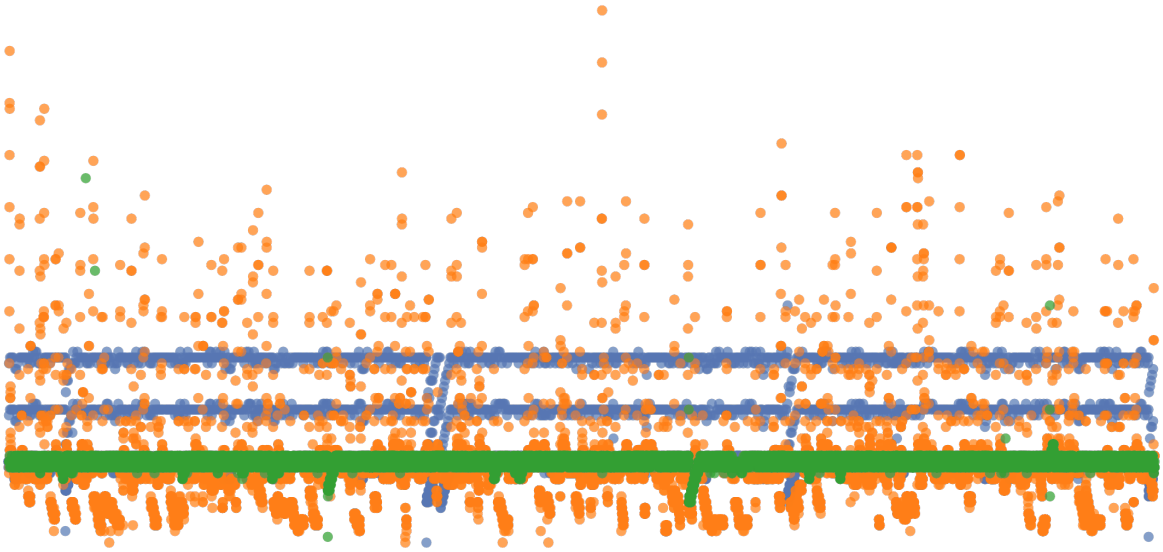
Temporal Confounding Effects in Virtual and Extended Reality Systems



Dissertation submitted for the degree of
Doctor rerum naturalium (Dr. rer. nat.)
at the University of Würzburg in Germany
Faculty of Mathematics and Computer Science

Submitted by Jan-Philipp Stauffert
Advised by Marc Erich Latoschik

ABSTRACT

Latency is an inherent problem of computing systems. Each computation takes time until the result is available. Virtual reality systems use elaborated computer resources to create virtual experiences. The latency of those systems is often ignored or assumed as small enough to provide a good experience.

This cumulative thesis is comprised of published peer reviewed research papers exploring the behaviour and effects of latency. Contrary to the common description of time invariant latency, latency is shown to fluctuate. Few other researchers have looked into this time variant behaviour. This thesis explores time variant latency with a focus on randomly occurring latency spikes. Latency spikes are observed both for small algorithms and as end to end latency in complete virtual reality systems. Most latency measurements gather close to the mean latency with potentially multiple smaller clusters of larger latency values and rare extreme outliers. The latency behaviour differs for different implementations of an algorithm. Operating system schedulers and programming language environments such as garbage collectors contribute to the overall latency behaviour. The thesis demonstrates these influences on the example of different implementations of message passing.

The plethora of latency sources result in an unpredictable latency behaviour. Measuring and reporting it in scientific experiments is important. This thesis describes established approaches to measuring latency and proposes an enhanced setup to gather detailed information. The thesis proposes to dissect the measured data with a stacked z-outlier-test to separate the clusters of latency measurements for better reporting.

Latency in virtual reality applications can degrade the experience in multiple ways. The thesis focuses on cybersickness as a major detrimental effect. An approach to simulate time variant latency is proposed to make latency available as an independent variable in experiments to understand latency's effects. An experiment with modified latency shows that latency spikes can contribute to cybersickness. A review of related research shows that different time invariant latency behaviour also contributes to cybersickness.

Zusammenfassung

Latenz ist ein inhärentes Problem in Computersystemen. Jede Berechnung benötigt Zeit bis das Ergebnis verfügbar ist. Virtual Reality Systeme verwenden komplexe Rechenresourcen um virtuelle Erfahrungen zu erstellen. Die Latenz dieser Systeme wird oft ignoriert oder als klein genug angesehen um eine gute Erfahrung zu ermöglichen.

Diese kumulative Doktorarbeit ist aus peer reviewten Forschungspublikationen zusammengesetzt, die das Verhalten und Effekte von Latenz erforschen. Gegensätzlich zu der landläufig verwendeten Beschreibung als zeitinvariante Latenz, wird gezeigt, dass Latenz fluktuiert. Wenige Forscher haben dieses zeitvariable Verhalten genauer untersucht. Diese Arbeit erforscht zeitvariable Latenz mit einem Fokus auf zufällig auftretende Latenzspitzen. Latenzspitzen wurden sowohl für die Ausführung kleiner Algorithmen als auch als End-to-End Latenz in kompletten Virtual Reality Systemen beobachtet. Die meisten Latenzmessungen sammeln sich nahe eines Durchschnittswertes wobei es mehrere Ballungen von höheren Latenzwerten gibt und einige seltene extreme Ausreißer. Das Latenzverhalten unterscheidet sich bei unterschiedlichen Implementierungen eines Algorithmusses. Betriebssystemsscheduler und Programmiersprachenumgebungen wie Garbage Collektoren tragen zu dem Gesamtlatenzverhalten bei. Diese Arbeit beschreibt diese Einflüsse am Beispiel unterschiedlicher Implementierungen von Nachrichtenaustausch.

Die Vielfalt an Latenzquellen resultiert in einem unvorhersehbaren Latenzverhalten. Das Messen und Dokumentieren von Latenz ist wichtig in wissenschaftlichen Experimenten. Diese Arbeit beschreibt etablierte Ansätze um Latenz zu messen und schlägt einen verbesserten Ansatz vor um detaillierte Messungen zu sammeln. Diese Arbeit beschreibt wie gemessene Daten mit Hilfe eines stacked z-Testes in separate Cluster geteilt werden können um das Verhalten besser beschreiben zu können.

Latenz in Anwendungen der Virtuellen Realität kann die Erfahrung in verschiedenen Arten verschlechtern. Diese Arbeit fokusiert sich auf Cybersickness als einen bedeutenden negativen Einfluss. Es wird ein Ansatz vorgestellt um Latenz zeit-variant zu simulieren damit Latenz als unabhängige Variable in Experimenten, die Auswirkungen von Latenz untersuchen, verwendet werden kann. Ein Experiment mit modifizierter Latenz zeigt, dass Latenzspitzen zu Cybersickness beitragen können. Ein Review verwandter Arbeiten zeigt, dass auch andere Latenzverhalten Cybersickness negativ beeinflussen können.

Contents

Chapter 1

INTRODUCTION

Virtual reality promises to deliver new horizons — virtual worlds that immerse the user in a way not possible before [Sut65] and provide new interaction paradigma to communicate with the computer [BRC96] and with other humans [Lat+19]. A key factor in the presentation is how responsive it is, i.e., how fast the system can react to input. Powerful technical machinery is needed to fulfil the promise. However fast they are, their processing incurs a latency, a time delay between input and output. The larger the latency, the more the experience suffers.

Virtual worlds become ever more convincing as technology advances. Users feel more present in a virtual world that provides a believable image of the environment and of the user itself [Wen+20]. The aspiration to create ever more detailled experiences demand more processing power of the computers that supply this experience. These computers need to get faster and the algorithms running on them more clever to uphold low latency in spite of increased demands.

The growth of computational power comes at a cost of complexity in both hardware and software. More complexity makes it hard to determine the runtime of tasks. Increased bandwidth is often traded for decreased latency predictability [Pat04]. Research shows that Virtual Reality applications require low latency systems but often ignores that modern computer systems may provide a certain average latency but the real latencies vary considerably around this average [SNL20a].

1.1 My work and their support

Research is a collective effort. This thesis will describe achievements in the plural form as “we” even though the research and writing of the text surrounding the reprinted papers is done by the author as attested in the Eidesstattliche Erklärung. Research builds upon other people’s research which is cited as necessary. There are other types of input that are not as obvious like general ideas of which topics to include, what direction research is headed in, that are the result of discussions among collegues and especially my supervisors Prof. Dr. Marc Erich Latoschik and Dr. Florian Niebling. They shaped the research and resulting papers and with it this thesis.

1.2 Contribution

This thesis discusses how latency varies over time, how to measure it, how to describe it and what consequences arise from latency and latency spikes. Latency is

an inherent property of computer systems and can cause negative effects. It is therefore important to understand latency and its implications.

This thesis provides proof that the author is able to conduct scientific research by including published, peer reviewed papers investigating the topic. This thesis is a cumulative thesis connecting scientific contribution on the field of latency in virtual reality applications to illustrate different aspects of latency.

Some research is only understood by some other researchers deeply involved in the topic [Lim98]. Latency, in contrast, is an important topic for virtual reality and needs to be accessible by more than a selected few. It needs to be known how to measure latency and that high latency needs to be combatted. The reason why there is high latency in a virtual reality system can be very specific to the used system and needs to be tested and improved on a case by case basis. The negative impact on the user, however, is always present. No researcher should harm users of their systems because they did not understand that measuring latency is important or the ignorance of not knowing how to measure.

This thesis will describe how we came to the conclusion that latency can be harmful and that it is necessary to measure and control latency. To guide this research, we devised two research questions.

Latency cannot be understood without observing it over time. Many researchers treat latency as if it is time invariant but in turn miss an important part of the dynamic. The first research question therefore guides the exploration to describe the real latency dynamics.

R_1 **How does latency behave in real-time interactive systems?**

We approach this question by decomposing it into two smaller research questions.

$R_{1.1}$ **How to measure latency?**
$R_{1.2}$ **How to describe latency?**

$R_{1.1}$ describes observing the latency but the measurement data alone is not enough to describe latency behaviour. There needs to be a way to make the measurement data understandable by humans, which is inquired in $R_{1.2}$.

With an idea of latency's behaviour, we can conduct experiments that research the effect of latency as described in the second research question:

R_2 **What are the effects of latency?**

Conducting experiments with latency necessitates the ability to use latency as an independent variable that can be varied. We therefore devise another research question necessary for R_2:

$R_{2.1}$ **How to simulate latency?**

Simulation allows the addition of latency on top of the always present latency in order to provide different latency conditions for experiments.

To summarise, the research questions are:

R_1 **How does latency behave in real-time interactive systems?**
- $R_{1.1}$ **How to measure latency?**
- $R_{1.2}$ **How to describe latency?**

R_2 **What are the effects of latency?**
- $R_{2.1}$ **How to simulate latency?**

1.3 Chapter Overview

These research questions guide the structure of this thesis. The following chapters discuss different research questions or different aspects of these questions.

Chapter 2 defines the terms "latency" as it emerges from "systems". It discusses why latency exists and provides the basis for the other chapters.

Chapter 3 introduces the temporal variability of latency. It discusses different systems and how latency behaviour differs between them. The observed systems are small and restricted to show how different latency behaves even with small differences

Chapter 4 provides approaches to measure latency of whole virtual reality systems. Approaches differ in the complexity and amount of information collected.

Chapter 5 offers a way to describe the latency's time variant behaviour.

Chapter 6 creates artificial latency jitter to provide a means to test the effect of varying latency in experiments.

Chapter 7 discusses effects of latency jitter with an experiment and related work.

Chapter 8 discusses the findings of the previous chapters and possible limitations.

Chapter 9 concludes the thesis with a brief summary of the contents.

Some research papers were published as part of this cumulative PhD thesis. Most of the papers are reprinted as part of this thesis. The author's contribution is described next to the reprinted paper and in Appendix A. The nature of a cumulative thesis means that some information is found in multiple papers such as a description of related work. The most thorough description is found in the paper "Latency and Cybersickness: Impact, Causes, and Measures. A Review" [SNL20a] in Chapter 7. This text being an addition to the papers therefore only tries to connect the papers and provide additional information where necessary without repeating information that is already present in the papers themself.

Chapter 2

DEFINITIONS

This thesis describes latency in computer systems with a focus on systems for virtual reality. Let us understand what latency means, what special latencies we are interested in and what these systems are that we discuss before diving into the analysis of latency.

2.1 Latency

Dictionaries know two definitions of latency. The first definition describes hidden or dormant processes that wait until they are fully developed. This meaning is most often used with diseases. They incubate und wait until effects are evident.

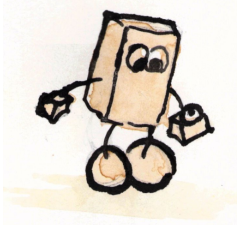
This thesis builds upon the second definition of latency, which describes latency as a delay in computation. The definitions are:

> "[uncountable, countable](computing) the delay before data begins to move after is has been sent an instruction to do so"
>
> Oxford Learner's Dictionary [Oxf]

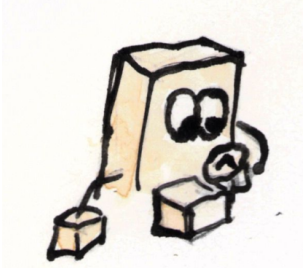
> "COMPUTING a measurement of delay in a system, especially the length of time it takes computer information to get from one place to another"
>
> MacMillan Dictionary [Mac]

Both definitions look at data that is present at one place at a certain point in time. It is then moved to arrive at a certain other place at a point in time after the initial time. The difference of the times or the measurement thereof is called latency. We will refer to the time difference as latency. The measurement of this time difference is described as a latency measurement.

As an analogy, imagine a piece of data to be animate.



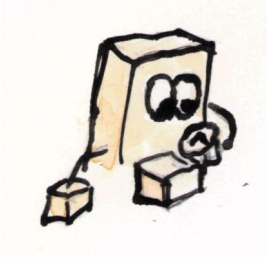
It looks at its watch ...



and enters a vehicle.

It drives to the beach ...

and looks at its watch again.

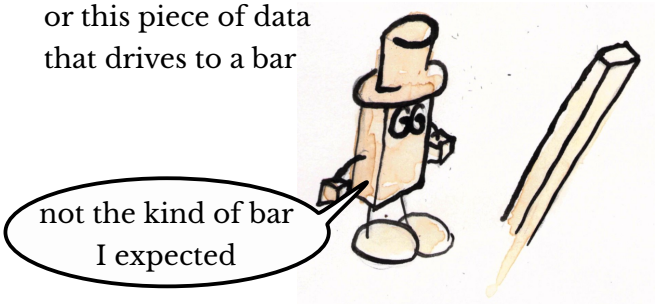


The time difference between the two times looked at the watch is the latency of the transportation. In other words: the roadtrip to the beach delays the arrival at the beach in comparison to already being there in the first place.

We often do not look at the data but at the processing time. In our example that would be the roadtrip. It describes an algorithm running that usually transforms and moves data. In our example, there could be other types of data,

like this piece of data that drives to a bar

or this piece of data that drives to a bar



We usually describe a similar set of data that travels to a restricted number of places with a common way of transportation. The example shows a car. The further described cases usually revolve around the execution of software in a computer system to act on data.

With the plethora of potential varying data and destinations, it becomes obvious that latency varies and it is difficult or impossible to describe its behaviour.

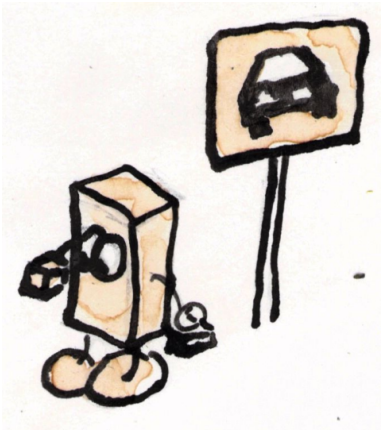
One day in our example, there may be construction work or a malevolent traffic light. The latency increases. The next day, everything is back to normal.

This transportation or processing of data is done with limited resources. Planning the best usage of these resources is a problem described in mathematics as the minumum latency problem. Blum et al. [Blu+94] describe it as a problem of a repairman or disk head scheduling its visits.

> "Given a set of n points, a symmetric distance matrix (d_{ij}), and a tour which visits the points in some order, let the latency of a point p be the length of the tour from the starting point to p. Let the total latency of the tour be the sum of the latencies of all points." Blum et al. [Blu+94]

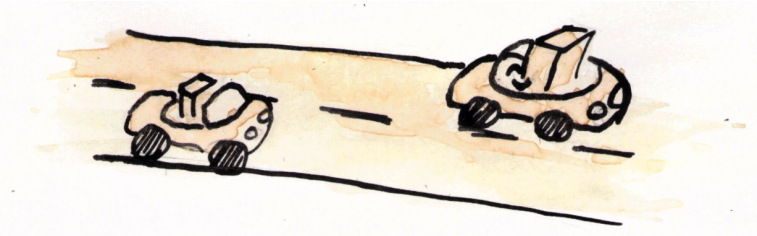
This definition presents in a more abstract way that one total latency is assembled from many smaller latencies. Each part contributes its own to the overall latency.

There is a limited resource that can work off requests one after the other. Work packets that are not currently worked on need to wait and therefore have their latency increased. Assume the data to not own a car each but use a taxi service that only owns few cars. While one piece of data rides the car, the others have to wait. Sometimes it is beneficial for the entirety of all data to only carry one piece of data halfways and then continue to transport another piece of data.



Similar, a processor has only a restricted amount of cores that can process data. The operating system schedules applications, thereby deciding which data gets processed first. The analysis of this problem finds it NP hard. We look further into the effects of this difficulty when we look at operating schedulers. Those schedulers face the same problem of needing to find a minumum latency schedule but need to optimize for different needs.

There often is a discrepancy between bandwidth and latency as nicely described in Patterson [Pat04]. Bandwidth is easier to increase with money and profits from improvements on latency while latency is limited by physical constraints and often does not benefit from improved bandwidth. Imagine the taxi service to increase its bandwidth by ordering more cars. It can now transport more data. Improving the latency of the transportation itself is more involved as it needs to find a way to make the cars faster. Rumble et al. [Rum+11] chimes in that latency — here in networking — has seen less improvement than bandwidth or memory size.



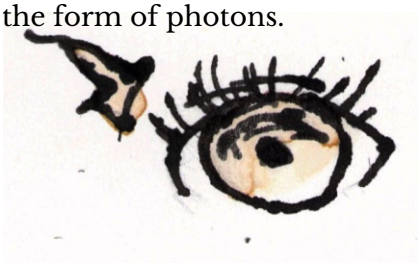
When describing latency in computers, the term often pops up to describe networking. For example Cardwell, Savage, and Anderson [CSA00] divides TCP latency into connection establishment latency and data transfer latency. The connection establishment latency is the time it takes to exchange the initial handshake to establish the connection. The transfer latency is the time from sending the requested data until the data is acknowledged. In line with the definitions above, both descriptions denote a time

until some feat is accomplished. Network latency can be very specific to certain computers [HVCT10].

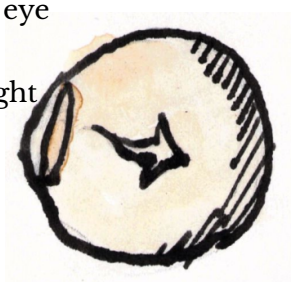
The movement of data from one place to another by ways of computing is not restricted to computer systems. The human processes data in the form of information arriving at the eye as photons or at the hands as tactile stimulation. More data is received by means of other senses. This data is shipped to the brain partially preprocessed where it is combined, connected, transformed to reach its final destination triggering a response. The response may be a movement, an utterance, a thought or a lot of other things.

The latencies we describe are often composed of many parts. Consider seeing:

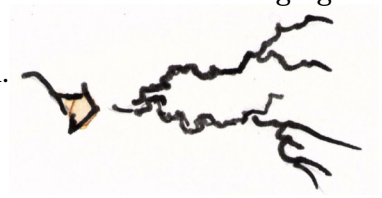
Data arrives at the eye in the form of photons.



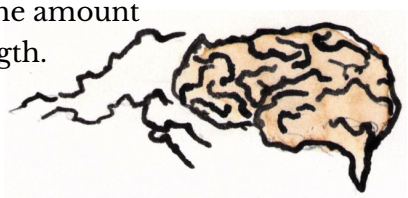
The latency from entering the eye until the arrival at the retina is proportional to the speed of light and the size of the eye.



The light is converted to an electrical stimulus by a receptor cell.



The conversion takes time. The time may be dependent on the amount of incoming light and its wavelength.



There is a latency from the creation of the electric stimulus until it reaches the brain. The optic nerve forwards the data but needs time to do so. The transmission speed of nerves is $34^{m}/_{s}$ [Nic97].

This simplifying example shows how the latency between the data arrival and the eye until its arrival in the brain is composed of multiple steps each contributing to the overall latency. The information is then processed in the brain with a delay but processes different information with different latencies [WM98]. Compensation of the latency is added later like in the motor actions triggered as a response [Pur+98]. This plethora of latencies in smaller parts of a computation to combine to one latency for a bigger part that may be different depending on what computational path is taken makes it important to always state which latency is reported.

Delays in the human processing are not subject of this thesis but only an example of other appearances of latency. The human has adapted to how own processing latency but as we will see not necessarily to external processing latency.

2.2 Systems

We often use the term "system" in a loose way, describing things that work together and are perceived as parts of this compound. The necessity of grouping and abstracting stems from the way humans learn. We are able to create abstractions from few samples. These abstractions then allow to adapt knowledge gathered from observed data to new, unobserved phenomena [Ten+11]. A system is the description of such an abstraction and as such, we give a definition below to provide a readymade definition and will then describe examples of the systems relevant to this thesis so the reader can generalize from there to find an own intuitive understanding.

There is the intuitive understanding that a system is a collection of cohesive parts. Wikipedia [Wik20] defines a system with reference to Backlund [Bac00] as

> "A system is a group of interacting or interrelated entities that form a unified whole" Wikipedia [Wik20]

Backlund's paper, however, warns that such a definition may be not precise enough and does not exclude everything that is not a system. His definitions is both more precise and harder to understand:

> "A system consists of a set, M, and a non-empty set of relations on M, R, satisfying the following conditions:
>
> 1. $|M| \geq 2$.
> 2. From every member of M there is a path to every other member of M.
>
> Let a and b any elements of M:
>
> 1. There is a path from a to b, if $(a, b) \in R_1$, where $R_1 \in R$.
> 2. Let the relation $R_2 \in R$ and the degree of R_2 be n, where $n > 2$. Let $a_1, a_2, \ldots, a_n$ be members of M. Let $\exists x(x \in \{a_1, a_2, \ldots, a_n\} \wedge x = a) \wedge \exists y(y \in \{a_1, a_2, \ldots, a_n\} \wedge y = b)$". Let $R_2(a_1, a_2, \ldots, a_n)$. Then there is a path from a to b, and there is a path from b to a.
> 3. There is a path from a to b, if there is an element $c \in M$ such that there is a path from a to c and there is a path from c to b."
>
> Backlund [Bac00]

He goes on to define that parts of a system that satisfy the definition are subsystems.

The definition states that a system is made from parts that have relations with each other. The relations need to be in a way to form a cohesive together where each part is connected to the rest either on a direct route or via relations to intermediate parts. No part may be isolated. Please refer to the paper for more details and a comprehensive explanation of how the definition can be understood.

The understanding of what a system describes becomes easier when talking about more concrete applications. The definition of Englander [Eng09] will be sufficient and more intuitive for the rest of this text.

> "A collection of components linked together and organized in such a way as to be recognizable as a single unit" Englander [Eng09]

2.3 Motion To Photon Latency

We will try to understand latency behaviour in the following chapters first by observing latencies of small, restricted mechanics. For the most part, however, we will talk about motion to photon latency. This is the latency that best describes the expected latency as experienced in virtual reality applications. It tries to span most of the latencies involved in the simulation of the virtual worlds and their interaction with the user.

So what is "motion to photon latency"? The name itself indicates a motion and a photon - here meant as the emission of a photon - as two points in time. The motion to photon latency is the delay in between. A patent issued to Qualcom defines

> "M2P latency is the time delay from a user's movement (e.g., head, eyes, or other body parts) to the time when a virtual reality scene is updated on a display" Quach et al. [Qua+18]

The document further stresses the importance of a low motion to photon latency to reduce unwanted effects such as cybersickness. We will discuss effects in more detail in Chapter 7.

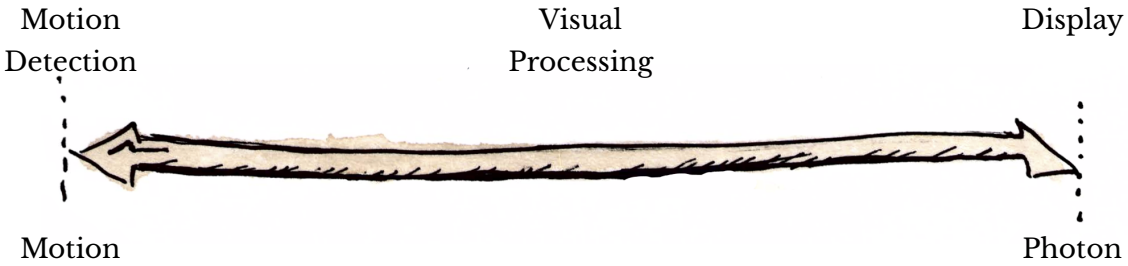
The definition restricts the source of movement to being created by the user of the virtual reality system. This is overly restrictive. A real use case will most often see a human operator that moves tracked objects. The human may be replaced by a synthetic movement from a pendulum's swing or a motor when measuring latency. This allows to define specific movement patterns that are replicatable. We will discuss measuring latency in Chapter 4. The important part of the movement to be usable to determine a motion to photon latency is its possible transferability to the computer. This is most often done with tracked objects that either report the movement themself to the computer (inside-out tracking) or are followed by some tracking system like a camera that then reports the movement to the computer (outside-in tracking) [Pin+02]. The amount of information transfered can vary from the positions and orientations over time to only one of position and orientation to binary information that indicates the presence or absence of movement or the tracked object reaching a certain point in space. Alternatives to tracked objects as the source of movement in the definition are mechanisms that can pick up a movement such as a light barrier.

Displays used for virtual reality applications vary widely. Virtual reality may use big screens, projections or head-mounted displays [Cze+97]. The motion to photon definition mentions the photon that arises from such a display because a photon is emitted at a specific point in time. The problem that a display updates its regions at different times is often neglected [PMK11]. The measurement of a motion to photon latency assumes that all parts of the screen are updated at the same time. An assumption that is justified by the much smaller update time for an entire display than for the total motion to photon latency. For our considerations, it is good enough to follow this assumption of instantaneous update of the whole screen but bear in mind that this might influence latency values.

Both the origin of the movement and the photon emitting device is not strictly defined. Motion to photon latency measurements can therefore measure very different scenarios. Some tracked objects use algorithms to reduce latency with prediction while others don't. An example is asynchronous timewarp [Wav16] which reduces the latency of the HMD orientation by warping the rendered image according to the most current tracking information. The shown content still uses older tracking information.

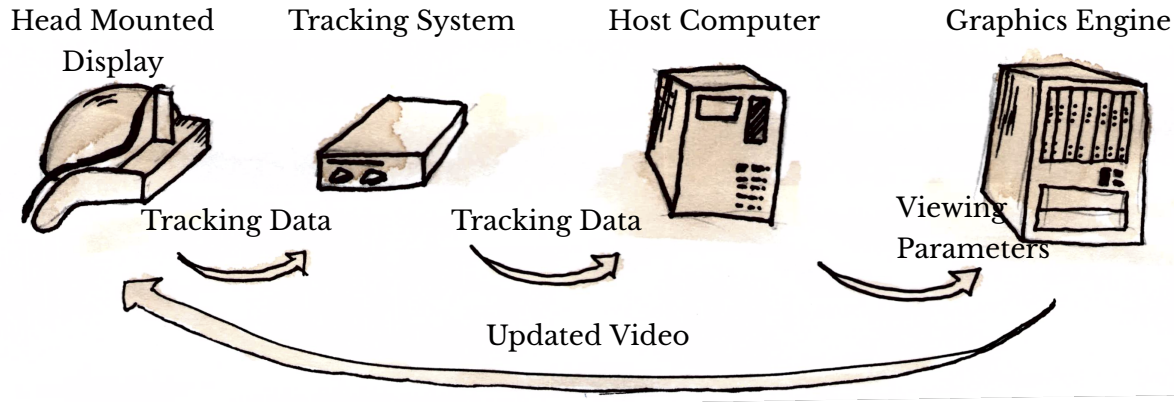
In between the movement and the photon corresponding to this movement, there are many processing steps. These steps are dependant on the hard and software used. Some papers that describe measuring motion to photon latency provide a rough explanation what between the motion and the photon emittal happens. We show adaptions of images in the original papers for illustration.

Choi et al. [Cho+18] describes it as



One step to register the motion and one to emit the photon. The description acknowledges that both picing up the movement and displaying it on a screen incurs a latency. Then there are some computations that convert the movement information into a pictoral representation.

Mine [Min93] lists four processing steps



He describes the visual processing step from before with two stages: application host and image generation. The distinction originates from the usage of separate machines back when the paper was published that do the general computation and the graphics procession. Today, these parts are more integrated but still separate in a computer as the main CPU and the graphics card.

Papadakis, Mania, and Koutroulis [PMK11] describes the pipeline in a similar way but describes the receival of tracking information with two parts: the tracker that picks up the motion and sends it to the host computer and a tracker driver that receives the information and forwards it to the simulation.

