, both values were slightly higher for version 4 than for other versions.

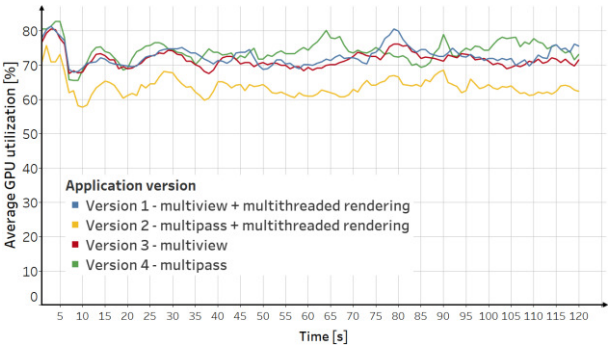

Figure 2: Average GPU utilization percentage over time for the Polonez VR application.

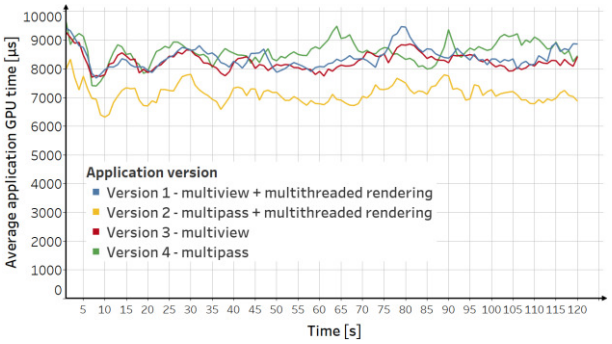

Figure 3: Average application GPU time over time for the Polonez VR application.

Graphs for other metrics measured for this application have not been included as they maintained very similar values over time and did not differ significantly between different versions. However, their average values have been included in table 1.

Table 1: Average values of measured metrics for each version of the Polonez VR application

Metric	Version 1, multiview + multithreaded rendering	Version 2, multipass + multithreaded rendering	Version 3, multiview	Version 4, multipass
CPU utilization [%]	27.00	33.83	29.51	38.83
GPU utilization [%]	72.78	63.77	71.67	74.29
CPU level	2	2	2	2
GPU level	3.04	2.99	3.02	3.10
App GPU time [µs]	8396	7134	8253	8603
Frames per second	72	71.99	71.98	72
Stale frames per second	0.01	0.04	0	0
USS [MB]	904.66	910.40	871.22	873.59
PSS [MB]	913.84	920.25	880.47	884.39

4.2. Results for the second application

Figures 4-9 contain graphs showing results for the second application – XR Interaction Toolkit Examples. In order to make the graphs more readable, a moving average has been applied to the relevant data. This means that each point on the graph represents an average of the value from that second and the values from two previous seconds if applicable. This aims to reduce the effect of short-term fluctuations, which were more common for this application than for the previous one.

The average CPU utilization for each version has been presented on figure 4. Similarly to what was seen for the first application, CPU utilization was lower for versions using multiview rendering. There was no noticeable difference between version that used multithreaded rendering and versions that did not use it.

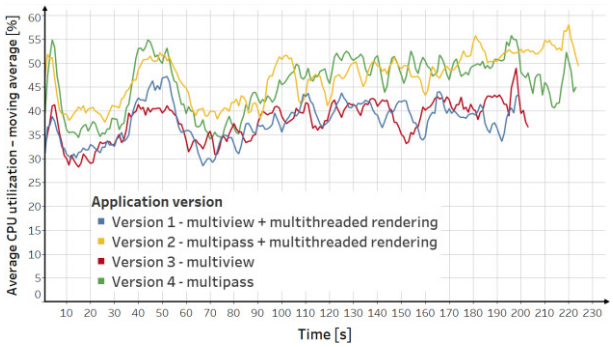

Figure 4: Average CPU utilization percentage over time for the XR Interaction Toolkit Examples application.

Figure 5 shows a graph of the average CPU level of the application. Very high fluctuations can be noticed for all configurations, but versions using multithreaded rendering generally maintained lower CPU levels. Values were lowest for version 2 and highest for version 4.

