Vrije Universiteit Amsterdam

Bachelor's thesis

Evaluating performance of MVEs on VR with an experiment framework

---

Author: Joachim Bose (2736851)

1st and daily supervisor: Jesse Donkervliet
2nd reader: Daniele Bonetta

A thesis submitted in fulfilment of the requirements for the VU Bachelor of Science degree in Computer Science

August 21, 2024

Abstract

Standalone Virtual Reality (VR) devices are new and have received interest from industry (Meta invested 42 billion dollars (1)) and academia due to their potential in digital twins and the 248 billion dollar gaming market (2). To expand on current applications, standalone VR should support modifiable virtual environments (MVEs), where the virtual worlds can be terraformed on a large scale. However, VR has performance requirements to prevent motion sickness and MVEs have strict performance limitations. In this work, we investigate the performance of MVEs in VR by designing, implementing, and conducting experiments using Yardstick-VR, a performance evaluation framework for an MVE in VR called Opencraft2-VR. We found the Meta Quest (MQ) 3 performs similarly to a gaming PC, where its predecessors MQ2 and MQ Pro had, respectively, 183% and 223% more frame times than Meta Quest 3 in a terrain workload. We find a low risk of motion sickness below a world size of 9, where popular (non-VR) Minecraft has 12, and it is limited by terrain synchronization time.

Contents

1

Introduction

Virtual reality (VR) has received a lot of attention from the academic and industrial community, as it is a powerful technology for applications that include games, education (3), and health (4). Furthermore, Meta has invested more than 40 billion dollars in the development of their Meta Quest lineup(1), showing great interest from the industry as well.

Virtual reality technology aims to fully immerse users in virtual worlds. To provide immersion, devices often use a head-mounted display (HMD) and hand tracking to give the user a view of the virtual world through their own eyes while being able to interact with this virtual world with their own hands. Modern VR devices use a standalone VR HMD like Meta's Quest lineup (5) and Apple's Vision Pro (6), which have onboard processors capable of simulating the virtual worlds on its own, rather than offloading this to another device.

The power of the virtual environment can be further increased by using modifiable virtual environments (MVEs). These are (usually) procedurally generated worlds and can be terraformed or modified on a large scale by its users, increasing the possibilities of what such worlds can represent. Existing popular MVEs include Minecraft, which is one of the most popular games in the world (7), which the community has modified and expanded to create entirely new games.

The applications identified for VR devices where MVEs technology could help include virtual training and education worlds specifically designed for the user to perform some exercise. This world may be cheaper and safer than a similar setup in actual reality, allowing more accessible training environments for more users, such as driving (8) where environments could be built procedurally to create an expansive driving simulator or crisis management(9, 10, 11) where buildings can collapse or break to create more realistic scenarios.

Academia also hopes that such trainable skills include teamwork, where multiple users are immersed in the same world, and interact and build on each other to complete a new task. Academia has proposed multiple applications for the so-called multi-user VR (9, 10, 11, 12).

1.1 Problem statement

VR devices are promising in their applications, but they have a flaw: motion sickness is common among users when performance drops below certain thresholds. Moreover, motion-to-photon latency (MTP latency) and frames per second (FPS) have been identified as key metrics in this system (13), where high latency and low FPS induce this motion sickness, making the experience uncomfortable.

(14) identified a knowledge gap about exactly these performance metrics in VR MVEs. Workloads related to MVEs are untested or documented, and we hope to find lessons or improvements for MVEs in VR in this knowledge.

Addressing this knowledge gap is nontrivial due to at least two identified reasons: Firstly, standalone VR hardware is a new device with counterintuitive behavior compared to its predecessors, shown by (14). Secondly, there is no standalone MVE to measure and compare with traditional platforms as the only MVE in VR is the closed-source Minecraft Questcraft, which means that such experiments will first have to be designed and built before addressing these gaps.

1.2 Research questions

This work attempts to help close the research gap explained by 1.1 by answering the following research questions.

RQ1. **How to design an experiment framework for comparing MVE performance on established gaming platforms and a standalone VR headset?** Because established VR MVEs like Minecraft Questcraft are not open source, a new one must be designed together with an experiment framework and workloads for running experiments with the MVE. However, this is not trivial, as workloads for MVE clients are not established.

RQ2. **How to implement such a design?** To properly address the research gap, experiments must be accurate by measuring an MVE running on actual VR devices. To do this, a version of the design from **RQ1** should be implemented. VR devices have a large amount of frameworks and software available, which will have to be considered and integrated into a product, in addition to the numerous errors associated with integrating these software systems.

RQ3. **How do MVEs perform on VR compared to traditional MVE platforms**? A knowledge gap about performance, should ideally be filled with measurements. Such measurements include comparison with other devices to place VR devices into a reference frame, some deeper measurement into pitfalls and performance problems, and some advice for future MVE developers targeting VR MVEs

1.3 Research methodology

To address the research questions in Section 1.2, we use a three-step methodology: we design a system, implement this system, and evaluate performance using experiments and the system. Firstly, we design Yardstick-VR an experiment framework with an application and workloads for evaluation (**RQ1**). Secondly, we implement Opencraft2-VR and a prototype of Yardstick-VR to carry out experiments (**RQ2**). Third, we used our prototype and Opencraft2-VR to obtain experimental results to shrink the knowledge gap discussed in Section 1.1 (**RQ3**).

1.4 Thesis contributions

When exploring the research questions, some contributions useful to a wider community presented themselves:

Knowledge contribution

KC1. **Design of Yardstick-VR** A benchmarking tool to conduct experiments for MVEs on VR. The design tool has an MVE capable of running on different platforms, including standalone VR and HMDs. Additionally, it has support for system and application level metrics.

KC2. **VR device reference frame** We provide a reference frame for performance of VR devices and a gaming PC. This is done through results found through experiments with Yardstick-VR's multi-platform support.

KC3. **Advice for VR MVE developers** Through experiments evaluating different design choices and workloads, we find some actionable insights for developing MVEs (in VR). We summarize and enumerate this advice in Section 5.7.

Technical Contributions:

TC1. **Opencraft2-VR** is a port of Opencraft2 (15) to Meta's Quest lineup of standalone VR devices, and is the first Open-source MVE in VR. The features of Opencraft 2 were also adapted to run on Android, creating an almost full port. It is built on an industry standard technology stack using Unity3d and OculusVR.

TC2. **A prototype of Yardstick-VR** An experiment tool for Android and Linux devices by extending the tools available inside Opencraft 2 and "Measuring the Metaverse"(14), and combining them to create a performance evaluation framework where experiments can be automated.

1.5 Plagiarism declaration

I declare that this thesis is my own original work and has not been assisted by anyone else.

1.6 Thesis structure

Chapter 2 sets the groundwork and foundation for the rest of the work by talking about work in the past and the ecosystem in which this work lives. Chapter 3 is conceptual about the design of the experiment framework Yardstick-VR. Chapter 4 discusses the technical details and technical problems encountered in implementing this design. Chapter 5 discusses the experiments conducted using this design.

If you are a VR developer, read some advice in Section 5.7. Sections 5.2 5.4 and 5.3 may also be of interest as they explore design decisions and MVE performance.

If you are a VR enthusiast, Sections 5.3 on a reference frame for VR devices may be of interest, in addition to Section 4.3.2 on the Opencraft2-VR VR runtime that Opencraft2-VR uses.

2

Background

This chapter introduces virtual reality background with system models and offloading strategies, and follows up by discussing modifiable virtual environment background with common terminology and a system model.

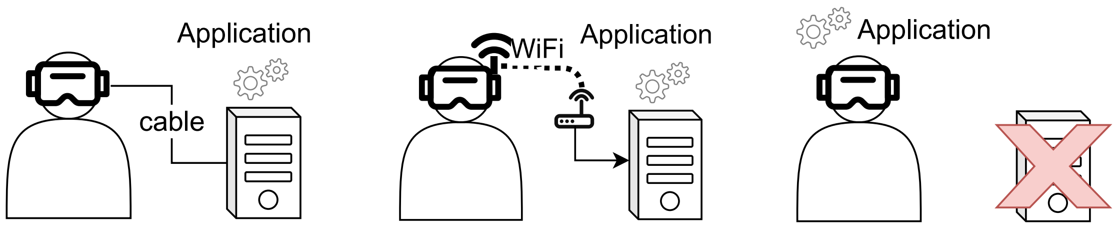
2.1 Virtual reality devices

A VR device is a device that provides the illusion to the user that they exist in a virtual world other than our own simulated by a computer. In this work, we focus on standalone head mounted displays (HMD). The display provides an accurate illusion to the user that they are looking into a virtual world by covering the user's entire field of view. To better meet the need for immersion and user comfort, VR technologists have redesigned the system multiple times over the years to push immersion.

2.1.1 Motion sickness and performance requirements

HMD are associated with motion sickness when the latency between the movement of the user's head and the movement on the screen in the HMD is too large (16). Users start noticing the latency when it increases above 23 ms (17), creating less comfortable and less immersive experiences. Furthermore, the frame rate or frames per second (FPS) should be higher than 120 HZ (18) to minimize symptoms of motion sickness. (19) warns VR developers about the negative effects of frame rate drops on motion sickness, we also define a spread of the frame time.

On modern VR devices such as Meta Quest 3 (MQ3), some tricks are used to mitigate motion sickness with frame rates below 120 HZ and frame times greater than 20 ms. These tricks include Camera time warp (20), Application Space Warp (21), and can fake more



(a) Tethered (i.e., wired) deployment.

(b) Wireless deployment

(c) A standalone HMD

Figure 2.1: Commonly used deployments of HMDs and applications. Applications can be streamed from a remote computer to the VR device using cable 2.1a or Wi-Fi 2.1b, or the application is rendered on the VR device hardware without offloading.

frames per second. The Meta Quest application marketplace guidelines suggest that VR applications should achieve a frame rate of at least or 36 frames per second (FPS) (22) that can then be upscaled with techniques to the desired refresh rate.