Wu, Dong, and Hoover [WDH13] shows an image that introduces buffers in between the stages. These buffers are used for message passing and depending on the setup can be buffers in memory for inter and intra process communication on one computer or buffers of network interfaces for communication between computers. This message passing does not only occur between the stages listed so far but also in the stages themself to facilitate the complexity that is necessary for virtual reality.

Kämäräinen et al. [Kä+17] shows a glance of how to split up some stages by describing the start and the end of the pipeline they used in more depth while the main processing that we usually focus on is sourced out to a server on the internet.

Feldstein and Ellis [FE20], just like Mine [Min93], recognises that the process is a cycle. The virtual reality as shown on the screen provoke the user to the next motion and the cycle repeats. Looking at the motion to photon latency is one part of the cycle of experiencing virtual worlds.

2.4 Latency Jitter

Latency is often described as one value. However, latency varies over time. Microcontrollers can show a deterministic execution time for applications. They execute only one application as described in its machine code. Desktop computers such as those used for virtual reality applications utilise a lot of optimisations to increase the amount of instructions executed per time. Many of those optimisations lead to a varying la-

tency. The complexity of the whole system with many sources of potential varying latency makes the observed latency unpredictable.

We will see that most of the observed latencies are usually gathered around a mean but there are outliers as well. We want to describe some of those optimisations to show how deeply rooted this indeterminism of latency is in todays computers.

Caching in the context of computer hardware usually refers to the usage of multiple layers of memory. Disks are slow compared to main memory. There is even faster memory closer to the processor. The faster the memory, the smaller it is, which is a cost and an implementation problem. The fastest memory are the processor registers followed by an L1, L2 and possibly L3 cache. The L1, L2, L3 caches load data from the main memory that is currently needed or may be needed soon after [CS06]. Not all processors have this elaborate cache hierarchy but many have at least one level of caches between the main memory and the processor. If the processor needs data that is not available in its closest cache, it will see to load it from slower memory. This is called a cache miss. Execution waits until this data is available. Optimisations see to fill the cache with data that will be needed [Pat+95]. Latency for an instruction is dependent on the cache behaviour. If there is a cache miss, the resulting latency will be higher.

Modern computers do not execute only one instruction at a time. They employ a superpipelined and superscalar processor architecture.

> "In a super pipelined machine of degree m, the cycle time is $1/m$ the cycle time of the base machine." Jouppi and Wall [JW89]

A simple instruction processing involves the fetching of the instruction, reading of the necessary registers, executing the instruction and writing the result back [BLM91]. This would be one cycle of the base machine described above. A super pipelined machine starts to fetch the next instruction after one was fetched, reads the registers for the next instruction after the registers for the previous one were read. At each time, there are as many instructions processed in the pipeline as there are pipeline stages — in the best case.

> "A superscalar machine of degree n can issue n instructions per cycle" Jouppi and Wall [JW89]

A superscalar pipeline has parts that are replicated to fetch multiple instructions at once or execute multiple instructions.

Those optimisations can achieve an increase in the throughput but require that instructions can be executed one after the other or in parallel. If one instruction relies on the result of the previous one, it needs to wait until the results are available. This reduces the benefits of superpipeline and superscalar machines. They will see to reorder instructions to mitigate this problem if possible. While this ensures a good pipeline utilisation when successful, it means that the developer can no longer be sure of the order

the instructions are processed in and looses the ability to estimate runtimes for a piece of code.

One kind of instruction that depends on previous results are conditional jumps. Speculative execution and branch predition lets processors assume the possible outcome before executing the instruction itself. The assumption of one possible case allows to already start to process instructions of the likely branch. If the assumption turns out to be false, all the speculative processing is discarded and the pipeline is filled with the correct instructions. A correct guess does not incur any additional execution latency while a wrong one does. This introduces uncertainty of the latency. The predictions may not be deterministic when other code is executed as well.

Hyperthreading sees to improve the pipeline utilisation by feeding instructions of two parallel threads into one physical processor. The operating system sees this one physical processor as two logical processors. If one thread needs to wait, e.g., for data due to a cache miss, the other one takes this free capacity. While this optimisation improves throughput even further, latency is not deterministic. An application can take a certain time to execute or less if the other thread needs to wait.

All of these factors can lead to time variant latency. The fluctuation in latency is called latency jitter.

Chapter 3

JITTER IN MESSAGE PASSING

This chapter shows the dynamic of one test scenario. It shows how latency differs in small processes. Larger systems will use this kind of processing in multiple places and accumulate more of the latency variability shown here.

The scenario is a simple message passing as is used in most applications. One example is the receival of mouse and keyboard input in the form of messages. Message passing is also employed to communicate between different applications and between parts of the same application. It is an essential part of virtual reality systems and computer systems in general. Virtual reality systems make excessive use of message passing due to their complex setups. The hardware communicates with the computer, involving drivers on the computer, a virtual reality application for the simulation. The simulation wants to maximise available performance by using multithreading which necessitates communication between the parallel executing threads. There is the communication with the graphics card to generate images which are sent to a screen.

The tests show that the latency behaviour differs between the approaches used for the message passing and the underlying, executing system.

3.1 Real-Time Systems

Real-Time systems require a certain timeliness for their execution. Each process is assigned a deadline after which it needs to have finished its computation.

> "Real-Time Systems are those in which the temporal aspects of their behavior are part of their specification. The correctness of the system depends not only on the logical results of the computation, but also on the time at which the results are produced." Bernat, Burns, and Liamosi [BBL01]

Real-Time systems and associated deadlines are described by Bernat, Burns, and Liamosi [BBL01] as

Hard where a missed deadline leads to system failure, potential harm or damage.

Firm where some deadlines may be missed but there is no value provided for late results.

Soft where some deadlines may be missed while still providing value for late results.

Only hard real-time systems can provide guarantees for latency. They provide a way to schedule each task to guarantee the required processing time before the required

deadline. It is possible to schedule tasks with these requirements if all tasks and their characteristics are known beforehand running on a single processor. Scheduling in multi-processor environments with more dynamics are NP-hard or NP-complete [Bur91]. There are heuristics to make multi-processor scheduling feasible in practice. Those schedulers might only provide a "minimum level of guaranteed performance" [Sta+95].

A real-time system will need to always regard the worst case processing time. Resource utilisation may be low if the worst case doesn't happen. Computer systems build for multimedia usage are usually optimised for throughput. They will not reserve capacity for worst case processing times nor care for processing deadlines. They will see to get as much work done as possible. Throughput comes at the cost of run-time predictability [McK08]. The systems used for virtual reality are not optimised to guarantee the necessary processing time to provide updated imaginary for every screen update.

Developers try to squeeze out the last drop of performance of a virtual reality system to provide better graphics, more effects or better simulations but ignore that computational deadlines may get missed if there are no buffers left to cushion longer processing times. The most obvious deadline is the graphics scanout when data is sent to the screen. If this deadline is missed, the next scanout will be in many milliseconds to come. Latency increases a lot in this case.

3.2 Jitter in Real-Time Systems

We published a poster on the IEEE VR 2016 that shows what kind of latency is to be expected for simple message passing. We compare different implementations in different programming languages and see their effect on a stock linux system and on a linux system with the PREEMPT_RT patch installed. The PREEMPT_RT patch tries to make the linux system real time capable. Following the explanation above, only a real time operating system can guarantee timeliness. We show that even our inspected real time system was not able to keep this promise.

The paper was published as Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems". In: *2016 IEEE Virtual Reality (VR)*. IEEE. 2016, pages 287–288

Reducing Application-Stage Latencies of Interprocess Communication Techniques for Real-Time Interactive Systems

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

ABSTRACT

Latency jitter is a pressing problem in Virtual Reality (VR) applications. This paper analyzes latency jitter caused by typical interprocess communication (IPC) techniques commonly found in today's computer systems used for VR. Test programs measure the scalability and latencies for various IPC techniques, where increasing number of threads are performing the same task concurrently. We use four different implementations on a vanilla Linux kernel as well as on a real-time (RT) Linux kernel to further assess if a RT variant of a multiuser multiprocess operating system can prevent latency spikes and how this behavior would apply to different programming languages and IPC techniques. We found that Linux RT can limit the latency jitter at the cost of throughput for certain implementations. Further, coarse grained concurrency should be employed to avoid adding up of scheduler latencies, especially for native system space IPC, while actor systems are found to support a higher degree of concurrency granularity and a higher level of abstraction.

Index Terms: D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming; D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Virtual Reality applications often consist of multiple aspects to handle input processing, simulations, artificial intelligence, or rendering etc. To fully take advantage of the available processing power of today's multicore and/or multi-CPU architectures and to avoid unnecessary blocking, the underlying software routines often will utilize concurrency and asynchronous behavior. At a certain point though, all parts have to cooperate and communicate to generate a consistent world state which usually uses some form of IPC technique. But IPC is heavily affected by scheduler latencies. Scheduler latency is almost unpredictable and might show spikes at uncontrolled points in time, resulting, e.g., in micro stutters. These outliers are presumably not perceived well and may cause increased simulator sickness.

Even without an explicit application concurrency, scheduling impacts the system as the available resources are shared in multiuser multitasking operating system (MMOS) commonly used today. Here, real-time operating systems (RTOS) give promises about an upper bound of scheduler latency. While they are common in, e.g., the embedded world or cyber-physical systems, today's VR applications usually run on an MMOS (e.g., simply, because there is enhanced graphics acceleration or many existing software solutions). We investigate how different programming languages and IPC techniques behave w.r.t latency jitter when applied on an MMOS compared to an RTOS.

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

IEEE Virtual Reality Conference 2016
19–23 March, Greenville, SC, USA
978-1-5090-0836-0/16/$31.00 ©2016 IEEE

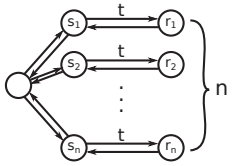


Figure 1: Schema illustrating the test programs: Each of the n senders is instructed at the same time to get the current timestamp, to send it to a receiver who additionally gets a timestamp and propagates back the elapsed time for logging.

2 PREVIOUS WORK

Frank et al. [3] found visual delay to be a major factor for simulator sickness. Ivkovic et al. [4] additionally found latency to influence the performance and experience of test subjects. While they conducted tests with a time invariant latency added, Teather et al. [8] found a reduced performance due to latency spikes. We assume therefore that latency spikes have a similar effect on the experience as degraded latency has. Motion-to-photon latency of VR environments has been measured using different methods, e.g. *sine fitting* [7]. Several current approaches optimize the rendering stage, e.g., using dynamic time warping or frameless rendering. Our research focuses on a different aspect of the latency problem, namely the latency that occurs prior to rendering, i.e., at the application stage of a VR system.

RT systems guarantee each process to be invoked within a certain time, therefore eliminating spikes in latency for IPC when the receiver has to wait an unbounded timespan until invokation. Most RTOSs are for embedded systems [6] with applications in robotics or industrial controllers. The Linux RT patch is a modification of the Linux Kernel that enables hard realtime capabilities [2]. Improvements in latency often inversely affects throughput. Real-time (getting started as quickly as possible) and real fast (getting done quickly once started) can be considered a design choice [5].

3 METHOD

We implemented a comparable test routine using two distinct IPC techniques and two programming languages: A thread based version using shared memory and mutex locks in C++ and Java and a message based communication with actors using Skala with Akka Actors and the C++ CAF library [1]. The routines start a variable number of pairs of threads/actors and tell the senders to get the current nanosecond time, read an integer from a random location in a memory block of 256k integers and send it to their peer thread/actor. The receiver waits for the hand-over, writes the received integer to another random position back in the memory block, reads the current nanosecond time and logs the difference of the two timestamps (see Figure 1). To ensure that all invoking threads/actors send at the same time, they are synchronized with a barrier. The pseudo-code is described in listing 1 and listing 2. The memory read/write is intended to reliably provoke cache misses for all cases.

The tests were conducted on a system running Ubuntu 14.04 with a Linux Kernel version of 3.14.57 with and without the RT

patch applied in a dual boot configuration. The CPU was an Intel© Core™ i7-2700K with 4 cores and hyperthreading. The employed JVM was an OpenJDK 2.5.6 with the Akka 2.3.11 library for Scala Actors. We increased the number of thread/actor pairs running at the same time by the power of two from 1 to 16. For each run, 10,000,000 samples were gathered.

```
barrier();
t = getTime()
x = readMemory(random)
send(t, x)
```

Listing 1: Sender reads the time and a random memory location and sends it to the receiver.

```
t1, x = receive()
writeMemory(random, x)
t2 = getTime()
log(t2-t1)
```

Listing 2: Receiver receives the information, saves the variable to a random memory location and calculates how much time has passed.

4 Results

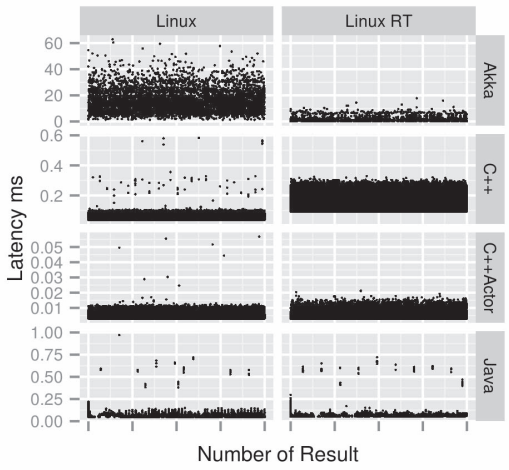


Figure 2: Plot of the latency distribution with 16 thread/actor pairs for all 4 test cases running on MMOS (left column) and on RTOS (right column) kernels. The x-axis shows the number of the result with the leftmost being the first gathered latencies while the rightmost were gathered at the end of the test run.

Figure 2 depicts the latency for each sample that took more than the median $+\,2 \cdot$ standard deviation. The C++ implementations are heavily affected by scheduler latency on the MMOS Linux and drastically reduce latency spikes using Linux RT at the cost of some lower overall performance only for the native threads. Both JVM-based implementations show repeated spikes, apparently caused by garbage collection. Here, Akka spikes are worse, as the message system seems to suffer under the many messages that are sent to log each result, therefore creating a lot of short-living objects. See Table 1 for comparison.

The threaded RTOS versions exhibit an increased latency with increasing number of threads, eventually surpassing the latency of outliers that happen with the MMOS versions. Therefore, a very fine-grained concurrency should be limited or an actor-based system should be applied, potentially due to the user-space implementation of these systems. We find the RTOS performing worse in the mean and median while also scaling worse with more threads.

Table 1: Comparison values for latencies with and without RT patch with 16 thread/actor pairs

	Max		Mean		Median	
		RT		RT		RT
Akka	62.9ms	17.8ms	29μs	4.1μs	6.7μs	2.2μs
Java	970.3μs	719.9μs	16.9μs	20.8μs	7.4μs	13.8μs
C++	583μs	329.1μs	13.1μs	29.8μs	3.4μs	5μs
C++ Actor	56.9μs	21.4μs	2.2μs	2μs	2.2μs	2μs

The latency, however, is better bound with the RTOS, not only for the C++ implementations. Without, there are repeatedly outliers that may deteriorate the user experience and may lead to simulator sickness.

5 Conclusion

Linux RT reduces latency jitter at the cost of some overall performance in the C++ case, an acceptable trade-off for VR systems. Additionally, system space IPC concurrency should be limited to a certain extent of granularity to reduce the impact of scheduler latency. Running on an RTOS, the Actor model provides a valuable alternative for an increased degree of concurrency granularity, specifically using the C++ implementation. Still, with our implementation based on the Java VM, latency spikes could not be lowered so far as their cause is not the system scheduler but the garbage collector (GC). Future work will evaluate if different GC implementations and parametrization can alleviate this problem.

Overall, while VR applications need concurrency and modularity to handle all required software tasks, communication can induce problems if proper care is not taken and adequate performance measures are not performed frequently as a standard procedure. We have only looked at a basic $n \times (1:1)$ IPC but see the need to extend the research to test the impact of different approaches as well as to extend the technical analysis with user-based perception studies to relate technical measures to perceived qualities, e.g., to see if and how it makes sense to trade performance for a bounded latency.

References

[1] D. Charousset, R. Hiesgen, and T. C. Schmidt. Caf-the c++ actor framework for scalable and resource-efficient applications. In *Proceedings of the 4th International Workshop on Programming based on Actors Agents & Decentralized Control*, pages 15–28. ACM, 2014.

[2] S.-T. Dietrich and D. Walker. The evolution of real-time linux. In *7th RTL Workshop*, 2005.

[3] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[4] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and mitigating the negative effects of local latencies on aiming in 3d shooter games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 135–144. ACM, 2015.

[5] P. E. McKenney. "Real Time" vs. "Real Fast": How to Choose? In *Ottawa Linux Symposium (July 2008), pp. v2*, pages 57–65, 2008.

[6] J. A. Stankovic. Real-time and embedded systems. *ACM Comput. Surv.*, 28(1):205–208, Mar. 1996.

[7] A. Steed. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pages 123–129, New York, NY, USA, 2008. ACM.

[8] R. Teather, A. Pavlovych, W. Stuerzlinger, and I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50, March 2009.

Copyright

Author Contributions

The author programmed the test software, conducted the measurements and took the lead in writing the manuscript.He provided critical feedback and helped shape the research, the analysis and the manuscript.

3.3 Jitter Results of Scheduler and Garbage Collector Choice

We extended this work on the SEARIS workshop 2016 to include comparisons of different operating schedulers and different garbage collectors when using the Java programming language.

Real time depends on the choice of scheduling that grants processes the necessary computing time to finish in time. Scheduling happens at operating system level or in the process itself.

The measurements show applications where scheduling of threads is left to the operating system with different scheduling algorithms. An alternative are actors that are executed by multiple operating threads but are scheduled on top of these threads by the application. Other things that are scheduled are parts that belong to the runtime environment of the application like garbage collectors. There is a plethora of available algorithms which show different latency behaviour when executing the same application. This paper sees to explore effects of these different choices.

The paper was published as Jan-Pilipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Reducing application-stage latencies for real-time interactive systems". In: *2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE. 2016, pages 1–7

Reducing Application-Stage Latencies For Real-Time Interactive Systems

Jan-Philipp Stauffert* Florian Niebling† Marc Erich Latoschik‡
University of Würzburg University of Würzburg University of Würzburg

ABSTRACT

Latency is a pressing problem in Virtual Reality (VR) applications. Low latencies are required for VR to reduce perceptual artifacts and cyber sickness. Additionally, latency jitter denotes the variance in the pattern of latency changes which additionally may cause unwanted effects. This paper analyzes latency jitter caused by typical inter-thread communication (ITC) techniques commonly used in todays computer systems employed for VR, the influence of the operating system scheduler, and the effect of different garbage collection (GC) methods to understand their effect on latency spikes, here for different Java Virtual Machines (JVM). We measure the scalability and latencies for various ITC techniques with an increasing number of threads and actors performing prototypical concurrent tasks. Four different benchmark implementations on a vanilla Linux kernel as well as on a real-time (RT) Linux kernel assess if a RT variant of a multiuser multiprocess operating system can prevent latency spikes and how this behavior would apply to different programming languages and ITC techniques.

We confirmed that scheduler and prioritization of the VR application both play an important role and identified the impact they have on the implementation strategies. Also, Linux RT can limit the latency jitter at the cost of throughput for certain implementations. As expected, the choice of a GC method also is critical and will change the latency patterns drastically. As a result, we suggest that coarse grained concurrency should be employed to avoid adding up of scheduler latencies and unwanted latency jitter for the native ITC case, while actor systems are found to support a higher degree of concurrency granularity and a higher level of abstraction.

Index Terms: D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming; D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Virtual Reality applications often consist of multiple components to handle input processing, simulations, artificial intelligence, or rendering etc. Non-functional software quality requirements like modularity, maintainability, and reusability can have an unforeseeable impact on the temporal behavior of software, especially for a Real-Time Interactive System (RIS), i.e., in Virtual, Augmented, and Mixed Reality (VR, AR, and MR) and computer games. Due to the complexity of many RIS applications, they are often split into different parts to foster cohesion and decoupling. To exploit todays' multi-core and multi-CPU architectures and to avoid unnecessary blocking, these parts often will be executed concurrently or they will be completely distributed [2, 15].

At a certain point though, all parts have to cooperate and communicate to generate a consistent world state which implies some sort of inter-process communication (IPC) or inter-thread communication (ITC). Hence it is critical to understand the impact of IPC/ITC on the resulting latency patterns. IPC/ITC is heavily affected by scheduler latencies. Scheduler latency is almost unpredictable and might show spikes at uncontrolled points in time, resulting, e.g., in micro stutters. As a result, latency and latency jitter can severely disturb the performance and experience of users and it can cause simulator sickness.

Even without an explicit application concurrency, scheduling impacts the system as the available resources are shared in multiuser multitasking operating system (MMOS) commonly used today. Here, real-time operating systems (RTOS) give promises about an upper bound of scheduler latency. While they are common in, e.g., the embedded world or cyber-physical systems, today's VR applications usually run on an MMOS. We investigate how different programming languages and ITC techniques behave w.r.t latency and latency jitter when applied to (a) an MMOS compared to (b) an RTOS. Three languages will be used: (1) C++ to create native binaries and (2) Java and Scala targeting the JVM. We finally compare threading with mutex locks to an actor model implementation.

2 RELATED WORK

An early discussion of simulator sickness is led by McCauley et al [17]. Frank et al. [9] found visual delay to be a major factor for simulator sickness. Ivkovic et al. [12] additionally found latency to influence the performance and experience of test subjects. While they conducted tests with a time invariant latency added, Teather et al. [22] found a reduced performance due to latency spikes. We assume therefore that latency spikes have a similar effect on the experience as degraded latency has. Users might be able to compensate for a predictable overall latency when it comes to interaction tasks (not for the perception though) but they can't compensate for unpredictable latency spikes.

Recent work has been done to reduce motion-to-photon latency of VR environments. Here, the overall goal is to apply the most current sensor reading as late as possible in the final graphics rendering stage to avoid, e.g., the application of an outdated camera projection. This latency cause has been measured using different methods, e.g. *sine fitting* [21]. Several current approaches optimize the rendering stage, e.g., using dynamic time warping or frameless rendering, *light sensing* [5], and *automated frame counting* [10]. Our research focuses on a different source for the latency problem, namely the latency that occurs prior to rendering, i.e., at the application stage of a VR system.

RT systems guarantee each process to be invoked within a certain time, therefore eliminating spikes in latency for ITC when the receiver has to wait an unbounded timespan until invocation. Most RTOSs are for embedded systems [20] with applications in robotics or industrial controllers. The Linux RT-Preempt patch modifies the Linux kernel to enable hard realtime capabilities [6]. This allows the comparison of software performance under both operating system flavors. Other operating systems exist only in either MMOS or RTOS variant. Improvements in latency often inversely affects throughput. Real-time (getting started as quickly as possible) and real fast (getting done quickly once started) can be considered a design choice [18].

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de
IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS) 2016
20 March, Greenville, SC, USA
978-1-5090-4275-3/16/$31.00 ©2016 IEEE

3 METHOD

We will examine differences in ITC latency jitter on a MMOS and RTOS to evaluate whether an RTOS can prevent critical latency spikes and how different programming languages and ITC concepts need to be adapted. We distinguish between two platforms: Native binaries are compared to bytecode running on the JVM. Additionally, the traditional multithreading approach with threads and mutexes is compared to the abstraction of actors.

Threads are used for concurrent flows of execution inside of one process. Here, mutexes allow threads mutually exclusive access to a resource with threads waiting for a resource being able to yield their execution time to another thread or process. Mutexes therefore allow for better real-time behavior than spinlocks that poll for a resource to be available [6]. Actors on the other hand provide an abstraction to facilitate parallel programming usually based on threads and lock-free communication as used for VR applications in [14]. Actors are entities that run in parallel and which solely communicate by message passing [13].

3.1 Test Routines

We implemented a comparable test routine using two distinct ITC techniques and two programming languages:

1. thread based using shared memory and mutex locks in C++
2. thread based using shared memory and mutex locks in Java
3. message based with actors using C++ with the CAF library [4]
4. message based with actors using Scala with Akka [16]

The routines start a variable number of pairs of threads/actors and tell the senders to get the current nanosecond time, read an integer from a random location in a memory block of 256k integers and send it to their peer thread/actor. The receiver waits for the handover, writes the received integer to another random position back in the memory block, reads the current nanosecond time and logs the difference of the two timestamps (see Figure 1). To ensure that all invoking threads/actors send at the same time, they are synchronized with a barrier. The pseudo-code is described in listing 1 and listing 2. The memory read/write is intended to reliably provoke cache misses for all cases. In larger applications, cache misses will certainly occur due to the increased code and large assets, therefore urging the processor to load data from the slower main memory instead of the much faster caches. All tests shown here collect 10,000,000 samples. Latency jitter is introduced by, among others, the OS scheduler, other processes and hardware interrupts that delay the communication.

The actor implementations use default settings. Akka uses by default a fork-join thread pool with work stealing and three times the amount of threads than processors as target amount. Threads are created or dismissed according to the work to do. The C++ Actor Framework uses by default a thread pool with the same amount of threads than processors with a central coordinating scheduler [4].

```
barrier();
t = getTime()
x = readMemory(random)
send(t, x)
```

Listing 1: Sender reads the time and a random memory location and sends it to the receiver.

```
t1, x = receive()
writeMemory(random, x)
t2 = getTime()
log(t2-t1)
```

Listing 2: Receiver receives the information, saves the variable to a random memory location and calculates how much time has passed.

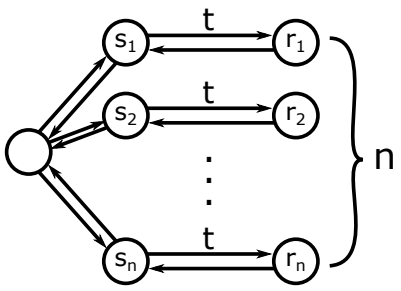


Figure 1: Schema illustrating the test programs: Each of the n senders is instructed at the same time to get the current timestamp, to send it to a receiver who additionally gets a timestamp and propagates back the elapsed time for logging.

3.2 Hardware

The tests were conducted on a computer running Ubuntu 14.04.3 with a Linux Kernel version of 3.14.57 with and without the RT-Preempt patch applied in a dual boot configuration. The CPU was an Intel© Core™ i7-2700K with 4 cores and hyperthreading. The employed JVMs were an OpenJDK 2.6.3 with the Akka 2.3.11 library for Scala Actors and for GC comparison additionally the Zing ZVM version 1.8.0-zing_15.09.0.0-b6.

3.3 Schedulers

Linux supports multiple schedulers with different use cases [3]. If no special scheduler is requested, Linux defaults to the "other" scheduler (`SCHED_OTHER`). Real-time scheduling is done with `SCHED_FIFO`, which implements a FIFO principle, `SCHED_RR`, a round robin approach, or recently `SCHED_DEADLINE`, which executes the thread with the earliest deadline first.

The C++ implementations will use the round-robin scheduler for it shows the best results in terms of limiting latency jitter for our use case. The implementations running on the JVM are evaluated both for the `SCHED_OTHER` and the `SCHED_RR` as they are impacted differently by the choice of the scheduler. `SCHED_DEADLINE` will be evaluated in later work.

The test programs are run with the round robin scheduler at priority 90, which is above most other processes with the exception of certain kernel processes like "watchdog" and "migration" that are essential for the proper functioning of the OS. When using the default scheduler, no further prioritization like nice values are used.

Hardware interrupts will nonetheless be served immediately urging other processes to wait. With threaded interrupts this time is held as short as possible with a big part then taken care of in a kernel thread that is subject to the scheduler [7].

The choice for a scheduler is a sensitive one. It should be evaluated which one performs best for the software at hand. While our test implementation with Scala/Akka performs better in terms of latency jitter with the default scheduler as shown below, it doesn't make any promises or efforts for real-time behavior and should not be favored for RT scenarios.

3.4 Vanilla vs. RT-Preempt

All four implementations were run on Linux with and without the RT-Preempt patch applied using 16 threads or actor pairs. Figure 2 depicts the latency for each sample that took more than the median $+\,2 \cdot$ standard deviation. Table 1 shows the absolute performance values for comparison.