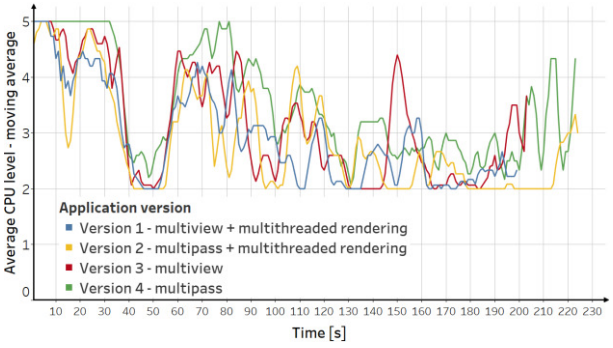

Figure 5: Average CPU level over time for the XR Interaction Toolkit Examples application.

The GPU utilization graph on figure 6 shows that high average GPU utilization was maintained by all versions of the application. Slightly lower values were observed for version 4 in the earlier periods of the measurement.

Figure 6: Average GPU utilization percentage over time for the XR Interaction Toolkit Examples application.

A graph of average application GPU time has been included on figure 7. The values for all versions frequently exceeded the threshold of 13,888 microseconds, which is roughly equal to the longest time in which a single frame can be rendered that still allows for displaying the targeted 72 frames per second. This means that the GPU was commonly not able to render the required frames on time. The lowest GPU time was recorded for version 2.

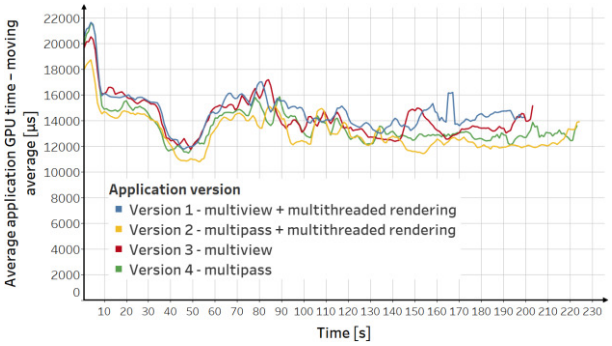

Figure 7: Average application GPU time over time for the XR Interaction Toolkit Examples application.

The graphs shown on figures 8 and 9 contain information about the average frame rate and the average number of stale frames per second. High fluctuations in frame rate can be seen, with significant amounts of stale frames and with all versions failing to maintain the desired rate of 72 frames per second. The highest number of frames per second and the lowest number of stale frames have been observed for version 2, but the values still suggest poor application performance. The other versions seem to have maintained similar FPS values.

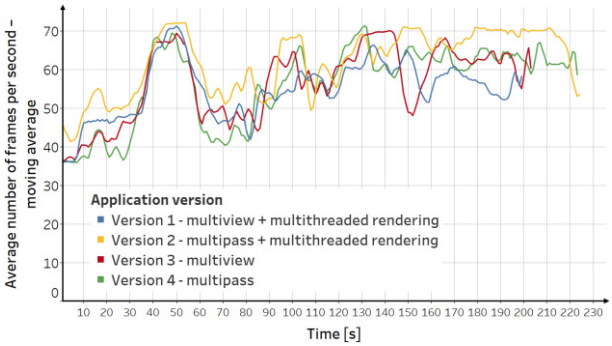

Figure 8: Average number of frames per second over time for the XR Interaction Toolkit Examples application.

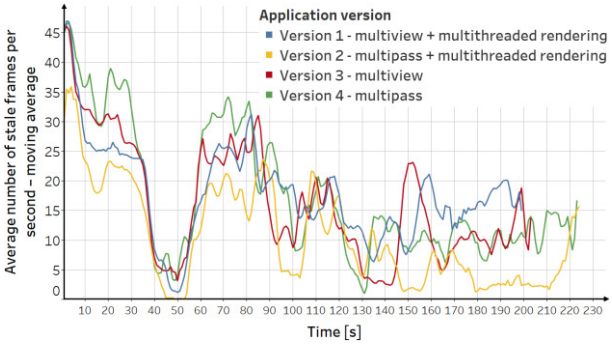

Figure 9: Average number of stale frames per second over time for the XR Interaction Toolkit Examples application.