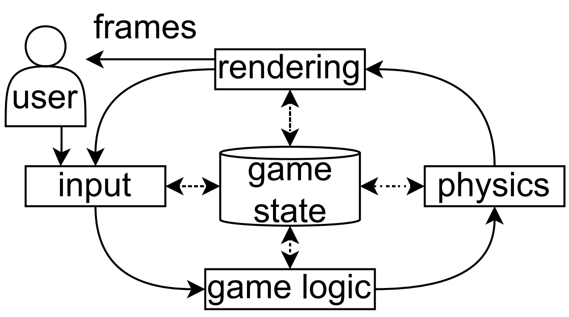
Based on Meta's guidelines, we define a performance requirement called **'par'** which applications should meet to mitigate motion sickness: As a user, I want 99% of the frame times to fall below 1/36 seconds or 27.7 milliseconds. To make par less arbitrary and more accurate, future work is needed to move par in the correct position.

2.1.2 Common VR deployments

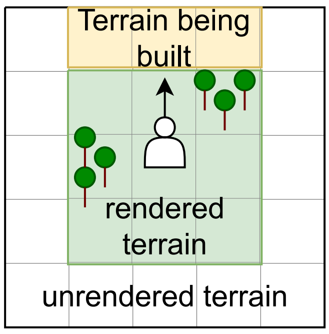
The first HMDs were tethered headsets, which do not have hardware to run games themselves, but require a more powerful computer to render and stream video to the HMD over a cable. Users found the tether and PC requirement uncomfortable, because they can get in the way and are a reminder of the real world around you, decreasing the immersive experience. Multiple efforts were made to remove them while meeting performance requirements.

The first non-tethered HMDs were standalone HMDs (Figure 2.1c), which have processors onboard capable of running the VR application, eliminating the need for offloading. This deployment was experimented with by putting an off-the-shelf phone into a head-mounted box with lenses, utilizing the phone's processor to render and simulate the environment. This was upgraded with hardware integrated into the headset, as in Pico Goblin and Meta Quest.

Due to the onboard processor, these HMDs also allow games to be offloaded over Wi-Fi (Figure 2.1b). This was viewed by many as the best design, because it allows a strong gaming computer to provide high-quality graphics to be sent to the VR device (23).



(a) A sketch of the game loop



(b) Terrain generation in MVEs

Figure 2.2: On the left a system model of an MVE showing the game loop. On the right, a common task in MVEs called terrain generation.

However, the view changed when (14) found that receiving the video stream costs energy and can lead to lower frames per second. In this work, we focus on the latter deployment configuration, where the device stands alone.

2.2 Modifiable virtual environments

Modifiable virtual environments (MVEs) are terraformable (procedurally generated) worlds, where the user can modify the world on a large scale. Minecraft is the most popular MVE, featuring a blocky terrain where the user can add and remove blocks to build nearly infinite structures.

2.2.1 System model

An MVE usually includes multiple components, as we sketched in a system model 2.2. An important part of an MVE is the game loop. This represents the feedback loop with the user, where the input of the user is inputted, processed, and the state of the game is updated.

In an MVE or other game, components of a traditional MVE could look something like Figure 2.2a. The user gives input to the input processor, this input has an effect on the game logic, which may set up some features of the physics engine to be rendered back to the user. The simulator component is defined by the components of the game loop that modify the game state, in our system model, this is all but the rendering component.

To keep the near-infinite terrain processable, an terrain loading protocol is used like the protocol sketched in 2.2b. The terrain is abstracted in a grid of areas or chunks, which can individually be loaded and unloaded when a player, respectively, approaches the area within render distance (a configurable parameter) or moves further than one view distance away.

2.2.2 Multi-user modifiable virtual environments

In Multi-user MVEs, multiple users play in the same world, usually from multiple machines connected to the Internet. The management of the users machines can be done in multiple ways, including the industry standard for games, which is a dedicated server providing service to some player (client) machines. This dedicated server can then simulate the world of the game by processing the input received from the client machines and returning the state of the game necessary to render the world and inform the players (24).

In traditional games, the player machines typically interact with the server through remote procedure calls (RPCs) where the server or the clients can perform a function call across the Internet, which may return some data. However, due to the nondeterministic nature of the internet, return delays larger than 60 milliseconds are not uncommon, making it difficult to keep the state on the client and the state on the server synchronized.

However, in MVEs the networking model is an active area of research (25). In this work, we use a networking model in which networked objects live in a server world and have a mimic in the client world called a ghost. The server then sends state updates over the network to the ghost object to keep the game synchronized.

3

Design of Yardstick-VR

This chapter answers research question **RQ1**: How to design an experiment framework for comparing MVE performance on established gaming platforms and a standalone VR headset? By designing an experiment framework for comparing MVE performance on established gaming platforms and standalone VR headsets called Yardstick-VR. Sections 3.1, 3.2, and 3.3 profile the use and elicit some requirements for Yardstick-VR , Section 3.4 looks at the design in total, and the rest of the sections look at specific components of this design overview.

3.1 Stakeholders

To best satisfy the needs of relevant stakeholders, they are enumerated and profiled below.

ST1. **Researchers**: These stakeholders use experiments and data to provide the world with better and more accessible VR systems and applications. This stakeholder wants to keep track of the state-of-the-art and advance it by producing papers like: (20, 24, 26). To produce these, they want data from scientifically found experiments and reproducibility.

ST2. **MVE Developers**: These stakeholders have project-ideas related to MVEs and may lack knowledge to turn their ideas into a product. These are technically skilled people and built popular MVEs like Minecraft. They have knowledge about technical details, design decisions and their effect on the MVE. To produce value game features and applicability to their products.

3.2 Use cases

The product aims to satisfy the needs of the stakeholders through the following use-cases:

UC1. **Comparing different MVE platforms.** A developer (**ST2**) has an idea for an MVE and wants to make a decision about what device to target or VR runtime to choose for his MVE. They pick up Yardstick-VR or investigate the results **ST1** found in a paper produced using Yardstick-VR. This provides them with a reference frame for every device, assisting in their decision and speeding up the process.

Similarly, researchers (**ST1**) may want to improve the performance of a platform and investigate what holds the platform back compared to other platforms. The researcher can pick up Yardstick-VR and find detailed information on the performance of the application in their target stack and evaluate their improvement on it too, speeding up the process by eliminating the need for the researcher to build Yardstick-VR from scratch.

UC2. **Push MVEs on VR to their limits and improve this limit.** The complexity of the world is related to the strength of the MVE as explained in 1, and thus, **ST2** wants to gain intuition about the feasibility of their idea. They can then pick up Yardstick-VR to gain an intuition about the strength of the state-of-the-art and whether his idea may work.

The **ST1** wants to improve this complexity to provide for **ST2** and advance the field. They pick up Yardstick-VR and see why the application breaks when it breaks. They can then look for improvements in this area to allow better performance.

UC3. **Evaluate effectiveness of deployments.** As **ST1** has new ideas about offloading, they would like to find out if this idea is feasible and improves the performance or complexity of MVEs. Their idea is to offload the game logic and terrain rendering from the HMD to a server over network. They can pick up Yardstick-VR and test the offloading mechanism and compare it with the traditional deployment.

3.3 Requirements

In general, Yardstick-VR is used like an experiment platform; this requires the need for a metric collector ❻, control over the target system, and a workload. We obtain some more detailed requirements (**RE**), and some nonfunctional requirements (**NFE**):

RE1. **Platform support.** To aid in **UC3** multiple platforms must be supported with the same workload. Although the workload to put the system under test may not change, the system under test may change. The underlying layers, such as runtimes, operating systems, and hardware, can be swapped out.

RE2. **Repeatable automated experiments.** As **ST1** wants repeatable experiment data in **UC1** **UC2** and **UC3**, automated experiments are required to reduce human error and assist and reduce researcher **ST1** time per experiment. The entire experiment and all its workload configurations should be runnable with one click of a button to allow other researchers to easily repeat the experiment without much assistance of the original researcher.

RE3. **Configurable workloads.** To push platforms to their limits in **UC2**, workloads within the MVE should have this capability. When workloads are configurable, these parameters should be easily modifiable by users of Yardstick-VR. This allows for experiments to be conducted faster, and decrease the iteration time of the project.

RE4. **Collectable metrics.** As the **ST1** values hard data and in all **UC2** **UC1** **UC3** Yardstick-VR it is used to compare multiple things using performance data. Detailed subcomponent level frame-time and hardware utilization metrics (CPU usage, GPU usage, memory, network usage) should be collectable to find when the system drops below acceptable performance, and why the system drops below acceptable performance.

NFE1. **Perform on par.** In Section 2.1.1 we defined par, the level a modern VR system should at least perform to mitigate motion sickness for its users. It is important that at least one VR system supported by the limits of Yardstick-VR is in the middle between the lowest load that Yardstick-VR can provide and the highest load that Yardstick-VR can provide.

NFE2. **Collector overhead.** The metric collector overhead must be kept low to keep the experiments close to real world scenarios. We define an arbitrary 5% overhead on the metric collector.