The C++ implementations are heavily affected by scheduler latency on the MMOS Linux and drastically reduce latency spikes using Linux RT. The native thread implementation sees a decrease

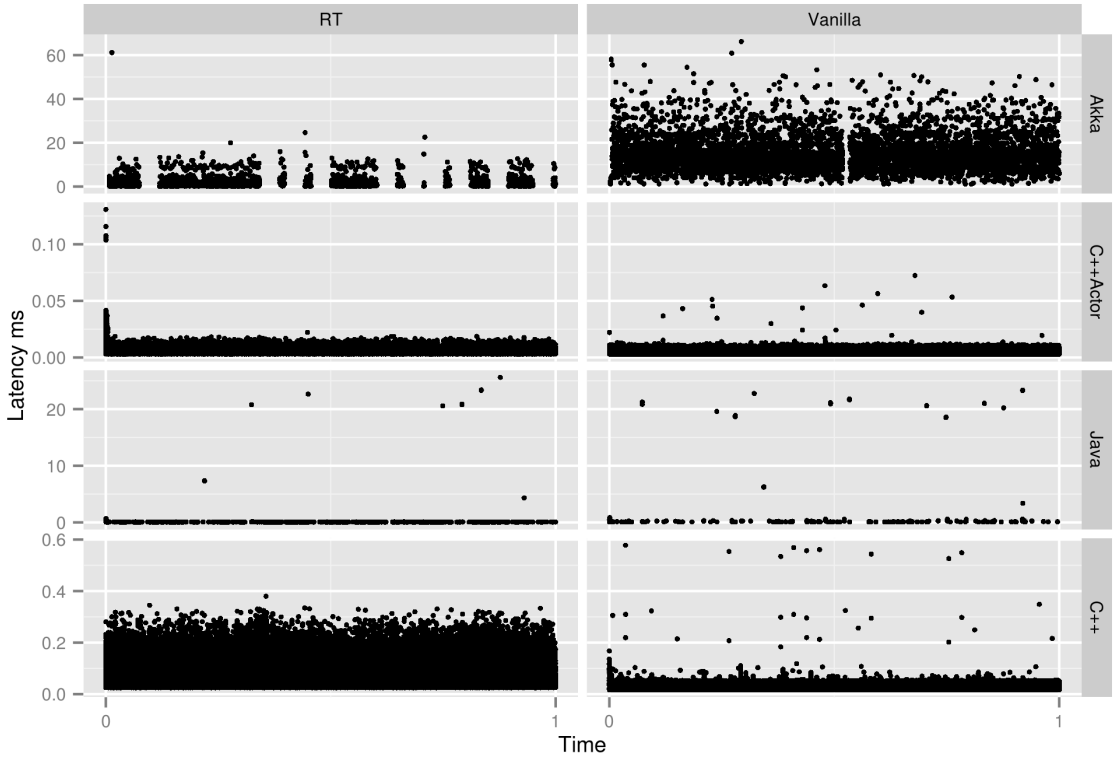


Figure 2: Plot of the latency distribution with 16 thread/actor pairs for all 4 test cases running on MMOS (right column) and on RTOS (left column) kernels. The x-axis describes the normalized time from start to the end of a test-run while collecting 10,000,000 samples.

	Max		Mean		Median	
	RT	Vanilla	RT	Vanilla	RT	Vanilla
Akka	67ms	63.4ms	1.9ms	15.9ms	915.8μs	14.5ms
Java	24.7ms	23.3ms	684.2μs	106.2μs	144.3μs	67.5μs
C++	380.2μs	577.9μs	48μs	34.8μs	43.1μs	35.3μs
C++Actor	130.8μs	72.5μs	4.1μs	3.7μs	3.5μs	3.2μs

Table 1: Comparison values for latencies with and without RT patch with 16 thread/actor pairs

in average performance with the RT patch but shows fewer outliers that are bound to a lower maximum latency. The C++ Actor implementation benefits from the reduced maximum latency while not suffering the same performance degradation as the native threads. However, there are spikes in the beginning of the test run on the RTOS which are not present on the MMOS. This hints to the actor initialization having more impact there.

3.5 Garbage Collectors

Preliminary studies with a modified version of the here presented tests showed the GC as a major impacting factor for latency jitter.

The Java Garbage Collection is as beneficial for the language as it poses problems. It allows for reliable software as a consequence of the nonexistence of memory corruptions and memory leaks. The well known problems of the garbage collection are temporary program stalls while the GC is conducting its work. Many application areas don't mind pauses in execution under 1 second. Due to Java's ubiquity especially in the business world, it has advanced to areas where latency plays a crucial role for the business value like in high frequency trading [19].

For our tests, we use the four different GCs that are implemented in the OpenJDK, which are the *Serial*, *Parallel*, *Concurrent Mark Sweep* (CMS) and *G1* GCs. Additionally, the Zing JVM [23], which promises pause free GC, is examined.

Figure 3 shows the measurements for the Scala/Akka implementation with Figure 4 showing the respective measurements for the Java thread implementation. The time was normalized to the range $[0;1]$ but different settings led to differing run times. With the exception of the Zing GC, every test has information added where the garbage collection took place. Java garbage collection is divided into two different steps, the *Young generation* (YG) GC and the *Old generation* (OG) GC, where short living objects are faster to

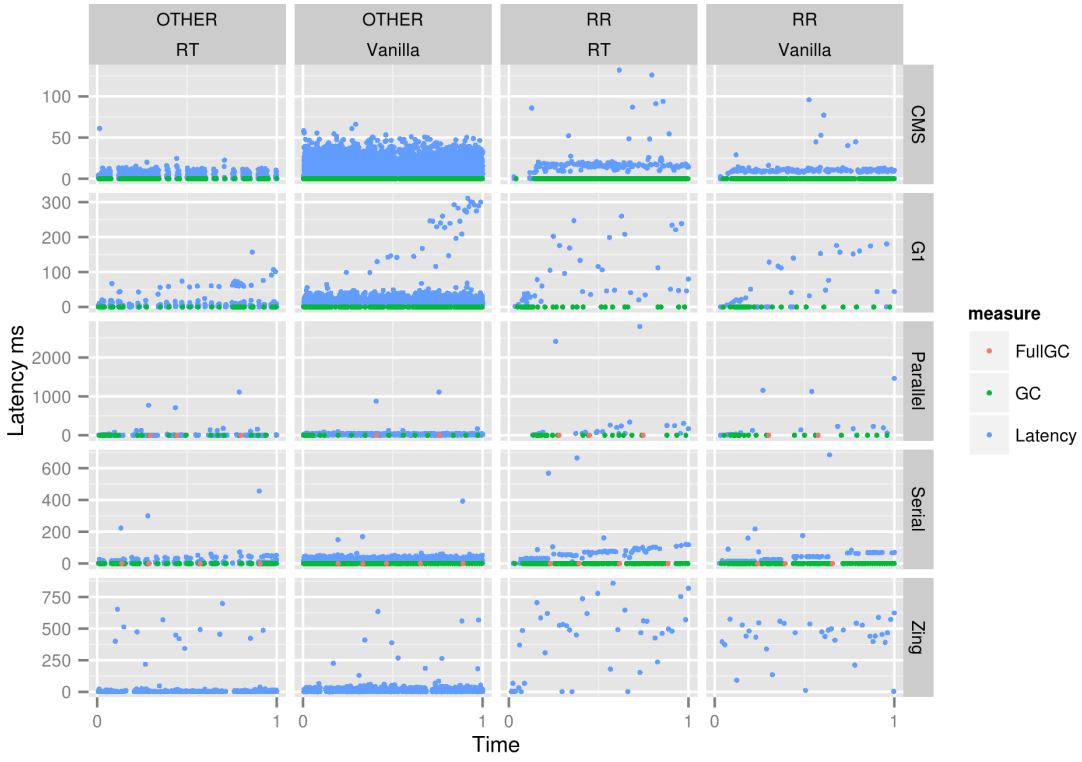


Figure 3: Comparison of different GCs for the Scala/Akka implementation using 16 actors with the default Linux scheduler (SCHED_OTHER) and the round robin scheduler (SCHED_RR) running on a Linux with and without the RT patch applied. The y-axis is differently scaled for each test to better convey the individual behaviour that would otherwise get lost due to the big difference in latency behaviour.

		Max		Mean		Median	
GC	Scheduler	RT	Vanilla	RT	Vanilla	RT	Vanilla
CMS	SCHED_OTHER	67ms	63.4ms	1.9ms	15.9ms	915.8μs	14.5ms
	SCHED_RR	191ms	212ms	18.3ms	12ms	14.9ms	9.5ms
G1	SCHED_OTHER	241ms	216.8ms	3.4ms	15.3ms	1.1ms	14.1ms
	SCHED_RR	235.9ms	210.6ms	45.1ms	8.8ms	28.9ms	454.8μs
Parallel	SCHED_OTHER	911ms	1110.2ms	3.1ms	17.1ms	1.8ms	14.8ms
	SCHED_RR	3756.6ms	192ms	212.7ms	46.3ms	62.3ms	13.8ms
Serial	SCHED_OTHER	631.8ms	385.1ms	4.2ms	16.4ms	1.9ms	14.9ms
	SCHED_RR	1423.5ms	717.4ms	90.8ms	42.6ms	58.6ms	24.4ms
Zing	SCHED_OTHER	744.3ms	387.5ms	11.2ms	13.7ms	2.7ms	11.4ms
	SCHED_RR	720.1ms	655.6ms	137.7ms	126.6ms	4.8ms	4ms

Table 2: Comparison values for latencies for the Scala/Akka implementation running with different GCs. The tests were conducted with and without RT patch with 16 actor pairs

get collected. Longer living objects and those surviving the YG GC are cleaned up less often but with more impact on the system. As the test routines are only shortlived and only create objects to pass the results around, they mostly don't provoke a *Full Garbage Collection* that sweeps the OC. Depending on the GC, this can happen nonetheless as is seen for the *Serial* and *Parallel* GC. There, spikes in latency are the result of the longer running full GC. If full GC is provoked for the other GCs, the impact is comparable.

The runtime and frequency of the full GC can be adjusted with the assignment of different ratios of the available memory to the YG and OG. This can be done to trade less frequent garbage collection for an increased stall time of the program but doesn't change the phenomenon of programm pauses in general. While this makes it possible to delay garbage collection in our tests until after the test run, we kept the memory assignment with default values to have our tests exhibit common behaviour.

Furthermore, the memory usage was not optimized for this case. It is possible to work around the GC by allocating memory without

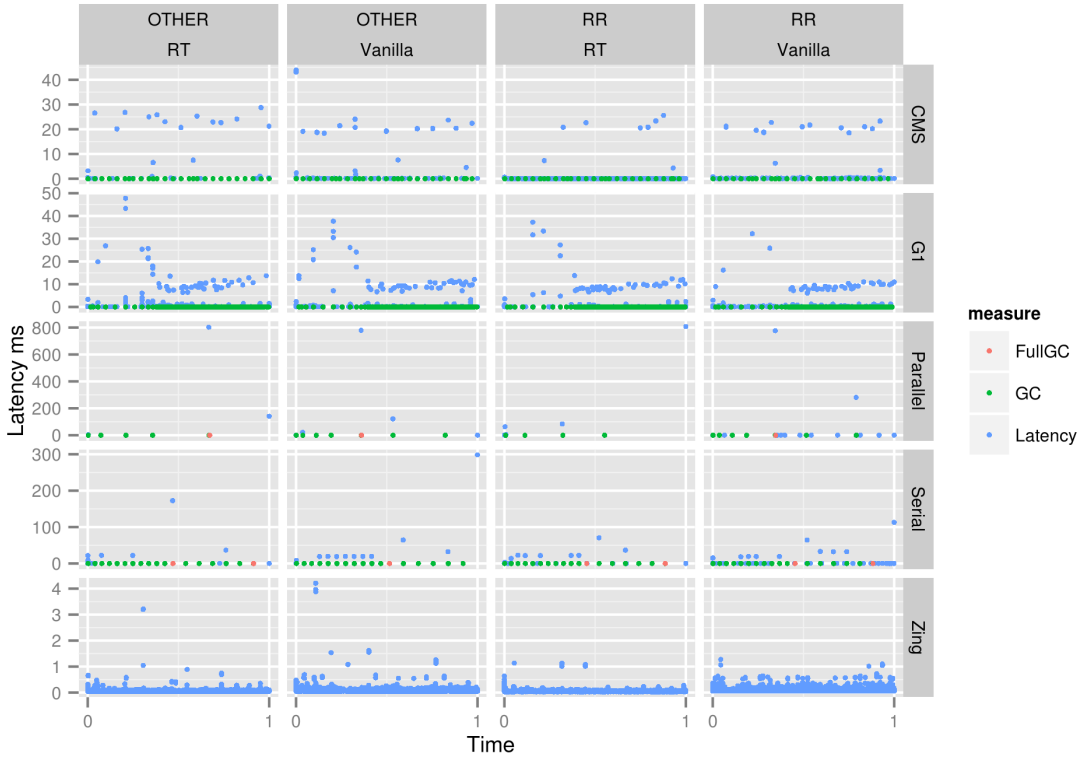


Figure 4: Comparison of different GCs for the Java thread implementation using 16 threads with the default Linux scheduler (SCHED_OTHER) and the round robin scheduler (SCHED_RR) running on a Linux with and without the RT patch applied. The y-axis is differently scaled for each test to better convey the individual behaviour that would otherwise get lost due to the big difference in latency behaviour.

its knowledge that has then to be managed manually. Additionally, the GC's work can be alleviated by reusing objects. Since these measures are not used in common Java programming, they are omitted for the comparison here.

The tests were run with both the FIFO and the round robin scheduler because the decision for the SCHED_RR is not as obvious as in the C++ tests.

4 Results

Both JVM-based implementations show repeated spikes, apparently caused by garbage collection. Here, Akka spikes are worse, as the message system seems to suffer under the many messages that are sent to log each result, therefore creating a lot of short-living objects. The values in Table 1 and Figure 2 show selected combinations of GC and scheduler (Scala/Akka: Concurrent Mark Sweep GC, SCHED_OTHER; Java: Concurrent Mark Sweep GC, SCHED_RR). The Akka measurements show far longer mean latencies which is a result of an implementation problem. The increasing delay occurred when changing the test program to not only send the measured latency but also the time when it occurred for logging reasons. The reason for this has to be assessed in the future. This problem makes it not possible to compare the Akka version to the other implementations but only to itself. Preliminary studies showed Akka's performance comparable or better than Java's but were not yet replicated with different GCs.

Both actor implementations show a better scaling with more parallelism. More threads lead to more mean and maximum latency, while the actor values are growing at a slower pace.

The Scala/Akka implementation shows more and larger latency spikes with the SCHED_RR which should support RT, but profits from the changes to Linux with the RT-Preempt patch if no RT scheduler is used. All test runs show outliers, which can be partially explained with full GC for the *Serial* and *Parallel* GC. The Java implementation has similar patterns for the *Serial* and *Parallel* GC, where repeated spikes are provoked by the full GC but besides this the latencies stay bound. The other GCs show very specific patterns with the *CMS* GC exhibiting repeated spikes, the *G1* GC starting with large latencies that go down to then slowly increase again and the *Zing* GC that has times with higher latencies around certain points in time though still very low latencies in comparison to other GCs.

We tried to introduce pauses of 1ms after each measurement to see if the OS and GC can make use of this time window of inactivity. The Scala/Akka implementation reduced its outliers roughly by a factor of four, which is probably due to skipping certain internal mechanics that cause latency spikes. It is more a working around spikes by hoping that certain characteristics fall into the pause than a real solution. The C++ implementation showed a slight increase in latency, which is explained by the added overhead of waking up the main application that then invokes all the threads that have been

GC	Scheduler	Max		Mean		Median	
		RT	Vanilla	RT	Vanilla	RT	Vanilla
CMS	SCHED_OTHER	57.1ms	25.9ms	4.7ms	141.6μs	201.8μs	74.2μs
	SCHED_RR	24.7ms	23.3ms	684.2μs	106.2μs	144.3μs	67.5μs
G1	SCHED_OTHER	37.1ms	33.7ms	1.1ms	623.5μs	128μs	109μs
	SCHED_RR	32.5ms	31ms	734.3μs	572.9μs	125.6μs	87.8μs
Parallel	SCHED_OTHER	324.4ms	750.8ms	14.8ms	27.7ms	405.3μs	563.4μs
	SCHED_RR	1447.5ms	761.4ms	299.6ms	172.8ms	58.1ms	49.9ms
Serial	SCHED_OTHER	304.8ms	32.6ms	38.8ms	70.9μs	21.7ms	66.8μs
	SCHED_RR	124.3ms	165.8ms	2ms	5.2ms	198.6μs	288.8μs
Zing	SCHED_OTHER	3.7ms	5.6ms	61μs	72.1μs	58.7μs	65.5μs
	SCHED_RR	2.9ms	3.3ms	50.8μs	90.9μs	49.3μs	83.9μs

Table 3: Comparison values for latencies for the Java thread implementation running with different GCs. The tests were conducted with and without RT patch with 16 thread pairs

idle in the mean time.

While we tried CPU pinning for the thread implementations, it didn't affect the measurements. CPU pinning describes the process of assigning a thread to a CPU. If this is not done, the scheduler is free to schedule a thread each time it is resumed on a different CPU. If a thread changes the CPU, the cache of the new CPU might not have pre-loaded the required data and additional time is wasted to load the program code and data. This is most likely due to our implementation being too small in terms of executable size to make a difference. Other use cases e.g. in big data applications found a significant improvement of throughput by assigning threads to CPUs [8].

5 Discussion

The threaded RTOS versions exhibit an increased mean and maximum latency with increasing number of threads, eventually surpassing the latency of outliers that happen with the MMOS versions. Therefore, a very fine-grained concurrency using many OS threads should be limited or an actor-based system should be applied. An actor-system implemented in user-space can make better use of the application's concurrency without burdening the OS scheduler with huge amounts of threads. We find the RTOS performing worse in the mean and median while also scaling worse with more threads. The scaling is negatively affected by the additional scheduling complexity in Linux RT where mutexes in the kernel lead to more context switches and therefore more overhead. The latency, however, is better bound with the RTOS, not only for the C++ implementations. Without, there are repeatedly outliers that may deteriorate the user experience and may lead to simulator sickness.

Applications can change their behaviour in different environments. Therefore, it is important to test every application in different settings to determine the best configuration. Here, the C++ Actors have an initialization cost under Linux RT and should therefore be created in program sections where latency outliers have lesser impact. With the unpatched Linux, this adaptation is not needed. Even more varying behaviour is shown by our Akka test, which performs better either on the patched or unpatched Linux depending on the scheduler and garbage-collector.

Furthermore, the JVM does a lot of optimizations behind the scenes [11] that we did not fully account for that can lead to different behaviour in other scenarios. The test applications presented here are small and run fast to allow quick testing of various settings. Larger applications might be handled differently and get more and more optimized the longer they are running. In the financial sector, some JVMs are "warmed up" for hours before they are put in production [1].

Only 1:1 communication with a certain amount of threads/actors in parallel at the same time was investigated. In a VR application, the communication might not be in sync. Input devices will report their measurements at different times and their messages are propagated through the program components in different paths. The amount of entities taking part in a communication will vary. This research is trying to start the evaluation of latency jitter sources for a restricted communication pattern, which will be present in more complicated settings as well.

The analysis only sheds light onto a very selected piece of VR applications. Even in this restricted test setting, there are many influencing factors that can provoke latency spikes that can only partially be avoided. Hardware/firmware interrupts can influence the CPU in a way that is not preventable from an application side. Bigger applications will suffer from even more sources of latency due to the underlying hardware and operating system, which makes controlling latency jitter even more difficult.

6 Conclusion

Linux RT reduces latency jitter at the cost of some overall performance in the C++ case, an acceptable trade-off for VR systems. Additionally, system space ITC concurrency should be limited to a certain extent of granularity to reduce the impact of scheduler latency. Running on an RTOS, the Actor model provides a valuable alternative for an increased degree of concurrency granularity, specifically using the C++ implementation. Still, with our implementation based on the Java VM, latency spikes could not be lowered so far as their cause is not the system scheduler but the GC. Different GCs provoke special latency patterns that need to be found out for every application anew and be considered. As long as a language running on the JVM is the choice for a VR project, the GC has to be accounted for.

Overall, while VR applications need concurrency and modularity to handle all required tasks, communication can induce problems if proper care is not taken and adequate performance measures are not performed frequently as a standard procedure. We have only looked at a basic $n \times (1:1)$ ITC but see the need to extend the research to test the impact of different approaches as well as to extend the technical analysis with user-based perception studies to relate technical measures to perceived qualities, e.g., to see if and how it makes sense to trade performance for lower latency spikes.

There is a need for special hardware and software to avoid latency jitter and provide an immersive experience. The VR community has identified the problem of latency and is working on it. We want to stress that low mean latency is not enough but the jitter has to be kept low as well.

References

[1] Azul Preps Java For Trading – Avoid Practice Trades Leaking Into Markets - Forbes. http://www.forbes.com/sites/tomgroenfeldt/2014/03/20/azul-preps-java-for-

trading-avoid-practice-trades-leaking-into-markets/#33d6bbbe2c78. Accessed: 2016-02-01.

[2] T. Arcila, J. Allard, C. Ménier, E. Boyer, and B. Raffin. Flowvr: A framework for distributed virtual reality applications. *Journees de lAFRV*, 2006.

[3] D. P. Bovet and M. Cesati. *Understanding the Linux kernel*. " O'Reilly Media, Inc.", 2005.

[4] D. Charousset, R. Hiesgen, and T. C. Schmidt. Caf-the c++ actor framework for scalable and resource-efficient applications. In *Proceedings of the 4th International Workshop on Programming based on Actors Agents & Decentralized Control*, pages 15–28. ACM, 2014.

[5] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence: Teleoper. Virtual Environ.*, 19(6):569–584, Dec. 2010.

[6] S.-T. Dietrich and D. Walker. The evolution of real-time linux. In *7th RTL Workshop*, 2005.

[7] J. Edge. Moving interrupts to threads [LWN.net], Oct. 2008.

[8] A. Foong, J. Fung, and D. Newell. An in-depth analysis of the impact of processor affinity on network performance. In *Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on*, volume 1, pages 244–250. IEEE, 2004.

[9] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[10] S. Friston and A. Steed. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):616–625, April 2014.

[11] V. Hork, P. Libi, A. Steinhauser, and P. Tma. DOs and DON'Ts of Conducting Performance Measurements in Java. pages 337–340. ACM Press, 2015.

[12] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and mitigating the negative effects of local latencies on aiming in 3d shooter games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 135–144. ACM, 2015.

[13] R. K. Karmani and G. Agha. Actors. In *Encyclopedia of Parallel Computing*, pages 1–11. Springer, 2011.

[14] M. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, pages 9–17, March 2012.

[15] M. E. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, pages 9–17. IEEE, 2012.

[16] Lightbend. Akka, 2016.

[17] M. McCauley, L. Hettinger, T. Sharkey, and J. Sinacori. *The effects of simulator visual-motion asynchrony on simulator induced sickness*. American Institute of Aeronautics and Astronautics, 2015/11/26 1990.

[18] P. E. McKenney. "Real Time" vs. "Real Fast": How to Choose? In *Ottawa Linux Symposium (July 2008), pp. v2*, pages 57–65, 2008.

[19] L. O. Ramirez. High frequency trading. Technical report, Working Paper, 2011.

[20] J. A. Stankovic. Real-time and embedded systems. *ACM Comput. Surv.*, 28(1):205–208, Mar. 1996.

[21] A. Steed. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pages 123–129, New York, NY, USA, 2008. ACM.

[22] R. Teather, A. Pavlovych, W. Stuerzlinger, and I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50, March 2009.

[23] G. Tene, B. Iyengar, and M. Wolf. C4: The continuously concurrent compacting collector. *ACM SIGPLAN Notices*, 46(11):79–88, 2011.

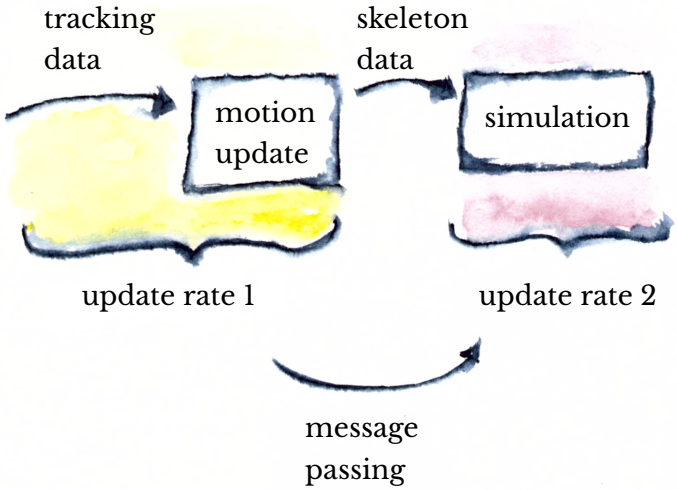
Copyright

Author Contributions

The author programmed the test software, conducted the measurements and took the lead in writing the manuscript.He provided critical feedback and helped shape the research, the analysis and the manuscript.

3.4 Applications of Message Passing

Tracking of user movement is important to present a convincing virtual representation of the user and react to user interaction. We contributed a chapter to the VR Developer Gems book describing the importance of avatar embodiment in virtual reality and how behaviour replication is implemented. The essence of the described mechanism is that tracking information arrives at its own cadence with the virtual reality simulation running at a different cadence. A part of the application receives the tracking information and forwards it as described in the above papers [SNL16a; SNL16c] to the main simulation. The book chapter describes how essential this mechanic is for embodiment.



The book chapter was published as Daniel Roth, Jan-Philipp Stauffert, and Marc Erich Latoschik. "Avatar Embodiment, Behavior Replication, and Kinematics in Virtual Reality". In: *VR Developer Gems, 1st Edition*. Edited by William R. Sherman. A K Peters/CRC Press, 2019, pages 321–346. url: `https://www.taylorfrancis.com/books/e/9781315157764/chapters/10.1201/b21598-17`

Copyright

Author Contributions

The author created the first draft explaining the receival and processing of tracking data as well as the pseudocode listings. He then developed this section in collaboration with the other authors.

3.5 Conclusion

Latency in even a small application varies. The latency pattern changes with the implementation and underlying systems such as the operating system. Some programming languages such as Java use a runtime that introduces additional and irregular latencies with a garbage collector.

We looked at message passing as one process that is present in many applications and essential for virtual reality applications that want to use as much computer performance as possible. Message passing is influenced by scheduling and multi processor coordination which make it susceptible to many of the described sources of latency and varying latency. With this essential building brick of larger applications showing such

varied latency patterns, we expect the effects to accumulate in such larger applications. We also showed an example where such message passing is in the critical path and can therefore impact the main application performance.

Discussed Research Questions

R_1 **How does latency behave in real-time interactive systems?** We find that latency behaviour differs for the same task if different algorithms are used. Latency spikes occur with different duration and frequency for different implementations. Operating system schedulers are one source for latency indeterminism. A real-time operating system reduces variability of latency values but does not eliminate it. Programming languages use different technologies like garbage collectors that introduce different latency behaviour. The same Java application shows different latency behaviour when used with different garbage collector implementations. Sources of latency variability can influence each other as shown with a Java application running on real-time and non-real-time linux operating systems using different scheduler implementations and different garbage collectors. As a consequence, we see that latency behaviour is tied to a specific implementation running on a specific system. Changes of hardware, operating system or implementation change latency behaviour for the same implemented concept.

$R_{1.1}$ **How to measure latency?** We show how to measure latency for small algorithms to receive latency measurements. The chosen task to time is message passing. Message passing is an important building block for virtual reality applications or in general applications that need to utilise multicore processors or distributed processing. Me We use the most accurate built in timing functionality and seek to reduce the influence of the timing instrumentalisation on the timed task. Nonetheless, timing on the same system as the application to time can introduce its own latency contribution.

Chapter 4

MEASURING

Measuring latency is a necessity to understand latency behaviour. Measurements can uncover problems in the implementation if it is too high [DL10]. Those problems might have gone unnoticed otherwise and can be fixed. Not all sources of latency can be fixed and so besides obvious errors in the implementation, it is important to know what latency to expect.

4.1 Overview of Measuring Approaches

Many researchers have conducted latency measurements with different approaches. The following paper shows an illustrated overview over those approaches to get a quick impression on what there is. A more textual description that discusses more nuances is done in the Frontiers paper on Cybersickness [SNL20a] in Chapter 7.

Here, we provide a quick overview over measuring latency to lower the barrier for other researchers to the topic. Measurement is always important and not everybody will be an expert in the topic. This paper facilitates basic knowledge. A visual explanation makes it more likely that interested readers approach the subject [Far18].

The paper was published as Jan-Philipp Stauffert, Kristof Korwisi, Florian Niebling, and Marc Erich Latoschik. “Ka-Boom!!! Visually Exploring Latency Measurements for XR”. in: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI EA ’21. Yokohama, Japan: Association for Computing Machinery, 2021. isbn: 9781450380959210 5. doi: 10.1145/3411763.3450379. url: https://doi.org/10.1145/3411763.3450379

Ka-Boom!!! Visually Exploring Latency Measurements for XR

Jan-Philipp Stauffert
University of Würzburg

Kristof Korwisi
University of Würzburg

Florian Niebling
University of Würzburg

Marc Erich Latoschik
University of Würzburg

ABSTRACT

Latency can be detrimental for the experience of Virtual Reality. High latency can lead to loss of performance and cybersickness. There are simple approaches to measure approximate latency and more elaborated for more insight into latency behavior. Yet there are still researchers who do not measure the latency of the system they are using to conduct VR experiments.

This paper provides an illustrated overview of different approaches to measure latency of VR applications, as well as a small decision-making guide to assist in the choice of the measurement method. The visual style offers a more approachable way to understand how to measure latency.