The average values of all of the measured metrics for this application have been included in table 2.

Table 2: Average values of measured metrics for each version of the XR Interaction Toolkit Examples application

Metric	Version 1, multiview + multithreaded rendering	Version 2, multipass + multithreaded rendering	Version 3, multiview	Version 4, multipass
CPU utilization [%]	37.68	46.77	37.82	45.32
GPU utilization [%]	89.64	90.09	89.27	85.24
CPU level	2.96	2.85	3.16	3.55
GPU level	4	4	4	4
App GPU time [µs]	14737	13050	14275	13689
Frames per second	55.09	61.97	56.59	55.58
Stale frames per second	17.79	10.82	16.51	18.11
USS [MB]	1213.17	1146.27	1196.52	1119.82
PSS [MB]	1224.01	1156.89	1206.93	1131.01

5. Discussion

After comparing the results for all versions of both applications several observations can be formulated. For both applications, versions using multiview rendering were characterized by a significantly lower average CPU utilization percentage. Differences between these versions and versions using multipass rendering ranged from 6.83 percentage points in the case of versions of the Polonez VR application using multithreaded rendering, to 9.32 percentage points in versions of the same application that did not use multithreaded rendering.

Versions using multithreaded rendering achieved a lower CPU utilization percentage for the first application, with differences ranging from 2.51 to 5 percentage points. A similar result was not observed for the second application.

For both applications average application GPU times were significantly lower in version 2, which combined multipass rendering and multithreaded rendering, compared to other versions. In the case of the first application, the GPU utilization percentage for this configuration was also lower, with the differences in the average values ranging from 7.9 to 10.52 percentage points. In the case of the second application, this version achieved the highest average frame rate and the lowest number of stale frames, while also maintaining the lowest average CPU level.

Small differences in memory usage between different configurations were shown by the USS and PSS metrics. For the first application, around 35 MB of additional memory was used by variants using multithreaded rendering compared to other versions, which amounts to a 4% increase. For variants of the second application, the highest memory usage was observed in version 1, which used multiview rendering and multithreaded rendering, while the lowest was observed in version 4, which used multipass rendering without multithreaded rendering.

For configurations where multithreaded rendering was not utilized, better performance metrics were

generally achieved by versions using multiview rendering. In the case of the first application, version 3 maintained much lower CPU utilization, as well as somewhat lower values of GPU utilization and GPU time. In the case of the second application, a slightly higher frame rate was observed for version 3, along with lower average CPU utilization and CPU level. Although GPU utilization and GPU time were relatively low in version 4, which used multipass rendering, this was not reflected by a higher frame rate.

When observing the changes of measured metrics over time, certain relationships between them can be noticed. In the case of the first application, which maintained the targeted frame rate, the GPU utilization graph was almost identical to the graph of application GPU time, but an opposite relationship can be observed for the second application. With the exception of a short period near the beginning of the measurement, during which high values of GPU time as well as GPU and CPU utilization could be observed, drops in application GPU time for specific versions appeared concurrently with rises in GPU utilization. This also coincided with rises in frame rate, as well as lower CPU levels and a higher CPU utilization percentage.