3.4 Design overview

Figure 3.1 shows the components of Yardstick-VR. Arrows represent the direction of the data flow; for example, A $\rightarrow$ B means that data flows from A to B. The colored boxes

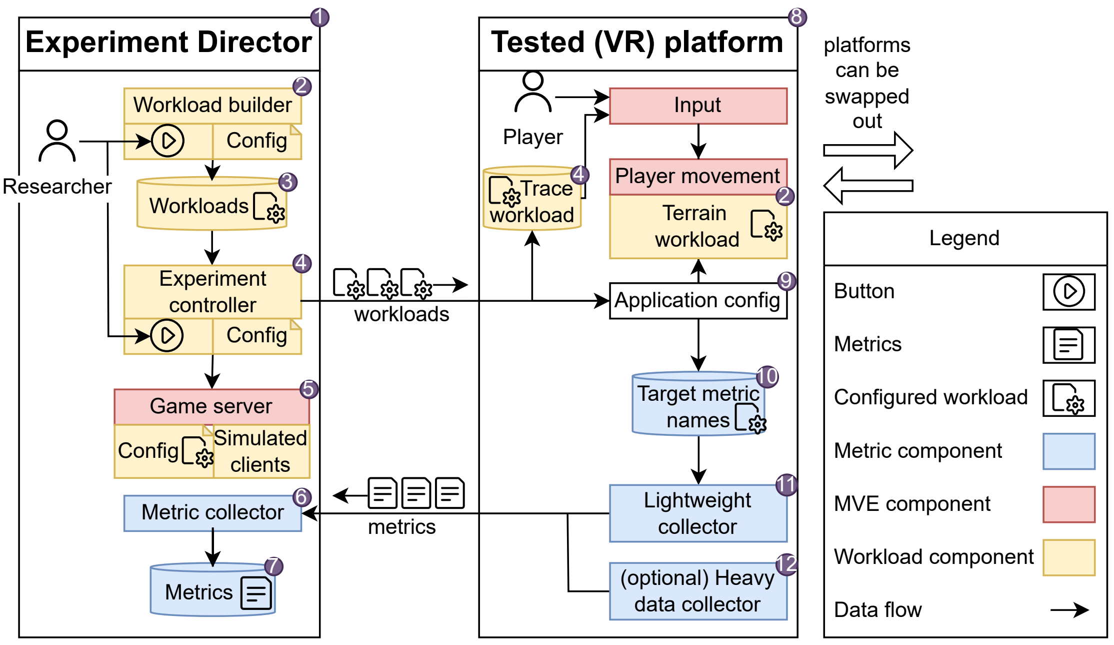


Figure 3.1: Design components

represent higher level parts of Yardstick-VR with a common goal; for example, a yellow box represents a component involved in configurable workloads, a red box represents a box involved in the MVE, and a blue component is one involved in the collection of metrics.

The left shows the experiment director, a device hosting most experiment infrastructure in 4 components: **(1)** The metric receiver ❻, which receives the metrics described in **RE4** from ⓫ and ⓬, and records them into the experiment director's disk (see Section 3.8); **(2)** The Game server (only when required by experiment like in **UC3**); **(3)** And the experiment controller ❹ (see Section 3.5), which starts and stops other components at the right time, this eliminates the need for the researcher to manage multiple tasks, helping **RE2**. **(4)** The workload builder takes a set of workload configurations and builds a set of executables for the target platform when required. This once again saves the researcher (**ST1**) the need to do some work and satisfies **RE2**.

The right shows the experiment subject, an MVE platform (maybe in VR) under test, containing an instance of the MVE (not shown). In this MVE, the handling of player input can optionally be overridden with a trace ❷ or terrain workload ❷. The workload and trace are configured through the application configuration ❾, which serves as the central configuration point where the configuration can be accessed by other components.

The platform under test ❽ can be swapped out by different platforms, such as a gaming PC or a smartphone, to compare these between multiple platforms. This satisfies the requirement **RE1**.

This multi-machine setup was chosen for three reasons: Firstly, this allows parts of the metric collection ❻ and the experiment controller ❹ to connect to the running application without taking too much CPU time away from the subject of the experiment to help with **NFE2**. Secondly, popular VR devices are very restrictive and do not easily allow applications to share files, making it somewhat difficult to export data and results for further processing. Third, **ST1** has to combine the data collected from multiple platforms when using Yardstick-VR in **UC1** to analyze the data; this step is made easier if all data are collected on the experiment director like in ❼.

To look at the overview of the design from a usage perspective, an example of **UC2** is sketched: The researcher configures the experiment builder ❷ for a number of differently configured workloads ❸ that increase in strength. They press the build button and give these executables to the experiment controller ❹ to test. The researcher can then connect their favorite VR device and press the Run button on the experiment controller. The controller loads the workloads onto the tested VR platform, where the application configurator ❾ sets up the metric collectors ⓫ and the trace ❹ or terrain workload ❷. The experiment controller then starts the game server ❺, if required, and a timer. When the timer runs out, the experiment controller loads a new workload and repeats the experiment process until all metrics are collected in ❼.

3.5 experiment controller

This component exists to satisfy **RE2** by automating the entire experiment-running process. The user of Yardstick-VR can give the experiment controller ❹ a set of workloads built with the workload builder. The experiment controller ❹ will then load the workloads onto the target system and start the workload. This task is platform-dependent and may have to be implemented multiple times. After starting the workload, it starts the metric collector ❻ and the game server (when required) and keeps track of the time programmed by the researcher. When time is up, the controller stops these processes and saves the data.

3.6 Terrain workload

To satisfy the requirement, **RE3**, Yardstick-VR is designed with a workload that pushes the terrain-building capability of the MVE running on the target platform to its limits

(**UC2**). The player is elevated above the world and given a constant speed. Then, as the MVE has to keep building terrain around the player while keeping up with a second parameter, render distance (see Section 2.2.1). The speed and view distance parameters do the same; thus, only one has to be configurable.

(24) has found that this terrain building is one of the most challenging tasks in MVE design, which is why it was specifically added as a workload for testing MVEs on our system, where replay of exploration traces tend to get stuck due to small inconsistencies.

3.7 Tracing workloads

As the terrain workload is quite niche, a workload that is more generally applicable and established by previous work (27) is input tracing. The MVE should support creation and replay of input traces, allowing researchers to easily create their own repeatable workloads.

3.8 metric collector

To satisfy the research requirements **RE4** and **RE2** A metric collector ❻ capable of collecting detailed low-level information about frame time with low overhead. Although the researcher wants to publish results measured on a platform with low overhead, to find hypotheses, the researcher typically does not need the low overhead.

The metric collector ❻ for Yardstick-VR has two modes: **(1)** high overhead high detail, used by the researcher to find hypotheses and has to follow scientific laws less strictly, and **(2)** low overhead, low detail mode, which is used by the researcher to create scientifically sound results and verify the hypotheses made earlier.

4

Implementation

In this chapter, we answer the research question **RQ2**: How to implement the design from Chapter 3? By implementing a prototype of Yardstick-VR

4.1 Implementation overview

In Figure 4.1 the overview of the implementation is shown. The right shows the tested VR device running Opencraft2-VR, and the left shows the experiment host running multiple processes. The dotted containers specify groups of components that live in the Unity ecosystem, and the arrows represent data flow.

For our system stack, we target the Meta Quest lineup running a Unity application ❾. We chose this combination because both are widely used in other works. Unity with VR is used in (28, 29, 30, 31, 32) and Quest HMDs are used in (14, 27, 33, 34). Additionally, Meta has more documentation and code available for this specific setup. Unity is a game engine editor and runtime that supports multiple platforms. Unity also has many tools available for extending its functionality that are used by many other components, and the chosen application is also built in Unity.

The Meta Quest lineup are established standalone VR HMDs running on Android, which allows us to use the Android debug bridge (ADB) to interface with them. Using ADB, we can connect experiment director to the HMD via USB for multiple functionalities. These functionalities include ADB logcat, a utility to listen to printed logs from applications on the HMD; ADB shell, which allows the experiment director to execute shell commands on the target HMD, which allows us to install and start programs.

For our application, we extended Opencraft-2 (15). Opencraft2 is fully open source and has been chosen from other options for a few reasons. Firstly, Opencraft2 is not in the

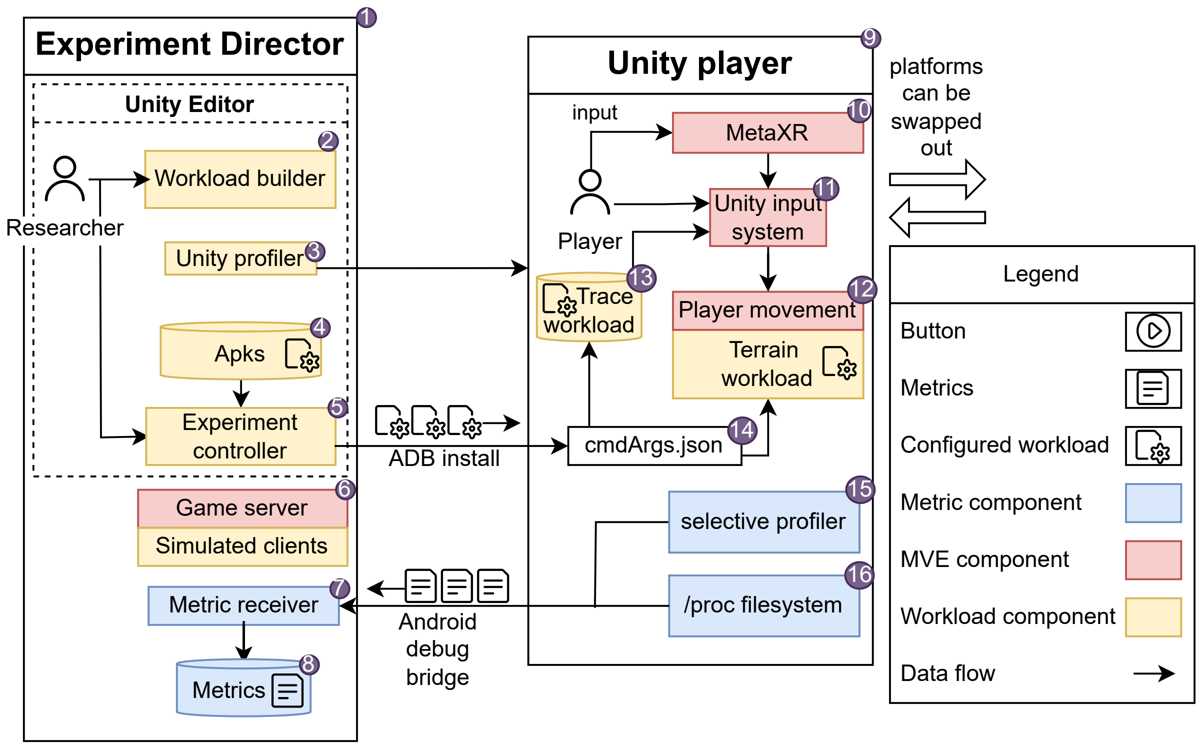


Figure 4.1: Overview of implemented components

Minecraft ecosystem, which does not allow open-source clients, unlike other MLGs such as Glowstone(35) or Opencraft 1 (36). Secondly, Opencraft2 is made for research and supports features including simulated players, advanced data collection methods, and input tracing unlike other open-source MVEs such as Minetest (37) or Terasology (38).