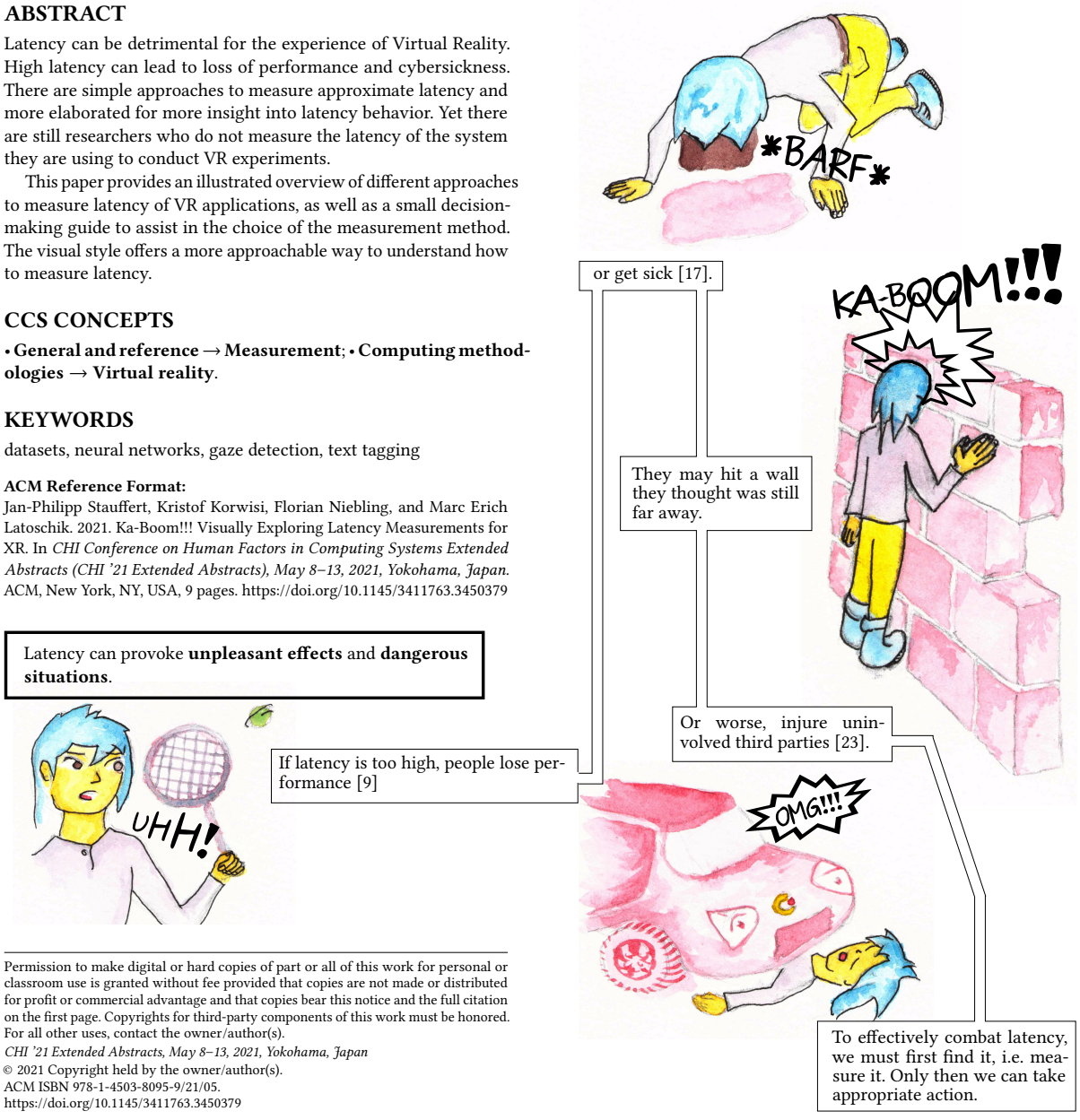
CCS CONCEPTS

• **General and reference** → **Measurement**; • **Computing methodologies** → **Virtual reality**.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

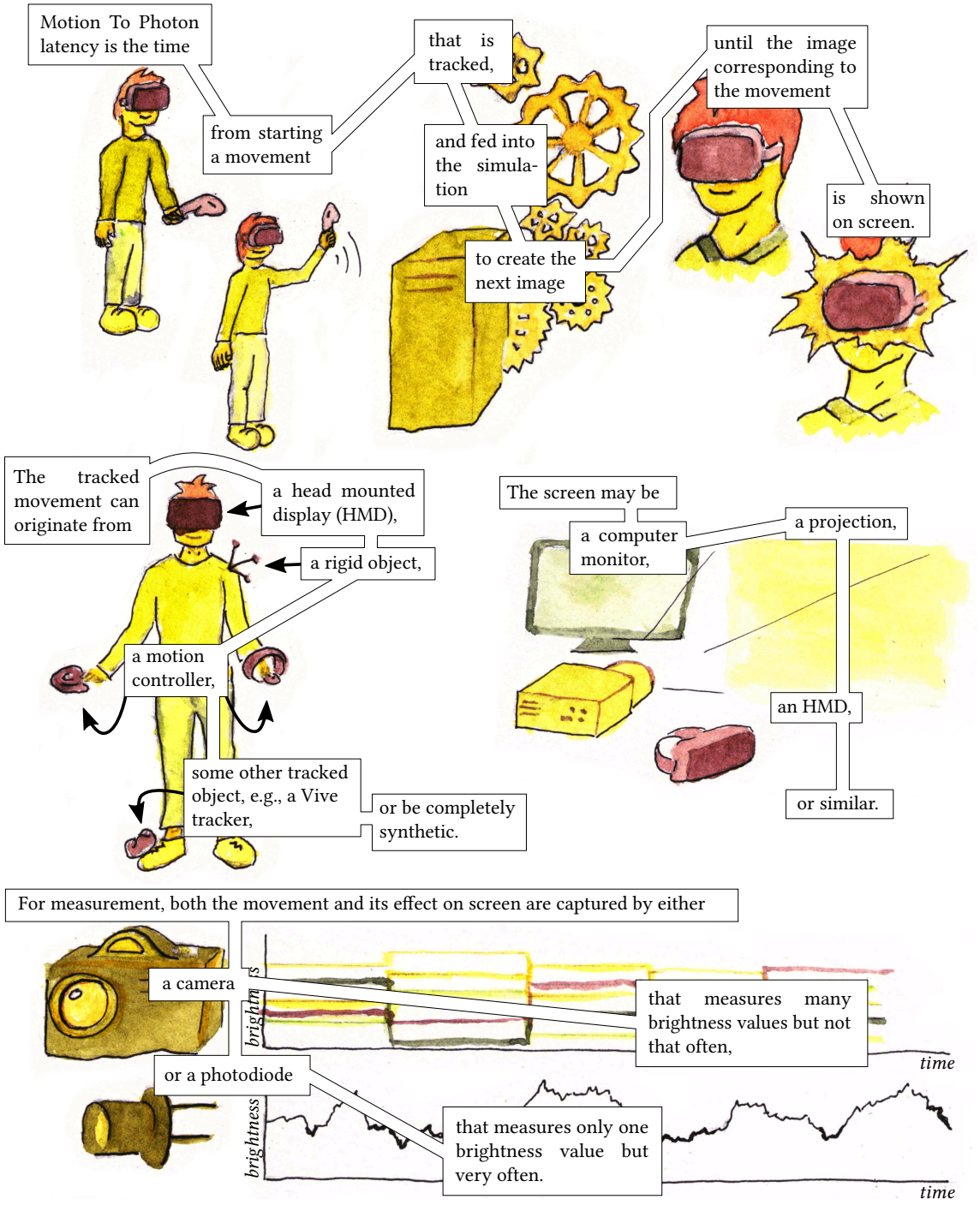
ACM Reference Format:
Jan-Philipp Stauffert, Kristof Korwisi, Florian Niebling, and Marc Erich Latoschik. 2021. Ka-Boom!!! Visually Exploring Latency Measurements for XR. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '21 Extended Abstracts), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3411763.3450379

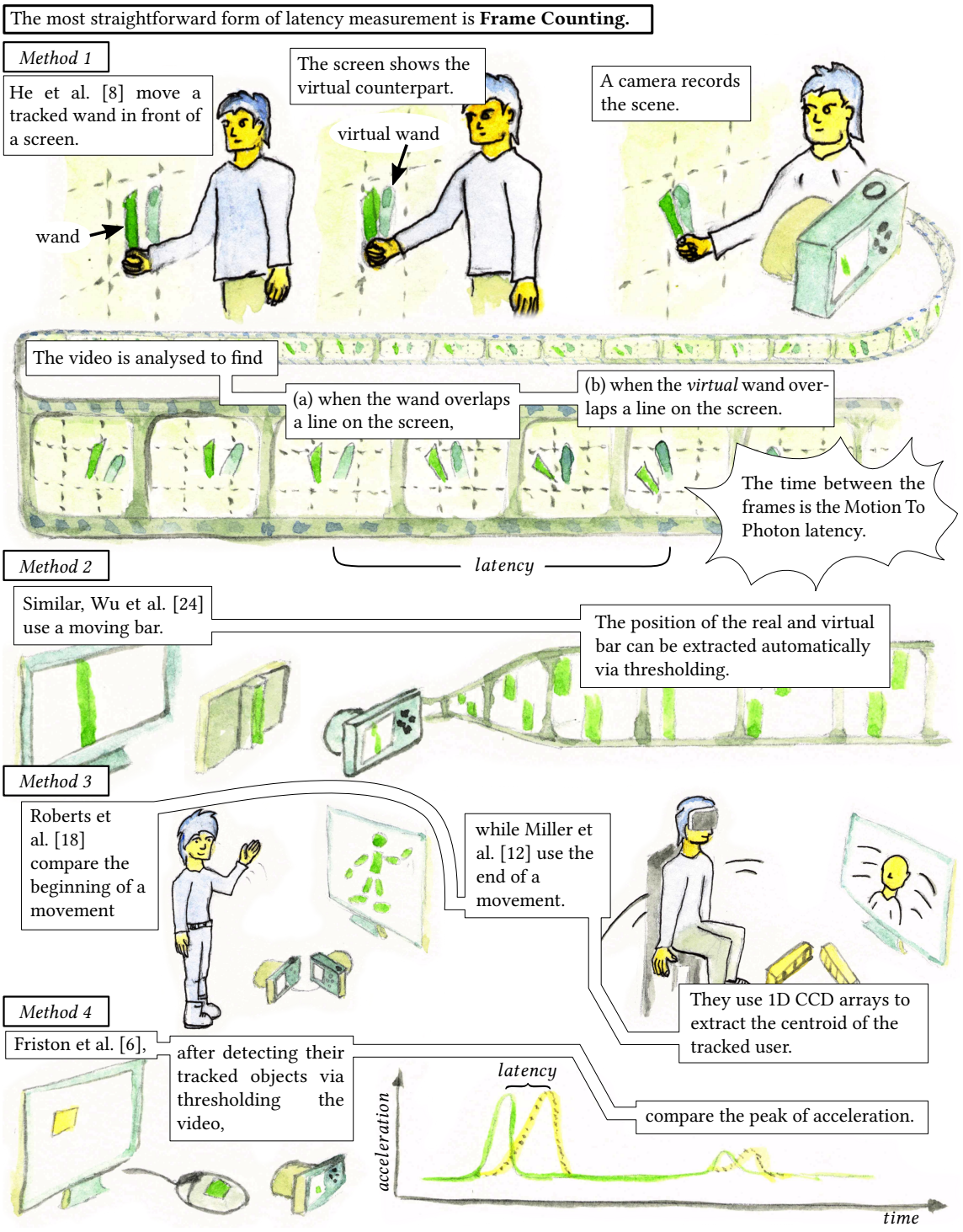


CHI '21 Extended Abstracts, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8095-9/21/05.
https://doi.org/10.1145/3411763.3450379

When talking about latency, we usually refer to **Motion To Photon Latency** or End To End Latency.




The most straightforward form of latency measurement is **Frame Counting.**
Method 1
He et al. [8] move a tracked wand in front of a screen.
The screen shows the virtual counterpart.
A camera records the scene.
virtual wand
wand
The video is analysed to find
(a) when the wand overlaps a line on the screen,
(b) when the *virtual* wand overlaps a line on the screen.
The time between the frames is the Motion To Photon latency.
latency
Method 2
Similar, Wu et al. [24] use a moving bar.
The position of the real and virtual bar can be extracted automatically via thresholding.
Method 3
Roberts et al. [18] compare the beginning of a movement
while Miller et al. [12] use the end of a movement.
They use 1D CCD arrays to extract the centroid of the tracked user.
Method 4
Friston et al. [6],
after detecting their tracked objects via thresholding the video,
compare the peak of acceleration.
latency
acceleration
time

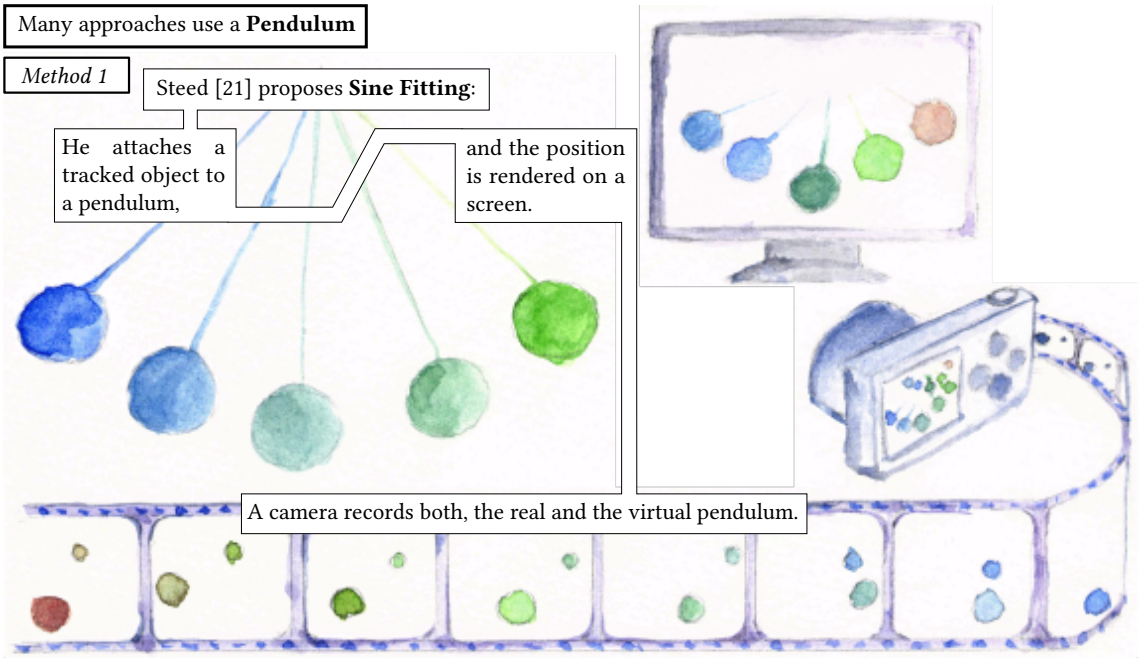
Many approaches use a **Pendulum**

Method 1

Steed [21] proposes **Sine Fitting**:

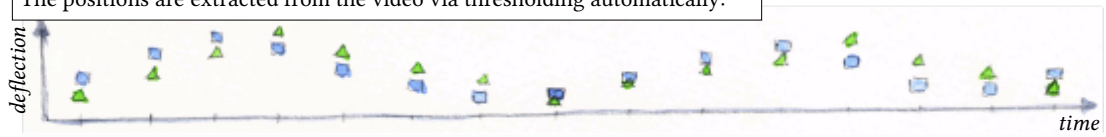
He attaches a tracked object to a pendulum,

and the position is rendered on a screen.



A camera records both, the real and the virtual pendulum.

The positions are extracted from the video via thresholding automatically:



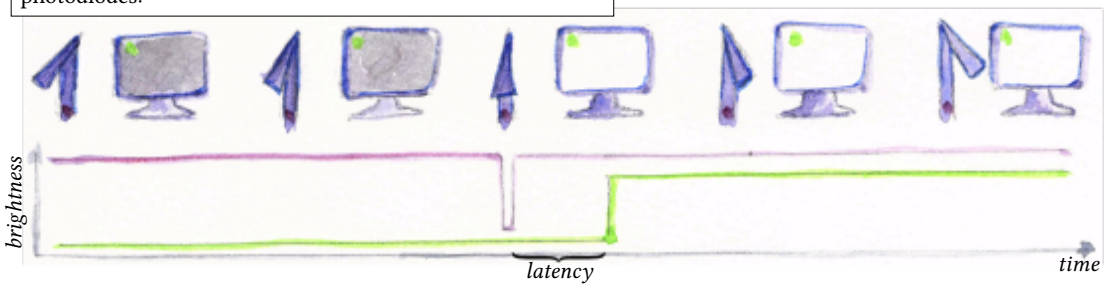
He fits a sine curve (e.g. $cos(2\pi * frequency * deflection + phase)$). The fit prevents inaccuracies of the thresholding, and low sampling frequency in the video.

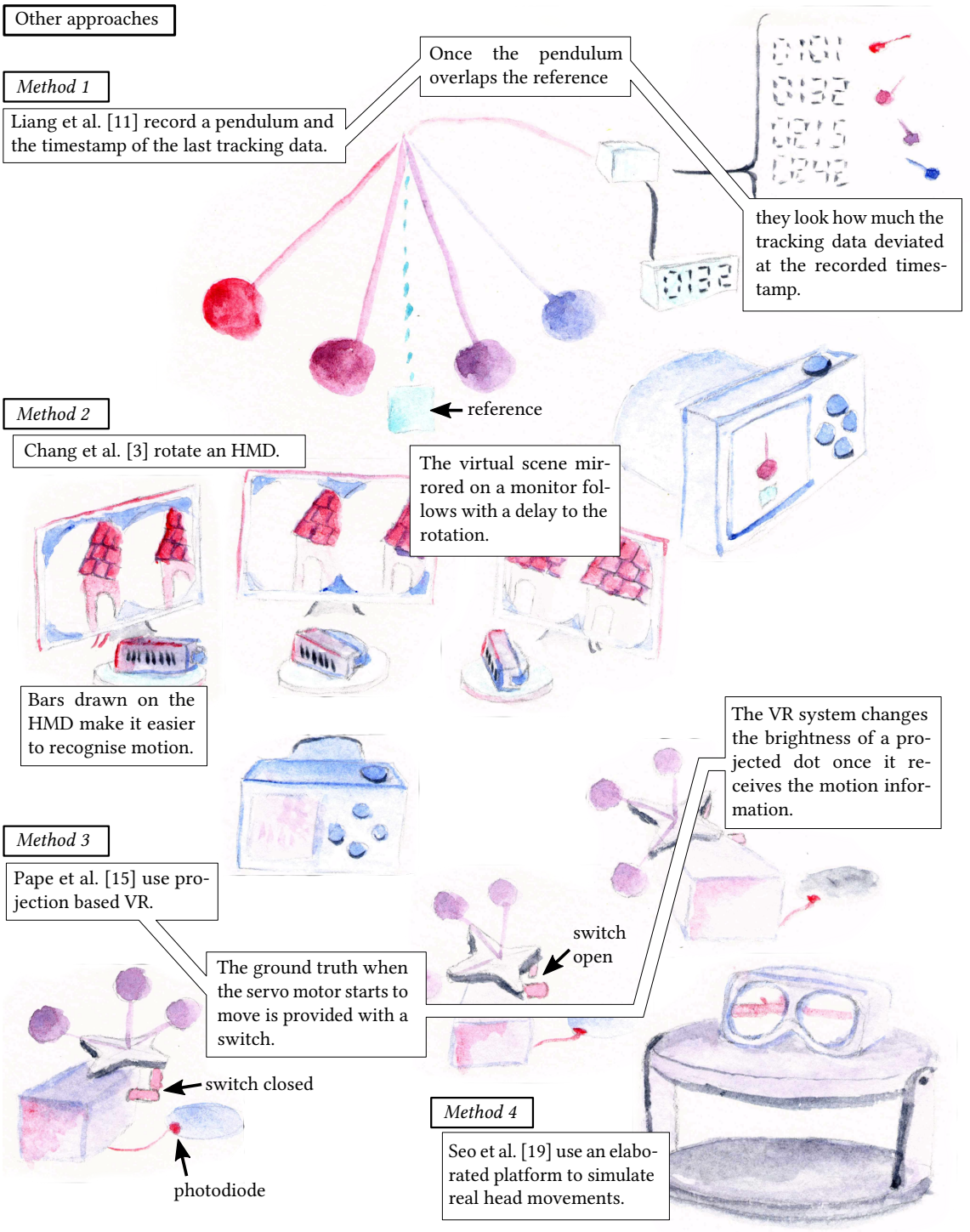
The motion to photon latency can be derived as

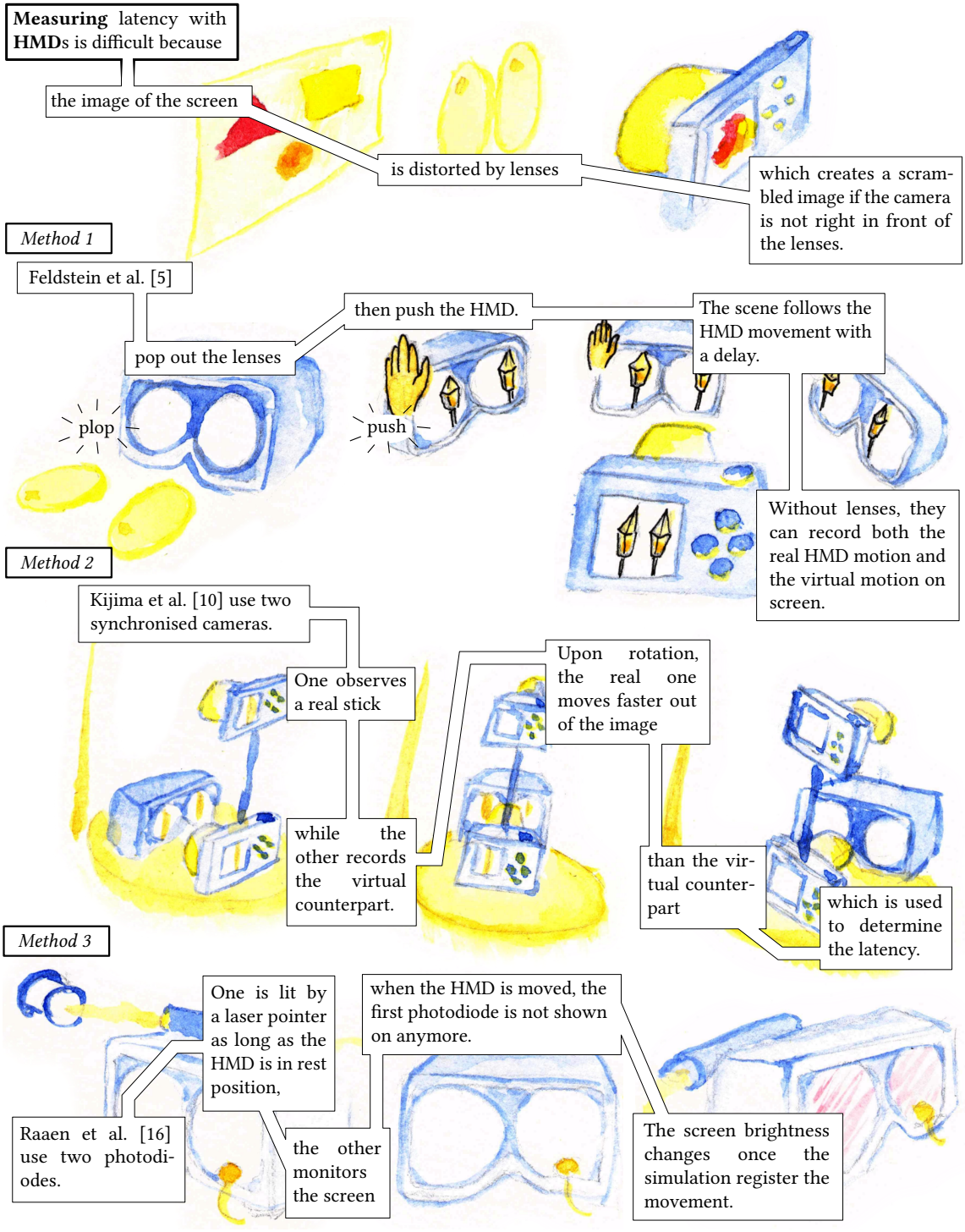
$$latency = \frac{phase}{2\pi * frequency * fps}$$

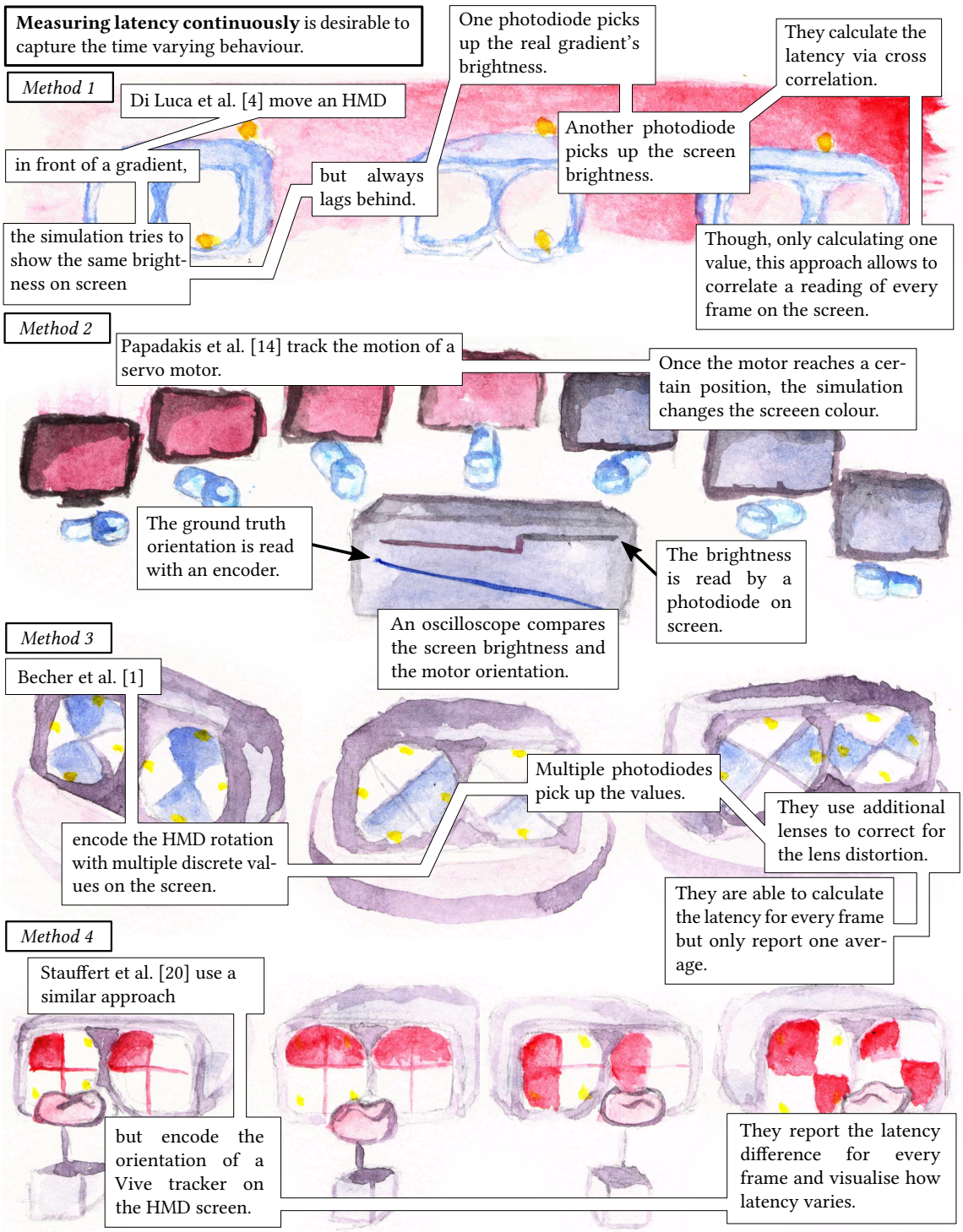
Method 2

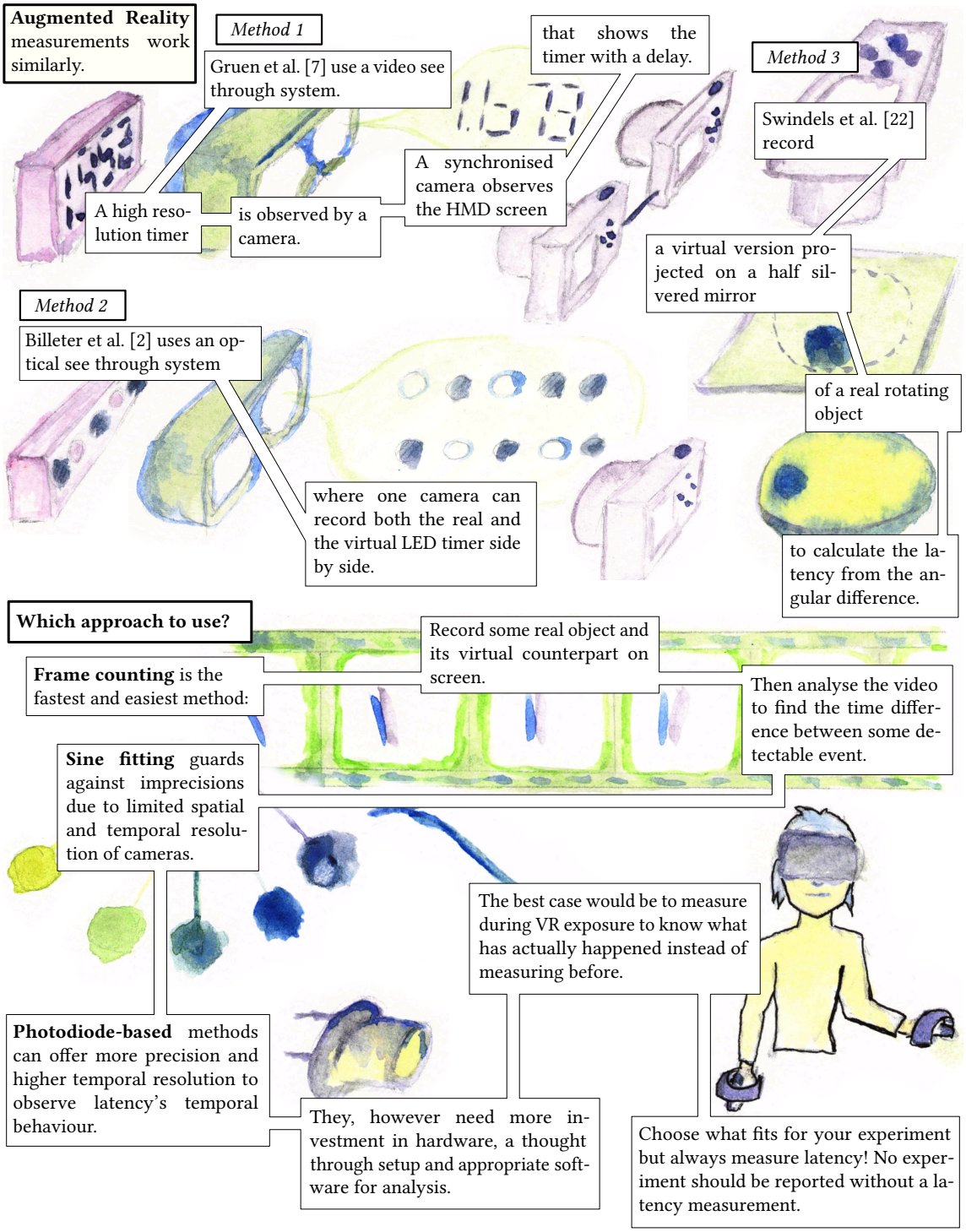
The predictability of a pendulum is used in other approaches to measure latency. Mine et.al. [13] uses a pendulum and two photodiodes:




Other approaches
Method 1
Liang et al. [11] record a pendulum and the timestamp of the last tracking data.
Once the pendulum overlaps the reference
they look how much the tracking data deviated at the recorded timestamp.
reference
Method 2
Chang et al. [3] rotate an HMD.
The virtual scene mirrored on a monitor follows with a delay to the rotation.
Bars drawn on the HMD make it easier to recognise motion.
Method 3
Pape et al. [15] use projection based VR.
The VR system changes the brightness of a projected dot once it receives the motion information.
switch open
The ground truth when the servo motor starts to move is provided with a switch.
switch closed
photodiode
Method 4
Seo et al. [19] use an elaborated platform to simulate real head movements.