High GPU utilization combined with frequent drops in frame rate during the tests of the second application, as well as the fact, that the configuration which achieved the highest frame rate also had the lowest GPU utilization in the case of the first application, could suggest that the observed drops in performance resulted mainly from overly high GPU load, which was reduced through the use of settings utilized in version 2. However, this interpretation would not be consistent with the observations described above. Periods during which GPU utilization was highest on average were also characterized by the highest number of frames per second and the lowest frame rendering time. Drops in performance also tended to coincide with rises in CPU level. This might suggest that the performance drops were mainly a result of bottlenecks related to the work of the CPU or to communication between the CPU and GPU, which prevented the capability of the GPU from being properly utilized. This kind of bottleneck would not necessarily be reflected in overall CPU utilization metrics, as it could involve only specific processes that are carried out sequentially and do not involve the entire CPU. Since CPU level is set dynamically depending on application requirements, this kind of situation would lead to an increase in CPU level in order to increase the CPU clock rate, which would in turn cause a drop in overall CPU utilization percentage, as increased clock speed would lower the relative load. This interpretation is supported by the fact that for version 2, which achieved the best performance, the lowest average CPU level was observed, perhaps due to it encountering a lower number of CPU bottlenecks. At the same time version 4, for which the highest CPU level and the lowest GPU utilization was observed, also suffered from the highest number of stale frames. However, not all of the encountered drops in performance can be explained by the high CPU level. This is especially visible in the case of version 1, for which a relatively low average CPU level was observed, which did not translate to a high frame rate. The relationship between CPU level and frame rate appeared to be stronger in the earlier parts of the measurements, where CPU levels were often higher, and where both versions which use multithreaded rendering and maintain lower CPU levels achieved better frame rates. Although version 2 maintained this advantage, version 1 performed very poorly in the latter periods of the experiment, despite maintaining a lower CPU level. This points at the possibility that performance was limited by different factors at different points during the experiments, which caused different effects on the results of different application versions. Both CPU bottlenecks and high GPU load seem to have played a relevant role.

To sum up the relationships between the observed metrics and the used configurations, multiview rendering seemed to reduce general CPU load. It also achieved slightly better overall performance metrics in versions not using multithreaded rendering. The lower CPU load might be a result of a lower number of draw calls, as images for both eyes were rendered with a single call.

In versions utilizing multithreaded rendering, slightly lower CPU utilization has been observed for the first application, and significantly lower CPU levels were maintained in the case of the second one, which seems to have had an impact on frame rate during some periods of the experiment. This may be a result of the fact, that multithreaded rendering splits CPU work related to rendering into a separate thread, which might help avoid bottlenecks and prevent the need for raising CPU clock speed, leading to a lower average CPU level.

Application versions combining multithreaded rendering and multipass rendering achieved the best overall performance metrics, including lower GPU time across both applications, lower GPU utilization in the case of the first application, and the lowest CPU level as well as the highest frame rate in the case of the second application. However, relatively high CPU utilization was also observed for these configurations.

6. Conclusions

The formulated hypotheses are only partially supported by the results of the performed tests. For configurations using multiview rendering, significantly lower values of CPU utilization were observed compared to counterparts using multipass rendering, which supports hypothesis H1, although other metrics were not always favorable for these configurations. For application versions that did not use multithreaded rendering, the use of multiview rendering led to slightly better results compared to multipass rendering, but in combination with multithreaded rendering, multipass achieved better results for several metrics, including number of frames per second.

Hypothesis H2, which predicted unambiguously better performance metrics for configurations using multithreaded rendering, has been rejected. Although the configuration with the best performance included multithreaded rendering and the use of this optimization was tied to lowered CPU utilization for the first application

and lower CPU level for the second, the combination of multithreaded rendering and multiview rendering achieved worse results in several metrics compared to its counterpart which did not include multithreaded rendering.

The performed study was affected by certain limitations that make it more difficult to generalise its conclusions. During the tests, only one device was used – the Meta Quest Pro headset, and only two applications were tested, both utilizing the Unity engine. The use of different hardware, a different engine or different project settings could change the effectiveness of the tested optimization methods. However, the results illustrate how relatively simple changes can have a relevant impact on performance in VR applications. Future studies could provide a broader understanding of the topic by carrying out tests covering a broader range of hardware platforms, applications and optimization methods.

References

[1] Y.-H. Qiu, Kai-Hu, X.-J. Luo, Application of Computer Virtual Reality Technology in Modern Sports, In 2013 Third International Conference on Intelligent System Design and Engineering Applications (2013) 362-364, https://doi.org/10.1109/ISDEA.2012.90.

[2] J. Schild, S. Misztal, B. Roth, L. Flock, M. Herkesdorf, K. Weaner, M. Neuberaer, A. Franke, C. Kemp, J. Pranqhofer, S. Seele, H. Buhler, R. Herpers, Applying Multi-User Virtual Reality to Collaborative Medical Training, In 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) (2018) 775-776, https://doi.org/10.1109/VR.2018.8446160.