4.2 The experiment controller

The experiment controller 5 lives in the Unity editor by extending the editor with an editor window. Even though other options like bash are more easily developed due to the slow Unity editor, the Unity Profiler was desired for the data collection. This profiler can only be controlled from the *UnityEditorInternal* namespace, which is only available inside an editor script such as one from the UI toolkit.

The implemented experiment controller is practically a loop over every workload provided by the researcher, and an exact implementation of the designed controller in Section 3.5. In addition to this workload, the researcher also provides a set of server configurations corresponding to the workload. The loop looks like this:

1. Start the game server 6 in the correct deployment.

2. Start the workload, if the target platform is Android, first install the workload on through ADB.

3. Start metric receiver ❼.

4. Await a timer while the experiment is running.

5. Stop the metric receiver and save the data ❽.

6. Stop the server gently, to allow proper cleanup of socket data.

7. Stop the workload, if the target is Android, we remove the workload too.

As the Unity Editor is single-threaded, busy waiting in the loop will cease all other functions of the editor during the experiment, including the Unity Profiler ❸ required to collect data. To avoid such a problem in the Unity player, developers usually use a coroutine. This looks like a function call and runs until it hits a yield statement, where execution is suspended. The Unity player then resumes the execution of the caller and resumes the execution of all coroutines at a specific location in the Unity player loop. This creates a nonparallel thread-like behavior, allowing non-busy waiting.

Coroutines are usually not available in Editor scripts like editor windows, luckily Unity provides an extended library to allow the functionality of these coroutines inside the Unity editor. The library is limited compared to the full implementation of the regular coroutines but works very well for our use case.

The interface with the researcher is implemented using the Unity UI toolkit, a framework to extend the functionality of the Unity editor by adding new scriptable editor windows featuring interactable elements such as buttons and input fields. This was used to create a window that features input fields for the path where workloads are stored, a path where servers are stored, and buttons to start the experiment for different targets.

4.3 Opencraft2 and Opencraft2-VR

To create Opencraft2-VR targeting the Meta Quest lineup on the Android operating system, a version running on an Android phone was first created (Subsection 4.3.1) because the phone was more open and problems were easier to fix. From there a VR version was built (see Subsection 4.3.2) which is used by the evaluation section. Some additional features were added to the VR version, which are elaborated in Sections 4.3.3 and 4.3.4.

4.3.1 Android version

Opencraft2 android implements only one new feature, but fixes some porting errors. The feature is command-line arguments packaged with the build. This is required for Android, as command-line arguments are not supported, which is required by Opencraft2 to configure the deployment.

We implemented a feature where the argument parser looks for a special file called "Assets/Resources/cmdArgs.json" and sources its arguments from there if no arguments are detected through the command line. For trace file and deployment file path arguments, the file in question can be placed in the Resources folder, and the filename (without path or extension) can be passed as an argument in cmdArgs.json. The parser will then look for the file in the Resources folder and load it correctly.

As the Unity build system in Unity is all hidden from the user, the platform switch and recompilation should be a one-click functionality. However, due to an internal Unity shader compiler error, the Unity editor kept crashing and corrupting the project. It seemed to be an AMD-GPU driver error and an update of the OS from Windows 10 to 11, which fixed the issue.

At some point, Opencraft2 was running, but nothing was rendered, except the player. This error was fixed more easily by switching the graphics API from Vulkan to OpenGL.

4.3.2 VR version

In the design, Opencraft2-VR is a modification of the player object and the player input system, to fit the new VR input system functionality into Opencraft2. This extension is aided by a library. Once the application was running, problems related to terrain rendering popped up again, as described in 4.3.2.

The VR extension of the player is done using the MetaXR plugin for Unity. This was not the first choice, as MetaXR is not open source and fewer headsets support this plugin compared to OpenXR. Moreover, OpenXR is also supported by (27) and OpenXR has received attention from the academic community. OpenXR seemed to have some input issues and was more difficult to get to work than expected, whereas the MetaXR package provides more assistance for this.

Opencraft-2 utilizes a custom shader to render the terrain and reduce data copy overhead. It took some time to realize where the shader enters the render pipeline and how the terrain was built. Shader compilation errors were difficult to fix with limited knowledge of shaders, and some small things were tried to "wish" the problem away.

Because this approach was not working, a different frontal assault strategy was chosen by choosing to write a new shader. The project contained a second broken shader with much less code, which was taken as a starting point. When implementing the vertex pass, problems started arising when mixing libraries in different shader languages called HLSL and CGINC. These languages have the same syntax but have different standard libraries that use many of the same names. The VR shader libraries extended the definition of the HLSL's standard library, but some were overwritten by the CGINC library, reverting the changes. Replacement of the entire CGINC standard library with only the necessary definitions fixed the issue.

4.3.3 Terrain workload

To implement the designed terrain workload 3.6, we use a simple hook in the player movement code that checks for a command-line parameter, assisted by the command-line parameter framework from the Opencraft2 base. If the flag "-playerFly True" is passed, the player movement is overridden with the designed behavior.

4.3.4 Configurable render distance

Using the same command-line parameter framework, a parameter "-renderDist" was implemented over the base of Opencraft2, which was a terrain configuration baked into the code by the compiler. The base Opencraft2 implementation caused a problem when workloads with more than one render distance were built, as Unity would rebuild large code sections. Changing render distances was sped up from $\tilde{2}0$ minutes to $\tilde{4}0$ seconds.

4.3.5 Terrain building bug

Additional to all the features, the most notable bug (among others) fixed by this work was a bug in terrain generation. The terrain generation works in a few steps, the first of which looks at the position of the players and the completed terrain areas and determines which areas need to be spawned; another system then takes this set and builds them concurrently. There was a problem where duplicates would be added to the list of areas to be built, and multiple threads would build the same area. A preventive fix improves performance.

4.4 Data collectors

Contrary to the design of Yardstick-VR, our prototype features 3 metric collectors instead of 2. These include: (1) the Unity profiler ❸ with large overhead measuring application-specific metrics, (2) a less overhead heavy selective profiler ⓯ measuring a subset of those metrics, and (3) the /proc file system ⓰ measuring system-level metrics scraped by the metric receiver ❼. The Unity profiler is already implemented and comes with the Unity editor, whereas the selective profiler is implemented by us.

Both the Unity profiler and the statistic writer use instrumentation from the base unity system, called profiler markers. These markers have a Begin() and End() function and a name. Unity placed many of these markers throughout the engine runtime, and many already exist. The Unity profiler profiles all markers, making it slow, which is why the lighter selective profiler measures only a configurable subset.

4.4.1 The selective profiler

The base selective profiler measures a subset of profiler markers and adds the last sample to a list for every frame. This becomes a problem when adding profilers called more than once per frame. We added a flag, which changed it to sum all the samples from the last frame instead.

Additionally, we changed the logging mechanism from adding all data to a buffer and saving that buffer on exit to writing every frame to file (when on Linux or Windows) or to ADB logcat (when on Android) that the system profiler picks up. This saves a copy step for the researcher in the Android case and saves a system slowdown when writing this large file to disk in the Linux & Windows case.

However, the selective profiler still had some issues, as it did not seem to measure profiler markers in subcomponents completely compiled by the burst compiler such as the Ghost update system (see Section 5.2). An awful workaround was found by placing an additional marker in this system.

4.4.2 The Unity profiler

This is a tool from the Unity ecosystem that has already been implemented by Unity with a nice graphical interface, which can be extended with the profiler analyzer package. It can compare datasets, find markers with the biggest differences, and investigate spikes in frame rate. However, it has some serious limitations, as it can only analyze up to 300 frames, which is less than 5 seconds on a 72 HZ refresh rate with Meta Quest 3.

4.4.3 The system profiler

To find system-level metrics such as CPU usage, memory usage, and GPU usage, we adapted the system from (14) to fit our needs. Some functionality was removed, such as the timer and host metrics, and some functionality was added to make it more automatable, including a filter for the statistic writer metrics coming through ADB logcat.

The system profiler is implemented in Windows PowerShell and uses the ADB shell to run commands on the connected Android device, it reads out the /proc file system files including /proc/stat and /proc/meminfo every second and dumps them to disk. These dumps can then be parsed to find detailed CPU usage for every core and memory usage.

5

Evaluation

This chapter focuses on **RQ3**: How do MVEs perform on VR compared to traditional MVE platforms? The experiments presented in this chapter are enumerated in Table 5.1 using the tools implemented in Chapter 4 to conclude 3 key findings.

KF1 **single-user Opencraft2-VR supports render distances up to 9, lower than default Minecraft.** (Section 5.2). We used Yardstick-VR to evaluate the performance of MQ3 in terrain workloads and find the limit of Opencraft2-VR in MQ3. We used terrain workloads (see Section 3.6) configured with increasingly larger render distances. We find that the maximum render distance that performs above par (see Section 2.1.1) is 9, which is below the Minecraft default render distance of 12 (64-bit Java edition), where Opencraft2-VR is also a less complex game. Additionally, we find that the VR device is limited by the MVE architecture, where synchronizing worlds is a particular problem. We advise MVE developers to consider terrain requirements with care when developing MVEs.

KF2 **Frame time in Opencraft2-VR in Meta Quest 3 is better than its predecessors and is similar to PC** (Section 5.3). We compare Meta Quest 2 (MQ2), 3 (MQ3) and Pro (MQP) with a gaming computer using Yardstick-VR. We conduct a stress test using our terrain workload and a high render-distance where mean frame time is far below the target 72 Hz. We show MQ3 and the PC to have similar frame time and significantly less frame time than the MQ2 and MQP. We advise developers of VR applications to target MQ3 instead of previous HMDs for better frame times.