Measuring latency with **HMD**s is difficult because
the image of the screen
is distorted by lenses
which creates a scrambled image if the camera is not right in front of the lenses.
Method 1
Feldstein et al. [5]
pop out the lenses
then push the HMD.
The scene follows the HMD movement with a delay.
plop
push
Without lenses, they can record both the real HMD motion and the virtual motion on screen.
Method 2
Kijima et al. [10] use two synchronised cameras.
One observes a real stick
while the other records the virtual counterpart.
Upon rotation, the real one moves faster out of the image
than the virtual counterpart
which is used to determine the latency.
Method 3
Raaen et al. [16] use two photodiodes.
One is lit by a laser pointer as long as the HMD is in rest position,
the other monitors the screen
when the HMD is moved, the first photodiode is not shown on anymore.
The screen brightness changes once the simulation register the movement.


Measuring latency continuously is desirable to capture the time varying behaviour.
Method 1
Di Luca et al. [4] move an HMD
in front of a gradient,
the simulation tries to show the same brightness on screen
but always lags behind.
One photodiode picks up the real gradient's brightness.
Another photodiode picks up the screen brightness.
They calculate the latency via cross correlation.
Though, only calculating one value, this approach allows to correlate a reading of every frame on the screen.
Method 2
Papadakis et al. [14] track the motion of a servo motor.
Once the motor reaches a certain position, the simulation changes the screeen colour.
The ground truth orientation is read with an encoder.
The brightness is read by a photodiode on screen.
An oscilloscope compares the screen brightness and the motor orientation.
Method 3
Becher et al. [1]
encode the HMD rotation with multiple discrete values on the screen.
Multiple photodiodes pick up the values.
They use additional lenses to correct for the lens distortion.
They are able to calculate the latency for every frame but only report one average.
Method 4
Stauffert et al. [20] use a similar approach
but encode the orientation of a Vive tracker on the HMD screen.
They report the latency difference for every frame and visualise how latency varies.


Augmented Reality measurements work similarly.
Method 1
Gruen et al. [7] use a video see through system.
A high resolution timer
is observed by a camera.
A synchronised camera observes the HMD screen
that shows the timer with a delay.
Method 3
Swindels et al. [22] record
a virtual version projected on a half silvered mirror
of a real rotating object
to calculate the latency from the angular difference.
Method 2
Billeter et al. [2] uses an optical see through system
where one camera can record both the real and the virtual LED timer side by side.
Which approach to use?
Record some real object and its virtual counterpart on screen.
Then analyse the video to find the time difference between some detectable event.
Frame counting is the fastest and easiest method:
Sine fitting guards against imprecisions due to limited spatial and temporal resolution of cameras.
The best case would be to measure during VR exposure to know what has actually happened instead of measuring before.
Photodiode-based methods can offer more precision and higher temporal resolution to observe latency's temporal behaviour.
They, however need more investment in hardware, a thought through setup and appropriate software for analysis.
Choose what fits for your experiment but always measure latency! No experiment should be reported without a latency measurement.

REFERENCES

[1] Armin Becher, Jens Angerer, and Thomas Grauschopf. 2018. Novel Approach to Measure Motion-To-Photon and Mouth-To-Ear Latency in Distributed Virtual Reality Systems. *arXiv:1809.06320 [cs]* (Sept. 2018). http://arxiv.org/abs/1809.06320 arXiv: 1809.06320.

[2] Markus Billeter, Gerhard Röthlin, Jan Wezel, Daisuke Iwai, and Anselm Grundhöfer. 2016. A LED-Based IR/RGB End-to-End Latency Measurement Device. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. IEEE, 184–188. https://doi.org/10.1109/ISMAR-Adjunct.2016.0072

[3] Chun-Ming Chang, Cheng-Hsin Hsu, Chih-Fan Hsu, and Kuan-Ta Chen. 2016. Performance Measurements of Virtual Reality Systems: Quantifying the Timing and Positioning Accuracy. In *Proceedings of the 24th ACM international conference on Multimedia*. ACM Press, 655–659. https://doi.org/10.1145/2964284.2967303

[4] Massimiliano Di Luca. 2010. New method to measure end-to-end delay of virtual reality. *Presence: Teleoperators and Virtual Environments* 19, 6 (2010), 569–584. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6797715

[5] Ilja T. Feldstein and Stephen R. Ellis. 2020. A Simple, Video-Based Technique for Measuring Latency in Virtual Reality or Teleoperation. *IEEE Transactions on Visualization and Computer Graphics* (2020). https://doi.org/10.1109/TVCG.2020.2980527 Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[6] Sebastian Friston and Anthony Steed. 2014. Measuring Latency in Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics* 20, 4 (April 2014), 616–625. https://doi.org/10.1109/TVCG.2014.30

[7] Robert Gruen, Eyal Ofek, Anthony Steed, Ran Gal, Mike Sinclair, and Mar Gonzalez-Franco. 2020. Measuring System Visual Latency through Cognitive Latency on Video See-Through AR Devices. (2020), 791–799.

[8] Ding He, Fuhu Liu, Dave Pape, Greg Dawe, and Dan Sandin. 2000. Video-based measurement of system latency. In *International Immersive Projection Technology Workshop*, Vol. 111. Citeseer. http://www.researchgate.net/profile/Dave_Pape/publication/2628137_Video-Based_Measurement_of_System_Latency/links/542166dc0cf203f155c66ad6.pdf

[9] Zenja Ivkovic, Ian Stavness, Carl Gutwin, and Steven Sutcliffe. 2015. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM Press, 135–144. https://doi.org/10.1145/2702123.2702432

[10] Ryugo Kijima and Kento Miyajima. 2016. Measurement of Head Mounted Display's latency in rotation and side effect caused by lag compensation by simultaneous observation — An example result using Oculus Rift DK2. In *2016 IEEE Virtual Reality (VR)*. IEEE, 203–204. https://doi.org/10.1109/VR.2016.7504724 ISSN: 2375-5334.

[11] Jiandong Liang, Chris Shaw, and Mark Green. 1991. On temporal-spatial realism in the virtual reality environment. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*. 19–25.

[12] Dorian Miller and Gary Bishop. 2002. Latency Meter: A device to measure end-to-end latency of VE systems. In *Stereoscopic Displays and Virtual Reality Systems IX*, Vol. 4660. International Society for Optics and Photonics, 7.

[13] Mark Mine. 1993. Characterization of end-to-end delays in head-mounted display systems. *The University of North Carolina at Chapel Hill, TR93-001* (1993). http://www0.cs.ucl.ac.uk/teaching/VE/Papers/93-001.pdf

[14] Giorgos Papadakis, Katerina Mania, and Eftichios Koutroulis. 2011. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry - VRCAI '11*. ACM Press, Hong Kong, China, 581–584. https://doi.org/10.1145/2087756.2087869

[15] Sebastian Pape, Marcel Krüger, Jan Müller, and Torsten W. Kuhlen. 2020. Calibratio: A small, low-cost, fully automated Motion-to-Photon Measurement Device. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. IEEE, 234–237. https://doi.org/10.1109/VRW50115.2020.00050

[16] Kjetil Raaen and Ivar Kjellmo. 2015. Measuring Latency in Virtual Reality Systems. In *Entertainment Computing - ICEC 2015*, Konstantinos Chorianopoulos, Monica Divitini, Jannicke Baalsrud Hauge, Letizia Jaccheri, and Rainer Malaka (Eds.). Vol. 9353. Springer International Publishing, Cham, 457–462. https://doi.org/10.1007/978-3-319-24589-8_40 Series Title: Lecture Notes in Computer Science.

[17] Lisa Rebenitsch and Charles Owen. 2016. Review on cybersickness in applications and visual displays. *Virtual Reality* 20, 2 (June 2016), 101–125. https://doi.org/10.1007/s10055-016-0285-9

[18] David Roberts, Toby Duckworth, Carl Moore, Robin Wolff, and John O'Hare. 2009. Comparing the End to End Latency of an Immersive Collaborative Environment and a Video Conference. In *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. IEEE, IEEE, 89–94. https://doi.org/10.1109/DS-RT.2009.43

[19] Min-Woo Seo, Song-Woo Choi, Sang-Lyn Lee, Eui-Yeol Oh, Jong-Sang Baek, and Suk-Ju Kang. 2017. Photosensor-Based Latency Measurement System for Head-Mounted Displays. *Sensors* 17, 5 (May 2017), 1112. https://doi.org/10.3390/s17051112

[20] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. 2020. Simultaneous Run-Time Measurement of Motion-to-Photon Latency and Latency Jitter. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, IEEE, Atlanta, GA, USA, 636–644. https://doi.org/10.1109/VR46266.2020.1581339481249

[21] Anthony Steed. 2008. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology (VRST '08)*. ACM, New York, NY, USA, 123–129. https://doi.org/10.1145/1450579.1450606

[22] Colin Swindells, John C. Dill, and Kellogg S. Booth. 2000. System lag tests for augmented and virtual environments. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*. ACM, 161–170. http://dl.acm.org/citation.cfm?id=354444

[23] Matthias Walter, Tim Wendisch, and Klaus Bengler. 2018. In the Right Place at the Right Time? A View at Latency and Its Implications for Automotive Augmented Reality Head-Up Displays. In *Congress of the International Ergonomics Association*. Springer, 353–358. https://doi.org/10.1007/978-3-319-96074-6_38

[24] Weixin Wu, Yujie Dong, and Adam Hoover. 2013. Measuring Digital System Latency from Sensing to Actuation at Continuous 1-ms Resolution. *Presence: Teleoperators and Virtual Environments* 22, 1 (Feb. 2013), 20–35. https://doi.org/10.1162/PRES_a_00131

Copyright

Author Contributions

The author conducted the literature research, planned the structure, painted the images and wrote the text.

4.2 Sine Fitting

Sine Fitting as described in the overview paper offers a convenient method to measure motion to photon latency as one average value. It uses the predictable motion of a pendulum to be able to use a camera of limited resolution to measure latency. The resolution may be limited both in spatial and in temporal regards. The approach was described by Steed [Ste08].

The usage may be difficult as the motion needs to be extracted. We published a poster and associated software that intends to make this process easier.

The paper was published as Jan-Philipp Stauffert, Florian Niebling, Jean-Luc Lugrin, and Marc Erich Latoschik. “Guided Sine Fitting for Latency Estimation in Virtual Reality”. In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, pages 706–707

2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)

Guided Sine Fitting for Latency Estimation in Virtual Reality

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Jean-Luc Lugrin‡ University of Würzburg
Marc Erich Latoschik§ University of Würzburg

ABSTRACT

Latency in Virtual Reality (VR) applications can lead to decreased performance and cybersickness. Multiple approaches exist to determine an average latency. Yet many scientific publications fail to report their system's latency despite the potentially detrimental impact. This paper extends Steed's sine fitting approach [13] by using a KCF tracking algorithm [2] to track the positions of physical objects in video recordings of VR systems. We provide a software for convenient usage. Our combination of sine fitting with KCF tracking allows to measure Motion-To-Photon latency of arbitrary tracking devices without any additional preparation or markers. The developed software is open source.

Index Terms: D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Each computation causes latency, i.e., a time delay between input and output. The latency most often described for VR applications is the Motion-To-Photon (MTP) latency that describes the delay between a movement and the corresponding effect shown on the display. Delays during the input-to-output processing not only risk to annoy users but they potentially might induce more severe consequences of visually-induced motion sickness (VIMS) and cybersickness [9]. Hence, a central requirement of VR systems is to measure and finally control a VR system's latency behavior to judge its performance before negative consequences arise. Many researchers fail to report the MTP latency behavior of their applications despite the possible negative consequences. This makes it hard to determine if the findings were due to the experimental setup or a result of poor application performance. There are multiple approaches to measure MTP latency with varying degree of effort to put into.

This paper builds upon Steed's approach to measure MTP latency with sine fitting [13] to determine an estimation of the VR application performance. The original paper records a tracked pendulum and its virtual counterpart to fit their movement with a sine curve. It calculates the time difference between their movements using the sines' phase difference. We propose to use a tracking algorithm - the KCF tracker - to determine the position of the tracked objects without the need of additional preparation or markers. This further simplifies the application of sine fitting. We provide a software based on Qt, OpenCV and the Ceres Solver to lower the barrier of use. The original paper provides a Matlab implementation that requires Matlab knowledge to run.

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:jean-luc.lugrin@uni-wuerzburg.de
§e-mail:marc.latoschik@uni-wuerzburg.de

2 RELATED WORK

Visual delay was found as a major contributing factor already in early simulators [5]. Latency causes cybersickness [3] and decreases performance [8]. Multiple approaches to measure MTP latency exist. He et. al. [7] employ manual frame counting. They record a tracked controller's movement and its virtual counterpart at the same time with a high speed camera. They count the time delay between movement discontinuities to infer the latency. Friston and Steed [6] propose to use image processing to automatically derive the latency in place of manual analysis. Steed [13] replaces the determination of discontinuities in the video with sine fitting. Steed attaches the tracked controller to a pendulum and fits the movement with a sine curve. The use of a continuous signal instead of detecting distinct events reduces the impact of inaccuracies due to limited temporal video resolution. Fitting a sine to the real controller's movement and its virtual image yields the MTP latency in the phase difference. Sine fitting allows to determine the turning point of a pendulum even if the geometric and temporal video resolution are insufficient to determine it directly from the video. The automatic video processing has in common that the tracked objects need to have distinct features like a LED attached. In contrast to the non invasive methods that rely on recording the scene from the outside, Di Luca [4] uses photodiodes attached to the Head-Mounted Display (HMD) in more involved hardware setups. Closer inspection of latency behavior shows that latency changes with time [10] and latency spikes affect user experience [12]. Description of this behavior requires more more in depth application analysis [11]. Putting more effort into the measuring produces better estimates. However, reporting a mean latency provides at least an estimate of the overall performance.

3 SINE FITTING

Sine fitting assumes a tracked object as part of the simulation to move in a sinosoidal pattern. Most often the tracked object is the motion controller the user employs as input modality to the VR application. It is attached to a string to hang from a fixed mount. Once pushed, it performs a sinosoidal movement. There is a display next to it that shows the virtual counterpart of the motion controller. Many VR applications have some representation of the motion controller so only the virtual camera needs to set up to provide a good view of the motion controller. The virtual motion controller follows the movement of the real motion controller so it too elicits a sinosoidal movement pattern. The processing time to do the tracking, simulation and displaying leads to the virtual motion controller having a temporal delay to the real motion controller. A video camera records both the real and virtual motion controller and their movements. A sine curve is fitted to the movements of both objects. The phase difference between both sine curves describes their temporal offset and therefore the MTP latency.

4 IMPLEMENTATION

We created a software based on Qt 5.13.2, OpenCV 4.1.1 and the Ceres solver 1.14 [1] for automatic calculation of MTP latency in VR systems, by comparing the sinosoidal movements of physical objects and their virtual counterparts in video recordings of these systems. Figure 2 shows a screenshot.

The usage is a follows: The video to be analyzed is loaded from disk. OpenCV tries to detect the video's frame rate given as frames

978-1-7281-6532-5/20/$31.00 ©2020 IEEE
DOI 10.1109/VRW50115.2020.00204

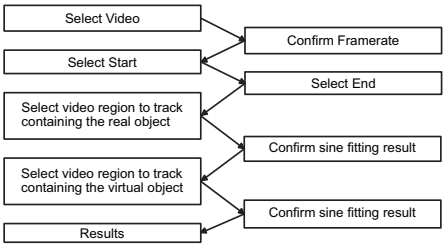


Figure 1: The application flow describing the required user interaction.

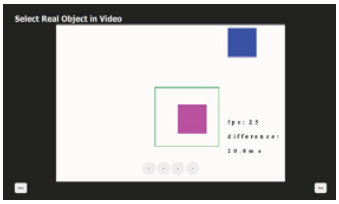


Figure 2: Application screenshot showing the selection of the real object in our synthetic test video.

per second (fps). The user can change the fps used for the calculation to account for inaccurate detection. The start and end time of the video is selected. Videos often show the experimenter's hands in the beginning and end to operate the setup which needs to get cut away. The user then selects a rectangular region in the video that shows the real object. A larger region results in a longer computation time for the tracking algorithm but provides better results. The user is asked to change the selection if the tracker looses the tracked object before all video frames are processed. A principle component analysis (PCA) converts the resulting 2d points to 1d values. The Ceres solver in turn fits a sine wave described by $e^{-dt} \cdot \cos(2\pi ft + \varphi)$ to the data using gradient descend. The decay term e^{-dt} serves as adjustment if the pendulum slows down over time. The frequency f is estimated beforehand by taking the largest frequency in the DFT (discrete fourier transform) transformed data and refined by Ceres. The fitted curve is overlayed over the tracked positions for visual inspection. Some times, the fit shows visible artifacts which necessitate choosing a different region to track and a repeat of the curve fitting. Once the real object's movement is processed, the user selects its virtual counterpart which undergoes the same procedure. The latency between the real object's movement and its virtual counterpart's movement results from the phase difference $\hat{\varphi}$ of both fitted sine waves as latency $= \frac{\hat{\varphi}}{2\pi f \cdot \text{fps}}$. Figure 1 shows a step by step overview.

5 Validation

First validations both with synthetic videos where two rectangles oscillated with a known time difference and two videos of Friston and Steed's mechanical latency simulator [6] that are available online show average estimation errors of 2.5 ms. This is similar to errors described for other analyses of the online videos.

6 Discussion

Results may vary depending on the video and tracking quality. We advice to repeat the analysis multiple times to not fall prey to outliers due to an inopportune choice of the region to track.

Sine fitting necessitates that the movement of a real object and its virtual counterpart are observable at the same time. Our approach is directly applicable to screen-based or projection-based VR setups, where target objects and their graphical counterpart are simultaneously visible. However, typical HMDs require an additional step. First, our approach has to be applied using a proxy display, e.g., a standard monitor. Then we have to calculate the latency offset between the target HMD and the proxy display, e.g., using a high speed camera that records both the monitor and the HMD screen. A solid color is displayed on both, then changed to another color. The camera detects the time difference between the second color shown on the respective screen. The time difference allows to derive the MTP latency from the real object to an HMD's screen. This approach needs the possession of a high speed camera while the sine fitting approach itself works with lower frame rates. The reason is a tradeoff in camera abilities between geometric and temporal resolution. Color changes need only little geometric resolution but high temporal resolution to determine time differences between two regions in a video. Capturing movement of a real object requires high geometric resolution which makes high framerates costly or impossible.

7 Conclusion

We suggest an improvement in usability of the sine fitting latency estimation for VR applications by using a KCF tracker. We provide the implementation as a downloadable Windows binary and provide full source access (`https://go.uniwue.de/autosine`).

References

[1] S. Agarwal, K. Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

[2] M. Danelljan, F. S. Khan, M. Felsberg, and J. v. d. Weijer. Adaptive color attributes for real-time visual tracking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1090–1097. IEEE. doi: 10.1109/CVPR.2014.143

[3] S. Davis, K. Nesbitt, and E. Nalivaiko. A systematic review of cybersickness. pp. 1–9. ACM Press. doi: 10.1145/2677758.2677780

[4] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence: Teleoperators and Virtual Environments*, 19(6):569–584, 2010. doi: 10.1162/pres_a_00023

[5] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[6] S. Friston and A. Steed. Measuring latency in virtual environments. 20(4):616–625. doi: 10.1109/TVCG.2014.30

[7] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin. Video-based measurement of system latency. In *International Immersive Projection Technology Workshop*, p. 111, 2000.

[8] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pp. 135–144. ACM Press, 2015. doi: 10.1145/2702123.2702432

[9] L. Rebenitsch and C. Owen. Review on cybersickness in applications and visual displays. *Virtual Reality*, 20(2):101–125, 2016.

[10] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems. In *Virtual Reality (VR), 2016 IEEE*, pp. 287–288. IEEE, 2016. doi: 10.1109/VR.2016.7504766

[11] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behaviour of virtual reality systems. In *Proceeding of the 22nd ACM Symposium on Virtual Reality Software and Technology (VRST)*, 2016.

[12] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Effects of latency jitter on simulator sickness in a search task. In *Proceedings of the 25th IEEE Virtual Reality (VR) conference*, 2018.

[13] A. Steed. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pp. 123–129. ACM. doi: 10.1145/1450579.1450606

Copyright

Author Contributions

The author wrote the software and poster.

4.3 Detailed Latency Measurements

Most approaches have the problem of only reporting one latency value though we showed that latency varies over time. There is sine fitting ([Ste08]) and cross correlation ([DL10]) that observe the time difference of a real object and its virtual counterpart over many samples and then calculate one value that fits the observed deviation the closest. Event based measurement detects an event in the motion of a tracked object and then relate this to an event in virtual reality ([Min93]). Few approaches can measure the end to end latency for every frame of the employed screen. The continuous measurement, however, is necessary to get more insight into the time variant behaviour of latency.

We created a measurement aparatus that measures latency for every frame displayed on an HMD screen. Our approach can be developed further to allow latency measurements during an experiment. The prototype can not yet fulfil this promise but the paper describes the next steps.

The paper was published as Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Simultaneous Run-Time Measurement of Motion-to-Photon Latency and Latency Jitter". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2020, pages 636–644

2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)

Simultaneous Run-Time Measurement of Motion-to-Photon Latency and Latency Jitter

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

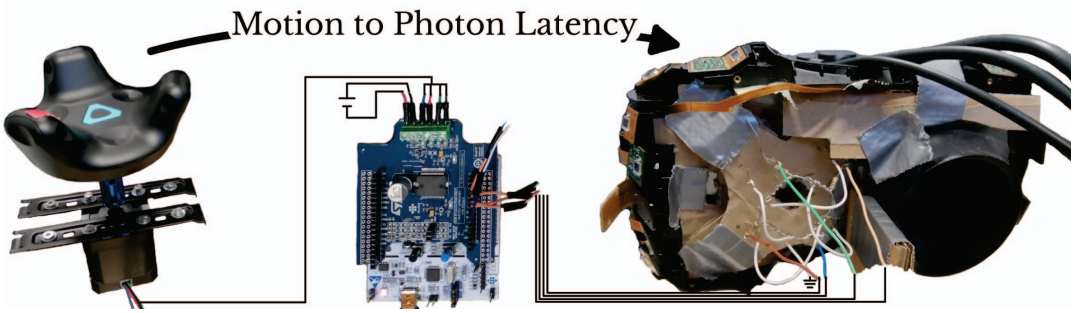


Figure 1: Illustration of the apparatus developed to measure the Motion-to-Photon latency. A microcontroller (middle) reads the difference between the rotation of a tracked controller as moved by a motor (left) and the reported rotation on the Vive display (right). The illustration shows the experimental prototype which disassembled the Head-Mounted Display for easier access to the internals, i.e., the lenses and displays.

ABSTRACT

Latency in Virtual Reality (VR) applications can have numerous detrimental effects, e.g., a hampered user experience, a reduced user performance, or the occurrence of cybersickness. In VR environments, latency usually is measured as Motion-to-Photon (MTP) latency and reported as a mean value. This mean is taken during some specific intervals of sample runs with the target system, often detached in significant aspects from the final target scenario, to provide the necessary boundary conditions for the measurements. Additionally, the reported mean value is agnostic to dynamic and spiking latency behavior. This paper introduces an apparatus that is capable of determining per-frame MTP latency to capture dynamic MTP latency and latency jitter in addition to the commonly reported mean values of latency. The approach is evaluated by measuring MTP latency of a VR simulation based on the Unreal engine and the HTC Vive as a typical consumer-grade Head-Mounted Display (HMD). In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time. We evaluate the accuracy of our apparatus by injecting a controlled artificial latency in a VR simulation. We show that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to dropped frames and an overall degraded system performance. The presented system can be used to monitor latency and latency jitter as critical simulation characteristics necessary to report and control to avoid unwanted effects and detrimental system performance.

Index Terms: D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

1 INTRODUCTION

Each instruction of a computer system has an associated execution time. As a result, any output that is calculated after changing user input will always be delayed, introducing latency into the human-computer interaction loop. The impact this delay causes on the system qualities of usability and user experience depends on the interactivity and potential real-time requirements of the human-computer interface and the employed interaction metaphor [6]. Even users of a spreadsheet software in a 2D graphical user interface (GUI) expect a timely response to a mouse click. The response should feel to be instantaneous. The conditions to experience feedback as instantaneous depend on several factors of the interaction metaphor.

Direct interaction metaphors are more sensitive to delays than indirect ones. Here, VR systems are specifically sensitive to delays between input and output processing since they directly and continuously couple input, including head movements, to the visual output. The overall latency in VR between an action, e.g, moving an input controller measuring hand or head movements, and its corresponding effect shown on a screen is denoted as Motion-to-Photon (MTP) latency. Unwanted delays during the input-to-output processing not only risk to annoy users but they potentially might induce more severe consequences of visually-induced motion sickness (VIMS) and cybersickness [25]. Hence, a central requirement of VR systems is to measure and finally control a VR system's latency behavior to judge its performance before negative consequences arise.

Deducing the latency from code inspection and counting the execution times of all instructions of a VR application is largely out of reach today. The interplay of all soft- and hardware sub-systems of modern general-purpose computer systems introduce a dynamic complexity that can only be observed as a black box [28]. Low-level system interrupts of the system's motherboard, modern CPU speed-up approaches including parallelization, caching, and branch prediction, as well as high-level operating system and application-layer aspects of concurrency, multi-threading and scheduling, and

2642-5254/20/$31.00 ©2020 IEEE
DOI 10.1109/VR46266.2020.00086

Table 1: Comparison of previous approaches to latency measurement. Camera based approaches are less intrusive but can't capture an object and an HMDs screen with the exception of augmented reality headsets [23]. Camera based approaches are not viable if the user wears an HMD as the view is obstructed. Photodiodes are more intrusive but provide high temporal resolution. Approaches that use photodiodes attached to the screen can allow to use a majority of the screen to display a VR experience. Only a small screen area needs to be reserved for the information that gets picked up by the sensor. Other approaches that use photodiodes have either only tested with a monitor screen instead of an HMD or require the HMD to follow specific movement patterns. Our approach is the only one that uses photodiodes with an HMD without the need of a predetermined HMD movement pattern. This potentially allows to measure latency while a user consumes a VR application.