[3] P. Prashun, G. Hadley, C. Gatzidis, I. Swain, Investigating the Trend of Virtual Reality-Based Stroke Rehabilitation Systems, In 2010 14th International Conference Information Visualisation (2010) 641-647, https://doi.org/10.1109/IV.2010.93.

[4] R. Lege, E. Bonner, Virtual reality in education: The promise, progress, and challenge, The JALT CALL Journal 16(3) (2020) 167-180, https://doi.org/10.29140/jaltcall.v16n3.388.

[5] E. Chang, H. T. Kim, B. Yoo, Virtual Reality Sickness: A Review of Causes and Measurements, International Journal of Human–Computer Interaction 36(17) (2020) 1658-1682, https://doi.org/10.1080/10447318.2020.1778351.

[6] L. Simón-Vicente, S. Rodríguez-Cano, V. Delgado-Benito, V. Ausín-Villaverde, E. Cubo Delgado, Cybersickness. A systematic literature review of adverse effects related to virtual reality, Neurología (English Edition) 39(8) (2024) 701-709, https://doi.org/10.1016/j.nrleng.2022.04.007.

[7] P. Cipresso, I. A. C. Giglioli, M. A. Raya, G. Riva, The Past, Present, and Future of Virtual and Augmented Reality Research: A Network and Cluster Analysis of the Literature, Frontiers in Psychology 9 (2018) 2086, https://doi.org/10.3389/fpsyg.2018.02086.

[8] F. Nusrat, F. Hassan, H. Zhong, X. Wang, How Developers Optimize Virtual Reality Applications: A Study of Optimization Commits in Open Source Unity Projects, In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (2021) 473-485, https://doi.org/10.1109/ICSE43902.2021.00052.

[9] J. Hogan, A. Salo, D. E. Rzig, F. Hassan, B. Maxim, Analyzing Performance Issues of Virtual Reality Applications, arXiv preprint arXiv:2211.02013 (2022), https://doi.org/10.48550/ARXIV.2211.02013.

[10] J. Wang, R. Shi, W. Zheng, W. Xie, D. Kao, H.-N. Liang, Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality, IEEE Transactions on Visualization and Computer Graphics 29(5) (2023) 2478-2488, https://doi.org/10.1109/TVCG.2023.3247057.

[11] Y. S. Sadek Hosny, M. A.-M. Salem, A. Wahby, Performance Optimization for Standalone Virtual Reality Headsets, In 2020 IEEE Graphics and Multimedia (GAME) (2020) 13-18, https://doi.org/10.1109/GAME50158.2020.9315059.

[12] J.-P. Stauffert, F. Niebling, M. E. Latoschik, Latency and Cybersickness: Impact, Causes, and Measures. A Review, Frontiers in Virtual Reality 1 (2020) 582204, https://doi.org/10.3389/frvir.2020.582204.

[13] N. Rendevski, D. Trajcevska, M. Dimovski, K. Velijanovski, A. Popov, N. Emini, D. Velijanovski, PC VR vs Standalone VR Fully-Immersive Applications: History, Technical Aspects and Performance, In 2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST) (2022) 1-4, https://doi.org/10.1109/ICEST55168.2022.9828656.

[14] M. Warburton, M. Mon-Williams, F. Mushtaq, J. R. Morehead, Measuring motion-to-photon latency for sensorimotor experiments with virtual reality systems, Behavior Research Methods 55(7) (2022) 3658-3678, https://doi.org/10.3758/s13428-022-01983-5.

[15] Y. Ryu, E.-S. Ryu, Overview of Motion-to-Photon Latency Reduction for Mitigating VR Sickness, KSII Transactions on Internet and Information Systems 15(7) (2021) 2531-2546, https://doi.org/10.3837/tiis.2021.07.013.

[16] M. Tytarenko, Optimizing Immersion: Analyzing Graphics and Performance Considerations in Unity3D VR Development, Asian Journal of Research in Computer Science 16(4) (2023) 104-114, https://doi.org/10.9734/ajrcos/2023/v16i4374.

[17] Q. Zhang, Advanced techniques and high-performance computing optimization for real-time rendering, Applied and Computational Engineering 90(1) (2024) 14-19, https://doi.org/10.54254/2755-2721/90/2024MELB0061.