KF3 **Offloading the simulator of Opencraft2-VR does not significantly impact frame time or CPU usage** (Section 5.4). We place a single-user Opencraft2-VR

KF	Section	Independent variable	Dependent variable	Workload
KF1	5.2	render distance	frame time	terrain workload
KF2	5.3	platform	frame time & CPU usage	terrain workload
KF3	5.4	deployment	frame time & CPU usage	terrain workload

Table 5.1: Experiment overview

on MQ3 in a client-server configuration (the server is offloaded) and compare them using Yardstick-VR to a standalone configuration. Our comparison shows that not all MVEs benefit from offloading the simulator to a server; in particular, Opencraft2-VR does not have significant differences between deployments.

5.1 Experimental setup

The experimental setup was performed with the implementation discussed in 4. The experiment runs for 120 seconds to collect numerous samples and frames. The Android devices (MQP, MQ3, MQ2) were connected to experiment director (an Asus Zenbook UM425I laptop), via USB through the Android Debug Bridge (ADB). Care was taken to fully update the devices before running the experiment. During the experiments, the VR headsets were laying still on the table, some paper was glued to the sensor between the lenses, which checks if the headset is being worn, a technique pioneered by (27).

In some experiments, a common desktop gaming PC is used. The Gaming PC consists of the following hardware and operating system (OS):

- AMD Ryzen 5 5600X
- Gigabyte B550 Aorus elite
- 16 GB DDR4 RAM (3200 MHz)
- MSI NVIDIA RTX 3060
- Seagate ST3500141cs: an HDD 5900 rpm made for home media servers
- OS: Linux Ubuntu 22.04 LTS (an unused fresh installation)

5.2 KF1 Opencraft2-VR supports render distances up to 9 due to performance problems associated with the MVE architecture

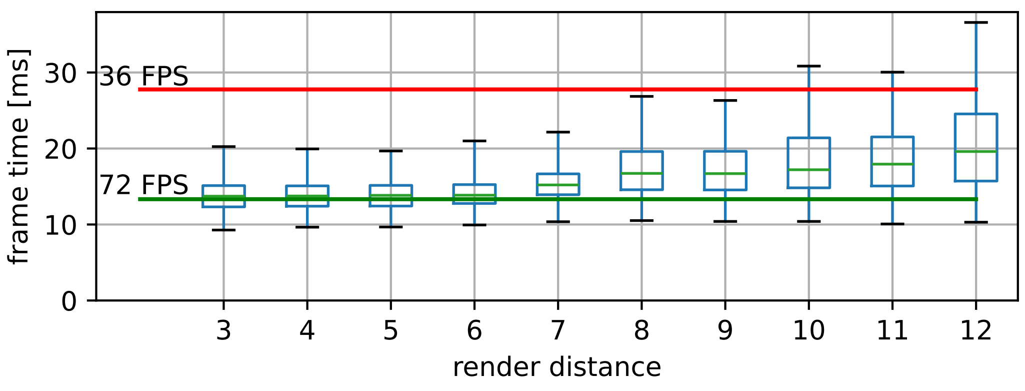
Section 1 describes the importance of world complexity, which we evaluate in this section to find improvements. We find that Opencraft2-VR terrain synchronization takes a lot of frame time, causing the maximum render distance for Opencraft2-VR to be not greater than 9, where Minecrafts default render distance is 12. A higher render distance causes the frame time to rise above par (see Section 2.1.1), introducing the risk of motion sickness.

We measured in an experiment using MQ3 with the terrain area generation workload described in 3.6, where more parts of the world are generated over time so that the system has an increasingly larger world to render. The experiment was repeated multiple times, each time varying the render distance. The experiment was conducted with the Opencraft2 data collector and the system level data collector and without the Unity Profiler to keep overhead to a minimum. The first 300 frames of the application were cut to focus the measurement on the stable state of the system.

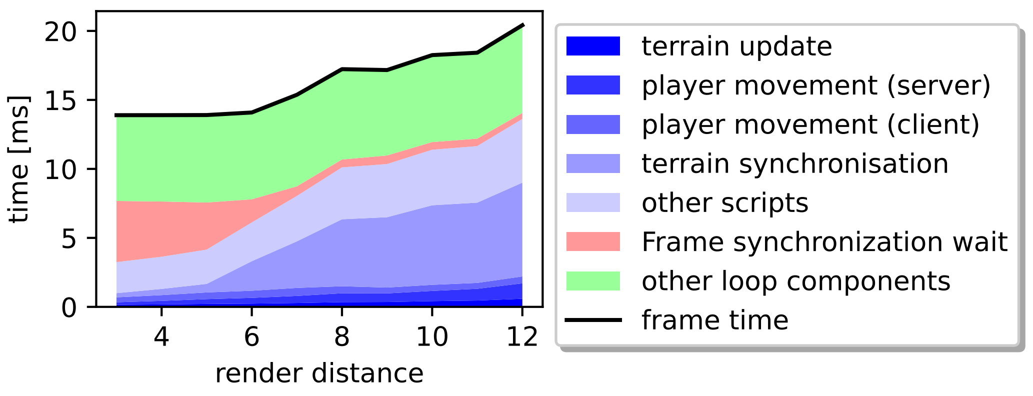
This experiment resulted in Figures 5.1a and 5.1b, where the horizontal axes contain the render distance, and the vertical axis represents the frame time in milliseconds. In 5.1a The data is displayed as box plots, where the whiskers represent the end of the first percentile and the end of the ninety-ninth percentile. 50% of the samples fall within the box, where the small green lines represent the median frame time. Outliers are not shown. The green and red lines represent the par defined in 2.1.1 by showing the frame time required for 36 FPS, and 72 FPS.

Figure 5.1a shows an increase in the median frame time over the render distance, where the ninety-ninth percentile also increases, while the lower percentile remains relatively stable. All medians are between the 36 FPS and 72 FPS lines, where the upper whiskers cross the 36 FPS line between the render distance 9 and the render distance 10.

The frame time in both figures is expected to increase as the render distance increases, the plot agrees. We expect this increase because the world the application has to keep track of is larger, and more world areas have to be generated at the same time. The upper whisker crosses the red line between render distance 9 and 10, which means that the render distance 9 is the highest render distance when the samples are below the red line and subsequently means that render distance 9 is the last render distance above par where the risk of users developing motion sickness is low.



(a) Opencraft2-VR frame time vs render distance on MQ3



(b) breakdown of frame time and biggest contributors

Figure 5.1: Analysis of frame time when exploring at increasing render distances

The upper whiskers of render distances 10 and 11 are not in the pattern strictly increasing, similar to 8 and 9. This suggests a lack of samples used for this plot, a limitation discussed in Section 7

The areas in Figure 5.1b represent the mean time that different components contribute to frame time. All components shaded in blue (the first seven components in the legend when read top-down) are all logic components of the game loop (see Section 2.1). The other colors are other components of the game loop. Some more detail and analyses are given for every component:

1. **Terrain building** This component is responsible for building the terrain layers from a random function and storing it in data structures. This area represents three subcomponents dividing the responsibility between them: A component which schedules areas to spawn, a component which builds the ground of those scheduled areas and a component

which builds trees on those scheduled areas.

The area that represents the terrain builder is small and does not increase fast when the render distance increases. The latter subcomponents are largely multithreaded through the C# job system. However, the C# job threads remain largely unused and mostly idle, suggesting that these components contribute relatively little time.

2. **player movement (client & server)**: This component is responsible for moving the player character according to the input of the user and the terrain. Due to the deployment of single user Opencraft2, which still uses client and server deployment, the difference for the server and client can be measured. It is implemented by first fetching all terrain areas, and then accessing only important parts

The time used by this script is constant over the render distance for the server, but increasing for the clients, the times spent in each server and client are still larger than the terrain update. The large time spent is supposedly due to the implementation fetching all terrain areas after some nonscientific extra measurements. The reason why the server time spent in the movement is unclear.

3. **terrain synchronization**: The terrain synchronization component in the figure is a C# job managed by a subcomponent of the networking library that keeps the world on the client and server synchronized. It does this by doing the game logic of specific objects on the server and pushing state updates to a replica (ghost object) on the client, the client can then interpolate between these state updates to allow additional updates for smoothness.

The terrain synchronization job is the fastest growing component when increasing the render distance. This component is also the cause of a large amount of variation in the frame time, as problem 5.1a shows. The linear increase of the frame time can be explained by the linear increase in the world size on which the terrain depends.

4. **Other scripts:** The time for this component has been calculated using the total time for the scripts in the player loop and subtracting all the other plotted script updates time. The area in the graph grows with the render distance, which means that there are still more components that are growing with the render distance; more work is required to identify these components and improvements to the system.

5. **Frame synchronization wait:** This component keeps the game loop frequency synchronized with the refresh rate of the headset. The component lives in the oculus XR plugin (see Section 4.3.2) which is not open source. A forum post (39) states the oculus

plugin will wait when the engine is producing faster than the headsets 72 display refresh rate to prevent too many frames too fast. Due to the closed source of the Oculus plugin used, we have not been able to verify the implementation or find more details for this component.

The figure shows that the area of the frame synchronization wait component starts large and shrinks fast. The explanation presented by (39) is applicable at small render distances, as the mean frame time is very close to the 72 FPS with high variance (see Section **KF1**). Samples with a frame time below the green line that the Oculus VR plugin will wait.

6. **Other loop components** This area represents the rest of the loop components of the game that are not updates to the game logic script. As the figure shows, the area representing these components is constant.

To conclude, Opencraft2-VR's 9 is low compared to (the most popular MVE) Minecraft's render distance of 12, where Minecraft also has a more complex and taller world compared to Opencraft2-VR. We advise MVE developers targeting VR applications to take care when setting higher render distance requirements, as they are not trivial to achieve.