Author	Method	Capture	Latency	Use HMD	HMD wearable	Measurement frequency	Temporal Resolution
Becher et al. [2]	Continuous	Photodiode	Mean, SD, Min, Max	yes	no	11 ms	Frame
Di Luca et al. [7]	Sine Fitting	Photodiode	Mean, SD	yes	no	20 Hz	Once
Friston et al. [11]	Event	Camera	Mean, SD, Min, Max	yes	no	Acceleration peak	Movement dependent
He et al. [14]	Event	Camera	Mean	no	no	Grid line crossed	Movement dependent
Kämäräinen et al. [16]	Event	Photodiode	Mean, SD	no	possible	16 ms - 100 ms	Once
Liang et al. [19]	Continuous	Camera	Mean	no	no	20 Hz	Frame
Mine [22]	Event	Photodiode	Mean	no	possible	Pendulum zero position	Movement dependent
Papadakis et al. [23]	Continuous	Photodiode	Mean, SD	no	possible	0.1 ms	0.1 ms
Sielhorst et al. [26]	Continuous	Camera	Distribution	AR	no	< 1 ms	< 1 ms
Steed [32]	Sine Fitting	Camera	Mean	no	no	25 Hz	Once
Wu et al. [35]	Continuous	Camera	Distribution	no	no	2 ms	2 ms
Zhao et al. [36]	Sine Fitting	Photodiode	Mean, SD	yes	no	11 ms	Once
Our approach	Continuous	Photodiode	Mean, SD, Distribution	yes	yes	11 ms	Frame

I/O communication (including networking) and the interplay of all these aspects induce a potential fluctuation in execution time. After all, real-time capabilities currently are not a central requirement for the typical consumer-grade general purpose computer systems. As a result, the overall MTP latency of VR systems is often reported as a mean value – if it is reported at all. However, each of the multiple hardware and software parts either contribute their own variable latency during run-time or the interplay of all parts creates a dynamic pattern of latencies not well represented by a mean value derived from sample runs.

Ideally, we would either be able to control and assure run-time behavior as provided by real-time systems, or to continuously monitor latency and latency jitter during execution of a VR system. For example, spikes in latency may influence and invalidate results during VR experiments since they induce a potential confound. The ability to detect latency spikes during experiments allows to sort out trials when the experimental condition is overshadowed by technical inconsistencies. Overall, monitoring latency in VR systems is a critical quality assurance measurement to optimize run-time behavior and to assess and guarantee good usability and user experience of VR systems.

Contribution

This paper introduces an apparatus that is capable of determining per-frame MTP latency to capture dynamic MTP latency and latency jitter in addition to the commonly reported mean values of latency. The approach is evaluated by measuring MTP latency of a VR simulation based on the Unreal engine and the HTC Vive as a typical consumer-grade Head-Mounted Display (HMD). We inject artificial latency in a VR simulation and show that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to an overall degraded system performance. In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time.

2 Related Work

Visual delay was found as a major contributing factor already in early simulators [10]. Time invariant latency causes cybersickness [5], decreases performance [15] and reduces presence [21]. Reoccuring latency spikes also have negative effects on performance [24, 33] and cybersickness [31]. Interruptions in VR can cause a break in presence [27].

Users are able to distinguish changes in latency for hand [9] as well as for head [8] movements that are faster than 33 ms. Building on this work, Mania et. al. test sensitivity to head tracking latency in virtual environments [20] where they show that differences of 15 ms are still distinguishable. The currently common VR displays running on 90 Hz would exceed this detectable threshold if tracking information is even one frame delayed.

The performance of VR applications is usually assessed by measuring MTP latency which tracks the time between an input on a certain input channel and the time it takes to show its effect on a display. He et. al. [14] employ manual frame counting. They record a tracked controller's movement and its virtual counterpart at the same time with a high-speed camera. They count the time delay between movement discontinuities to infer the latency. Steed [32] replaces the error prone determination of discontinuities in the video with sine fitting. Steed attaches the tracked controller to a pendulum and fits the movement with a sine curve. The use of a continuous signal instead of detecting distinct events reduces the impact of inaccuracies due to limited temporal video resolution. Fitting a sine to the real controller's movement and its virtual image yields the MTP latency in the phase difference. The still manual video analysis is replaced by a direct deduction of the sine's gradient with photodiodes by Di Luca [7] or image processing in Friston and Steed [11]. Papadakis et al. [23] correlate a continuous movement of a tracked object with a photodiode reading which is attached to a monitor screen. An oscilloscope shows measurements of the system's latency. Becher et al. [2] use multiple photodiodes on the HMD screen to pick up

the brightness encoded HMD orientation. A motor rotates the HMD to provide the base truth. The latency is derived from the difference of the orientation in reality and the orientation reported on screen. Kämäräinen et al. [16] use a photodiode and a simulated touch event to measure latency in a remote rendering application, inducing further latency through additional non-local network communication. All approaches report one mean latency value and if the approach allows it a standard deviation.

Latency, however, changes with time and shows repeated spikes [7, 28]. This is a result of the complexity of VR systems that often consist of multiple software components to handle various input and output modalities that run in parallel or on distributed machines [1, 18, 29]. Regarding latency as time invariant allows to only measure it once and claim that it will be the same at a later time. Sielhorst et al. [26] describe the latency behavior of an augmented reality system with the distribution of measured latencies. Their measurements exhibit infrequent outliers. Wu et al. [35] propose a camera-based approach to measure latency at a 1 ms resolution again showing the time variability of latency. If latency is to be assumed to change during runtime of a system, it is neccessary to measure latency spikes during user studies that might be influenced by this jitter.

Latency measurement either focuses on the latency between certain events, or use sine fitting to correlate a known movement to the measurement data. If the measurement is precise enough and can be repeated often, a continuous approach is taken. The capturing of latency measurements is either done with a camera that records both the tracked object and the result on a screen, or with photodiodes attached to a screen that are correlated with a known movement. Camera based measuring is less invasive but is hard to use with HMDs, as their screen is difficult to capture. Photodiodes are more invasive and need additional hardware to support their usage, but can potentially report measurements during runtime of a system, and not only in post hoc analysis. A comparison of the reported approaches is presented in Table 1.

Research shows that latency changes over time and users of VR applications can detect small changes in latency, but measuring approaches are restricted to report mean values. We introduce a setup that allows to measure MTP latency for every frame, that can be extended to work during VR experiments. We use this to describe a VR system's latency under different conditions.

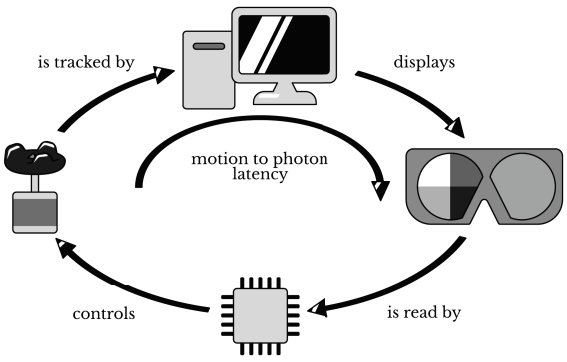


Figure 2: End-to-end latency in the setup: The top part shows MTP latency from a tracked controller to its representation on the HMD screen. The bottom shows the microcontroller comparing controller and display to calculate the MTP latency.

3 SETUP

We measure the time between a known real-world rotation of a tracked controller, and the effect of said rotation on a HMD screen. A motor rotates a tracked controller with a known speed. The tracker sends the position and orientation of the controller to a computer. The computer then calculates the motor angle from the orientation. This detected angle is rendered to a VR HMD screen, encoded into rectangles with certain brightness levels. Each processing and communication adds to the final latency both by our own software and by the hard and software we are using. A microcontroller is employed to drive the motor and to read the HMD screen using photodiodes. It calculates the difference between the known motor angle and the angle reported on the screen. Knowing the speed of the motor, introduced latency between movement of the controller to display of the correct orientation on the HMD screen can be deduced. An overview is shown in Figure 2.

We use an HTC Vive tracker, first and second generation, which is mounted on a NEMA17-01 2 phase hybrid stepper motor. The tracker orientates itself with an IMU and the Lighthouse tracking system at 120 Hz [17]. The tracking data is sent to the connected computer. A VR application based on the Unreal Engine 4.22.3 receives the data through the SteamVR/OpenVR connection. The microcontroller is a NUCLEO L152RE developer board with ARM Cortex M3 processor, attached to a motor driver shield X-NUCLEO IHM01A1. Four OSRAM BPW 21 photodiodes are attached to the microcontroller and HMD to read back the orientation of the tracked controller encoded on the HMD screen.

The Unreal Engine collects the most recent tracker orientation every frame and provides it for the subsequent user logic. We receive the orientation in euler angle form and convert it to a quaternion to extract the motor angle. This extraction needs a calibration before the experiment. The tracker rotates multiple times around its axis. Due to the intermittent Euler angle representation, the orientations form a line in quaternion space. The quaternion at the beginning of the line q_0 is used to normalize all other rotations. It is characterized by having a negative distance to its predecessor, with the distance being calculated using the dot product between itself and its predecessor. The calculation is done multiple times to account for sampling errors. All potential quaternions for q_0 are collected in a set S.

$$S = \{q_t | \langle q_t, q_{t-1} \rangle < 0\}$$

where q_t is the tracker's orientation at time t. The quaternion q_0 is the element of S with minimal w component.

$$q_0 = \arg\min_{p \in S} p(w)$$

A multiplication of an orientation q_t of the tracked controller with the inverse of q_0 removes the base orientation, i.e. the slope the tracker stands on. This assumes that the motor and attachement provide a rotation without nutation. The normalized quaternion is denoted q_n.

$$q_n, t = q_t \cdot q_0^{-1}$$

The motor/tracker angle a_t is then computed by taking the inverse cosine of the w component.

$$\text{angle}_t = cos^{-1}(q_{n,t}(w))$$

The stepper motor moving the tracked controller rotates at 0.9° per step. Microstepping increases the motor vibrations that influence the IMU part of the tracking significantly, and was therefore not used. The motor is attached to a small slope to prevent gimbal lock. The calculated controller angle is converted to its stepper motor step equivalent and encoded on the HMD, by rendering a brightness pattern to a specific area of the HMD screen. Four photodiodes attach to the display area to read back the brightness values rendered

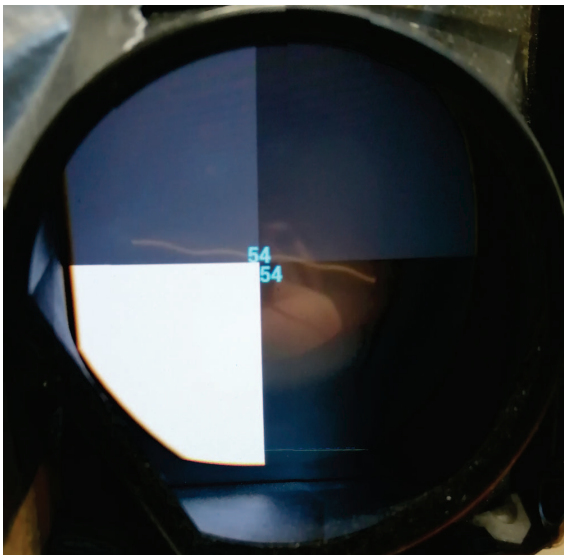


Figure 3: The Vive display with a number encoded as a brightness pattern. The lens is taken out, and the rectangles encoding the number are enlarged for a better view. The number displayed in the middle is for debugging only, as the photodiodes are attached at the border of the screen.

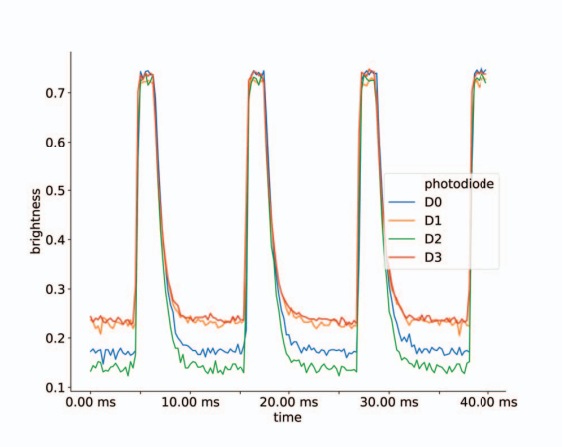


Figure 4: Brightness readings of the Vive display with four different brightness values shown as read by four photodiodes. The Vive display is black most of the time with a short burst of brightness to show the image. The maximum brightness represents the color shown on screen.

on the screen. The photodiodes are able to distinguish four different values consistently, leading to a total number of 256 (4^4) different values that can be encoded. The 400 possible steps per single rotation of the motor exceed the 256 distinguishable values and are therefore encoded modulo 200. Figure 3 shows an example number encoded on the display.

The Vive display is dark most of the time, with a light wave occuring every frame. The microprocessor reads the attached photodiodes at a frequency of 4 kHz, delivering approximately 44 brightness samples per frame for each photodiode, of which 16-17 are in the bright region of each frame. The bright region consists of a plateau of 6-7 samples. The remaining 10-11 samples describe the rising and diminishing brightness. Figure 4 shows an exemplar reading of the brightness levels of the various photodiodes during several frames of the HMD. The photodiodes have different dark levels due to variation in their attachment. The brightness reading rises once an image is shown. The absolute brightness units carry no meaning here, as the measurements are used for relative comparison between the different brightness levels, and are normalized in the figure. The computer performs a calibration with the microcontroller over its serial interface prior to the measurements, to ensure accurate and repeatable detection. The calibration phase starts to display a black HMD screen for the microcontroller to pick up which sensor values equal black. To be robust against small variations, the black threshold is set to be at 1.15 times the maximum measured black value. All readings above this threshold are then counted as belonging to a brightness level encoding a specific grey level of the display. The computer then presents each possible brightness combination to the microcontroller, while communicating which actual grey level is set over the serial interface. The microcontroller deduces for each sensor which measured brightness interval represents the respective number. The figure shows the four different brightness values used in our system, presented to one of the photodiodes each. Relatively large gaps between amplitudes of the different curves indicate that more intensity levels could be distinguished by the photodiodes. This optimization was not followed up upon, to keep the measurements as reliable as possible. During measurement, the microcontroller saves the two highest brightness values for each phase, while the brightness reading is above the black threshold for each respective diode. The encoded number is deduced at the fourth reading below the black threshold by averaging the two highest values and comparing the result to the brightness intervals calibrated during system startup. We discard a frame's reading if at least one photodiode doesn't detect the light flank end within 2 ms of the other diodes to guard against erroneous readings.

The microcontroller drives the stepper motor with two revolutions per second. Using a motor with 400 steps per revolution, one step of the motor is occuring every $\frac{1}{800}$s $= 1.25$ms. This value that depends on the employed stepper motor is a limiting factor determining the maximum possible accuracy of the measurements in the following experiments.

4 Experimental Measurements

We use our setup to observe a computer systems's baseline MTP latency behavior and MTP latency behavior under load. We validate the system setup with an experiment: Known artificial latency jitter is introduced at the application stage of our VR system, to determine if the additional latency is visible in the final readings compared to an established baseline containing no additional latency.

4.1 Experiment 1: Baseline

The baseline measurement describes the MTP latency behavior of our system. Common VR applications are expected to show a similar latency behavior.

4.2 Experiment 2: Artificial Latency Jitter

We introduce artificial latency jitter to see if manipulation at the application stage is visible in the final measurement. The introduced jitter discards position and orientation updates of the tracker every 16th frame for the duration of two frames. The implementation follows Stauffert et al. [31] but exchanges the probability distributions with fixed numbers to achieve the desired effect.

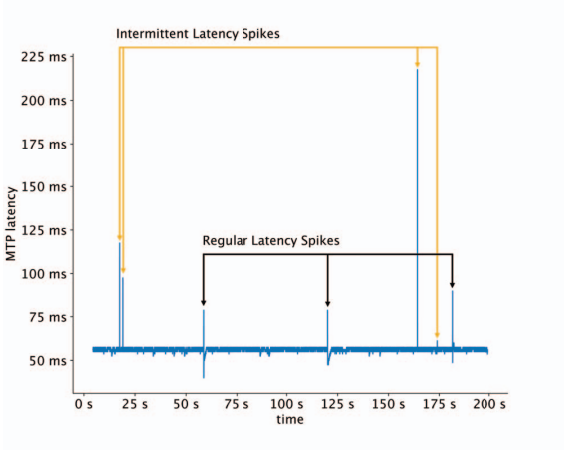


Figure 5: Difference in between the real tracker rotation and the reported tracker rotation on the HMD screen reported as delay in ms as deduced by the rotation speed of the tracker. The difference was measured for every frame of the HMD. Most latencies are close to the mean value with few latency spikes. There are regular reoccuring spikes and irregular spikes.

4.3 Experiment 3: Load Analysis

We stress the computer running the application "HeavyLoad" [13] that creates load on both the main processor and the graphics card. This leads to a CPU load of 100% on all cores, and a GPU load close to 100% causing other applications running concurrently to suffer.

We assume to encounter a fixed angular offset between the motor rotation and the motor rotation reported on screen, as long as the computer has sufficient free resources. Stressing the computer here with a background benchmark or in general with a demanding simulation should evoke similar spikes in angular difference as in the previous experiment, where latency was introduced in a predetermined period. The expected difference is a varying latency spike duration with varying occurrence.

5 Results

5.1 Baseline Measurements

Figure 5 shows parts of a test run. We discard the first four seconds of the motor accelerating. Most of the latency measurements gather around one value differing only by one stepper motor step difference equaling 1.25 ms. Repeated outliers to both sides represent disturbances in the tracking or processing resulting in latency jitter. We do not have any insight where the latency jitter originates. Examples of sources can be the tracking, different components like scheduling and background tasks, application stage like specifics in the employed engine, the display, or any stage in our setup.

The mean latency in the baseline test run was 56.14 ms with a standard deviation of 1.6 ms. There is no difference between using a Vive tracker of the first generation compared to the second generation version. The plots here show a test run with a tracker of the second generation.

We measure repeated latency spikes followed by a period of lower latency that degrades to the mean after few samples. Figure 6 shows a segment with two such patterns. These patterns occur approximately every 61.4 seconds. The measurement in Figure 5 features three occurrences with in between times of 61.443 ms and 61.432 ms. There are irregular outliers besides this periodic pattern.

5.2 Artificial Latency Jitter

Introducing artificial latency jitter every 16th frame for two frames shows the angular difference to increase in the first not updated frame and to increase further in the second not updated frame as was expected (cf. Figure 7). The smaller time differences visible in the graph are quantised to the measurement accuracy of 1.25 ms. The mean latency in the latency jitter test run was 58.20 ms with a standard deviation of 6.22 ms. The comparison plot in Figure 9 shows two bands representing the two introduced delays.

5.3 System Load

Measuring latency with the computer brought to its capacity shows many and irregular latency spikes. Compare Figure 8 for a visualisation. The load test had a mean of 54.51 ms with a standard deviation of 8.62 ms. In addition to the many outliers with increased latency, there are some latencies below the MTP latency mean.

5.4 Comparison

The histogram of latency measurements in Figure 11 shows the majority of the samples gathering around 55 ms latency as does the scatterplot of Figure 9. Frequent outliers are delayed only by a small amount of time, while a decreasing number of outliers exhibit higher latency. Note that the y scale of the figure is logarithmic to better show the distribution. The figure shows the result of a stacked-z test following Stauffert et al. [30]. It assumes that latencies are normally distributed and applies a z-test to detect outliers. The test is then recursively applied to the outliers again. A first outlier category to start at larger latencies indicate a bigger variation of the underlying distribution.

The baseline measurement describes a narrow normal distribution with outliers close to the mean. The outliers themselves have a small variation and therefore create multiple outlier groups that are each close to a mean. The samples around the mean contribute 88.1% of all the samples with the first outlier group containing 11.6%. The remaining 0.3%. The artificial latency condition shows the two introduced peaks in latency. Its distribution has a larger variation. The z-test separates the main values (76.1%) from the introduced latency jitter (23.9%). The load condition has a large variation which comprises 95.4% of the samples in the main part with 4.5% in the first outlier category. All conditions show rare extreme outliers.

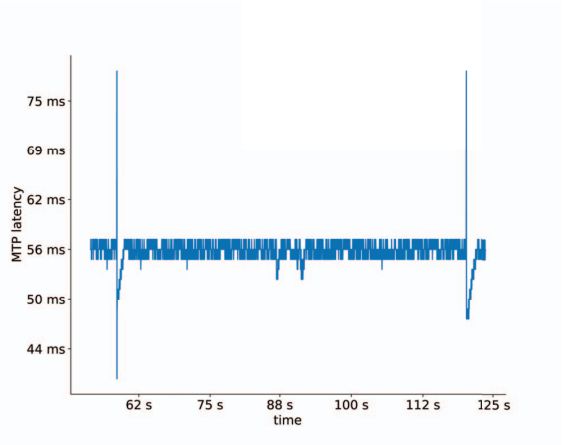


Figure 6: Detail of Figure 5 to show the repeated occuring pattern of latency spikes. We observe a regular pattern with a big spike in latency followed by lower latency that converges back to the mean latency.

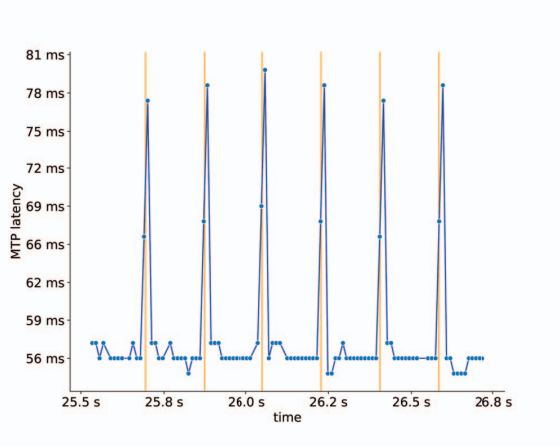


Figure 7: Measurements with artificial latency spikes: A spike, delaying tracking information for two frames, was introduced every 16th frame. The latency measurement returns the introduced pattern of one frame with an increased latency by 11 ms and the subsequent frame with an increased latency by 22 ms. The orange vertical lines mark the first delayed sample.

The quantile-quantile plot in Figure 10 shows how the distributions compare to one another. If a distribution is the same, its quantiles lie on a line as seen in the diagonal. All distributions have one or two big outliers at the 100% quantile. The lower quantiles are similar. The comparison between the baseline and the artificial latency jitter condition shows a similar distribution until the latency timing of the first spike. The second spike shifts the q-q plot further from the diagonal. The comparison between the artificial latency spike and the load condition show more percentage of the samples for the latency spike condition in the latencies of the provoked spikes and then more samples above those latencies in the load condition.

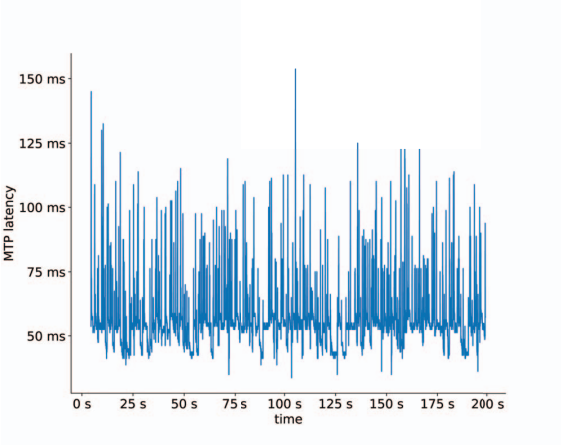


Figure 8: Measurements with artificial load: The processor and graphics card were stressed by a background software. Multiple irregular latency spikes result. The majority of samples still gather near the mean. Interesting is the increase in samples with latencies lower than the mean.

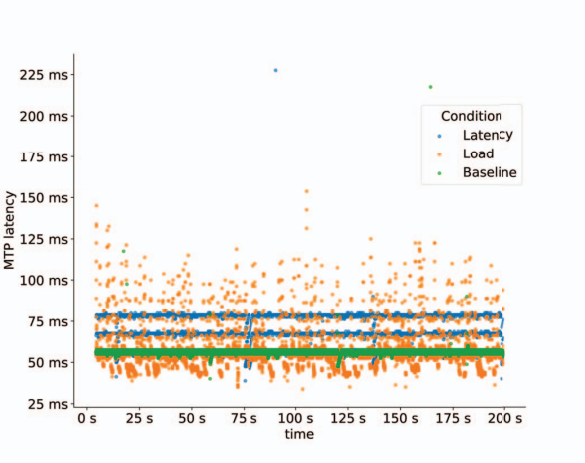


Figure 9: Scatterplot comparing the measurement runs with different conditions. The baseline run samples form a narrow band. The artificial latency jitter run samples show two additional bands that equal the two injected delays. The load condition shows multiple outliers. All conditions show some outliers.

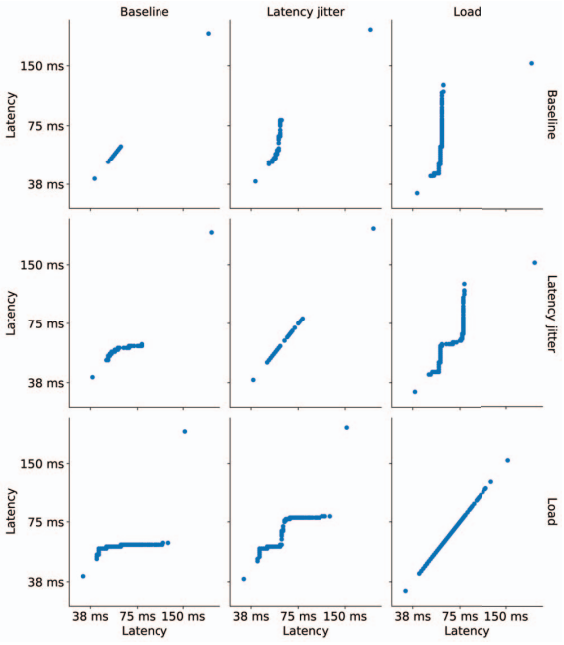


Figure 10: Quantile-quatile plot to compare the distribution of the different measurement runs. The distributions are similar, i.e., close to the diagonal, in their lower quantiles. The higher quantiles show the two introduced latency spikes in the latency jitter condition. The load condition has more samples in the higher quantiles. The few extreme outliers are clearly visible and independent of the condition.

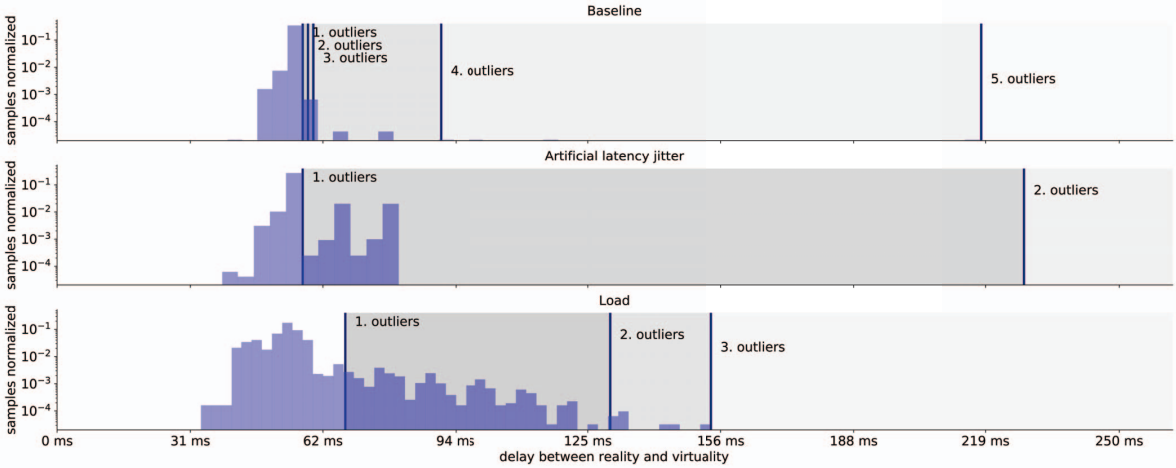


Figure 11: Illustration of the stacked z-test of the latency measurements for the three load conditions. Top: the baseline condition without any additional load. Middle: system behavior with injected latency jitter. Bottom: system behavior under load stress. Without any load, the majority of the latency measurements gather around a mean as a central tendency. The different latency distribution patterns depending on the load condition slowly move this mean to the right since the variance and hence the jitter increases. This mean does not accurately reflect the latency-related system behavior. The stacked-z test shows outliers of the assumed normal distribution and therefore encodes the variance as well. Notably, our approach detects that the no-load baseline condition is already affected by two groups of outliers which would result in repeated reoccurring lost frames not well represented by one mean value. Also, the injected latency (middle) is well detected and reported by our apparatus. The heavy-load condition at the bottom reports a system behavior which can be assumed to generate a severe negative impact on usability and user experience.

6 Discussion

The evaluation of our approach validates that the injected artificial but controlled latency is properly reflected in the measured data. We find that a normal test run carries both distinct patterns as shown in Figure 6 as well as less regular outliers. The stacked-z test works well to separate outliers. It separates the regular outlier pattern in the baseline run, separates the introduced latency spikes in the artificial latency jitter run and splits the majority of outliers from the rarer more extreme outliers in the load condition. A q-q plot proves to be a suitable tool to compare different behaviors. It allows to test the system with a baseline measurement and then compare what changes in more taxing situations.

A computation between motion and photon or an interaction of multiple parts takes longer and therefore misses one screen refresh to create the distinct pattern in Figure 6. This computation occurs in regular intervals. As the rotational difference between the real tracker and the reported rotation is below the mean difference after such a spike, we assume that the tracker recalculates its rotation based on the external reference system, and the samples afterwards that approach the mean value again are the increasing drift of the sensors. The nature of MTP latency entails that this can only be speculation. Other parts in the pipeline like drivers receiving the signal, the VR application, other applications in the background or the display could be sources of this emerging pattern which can only be analyzed further by separately analyzing the different subsystems.