Additionally, components implemented with a time complexity depending on the number of areas cause great problems when dealing with large worlds. Terrain synchronization and player movement times are examples of components with such time complexities. Terrain synchronization is the main problem that holds the frame time back. The terrain synchronization time grows fast and also brings a large amount of variance to the frame time. The player movement time also increases as the render distance increases, and is a good candidate for the next problem when the terrain synchronization time is reduced.

Because terrain synchronization and the client-server deployment are not necessary in single-user MVEs, MVE developers who want to build and simulate large worlds should attempt to find alternatives to continuously synchronizing unchanging worlds, as this architectural decision really holds Opencraft2 world complexity back.

5.3 KF2 Opencraft2-VR frame time is better on MQ3 than its predecessors and comparable to PC

As the performance of standalone devices is not yet adequately understood (see Section 1), it is important to compare these devices with traditional simulator platforms such as the PC, to create a reference frame. We show that MQ3 performance is comparable to PC and better than MQ2 and MQP.

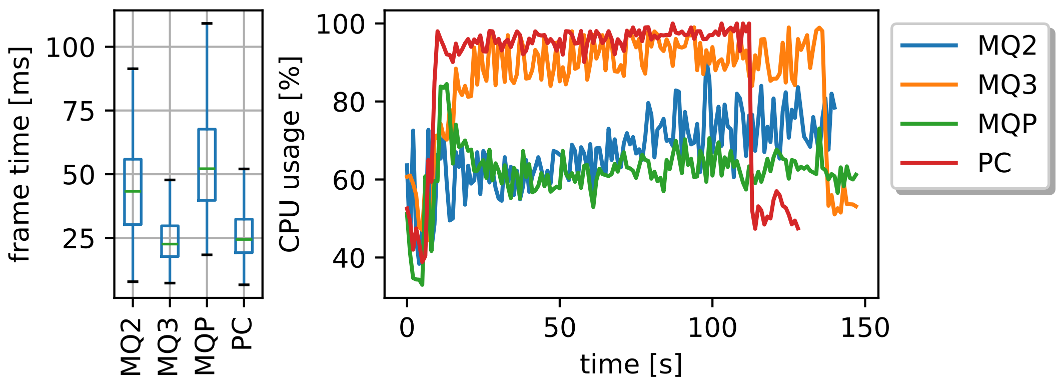


Figure 5.2: Frame time and CPU usage are different across different platforms.

To come to this conclusion, an experiment was run using our platform built in Chapter 4. The same workload, that is, the player flight workload, was run with a render distance of 18 for all bench-marked hardware. The four selected hardware platforms were MQ2, MQP, MQ3, and PC. The Unity profiler was turned off for the experiment, to mitigate overhead and provide more relevant results.

The experiment produced 5.2 similar to **KF1** in which the different platforms are enumerated on the horizontal axis and the frame time in milliseconds is shown on the vertical axis. The data is again shown in box plots where the whiskers represent the first and ninety-ninth percentile, the boxes show the range of 50% of the data, and the median is annotated with the green lines in these boxes.

The figure shows the mean frame time for MQ3 to be the lowest, closely followed by PC. The frame times of MQ2 and MQP are, respectively, farther away. The spread of frame times are also quite different, where the MQ2 and MQP have a significantly larger difference between their whiskers and larger boxes compared to the MQ3 and PC.

As applications running on different devices are the same, we identify the lower layers of the technology stack as the reasons for the difference in performance. However, the lower layers are proprietary and there may be many reasons in operating system, XR runtime, hardware, etc. We hypothesize that this difference is not due to the processor upgrade MQ3 has over its predecessors as the CPU utilization of the MQ3 is much higher than the CPU utilization of the MQ2 and MQP, which suggests that there is some other bottleneck in the system that keeps the CPU from delivering strong performance.

In conclusion, MQ3 is the best Opencraft2-VR platform of the evaluated selection. The MQ3 outperforms its predecessors, which is surprising as the MQP is Meta's flagship device and the MQ3 achieves performance surprisingly close to PC. We discourage MVE developers from targeting MQP and MQ2.

5.4 KF3 Single-user Opencraft2-VR should not be offloaded to a server

(40) shows the benefits of offloading terrain generation, we explore the possibility and effectiveness of offloading the server part of Opencraft2-VR to a new device. We show using an experiment that offloading does not increase the application performance at our render distance.

To conclude this finding, an experiment was run using our platform built in Chapter 4. The player flight workload was run using a render distance of 18, producing one data point where the server and the client run on MQ3 (onloaded) and one data point where the client runs on MQ3, and the server runs on the experiment director connected over Wi-Fi (onloaded) (see Section 5.1 for hardware details).

The experiment produced Figure 5.3, consisting of 3 subplots representing frame time (left) and CPU usage (right):

The left subplot shows the frame time on the vertical axis, with the configurations on the horizontal axis. The data is shown in box plots for each configuration, where the lower whisker is in the lowest 1%, the upper whisker is in the upper 1%, the box represents the range of 50% of the data, and the bar within the box represents the median. Below the boxes, some bar charts represent where this frame time is going within the application.

All areas in the bar charts shaded blue (the first 5 when reading the legend top-down) represent subcomponents in the game logic components of the game loop, where other colors represent subcomponents of other game loop components. The measured components include: the terrain update and terrain meshing which are related to building the terrain; the movement components which are related to calculating how a player should move through the world; the terrain synchronization update, which is a subcomponent of the network library that sends game state between the server and client components; and other scripts and the rest of the game loop which represent components not explicitly measured.

The right subplots show the CPU usage for both configurations. On the vertical axes, the CPU usage is displayed, where the horizontal axis displays the time over the experiment. The data is displayed with a line for every CPU core and one line (blue) for the mean of all CPU cores.

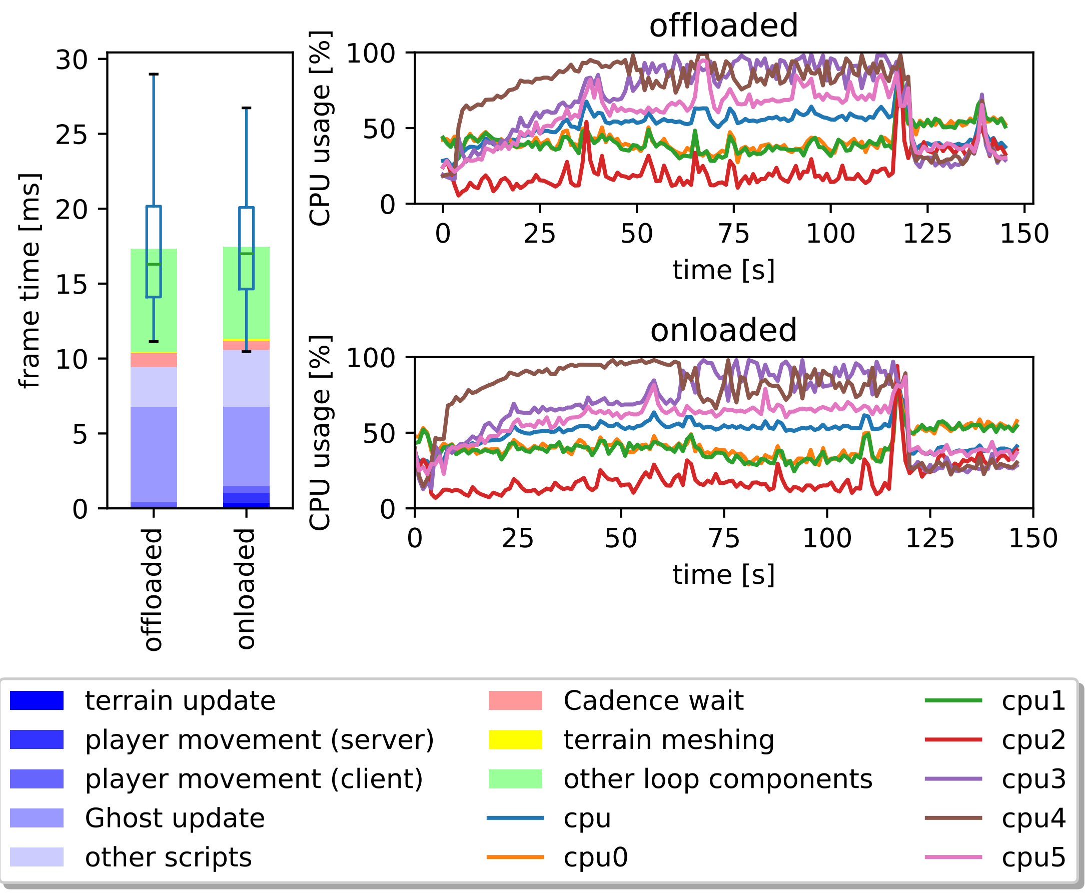


Figure 5.3: Difference in frame time and CPU usage between a deployment with an offloaded server and an onloaded server

The figure shows that the mean frame time is slightly lower when the server is offloaded (17.33 ms) compared to the configuration where the server is onloaded (17.46 ms). The time saved by eliminating the terrain update is around 2 ms, as shown by the darker blue components disappearing. However, this extra time is lost due to a larger terrain synchronization time, as shown by the larger area representing terrain synchronization. Some of the increasing subcomponents of the script component also shrink when the server is offloaded, which is being offset by some unmeasured components in the rest of the game loop. More insight into the missing components is needed in future work.

The CPU usage of the top core increases to 100%, after which it starts doing a strange dance with another core. It is hypothesized that this is the kernel switching some allocation strategy or memory swapping when memory starts running out. This hypothesis comes from the observation that the android out-of-memory killer killed both applications 110 seconds into the experiment, And the start of this dancing perfectly coincides with a lowering spike in memory usage. The evaluation of this repository has been left to future work, as it is hardly relevant.

In conclusion, offloading does not increase the application performance at our render distance. The CPU usage is similar, the time saved by eliminating components is offset by others, and the mean and median frame times do not change much when changing configuration. We do not recommend MVE developers to offload the server component to another machine when developing single-user MVEs.

5.5 Limitations and/or threat to validity

A limitation of this experiment setup when comparing different platforms is a different deployment for PC and Meta Quest devices. As Linux does not have ADB, an alternative to ADB USB was searched for Linux, but this came up empty-handed. This is the reason why the PC is connected with SSH (Secure shell) to the experiment controller over a 1GPS Ethernet cable to a Fritz box 4040 router, which is connected via Wi-Fi to the experiment controller.

An additional limitation to the PC is the required Linux OS, as it is the only OS supported by our system-level data collector. The common OS for gaming on desktop form factor devices is Windows, which does not have the /proc file system the data collector is based on. This created the need to dual boot Linux (the original PC had windows), but the B.Sc. thesis student would not like his operating system and storage drives to be messed with, so an off-the-shelf (slow) HDD was used.

A limitation to Opencraft2-VR is its memory usage; when running render distances which are challenging for Quest devices, but not challenging enough to slow the simulator down, the out-of-memory killer kills Opencraft2-VR, which strictly limits the time we can spend to measure frame times over longer application lifetime. A fix for this would be to implement a feature in which terrain areas far away from the player can be removed to allow memory to be reclaimed; more future work is needed to implement such a feature.

5.6 Conclusion

In conclusion to our research question **RQ3**, VR is ready for MVEs, but Opencraft2-VR can still improve VR performance. We conclude this as the performance of the MQ3 in Figure 5.2 is comparable to the gaming computer, which is the origin platform of the most popular MVE (Minecraft). The problems with terrain synchronization discussed in **KF1** should be solved, which was again shown to be a problem in **KF3**, as there is much to gain by fixing this problem. We advise VR MVE developers to refrain from targeting MQ2 and MQP and suggest MQ3; we also suggest finding a more performant solution for keeping large worlds synchronized; and suggest a fully central world when targeting a single-user MVE.

5.7 Advice for MVE VR developers

To assist future work and MVE developers, we summarize the advice found while answering these research questions for researchers and developers in this field. We present this advice in a few pieces of advice **(PA)**:

PA1. Target the Meta Quest 3 instead of the older Meta Quest 2 and Pro. In Section 5.3 we found a significant improvement for the Meta Quest 3 compared to its predecessors when stressed by our environmental workloads.

PA2. Consider minimizing terrain updates or eliminating the need for them if possible. In 5.2 we found large parts of the frame time spent on synchronizing terrain in the client world with terrain in the server world running on the same VR HMD. We advise merging these worlds when running a local game, and to take care in picking your update strategy developing a client server deployment.

PA3. Do not assume offloading terrain building to a server always improves performance. In 5.4, we found offloading Opencraft2-VR server to an edge device did not improve performance. This is counterintuitive as we have seen performance improvements with similar optimizations like in (40).

6

Related Work

To the best of our knowledge, we are the first to build an MVE on VR in an academic work. The commercial MVE Minecraft(41) and its community (42) have made VR versions, but none of these has been documented in a paper. Other works have built other VR applications and analysed the performance of other MVEs. Furthermore, (43) has investigated technical challenges involving scalable VR environments but has not yet built a system to address these challenges.

Other works have investigated VR systems and applications, which we summarise and compare in Section 6. The works related to the MVE part of this work are enumerated in Section 6.

Related VR work

Many other works have built standalone VR applications, but none of them are MVEs like Opencraft2-VR. This is an important distinction, as the performance profile of an MVE is much different than traditional games (24).

(14) investigated VR performance in and between different offload strategies. In standalone HMDs, they found that GPU usage is significantly higher than CPU usage. They also show that wired and wireless offloading may negatively affect the frame time and identify network requirements for wireless offloading.

(44) builds PlayBricks, a standalone VR application where users can create buildings from shapes to demonstrate a performance problem when 2000+ cubes are placed in the rendered environment. The PlayBricks application features an optimisation to a problem created by Meta's Application spacewarp optimisation, which significantly increases the amount of cubes able to exist in a scene.

(45) Built two applications for standalone VR: WeldVR and SprayVR, where the user can exercise welding and spraypainting in a cheaper environment. During their implementation, they encountered multiple problems including large script updates like we found in 5.2. (45) also found additional performance problems unexplored by us involving rendering.

(46) Rebuilt A VR application from (47) for standalone VR, which suffered a performance problem and found that the standalone MQ2 was much weaker than the PCVR setup used by the original application.

Additional non-standalone VR applications have also been developed by (8, 9, 10, 11, 23, 34, 48, 49) with offloaded configurations like WiFi offloading or tethered offloading.

Related MVE work

However, no work has created an MVE on VR like Opencraft2-VR, some works did find overlapping ideas. We show how MVEs can be limited by deployment and implementation when scaling world size and similarly, other works have shown limitations of MVEs through experimental data and benchmarking.

(50) developed Yardstick, a benchmark for Minecraft-like services, where the server performance of MVE was tested with high player counts. Using their benchmark, they found similarly to us that terrain synchronisation between worlds can be a problem for the network, where we concluded that these synchronisations can strain the compute hardware available. Furthermore, (50) found how minecraft-like services are poorly paralelised, have different performance profiles between different server implementations and scale to hundreds of players.

(24) went further and developed Meterstick, a similar benchmark for Minecraft servers. Using Meterstick, the authors found how environmental workloads like terrain generation were hard for the systems under test and caused a lot of performance variability. (40) found an improvement for this and increased performance with a serverless extention, which provides additional compute resources when needed.

7 Conclusion

Modifiable virtual environments in virtual reality are desirable, as they may help bring existing VR technologies to the next level. Virtual reality is a fast evolving technology with strong performance requirements, for which the exact performance and combination with MVEs remain undocumented. In this work, we try to narrow this research gap with Opencraft2-VR, an MVE in VR, and Yardstick-VR an experiment framework for this MVE by answering a few research questions.

RQ1 How to design an experiment framework to compare MVE performance on established gaming platforms and a standalone VR headset? In Chapter 3 we designed a multi-machine setup that supports multiple platforms including standalone VR HMDs. We looked at the needs of the stakeholders in this system and their requirements. There we found a requirement and designed for automated experiments with an experiment controller; we found a requirement for workloads, and designed a workload that pushes the complexity of the MVE world to its limits; we found a requirement for overhead when collecting data, and found a solution with a light data collector and heavy data collector.

RQ2 How to implement the design of Chapter 3 was answered by Chapter 3, where we implemented a prototype of the design. We successfully implemented Opencraft2 for Android and Opencraft2-VR, we also implemented the experiment framework infrastructure with the experiment controller, and the data collectors.

RQ3 How do MVEs perform on VR compared to traditional MVE platforms? In 5 we found the newest VR devices (Meta Quest 3) handle our terrain workload as well as a PC, but Opencraft2-VR has limitations. Performance problems involving world synchronization between client and server worlds increase frame time and risk of motion sickness. Additionally, we found the previous VR devices (Meta Quest Pro, Meta Quest 2) perform worse than their newer versions. An attempt to improve performance by offloading the server did not yield measured improvements worth the cost.

Limitations and future work

When writing this work, we found multiple limitations that restrict our ability in our results. We stress and summarize them here to reduce the risk of incorrect interpretation and conclusions.

No implementation of Application Spacewarp

We use Application Space warp(21) when defining par in 2.1.1, but Application space warp has to be specifically enabled for VR applications built on Unity. Due to the late realization of this fact, Application Space warp has yet to be implemented in Opencraft2-VR and **KF1** may change depending on if application space warp is enabled.

Hardware limitations for PC

As described in 5.5, we used a PC with an outdated HDD and an unconventional Linux Operating system as most gaming PCs use a faster SSD and Windows. As our application barely interacts with storage, we do not expect the impact of these choices to be large as Ubuntu is an established OS for other applications than games.

Terrain despawn mechanism

When implementing and designing our player flight workload in 3.6. We were restricted by a feature despawning and reclaiming memory of spawned terrain, which went out of view distance. This caused practically the same problems as a memory leak, and the workload could not run for longer than 2.5 minutes on some render distances. This limited our findings as smaller sample sizes may produce statistically insignificant results.

Experiment reruns

When running experiments, common to standard practice, the experiment was not run multiple times, which restricts us in the same way. To increase statistical significance the experiments should be run the common 5 times to increase and strengthen this statistical significance.

Measured metrics

In 5.2 We find the script update component (a subcomponent of the player loop) still has scripts increasing in time when render distance grows. From this, we concluded that more work is needed to identify these scripts, as they may lead to problems when the Ghost update problem is solved.