Our prototype setup uses photodiodes directly attached to one display of the Vive HMD. Mounting the photodiodes on the lens would allow easier baseline measurements of VR hardware. However, tests revealed that it produced less distinct patterns. It would also have an impact on the usability and would potentially induce measurement artifacts since then the photodiodes and cabling would be visible by the users, hence we did not follow this approach but placed the photodiodes directly on the display. Today's HMDs usually don't utilize all pixels of the displays. The circular lenses only cover parts of the rectangular displays, leaving unused pixels around the display edges, prominently in the corners of the rectangular screens as can be seen in Figure 3. The photo illustrates this partial coverage of the screen clearly in the bottom left corner. It shows how the circularly shaped rendered image leaves the corner of the rectangular display dark. A similar partial coverage is detected for the other three corners not seen so prominently in Figure 3. However, physical access to these unused areas of the screens requires a more invasive approach to the hardware of the HMD. For example, with our currently used HMD the photodiodes would require to get attached via drilled holes on the side of the HMD chassis enclosing the view cones. In addition, this placement requires rendering to the uncovered areas. We currently refine our prototype and investigate other HMD types to support these features.

We evaluated the feasability of our approach with the setup shown in Figure 1. The Vive HMD is disassembled to get access to the display. Photodiodes are then attached to measure brightness in distinct regions of the screen. This setup can be extended as shown in Figure 12 to be used during normal VR usage: The photodiodes are attached at the corners of the display as described before. Most of the screen can then be used by the actual VR application, with only a small area reserved for the brightness readings. The better the photodiodes are attached, the smaller the reserved region needs to be, as adjacent brightness values don't influence the readings as much. The encoded data can be rendered over the scene in the same way a user interface is rendered on top of a 3D computer game. The wires can be led along the cable of the HMD. Some tracked object needs to be in the tracked space to follow a known rotation. The tracked object can be placed in a corner to not obstruct the users pathway.

The tracked movement was chosen to be a rotation, which is not common for human movement. Input devices, however, include algorithms to improve tracking for human motion [12]. Start and stop movements show an initial and settling delay [4], which we

ignore, by disregarding the first measurements when the tracked controller accelerates and stopping the measurement before the tracked controller is brought to rest again. The rotation has the benefit of providing a continuous signal that is easy to control with a microcontroller. Non continuous movement patterns potentially introduce more tracking artifacts due to inertia influencing the sensors inconsistently.

We chose to use a Vive tracker as the tracked device fixed in the setup. This frees the normal motion controllers and the HMD to get used in an experiment. The HMD only needs few additional wires to attach the sensors which can be guided alongside the existing cables. The measured latency can only be taken as a guideline and does not need to express the latency between a movement of the HMD and its respective effect on screen. Some optimizations such as asynchronous timewarp [34] only apply to the HMD to reduce the perceived latency. Late-latching [3] renders controllers with updated positions, which are not reflected in our approach.

The latency mean of the measurements with load is lower than the one without load. The plot shows many outliers with increased latencies, but there are more readings below the mean as well. A possibility is that a system under load might be pressed to sometimes read the tracker information later, and therefore closer to the next display scanout. The effect could be similar to an unvoluntary late-latching.

The described setup measures the total MTP latency but does not provide insight into where the latency originates. Finer details can be obtained by feeding the tracker data directly into the microcontroller, by means of its exposed pins to estimate the tracker influence or cut out the tracker of the setup by providing the tracking signal directly from the microcontroller via e.g., the audio input as in the approach by Di Luca [7].

Our setup assumes the motor to actuate the change in position perfectly, and without time variant latency. We did not test the motor and motor circuit latency which needs to get deduced from the measured end to end latency.

The setup is not restricted to HMDs, but can be used for large screen systems as well. The aquisition of screen brightness values is currently fitted to the Vive display and would need to get adapted for different displays.

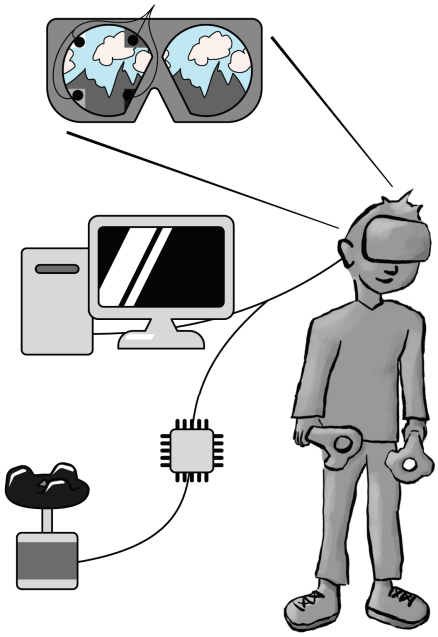


Figure 12: Proposed extension: The photodiodes are positioned at the rim of the screen to be minimally disturbing the experience. If the HMD allows to render outside the visible area, attach them there. The wires that connect to the photodiodes can be led alongside the HMD cable.

7 Conclusion

Latency in Virtual Reality (VR) applications can have numerous detrimental effects, e.g., a hampered user experience, a reduced user performance, or the occurrence of cybersickness. Today, measuring latency in VR systems usually is restricted to report mean values sampled over some dedicated and often isolated application runs which do not accurately reflect overall system behavior under varying load conditions. This paper introduced an apparatus to continuously measure per-frame latency and that therefor is able to capture latency spikes and hence latency jitter. A microcontroller drives a motor with an attached Vive tracker. The tracker's orientation is encoded by a VR application onto parts of the HMD screen as regions of different brightness. The microcontroller reads this data with photodiodes and calculates the difference between the real tracker orientation and the reported orientation as MTP latency. The apparatus can pick up the addition of artificial latency into a VR application for validation.

We evaluated our approach under three different load conditions. We inject artificial latency in a VR simulation and showed that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to an overall degraded system performance. In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time. We stress the fact that continuous measurements of latency behavior of VR application should be a central means to monitor and control an important characteristic of VR systems to assure a high usability and user experience and hope we could contribute an adequate approach to implement such measurements. The source code for the setup is available at `https://github.com/Nighink/latency-rotation`.

References

[1] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. FlowVR: a middleware for large scale virtual reality applications. In Euro-par 2004 Parallel Processing, pp. 497–505. Springer Berlin Heidelberg, Berlin, Heidelberg.

[2] A. Becher, J. Angerer, and T. Grauschopf. Novel Approach to Measure Motion-To-Photon and Mouth-To-Ear Latency in Distributed Virtual Reality Systems. arXiv:1809.06320 [cs], Sept. 2018. arXiv: 1809.06320.

[3] A. Binstock. Optimizing vr graphics with late latching. `https://developer.oculus.com/blog/optimizing-vr-graphics-with-late-latching/`, 2015.

[4] C.-M. Chang, C.-H. Hsu, C.-F. Hsu, and K.-T. Chen. Performance measurements of virtual reality systems: Quantifying the timing and positioning accuracy. In Proceedings of the 24th ACM international conference on Multimedia, pp. 655–659. ACM, 2016.

[5] S. Davis, K. Nesbitt, and E. Nalivaiko. A systematic review of cybersickness. pp. 1–9. ACM Press. doi: 10.1145/2677758.2677780

[6] J. Deber, R. Jota, C. Forlines, and D. Wigdor. How much faster is fast enough?: User perception of latency & latency improvements in direct and indirect touch. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15, pp. 1827–1836. ACM, New York, NY, USA, 2015. doi: 10.1145/2702123.2702300

[7] M. Di Luca. New method to measure end-to-end delay of virtual reality. Presence: Teleoperators and Virtual Environments, 19(6):569–584, 2010. doi: 10.1162/pres_a_00023

[8] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes in latency during head movement. In Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Communication, Cooperation, and Application Design-Volume 2 - Volume 2, pp. 1129–1133. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1999.

[9] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 43, pp. 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999. doi: 10.1177/154193129904302203

[10] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. Human Factors: The Journal of the Human Factors and Ergonomics Society, 30(2):201–217, 1988.

[11] S. Friston and A. Steed. Measuring latency in virtual environments. Visualization and Computer Graphics, IEEE Transactions on, 20(4):616–625, 2014. doi: 10.1109/TVCG.2014.30

[12] E. Gach. Lighthouse tracking examined. `https://kotaku.com/valve-updates-steam-vr-because-beat-saber-players-are-t-1832536574`, 2019.

[13] J. S. GmbH. Heavyload v3.5. `https://www.jam-software.com/heavyload/index.shtml`, 2019.

[14] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin. Video-based measurement of system latency. In International Immersive Projection Technology Workshop, p. 111, 2000.

[15] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pp. 135–144. ACM Press, 2015. doi: 10.1145/2702123.2702432

[16] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui. Dissecting the end-to-end latency of interactive mobile video applications. In Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications - HotMobile '17, pp. 61–66. ACM Press. doi: 10.1145/3032970.3032985

[17] O. Kreylos. Lighthouse tracking examined. `http://doc-ok.org/?p=1478`, May 2016.

[18] M. E. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on, pp. 9–17. IEEE. doi: 10.1109/SEARIS.2012.6231175

[19] J. Liang, C. Shaw, and M. Green. On temporal-spatial realism in the virtual reality environment. In Proceedings of the 4th annual ACM symposium on User interface software and technology, pp. 19–25.

[20] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization, APGV '04, pp. 39–47. ACM, New York, NY, USA, 2004. doi: 10.1145/1012551.1012559

[21] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In IEEE Virtual Reality, 2003. Proceedings., pp. 141–148, March 2003. doi: 10.1109/VR.2003.1191132

[22] M. Mine. Characterization of end-to-end delays in head-mounted display systems.

[23] G. Papadakis, K. Mania, and E. Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry, pp. 581–584. ACM, 2011.

[24] K. S. Park and R. V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In Virtual Reality, 1999. Proceedings., IEEE, pp. 104–111. IEEE, 1999. doi: 10.1109/VR.1999.756940

[25] L. Rebenitsch and C. Owen. Review on cybersickness in applications and visual displays. Virtual Reality, 20(2):101–125, 2016.

[26] T. Sielhorst, W. Sa, A. Khamene, F. Sauer, and N. Navab. Measurement of absolute latency for video see through augmented reality. In 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pp. 215–220. ISSN: null. doi: 10.1109/ISMAR.2007.4538850

[27] M. Slater and A. Steed. A virtual presence counter. Presence: Teleoperators & Virtual Environments, 9(5):413–434, 2000.

[28] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies for real-time interactive systems. In 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS). IEEE Computer Society. doi: 10.1109/SEARIS.2016.7551584

[29] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems. In Virtual Reality (VR), 2016 IEEE, pp. 287–288. IEEE, 2016. doi: 10.1109/VR.2016.7504766

[30] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behavior of virtual reality systems. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology, pp. 47–50. ACM, 2016.

[31] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Effects of latency jitter on simulator sickness in a search task. In 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp. 121–127. IEEE, 2018. doi: 10.1109/VR.2018.8446195

[32] A. Steed. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology, VRST '08, pp. 123–129. ACM, New York, NY, USA, 2008. doi: 10.1145/1450579.1450606

[33] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and S. I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In 3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on, pp. 43–50. IEEE, 2009. doi: 10.1109/3DUI.2009.4811204

[34] J. Van Waveren. The asynchronous time warp for virtual reality on consumer hardware. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology, pp. 37–46. ACM, 2016.

[35] W. Wu, Y. Dong, and A. Hoover. Measuring Digital System Latency from Sensing to Actuation at Continuous 1-ms Resolution. Presence: Teleoperators and Virtual Environments, 22(1):20–35, Feb. 2013. doi: 10.1162/PRES_a_00131

[36] J. Zhao, R. S. Allison, M. Vinnikov, and S. Jennings. Estimating the motion-to-photon latency in head mounted displays. In 2017 IEEE Virtual Reality (VR), pp. 313–314. ISSN: 2375-5334. doi: 10.1109/VR.2017.7892302

Copyright

Author Contributions

The author created the final experimental setup, programmed the necessary software and conducted the measurements. He wrote the description of the setup and measurements and took the lead in writing the manuscript. He provided critical feedback and helped shape the research, the analysis and the manuscript.

4.4 Conclusion

We provided an illustrated overview over different approaches to measure latency, making the topic more approachable to readers. Latency measurement approaches may be quick to conduct with frame counting [He+00], require additional evaluation steps like sine fitting [Ste08] or involved hardware setups [BAG18]. We provided software to make sine fitting easier to conduct [Sta+20] and provide an approach to detailed latency measurements [SNL20b]. This way, we make latency measurement more approachable and easier to conduct. We provide an approach for detailed latency measurement if this is necessary.

Discussed Research Questions

R_1 **How does latency behave in real-time interactive systems?** We see that latency varies around a mean value with some outliers. The outliers gather in multiple normally distributed clusters with rare single outliers of even higher latency. Our elaborated system allows detailed measurement of motion to photon latencies in virtual reality applications. The described latencies behave similar to the latencies described in other chapters.

$R_{1.1}$ **How to measure latency?** We provide an overview of different latency measurement approaches by other researchers. The paper provides a lightweight overview of different approaches grouped by similar methods. The reader gets an impression of the field to then dive deeper into the topic.

We identify Steed's sine fitting [Ste08] as a reliable and easy way to measure latency for every researcher in the virtual reality domain. We provide a software to easier conduct the Sine Fitting approach that is quick to conduct and provides one estimate of time invariant latency. We hope to lower the bar for researchers to conduct and report latency measurements.

On the other side, we provide a more involved latency measuring approach that requires hardware instrumentalisation. This setup can measure detailed latency behaviour to drive insight into latency behaviour.

$R_{1.2}$ **How to describe latency?** We discuss the measured data of our complex measuring setup in the way described in the next chapter.

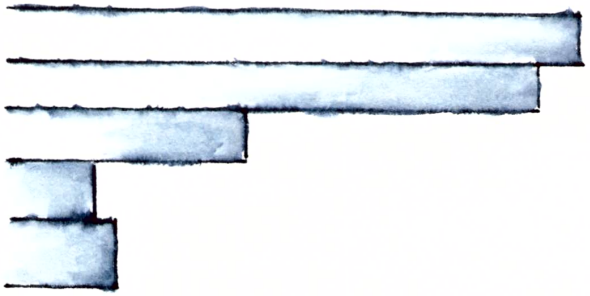
Chapter 5

JITTER DESCRIPTION

Now that we can measure latency, we need to describe what we measure. A good description captures as much of the latency characteristic as possible while needing little space. A good description is easy to understand and quick to grasp. Unfortunately, we have not yet found a description that captures all the different aspects of latency.

Our paper discussing latency and cybersickness in Chapter 7 summarises how latency is usually reported: Most researchers report one mean value with an optional standard deviation and occasional minumum and maximum values. Few illustrate the latency dynamics with additional explanations like histograms or scatter plots. The surveyed papers are all papers that revolve around measuring latency.

papers surveyed	27
mean	25
standard deviation	11
min / max value	4
histogram	5

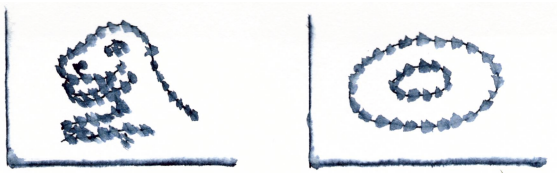


Papers that report only latency to show the fidelity of their system are less likely to go to length to report detailed latency behaviour. Most studies don't report latency at all. The 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) attracts researchers that are likely to have heared of negative effects of latency. 85 papers conducted a user study out of 104 published papers in total. Only 6 reported the latency of their employed VR system. Those 6 papers all reported one average latency value with one also reporting a standard deviation.

The problem with reporting only a mean value and a standard deviation is that it hides the data's real shape. Anscombe [Ans73] showed four sets of points that share their mean and standard deviation of the x and y coordinates but are obviously different. The image below shows the general point distribution.

Matejka and Fitzmaurice [MF17], following the idea of Anscombe, show how to generate different shapes that share the same statistical measures. They portrait datasets

similar to the sketches below.



5.1 The Stacked-Z Test

The examples show, how frail mean values are when used to describe complex data. Our own measurements show that latency measurements gather around a mean which would make the usage of mean values expressive. This, however, depends on the measured system. We [SNL16b], as well as Pape et al. [Pap+20] and Sielhorst et al. [Sie+07] measured distributions that resemble a mixture of multiple normal distributions. Here, one reported mean value could obfuscate the true behaviour.

We proposed a way to visualize latency behaviour that incorporates the spiking nature of latency. The proposed solution assumes that latency is normally distributed and the outliers of this normal distribution again follow a normal distribution. The approach allows to group latency measurements into groups of outliers. This way to report latency conveys more information on the behaviour than the currently most employed approach of reporting mean and standard deviation values.

The paper was published as Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. “Towards comparable evaluation methods and measures for timing behavior of virtual reality systems”. In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. 2016, pages 47–50.

Towards Comparable Evaluation Methods and Measures for Timing Behavior of Virtual Reality Systems

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

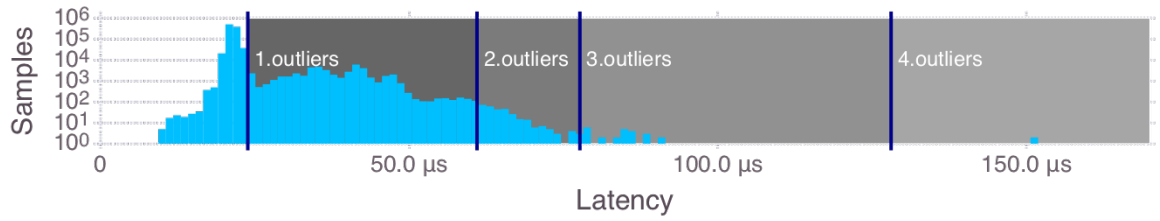


Figure 1: *Histogram visualization illustrating the distribution and categorization of latency measures using a logarithmic y-axis. The example shows the results gained by a modified z-score test.*

Abstract

A low latency is a fundamental timeliness requirement to reduce the potential risks of cyber sickness and to increase effectiveness, efficiency, and user experience of Virtual Reality Systems. The effects of uniform latency degradation based on mean or worst-case values are well researched. In contrast, the effects of latency jitter, the distribution pattern of latency changes over time has largely been ignored so far although today's consumer VR systems are extremely vulnerable in this respect. We investigate the applicability of the Walsh, generalized ESD, and the modified z-score test for the detection of outliers as one central latency distribution aspect. The tests are applied to well defined test cases mimicking typical timing behavior expected from concurrent architectures of today. We introduce accompanying graphical visualization methods to inspect, analyze and communicate the latency behavior of VR systems beyond simple mean or worst-case values. As a result, we propose a stacked modified z-score test for more detailed analysis.

Keywords: virtual reality, latency, outlier, cyber sickness

Concepts: •**Software and its engineering** → **Virtual worlds software;** •**General and reference** → *Metrics;* •**Computer systems organization** → *Reliability;* Multicore architectures; Real-time system architecture;

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

VRST '16,, November 02-04, 2016, Garching bei München, Germany
ISBN: 978-1-4503-4491-3/16/11 ...$15.00
DOI: http://dx.doi.org/10.1145/2993369.2993402

1 Introduction

Virtual reality applications are complex systems that consist of multiple interdependent parts to handle input, simulation and output. Thereby it has to be ensured that the processing is fast enough to allow for a fluid experience. In computer science in general and VR in particular timing behaviour is compared in regards to average case or to the worst case. We argue that this is not enough when it comes to latency and latency changes in particular. We need additional and more detailed information to analyse, detect, communicate and compare timing behaviour of systems which require a high timeliness in VR and related fields.

Latency spikes are not yet understood enough with respect to cyber sickness. For an in-depth analysis of their effects on VR users, latency spikes have to be measured as well as separated from the expected latency distribution inherent in the system.

In this paper, we look into this second step on how to separate outliers from other measurements and find descriptive visualisations.

The contributions of the work presented here are as follow:

1. We assess different outlier tests on their suitability to extract outlier data from latency measurements.

2. We develop, test, and propose recursive application of outlier tests based on a repeated computation of outliers on outliers to get multiple levels of severity for outliers.

3. We present visualization examples suitable to inspect and communicate latency and latency jitter data and patterns not easily captured in single measurement values currently available (see Figure 1 as an example).

This paper is structured into first discussing related research, followed by the introduction and description of our test data that will be used to assess the proposed methods that are explained afterwards. In the end there is a discussion of the findings with a conclusion and ideas for future work extending the research presented here.

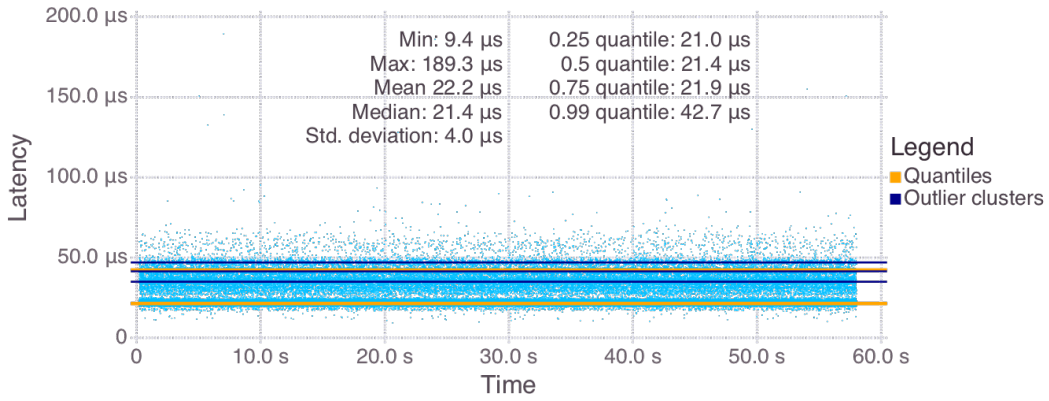


Figure 2: *The measured latencies plotted over time. The point size is chosen small to better show the structure of the data. The orange lines show the 25%, 50%, 75% and 99% quantiles. The first three quantiles are close enough together to not allow a distinction between their respective lines. Most of the measurements are below 24µs. Outliers cluster around certain values indicated by the blue lines.*

2 Related Work

Simulator sickness is a problem of VR applications where users are experiencing symptoms such as nausea [Kennedy et al. 1993]. Visual delay was found as a major contributing factor already in early simulators [Frank et al. 1988]. Latency also influences the performance of test subjects both if time variant latency is added [Ivkovic et al. 2015] or for latency spikes [Teather et al. 2009]. The assumption is consequently that latency spikes influence simulator sickness with a similar impact as the better researched time invariant latency.

The performance of VR applications is usually assessed by measuring motion-to-photon latency which tracks the time between an input on a certain input channel and the time it takes to show its effect on a display. Approaches to measure this latency are sine fitting [Steed 2008], light sensing [Di Luca 2010] and automated frame counting [Friston and Steed 2014]. In this paper, the focus is on latency that is contributed by the VR application, a subset of motion-to-photon latency. While there are many optimization techniques for the rendering stage like frameless rendering [Bishop et al. 1994], latency at the application stage is yet less researched.

VR systems often consist of multiple software components to handle various input and output modalities that run in parallel or on distributed machines [Latoschik and Tramberend 2012; Allard et al. 2004]). This facilitates latency jitter, especially in the communication of the different modules [Stauffert et al. 2016].

Outliers are defined as "observations that deviate so much from other observations as to arouse suspicion that it was generated by a different mechanism." [Hawkins 1980]. In our case, we assume that outliers are also caused by factors outside of the application such as interrupts or other software running on the same computer. They are equally dependent on the application that is examined and its surroundings which is why it is not possible to find one fixed threshold that works for all applications.

Here, we examine the results using the Walsh [Walsh and others 1950], the generalized ESD [Rosner 1983] and the modified z-score [Iglewicz and Hoaglin 1993] outlier test on their suitability to extract outliers from latency measurement data. See [Hodge and Austin 2004] for a discussion of different approaches and application fields.

3 Method

We adapted the method of [Stauffert et al. 2016] to obtain latency measurements. The test measures the time of message passing between two pairs of actors, a common task for VR systems that need to employ parallelism to maximise performance. While the additional actor scheduling will produce its very unique latency jitter and distribution patterns, this would be equally true for any alternative concurrency scheme.

As a testing platform, we use a Raspberry Pi 2 running Raspbian on a Linux kernel (version 4.4.9) with the kernel timer resolution set to 1000Hz for lower response times. The tests are based on the C++ actor library CAF [Charousset et al. 2014].

Figure 2 shows the measured latencies for each communication over time. Most measures fall in a small range indicated by the orange lines for the 25%, 50% and 75% quantiles that are too close together to be distinguishable on the plot. It is evident, that they are not sufficient to describe the distribution. Blue lines indicate clusters of outliers as can be found by peaks in the histogram depicted in Figure 1.

We will analyse the data in non overlapping windows. The time window size here is chosen arbitrarily as 1s. This was done to allow to compare the results of our tests for multiple time frames. The size of the time window needs to be chosen dependent on the property that is measured. It has to be wide enough to contain enough samples to conduct the tests but needs to be small enough to preserve temporal descriptiveness. Applications will try to keep the window as small as possible to be able to attribute certain events to outliers and react timely.

In the following, we are looking for a suitable test to classify outliers. We conduct the three examined tests over non overlapping time windows of one second with the total gathered data spanning one minute. This is to show the performance over multiple time slices that follow the same underlying mechanics but can change due to outside factors.

3.1 Distribution of Measurements

The samples describe the interference patterns of the algorithmic base that the tests are build upon. We expect the frequencies de-

scribing the sending and receiving algorithms to be interfered by operating system frequencies, other software frequencies as well as hardware influences. While the interference pattern does not follow a normal distribution, we expect the message passing algorithm on its own to approach a normal distribution for a sufficiently long measurement interval. Extraneous influences then lead to a skewing of the distribution. Consequently, our measures do not follow a normal distribution as tested with the Anderson Darling test provided in the R library "nortest" [Gross and Ligges 2015] with a p-value of $< 2.2e-16$.

3.2 Walsh Outlier Test

The Walsh outlier test [Walsh and others 1950] is a nonparametric test to detect multiple outliers. In contrast to many other statistical tests that require a normal distribution, this test works on data that is not normally distributed.

The test shows whether a suspected amount of outliers is present in the data with a level of significance that is dependent on the sample size ($\alpha = 0.1$ if $60 < n \leq 220$ and $\alpha = 0.05$ if $n > 220$). The test is run with an increasing number of suspected outliers where the highest amount of suspected outliers k that still test positive is taken.

The k largest latencies are then classified as outliers. The test is computed by determining the values

$$\begin{aligned} c = \lceil\sqrt{(2n)}\rceil; r = k + c; \qquad b^2 = \frac{1}{\alpha} \\ a = \frac{1 + b\sqrt{\frac{c-b^2}{c-1}}}{c - b^2 - 1} \end{aligned} \tag{1}$$

If $X_{n+1-k} - (1+a)X_{n-k} + aX_{n+1-r} > 0$ then the k largest points are outliers, where X_i are the sorted values of the time window such that $X_1 < X_2 < \cdots < X_n$.

The criteria tests differences of samples therefore being sensitive to samples that are distant from others. However, if there are more outliers in close neighborhood these are not detected.

The Walsh test therefore helps to capture the extreme outliers in time windows. There are some windows where no outlier is found because they are grouped too densely even though the same latency was flagged to be an outlier in a different time window.

3.3 Generalized ESD Outlier Test

As described, we expect the observed algorithm to approach a normal distribution that gets influenced by outside factors. Therefore, we try a different outlier test that assumes normal distribution, which promises to separate the samples of the expected normal distribution from samples that were influenced.

We recursively repeat the generalized ESD test [Rosner 1983] on the outliers to distinguish between outliers and outliers of outliers. This leads to a separation of the majority of the measurements from their outliers but then additionally identifies severe outliers. Assuming the influencing factors themselves approach a normal distribution, we separate different influences and their accumulated interference patterns from each other. This coincides with the visual impression from Figure 2 where most outliers are clustered above the mean values with few extreme values.

To determine whether a measured latency x_i of the observed time window is an outlier, the values R_i and λ_i are calculated

$$\begin{aligned} R_i &= \frac{\max|x_i - \bar{x}|}{\sigma} \\ \lambda_i &= \frac{(n-i)t_{p,n-i-1}}{\sqrt{(n-i-1+t^2_{p,n-i-1})(n-i+1)}} \\ p &= 1 - \frac{\alpha}{2(n-i+1)} \end{aligned} \tag{2}$$

where $t_{p,\nu}$ is the $100p$ quantile of the t distribution with ν degrees of freedom, $\bar{x}$ the sample mean, σ the sample standard deviation and α the significance level here set as 0.05. The largest i that satisfies $R_i > \lambda_i$ is the number of outliers in the sample.

We tried to substitute the mean and standard deviation over the values of the time window with the respective functions over the complete test run. This would allow to run the application once to gather a representative mean and standard deviation of the values and then use those for subsequent test runs. This, however, increases the lower threshold and therefore performs worse in detecting outliers.

3.4 Modified z-score outlier Test

The last test conducted is a modified z-score outlier test [Iglewicz and Hoaglin 1993], which assumes normal distribution as well. The modified z-score Z_i is computed for each value x_i in the time window to be

$$\begin{aligned} Z_i &= \frac{0.6745(x_i - \tilde{x})}{\text{MAD}} \\ \text{MAD} &= \text{median}(|x_i - \tilde{x}|) \end{aligned} \tag{3}$$

Where $\tilde{x}$ is the median over all samples and MAD the median absolute deviation.