References

[1] **Meta Platforms Has Spent $46 Billion on the Metaverse Since 2021, But It's Spending Twice As Much on This 1 Thing**. iii, 1

[2] Precedence research. **Video Games Market Size To Attain USD 664.96 Billion By 2033 — precedenceresearch.com**. `https://www.precedenceresearch.com/video-game-market`. [Accessed 11-08-2024]. iii

[3] Ghaliya Alfarsi. **A Review of Virtual Reality Applications in an Educational Domai**. *International Journal of Interactive Mobile Technologies (iJIM)*, **15**, 11 2021. 1

[4] Mohd Javaid and Abid Haleem. **Virtual reality applications toward medical field**. *Clinical Epidemiology and Global Health*, **8**(2):600–605, 2020. 1

[5] Meta. **Meta quest 3: New mixed reality VR headset – shop now**. 1

[6] Apple. **Apple vision pro**. 1

[7] **The 10 Best-Selling Video Games of All Time**. 1

[8] Yining Lang, Liang Wei, Fang Xu, Yibiao Zhao, and Lap-Fai Yu. **Synthesizing Personalized Training Programs for Improving Driving Habits via Virtual Reality**. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 297–304, 2018. 1, 36

[9] Aurélie CONGÈS, Alexis EVAIN, Frédérick BENABEN, Olivier CHABIRON, and Sébastien REBIÈRE. **Crisis Management Exercises in Virtual Reality**. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 87–92, 2020. 2, 36

[10] Annette Mossel, Mario Froeschl, Christian Schoenauer, Andreas Peer, Johannes Goellner, and Hannes Kaufmann. **VROnSite: Towards immersive training of first responder squad leaders in untethered virtual reality**. In *2017 IEEE Virtual Reality (VR)*, pages 357–358, 2017. 2, 36

[11] Janne Heirman, Shivam Selleri, Tom De Vleeschauwer, Charles Hamesse, Michel Bellemans, Evarest Schoofs, and Rob Haelterman. **Exploring the possibilities of Extended Reality in the world of firefighting**. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pages 266–273, 2020. 2, 36

[12] Jonas Schild, Sebastian Misztal, Beniamin Roth, Leonard Flock, Thomas Luiz, Dieter Lerner, Markus Herkersdorf, Konstantin Weaner, Markus Neuberaer, Andreas Franke, Claus Kemp, Johannes Pranqhofer, Sven Seele, Helmut Buhler, and Rainer Herpers. **Applying Multi-User Virtual Reality to Collaborative Medical Training**. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 775–776, 2018. 2

[13] Jacqueline M. Fulvio, Mohan Ji, and Bas Rokers. **Variations in visual sensitivity predict motion sickness in virtual reality**. *Entertainment Computing*, **38**:100423, 2021. 2

[14] Matthijs Jansen, Jesse Donkervliet, Animesh Trivedi, and Alexandru Iosup. **Can My WiFi Handle the Metaverse? A Performance Evaluation Of Meta's Flagship Virtual Reality Hardware**. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23 Companion, page 297–303, New York, NY, USA, 2023. Association for Computing Machinery. 2, 4, 7, 15, 21, 35

[15] Jerrit Eickhoff. **Polka: A Differentiated Deployment System for Online and Streamed Games, Meta-verses, and Modifiable Virtual Environments**, mar 2024. 4, 15

[16] Minxia Yang, Jiaqi Zhang, and Lu Yu. **Perceptual Tolerance to Motion-To-Photon Latency with Head Movement in Virtual Reality**. In *2019 Picture Coding Symposium (PCS)*, pages 1–5, 2019. 5

[17] Kevin Boos, David Chu, and Eduardo Cuervo. **FlashBack: Immersive Virtual Reality on Mobile Devices via Rendering Memoization**. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 291–304, New York, NY, USA, 2016. Association for Computing Machinery. 5

[18] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. **Effect of Frame Rate on User Experience, Performance, and Simulator Sickness in Virtual Reality**. *IEEE Transactions on Visualization and Computer Graphics*, **29**(5):2478–2488, 2023. 5

[19] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. **Effects of Latency Jitter on Simulator Sickness in a Search Task**. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 121–127, 2018. 5

[20] Jann Philipp Freiwald, Nicholas Katzakis, and Frank Steinicke. **Camera time warp: compensating latency in video see-through head-mounted-displays for reduced cybersickness effects**. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, VRST '18, New York, NY, USA, 2018. Association for Computing Machinery. 5, 9

[21] Jian Zhang, Neel Bedekar, Leonard Tsai, and Xiang Wei. **Introducing Application SpaceWarp**, 2021. 5, 38

[22] Meta. **Meta Quest Virtual Reality Check (VRC) Guidelines | Oculus Developers**. 6

[23] Alec Rohloff, Zackary Allen, Kung-Min Lin, Joshua Okrend, Chengyi Nie, Yu-Chia Liu, and Hung-Wei Tseng. **OpenUVR: an Open-Source System Framework for Untethered Virtual Reality Applications**. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 223–236, 2021. 6, 36

[24] Jerrit Eickhoff, Jesse Donkervliet, and Alexandru Iosup. **Meterstick: Benchmarking Performance Variability in Cloud and Self-hosted Minecraft-like Games**. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, ICPE '23, page 173–185, New York, NY, USA, 2023. Association for Computing Machinery. 8, 9, 14, 35, 36

[25] Jesse Donkervliet, Jim Cuijpers, and Alexandru Iosup. **Dyconits: Scaling Minecraft-like Services through Dynamically Managed Inconsistency**. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 126–137, 2021. 8

[26] Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron Lefohn, and David Luebke. **Perceptually-based foveated virtual reality**. In *ACM SIGGRAPH 2016 Emerging Technologies*, SIGGRAPH '16, New York, NY, USA, 2016. Association for Computing Machinery. 9

[27] Radu Apsan, Damla Ural, Paul Daniëlse, Vlad-Andrei Cursaru, Eames Trinh, Jesse Donkervliet, and Alexandru Iosup. **Towards a Workload Trace Archive for Metaverse Systems**. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, ICPE '24 Companion, page 204–210, New York, NY, USA, 2024. Association for Computing Machinery. 14, 15, 18, 24

[28] Sa Wang, Zhengli Mao, Changhai Zeng, Huili Gong, Shanshan Li, and Beibei Chen. **A new method of virtual reality based on Unity3D**. In *2010 18th International Conference on Geoinformatics*, pages 1–5, 2010. 15

[29] Alexander Novotny, Rowan Gudmundsson, and Frederick C Harris Jr. **A Unity Framework for Multi-User VR Experiences.** In *Conference on Computers and Their Applications*, **69**, pages 13–21, 2020. 15

[30] Cosmina Cosmina. **A Glance into Virtual Reality Development Using Unity**. *Informatica Economica*, **22**:14–22, 09 2018. 15

[31] Jichao Wang, Lucas Phillips, John Moreland, Bin Wu, and Chenn Zhou. **Simulation and visualization of industrial processes in unity**. In *Proceedings of the Conference on Summer Computer Simulation*, SummerSim '15, page 1–7, San Diego, CA, USA, 2015. Society for Computer Simulation International. 15

[32] Jack Brookes, Matthew Warburton, Mshari Alghadier, Mark Mon-Williams, and Faisal Mushtaq. **Studying human behavior with virtual reality: The Unity Experiment Framework**. *Behavior Research Methods*, **52**(2):455—-463, April 2020. 15

[33] Diar Abdlkarim, Massimiliano Di Luca, Poppy Aves, Mohamed Maaroufi, Sang-Hoon Yeo, R. Chris Miall, Peter Holland, and Joeseph M. Galea. **A methodological framework to assess the accuracy of virtual reality hand-tracking systems: A case study with the Meta Quest 2**. *Behavior Research Methods*, **56**(2):1052–1063, February 2024. 15

[34] Manuel Trinidad-Fernández, Benoît Bossavit, Javier Salgado-Fernández, Susana Abbate-Chica, Antonio J. Fernández-Leiva, and Antonio I. Cuesta-Vargas. **Head-Mounted Display for Clinical Evaluation of Neck Movement Validation with Meta Quest 2**. *Sensors*, **23**(66):3077, January 2023. 15, 36

[35] Glowstone project. **Glowstone**. 16

[36] AtlargeResearch. **Opencraft 1**, October 2022. 16

[37] Minetest. **Minetest**. 16

[38] The Terasology Foundation. **Terasology**. 16

[39] jj unity. **Other - XREarlyUpdate spikes any news?**, 2020. 27, 28

[40] Jesse Donkervliet, Javier Ron, Junyan Li, Tiberiu Iancu, Cristina L. Abad, and Alexandru Iosup. **Servo: Increasing the Scalability of Modifiable Virtual Environments Using Serverless Computing**. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*, pages 829–840, 2023. 30, 33, 36

[41] **Minecraft**. 35

[42] **Vivecraft**. 35

[43] Andreas Haeberlen, Linh Thi Xuan Phan, and Morgan McGuire. **Metaverse as a Service: Megascale Social 3D on the Cloud**. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC '23, page 298–307, New York, NY, USA, 2023. Association for Computing Machinery. 35

[44] Youssef Samir Sadek Hosny, Mohammed A.-M Salem, and Ahmed Wahby. **Performance Optimization for Standalone Virtual Reality Headsets**. In *2020 IEEE Graphics and Multimedia (GAME)*, pages 13–18, 2020. 35

[45] Nikola Rendevski, Konstantin Veljanovski, and Naile Emini. **FPS Performance Factors in Standalone Virtual Reality Applications**. In *2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*, pages 349–352, 2023. 36

[46] Sritrusta Sukaridhoto, Amma Haz, Evianita Fajrianti, and Rizqi Putri Nourma Budiarti. **Comparative Study of 3D Assets Optimization of Virtual Reality Application on VR Standalone Device**. *International Journal on Advanced Science, Engineering and Information Technology*, **13**:999, June 2023. 36

[47] Amma Liesvarastranta Haz, Muhtadin, I Ketut Eddy Purnama, Mauridhi Hery Purnomo, and Sritrusta Sukaridhoto. **Virtual Reality Application for Co-Bot Training**. In *2022 International Electronics Symposium (IES)*, pages 644–650, 2022. 36

[48] Ocean Hurd, Sri Kurniawan, and Mircea Teodorescu. **Virtual Reality Video Game Paired with Physical Monocular Blurring as Accessible Therapy for Amblyopia**. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 492–499, 2019. 36

[49] Marc Herrlich, Ronald Meyer, Rainer Malaka, and Helmut Heck. **Development of a Virtual Electric Wheelchair – Simulation and Assessment of Physical Fidelity Using the Unreal Engine 3**. In Hyun Seung Yang, Rainer Malaka, Junichi Hoshino, and Jung Hyun Han, editors, *Entertainment Computing - ICEC 2010*, pages 286–293, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 36

[50] Jerom van der Sar, Jesse Donkervliet, and Alexandru Iosup. **Yardstick: A Benchmark for Minecraft-like Services**. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, page 243–253, New York, NY, USA, 2019. Association for Computing Machinery. 36

Appendix A

Code repository

The used code of Yardstick-VR can be found at https://github.com/atlarge-research/Opencraft-2/tree/Opencraft2-VR. The code for the metric receiver can be found at https://github.com/atlarge-research/measuring-the-metaverse/tree/opencraft2-VR-measurements.