We changed this test to not take the median absolute deviation and median of the samples in the window to be tested but of all the measured samples. This allows to run an application using the measurements to determine the absolute median deviation and the median of this sample and use those values for subsequent application runs to assess the performance. This changes equation 3 to calculate the median and MAD for the values of the first run w_i with

$$\begin{aligned} \tilde{w} &= \text{median}(w_i) \\ \text{MAD}_w &= \text{median}(|w_i - \tilde{w}|) \end{aligned} \tag{4}$$

and then calculate the z-scores for all subsequent runs with

$$Z_i = \frac{0.6745(x_i - \tilde{w})}{\text{MAD}_w} \tag{5}$$

The threshold was chosen to be 3.5 as suggested by the authors. Recursive use of this test yields more gradations of outliers than the generalized ESD test. Here, we distinguish between the main part of the outliers, an area above with only few outliers and the rare extremes.

4 Discussion

The Walsh outlier test only captures few extreme outliers. These extremes are supposed to have the most impact and are therefore the most interesting for further examination.

The generalized ESD and modified z-score tests are similar to each other. Both support the classification of outliers into multiple levels by stacking them so they can be used to determine how severe an outlier is. The modified z-score allows for finer separation. Additionally, it allows to determine base values like the MAD and median for one test run to establish a base line that can then be used for subsequent runs. When trying the same with the generalized ESD test, the lowest threshold to classify outliers moves up to then yield a value more distant than the threshold we would have chosen by inspecting the histogram.

5 Conclusion

VR applications get optimised for mean and worst case behaviour, which we argue is not enough to capture latency behaviour as it does not account for different patterns of latency outliers.

We have discussed three tests to find outliers in latency measurements. The measurements here were taken as the time needed for the communication of two actors, a common process in VR systems that consist of multiple parts. The measurements were then analyzed in time windows to assess how good they detected outliers over time.

The Walsh test allows to catch the extreme outliers. Those are supposed to have the most impact on an application's performance. Finer analysis is offered by using a stacked modified z-score test that groups outliers into categories of different severity.

We propose to first establish a base line by running an application once to calculate the median and MAD over the latency samples. Subsequent runs can then be analyzed to determine what category of outlier a latency sample belongs to.

The proposed recursive application of outlier tests by repeating a test on the detected outliers multiple times yields several categories of outliers. These different levels of severity can then be used to evaluate an application on multiple scales.

6 Future Work

The impact of latency spikes on cyber sickness is not yet tested, which is necessary to evaluate the impact of the measured latency spikes on the user. We have laid a base to measure latency spikes and gather outliers. This data can then be used to correlate it with symptoms of cyber sickness shown in tests where such spikes are artificially added, enhanced or altered in their pattern. Real-time systems where far less latency spikes are observed as discussed in [Stauffert et al. 2016] can be used to test against.

Using the gathered outlier data, it will be possible to derive an outlier fingerprint of an application on a specific hardware. Comparing this to a different application's outlier fingerprint running on the same hardware might open up the possibility to compare applications by their latency behavior, benchmark them and propose improvements.

We have used different visualisation methods to allow for an intuitive interpretation of outliers. Future research has to use user studies to show how intuitive these graphs are and how convenient they are to spot unwanted application behaviour.

References

ALLARD, J., GOURANTON, V., LECOINTRE, L., LIMET, S., MELIN, E., RAFFIN, B., AND ROBERT, S. 2004. FlowVR: a middleware for large scale virtual reality applications. In *Europar 2004 Parallel Processing*, Springer, 497–505.

BISHOP, G., FUCHS, H., MCMILLAN, L., AND ZAGIER, E. J. S. 1994. Frameless rendering: Double buffering considered harmful. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM, 175–176.

CHAROUSSET, D., HIESGEN, R., AND SCHMIDT, T. C. 2014. CAF - the C++ Actor Framework for Scalable and Resource-Efficient Applications. ACM Press, 15–28.

DI LUCA, M. 2010. New method to measure end-to-end delay of virtual reality. *Presence 19*, 6, 569–584.

FRANK, L. H., CASALI, J. G., AND WIERWILLE, W. W. 1988. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society 30*, 2, 201–217.

FRISTON, S., AND STEED, A. 2014. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on 20*, 4, 616–625.

GROSS, J., AND LIGGES, U. 2015. *nortest: Tests for Normality*. R package version 1.0-4.

HAWKINS, D. M. 1980. *Identification of outliers*, vol. 11. Springer.

HODGE, V. J., AND AUSTIN, J. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Review 22*, 2, 85–126.

IGLEWICZ, B., AND HOAGLIN, D. 1993. *Volume 16: how to detect and handle outliers, The ASQC basic references in quality control: statistical techniques, Edward F. Mykytka*. PhD thesis, Ph. D., Editor.

IVKOVIC, Z., STAVNESS, I., GUTWIN, C., AND SUTCLIFFE, S. 2015. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. ACM Press, 135–144.

KENNEDY, R. S., LANE, N. E., BERBAUM, K. S., AND LILIENTHAL, M. G. 1993. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The international journal of aviation psychology 3*, 3, 203–220.

LATOSCHIK, M. E., AND TRAMBEREND, H. 2012. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, IEEE, 9–17.

ROSNER, B. 1983. Percentage Points for a Generalized ESD Many-Outlier Procedure. *Technometrics 25*, 2 (May), 165.

STAUFFERT, J.-P., NIEBLING, F., AND LATOSCHIK, M. E. 2016. Reducing Application-Stage Latencies For Real-Time Interactive Systems. In *9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, IEEE Computer Society.

STEED, A. 2008. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, NY, USA, VRST '08, 123–129.

TEATHER, R. J., PAVLOVYCH, A., STUERZLINGER, W., AND MACKENZIE, S. I. 2009. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, IEEE, 43–50.

WALSH, J. E., AND OTHERS. 1950. Some nonparametric tests of whether the largest observations of a set are too large or too small. *The Annals of Mathematical Statistics 21*, 4, 583–592.

Copyright

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

Author Contributions

The author conducted the selection and evaluation of suitable statistical tests and took the lead in writing the manuscript. He provided critical feedback and helped shape the research, the analysis and the manuscript.

5.2 Conclusion

We propose the stacked-z test to separate measured latency values into a cluster around the mean value, multiple clusters with larger latency values and few extreme outliers. An accompanying visualisation with a histogram improves the understandability. This is a first step that identifies necessary information for a comprehensive reporting of time-variant latency.

Discussed Research Questions

$R_{1.2}$ **How to describe latency?** We propose to describe latency behaviour as showing the majority of latencies normally distributed around a mean with multiple normally distributed clusters of longer latencies. There are rare extreme latency outliers. Chapter 7 shows that other researchers find similar patterns but describe them in different ways. Our description uses repeated use of a z-outlier-test: Each latency value is tested if it is an outlier with the z-score

$$z_i = \frac{0.6745(x_i - \tilde{x})}{\text{median}(|x_i - \tilde{x}|)}$$

A z-score above 3.5 indicates an outlier. The test is repeated on the outliers to determine outliers of outliers and so forth. This separates the normally distributed clusters. The result is visualised by annotating a histogram of the latency values with the found cluster separators.

Chapter 6

SIMULATION

Simulation of latency is necessary to be able to run experiments that research effects of increased latency. Adding a fixed amount of latency is done with a ringbuffer ([PMK11]). Waltemate et al. [Wal+16] describe the usage as

> "This buffer was filled with incoming motion capture frames, and as soon as one of these frames was older than the desired latency offset, it was emitted and used." Waltemate et al. [Wal+16, p. 3]

This approach, however, only simulates a time invariant latency increase. St. Pierre et al. [SP+15] describe their implementation that can simulate time-variant latency as

> "The program manipulated latency by buffering captured images in integral units of the frame capture rate [...]. A constant buffer size was used to add a consistent latency [...]. A varying buffer size was used to add a varying latency" St. Pierre et al. [SP+15, p. 2]

Their approach allows to simulate time-variant latency but leaves room for interpretation on how the implementation handles this shortening of the buffer. It also looks to be dependent on a regular arrival of tracking data. The tracking data, however, is also influenced by varying latency in its processing and sending data. The assumption of a regular tracking data arrival is optimistic. The implementation based on this assumption may hide latency variation of the incoming tracking data.

6.1 Simulation of Latency Spikes

We proposed a way to simulate latency that models both a time invariant part and a time-variant part on top of it. The time-variant part is described as consisting of latency spikes. The spike duration and the inter-arrival times are drawn from a probability density function describing the respective behaviour.

The paper was published as Jan-Philipp Stauffert, F. Niebling, and M. E. Latoschik. "A Latency and Latency Jitter Simulation Framework with OSVR". in: *2017 IEEE 10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. 2017, pages 1–7

A Latency and Latency Jitter Simulation Framework with OSVR

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

ABSTRACT

Latency is a pressing problem in Virtual Reality (VR) applications. Low latencies are required for VR to reduce perceptual artifacts and cyber sickness. Latency jitter, i.e. variance in the pattern of latency, prevent coping mechanisms as users can't adapt.

Low latency is a fundamental timeliness requirement to reduce the potential risks of cyber sickness and to increase effectiveness, efficiency, and user experience of Virtual Reality Systems. The effects of uniform latency degradation based on mean or worst-case values are well researched. In contrast, the effects of latency jitter, the distribution pattern of latency changes over time has largely been ignored so far, although today's consumer VR systems are extremely vulnerable in this respect.

In this paper, we propose to create a model of latency and latency jitter with empirical distributions as well as a method of using those models to inject latency. The process of creating a latency model is demonstrated with an example of gathering and converting latency samples from an example application. We show how to simulate latency and motivate to use it in middleware to allow for less intrusive latency effect evaluations.

Index Terms: D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming; D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Virtual Reality applications often consist of multiple components to handle input processing, simulations, artificial intelligence, or rendering etc. Non-functional software quality requirements like modularity, maintainability, and reusability can have an unforeseeable impact on the temporal behavior of software, especially for a Real-Time Interactive System (RIS), i.e., in Virtual, Augmented, and Mixed Reality (VR, AR, and MR) and computer games. Due to the complexity of many RIS applications, they are often split into different parts to foster cohesion and decoupling. To exploit today's multi-core and multi-CPU architectures and to avoid unnecessary blocking, these parts often will be executed concurrently or they will be completely distributed [2, 12].

Each computation in each application part takes time with the orchestration and synchronisation adding additional overhead. The computations cause a latency between input data entering the system and output data based on them exiting the system.

In theory, the created latency can be determined deterministically by inspecting the system and the control paths taken. In practice, today's hard- and software is too complex to determine how its latency behaves. It is agreed upon that more latency, i.e. a bigger time discrepancy between input and the resulting output, lead to a decreased performance operating a system and especially in VR to decreased immersion and acceptance due to an increase in cyber sickness [10, 18].

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

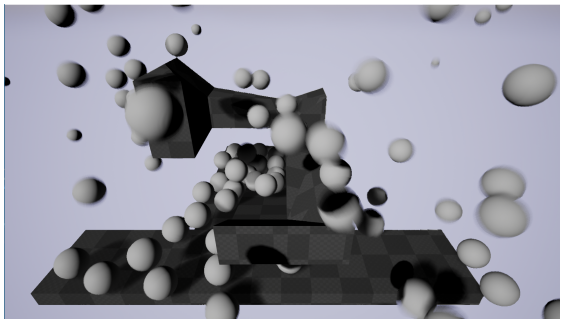


Figure 1: Example scene to create latency measurements. Spheres are spawned every second and collide with each other as well as with the environment.

While time invariant latency is well researched, this paper focuses to create a model for latency jitter. Latency jitter describes latency that changes over time, here with the focus on latency spikes. Latency jitter leaves the user unable to adapt as it is constantly changing.

To provide a tool for further research of latency spikes, this paper proposes to use the here introduced latency models as a basis to simulate latency. This simulated latency can be inserted into selected parts of an application to enable detailed observations.

The contribution of the work presented here are as follow:

- Description of how to create a model for latencies from measured data
- Description how to use a latency model to simulate latency

This paper is structured into first discussing related research, followed by the description of our method to model and simulate latency and latency jitter. We will then introduce a latency injection system that allows for the introduction of latency and latency jitter based upon our model without changes to the VR application. In the end there is a discussion of the findings with a conclusion and ideas for future work extending the research presented here.

2 RELATED WORK

Simulator sickness is a problem of VR applications where users are experiencing symptoms such as nausea [11]. While some users are more sensible, there are certain factors that make simulator sickness worse for most users. Visual delay was found as a major contributing factor already in early simulators [8]. Latency also influences the performance of test subjects both if time variant latency is added [10] or for latency spikes [18]. The assumption is consequently that latency spikes influence simulator sickness with a similar impact as the better researched time invariant latency.

The performance of VR applications is usually assessed by measuring motion-to-photon latency which tracks the time between an input on a certain input channel and the time it takes to show its

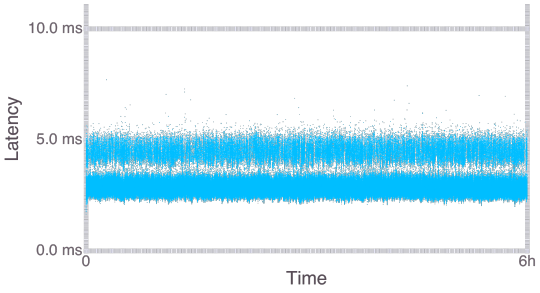


Figure 2: Latency measurements of the physics simulation in our test scene, scaled to show the mean latency with most of the samples and above a region with sparser sample density.

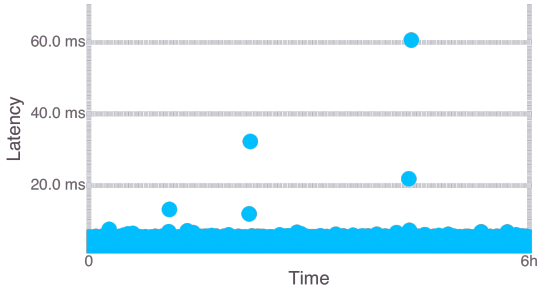


Figure 3: Latency measurements of the physics simulation in our test scene, scaled to show all values with bigger point size to make the outliers visible.

effect on a display. Approaches to measure this latency are sine fitting [17], light sensing [5] and automated frame counting [9]. In this paper, the focus is on latency that is contributed by the VR framework and its internal software processes with their interactions, a subset of motion-to-photon latency. While there are many optimization techniques for the rendering stage like frameless rendering [3], latency at the application stage is yet less researched.

Latency has been injected into virtual environments to evaluate its effect on task performance, presence, and other factors. Most of these experiments delay tracker input data by a controllable amount of time units — frames or multiples of the tracker sampling rate — by employing a ring buffer or other FIFO data structures either inside the tracker itself, its software driver, or the VR application. Experiments are then performed with different, yet constant per experiment, amounts of latency artificially injected into the system.

Ellis et. al. tested distinguishability of changes in latency for hand [7] as well as for head [6] movements. They employ custom tracker drivers to ensure a low base latency and to provide the ability to add custom latency to their input devices. Building on this work, Mania et. al. test sensivity to head tracking latency in virtual environments [13]. Meehan et. al. studied the effects of latency on presence in stressful virtual environments [14]. To do user studies with different latency settings, they adapted their VRPN client implementation to delay tracker input data by a fixed amount of time to add constant end-to-end latency to their system, enabling controlled experiments with 50ms and 90ms of latency respectively. Other studies that control latency, e. g. performed by Allison et. al. [1], or more recent work on latency control by Papadakis et. al. [15] as well as by Waltemate et. al. [19], also only allow for the insertion of constant latency by delaying tracker input data using ring buffers.

Time invariant latency, however, ignores that latency in applications changes over time. The effects of latency jitter are far less researched as discussed above. To describe the effects that can be observed, a model of time variant latency is needed. Additionally, there needs to be a way to introduce time invariant latency to then allow more research of the effects on the systems and for the users.

3 METHOD

Distinguishing the work presented here from past research outlined in the previous section, we are presenting a method to inject latency jitter (instead of constant latency) into VR applications, based upon a model created from measurements of existing VR systems, without the need to change the VR application itself.

Our approach is split into two parts:

- Modelling latency by deriving an empirical distribution from measurements
- Using a latency model and a latency injector, based upon established VR middleware, to simulate latency

We discuss latency modelling at the example of timing the physics calculation of a test scene created with the Unreal Engine 4.

Afterwards, latency simulation based on a latency model is discussed at the example of creating an OSVR plugin to delay tracking data without needing to alter the affected application.

3.1 Latency Model creation

The latency model described here is based on latency measurements at the application stage. We explain our approach with example measurements taken with a test scene made with the Unreal Engine 4. This offers measurements relateable to real world VR scenarios.

The measurements are divided into outlier groups to separate expected latencies close to the mean value from the less often occuring latency spikes. Latency spikes are further categorised by recursively applying the outlier detection algorithm as described by Stauffert et al. [16]. For each category of detected outliers, two empirical distributions are derived. One describes the duration of the latency spike. The other describes how much time passes between latency spikes.

Figure 4 shows the process of deriving a model with graphs demonstrating how example measurements are transformed at every step. The following sections will further explain the steps undertaken.

3.1.1 Measurements

The basis for the latency model are latency measurements. Here, we demonstrate how to measure them, and what appearance latency measurements have.

We created a scene with the Unreal Engine 4, where a physics enabled sphere is spawned every frame. The balls collide with the sparse environment and with themself, and get despawned once they fall through holes in the environment. An overview of the scene is given in figure 1.

The engine allows to create objects that are updated at different times during the game loop. Utilising this, we register two functions of an object. One gets invoked before the physics calculation, the other after the physics system has updated. With the two invocations, the elapsed time in between is measured which is taken to create a latency profile for the engine execution.

With this setting, a big part of the computational time of the application to survey is measured. Measurement, however, can be conducted for far smaller parts such as the duration of AI computation for only one object in a scene or one small algorithm.

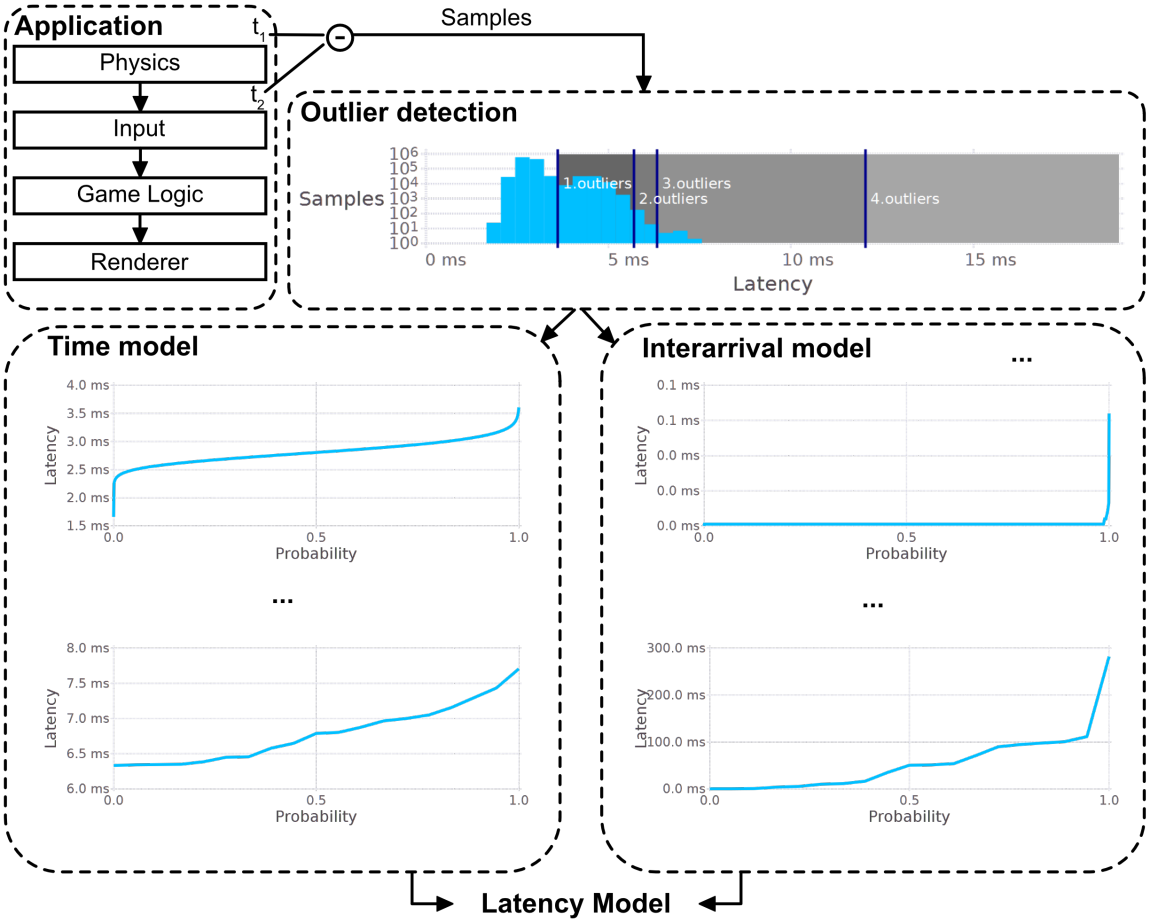


Figure 4: Steps to create a latency model: Latency samples are gathered in the application. Outliers are extracted and sorted into different categories. The distributions for the duration and interarrival times or latencies are expressed by their inverted cdf.

Figure 2 shows a scatter plot of the latencies measured for each frame. Note that most samples gather around a mean with less samples above the mean latency. Figure 3 shows all samples including the extreme outliers. The mean is 2.93ms, variance 0.2ms^2, min 1.6ms, max 60.6ms.

3.1.2 Outlier detection

The measurements are separated into outlier categories. For this, we recursively use the modified z score test to determine outliers as proposed for latency data by Stauffert et al. [16]. They calculate the z-score for each sample to determine if it is an outlier. Once, all outliers are found in the dataset, the algorithm is applied to separate the outliers from the outliers of the outliers. This is repeated recursively until no outliers can be found anymore.

The z-score over all samples x_i is calculated with

$$Z_i = \frac{0.6745(x_i - \tilde{x})}{\text{MAD}} \\ \text{MAD} = \text{median}(|x_i - \tilde{x}|) \tag{1}$$

Samples with a Z-score larger than 3.5 are detected as outliers. For the example data, after the fourth application of the algorithm, no more outliers can be found.

3.1.3 Distribution derivation

An empirical distribution is derived for each outlier category and the samples around the mean. The distributions are described by their cumulative density function (cdf). For convenient generation of samples from the distribution, the cdf needs to get inverted. The quantile function represents the inverse cumulative distribution function and is faster to compute than first generating the cdf and then inverting it. The quantile function is sampled to avoid the need to save all latency samples but be able to represent the distribution with a small amount of values. Using more samples when sampling the quantile function leads to a more precice representation of the observed distribution.

The result from sampling the quantile function is an array of latency values. To draw a random value that is distributed by the empirical distribution, a random uniformly distributed number is drawn. The random number needs to be between 0 and the number of samples used for sampling the quantile function. This random number is used to index into the array.

The latency model proposed here consists of empirical distributions for each outlier category for both the latency duration and the time in between the observed latency spikes.

3.2 Latency Injection

This section describes how to use a latency model as described above to simulate latency. The discussion is lead with an example of an OSVR plugin where additional latency is simulated for processing tracking data. While the example helps to describe the process, the approach isn't limited to an implementation in a middleware, but can be used to simulate latency in parts of the application itself. We will describe the benefits there are in simulating latency in a middleware like OSVR, before presenting our latency simulation approach.

3.2.1 Placement in middleware

OSVR provides a middleware supporting many devices, presenting a uniform API to applications abstracting away differences. Introducing latency simulation in this layer allows for little intrusion of the application code. This enables the evaluation of different amounts of latency jitter in existing VR systems. The existing software does not need to get modified in the process.

Many VR applications couple physics, logic and the renderer very tightly to guarantee that for each frame, all systems have updated their state. Introducing latency there is more prone to influence the whole application performance. Middleware like OSVR runs parallel to and decoupled from the application. This allows to introduce latency into selected tracking devices without affecting others. Having only one device input modified promises to analyse effects of latency in different modalities with more detail.

Compare figure 5 for an overview where a latency simulating plugin is located within OSVR. OSVR contains plugins that handle the connection to various tracking devices. The data is either directly forwarded to an application or modified by another plugin, called analysis plugin. Introducing latency in an analysis plugin allows for targeted modification of tracking data by only delaying data of one selected device. OSVR works like a dataflow environment, where data is gathered at the input devices, then processed by plugins in a configurable order until it eventually gets delivered to an application. The application doesn't need to be changed as it can't distinguish whether its data stems directly from the hardware plugin or was tampered with on the way.

3.2.2 Algorithm

We implemented a configured device plugin. This is a plugin that can be used multiple times with different configurations to allow binding to different other devices with individual latency behaviour.

The latency plugin simulates a component in the system that receives data, works with it and then forwards it. The delay between receiving and forwarding a dataset d_l consists of the expected delay $\bar{d}_l$ and a time d_s indicating a spike in latency. If the simulator shall only simulate latency spikes, the expected mean latency to simulate $\bar{d}_l$ is set to zero.

The simulator itself introduces latency by simply executing code. Care has to be taken that the simulated latencies are significantly larger than the values that are introduced by the execution of the simulator. We will discuss this and similar problems at the end.

A sample code describing the execution is shown in listing 1: Initially, a time for the first latency spike is determined. The simulator then receives a dataset and decides d_l.

First, it is initialised with $\bar{d}_l$. If the current time is larger than the time, a latency spike was scheduled, a sample for the latency spike length is drawn from the respective distribution. This length is added to the delay time. Afterwards, the time of the next latency spike is computed by adding a randomly drawn sample of the spike interarrival distribution to the current time.

The simulator then delays further processing of the data by other components by sleeping for the calculated time. Note that a sleep only guarantees that the execution is not resumed before the given amount of time has passed. It might take longer than requested until execution is continued.

This method assumes that there is a buffer in place that stores incoming tracking data until they get processed. This buffer might hold only the most recent data and discard older data elements if new ones arrive, or be implemented with a queue. If the simulated working time exceeds the arrival time, the queue might fill up and overflow.

3.3 Evaluation

For first evaluations, we created an OSVR hardware plugin that continuously sends timestamps. These timestamps are received in an OSVR client application and the difference of the received timestamp to the time it is received is saved.

We gather the latency samples collected with the described method with and without an additional latency injector plugin in the pipeline. In total, we compare three different scenarios:

- Without latency injector

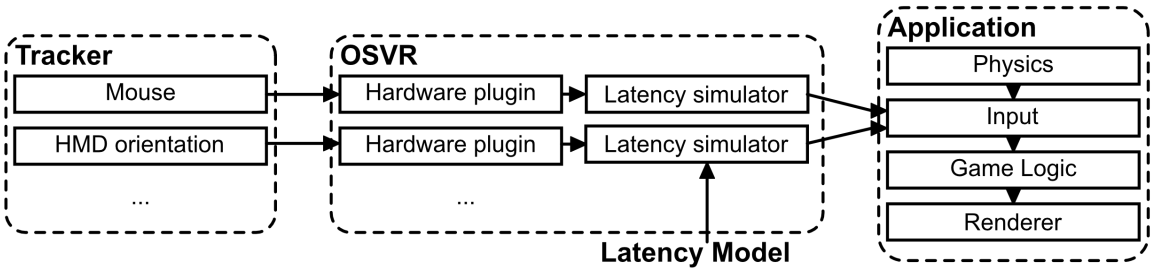


Figure 5: Latency can be injected in the middleware, that administrates the hardware, without needing to change the application.

```
next_spike = now() + interarrival_distribution.draw_value();
while data = receive_data() {
  worktime = mean_work_time;
  if now() + worktime > next_spike {
    worktime += spike_distribution.draw_value();
    next_spike = now() + interarrival_distribution.draw_value();
  }
  sleep(worktime);
  forward_data(data);
}
```

Listing 1: Simple latency simulator

- With latency injector based on a latency distribution that never causes spikes, i.e. reducing the latency injector to be a pass through stage
- With latency injector based on a latency distribution simulating only spikes

We expect to find higher latency outliers with the latency injector in place with more latency outliers when using a latency distribution.

For comparing the two measurements, we use QQ-plots. QQ-plots are used to compare two distributions with each other. Usually, this is done to see how well one distribution fits another. As we already use the quantile function to model latency jitter, the plot shows the quantiles of the measurements without latency on one axis against the quantiles of the measurements with latency on the other axis. A line above the bisector signals more latency spikes with latency injected. See figure 6 for an example of one of these plots.

We find that the mean latency is the same for all cases.

With the latency injector only working as a pass through for messages, there are more extreme spikes with the lower outlier categories exhibiting similar distributions to the run without the added plugin.

Having the latency injector simulating only spikes, there are more spikes visible in the QQ-plots for the outlier categories than without the plugin. The means match and the first outlier category distribution is similar.

This means, our latency simulator is able to simulate latency spikes with little overhead, leaving the latencies around the mean mostly unchanged.

4 Discussion

We proposed a method to model latency and latency jitter with an empirical distribution.

In the introduction, we stated that systems elicit non deterministic latency behaviour due to the complexity of today's computers and applications. Likewise, introducing additional latency into a system can lead to different behaviour. For this discussion, we assume that VR systems consist of different parts that run partly in parallel and exchange state at synchronisation points. The timing of these synchonisation points plays a crucial role of how latency injection into an application is visible to the user.

Latency injected into an application could not change the application at all. If it affects a part of an application that has sufficient buffer time between the time its own execution has finished and the time point it needs to synchronise, this free buffer time can swallow the latency. Similarly, if a part with additional latency injected urges another part to wait but this dependent part has sufficient buffer time, the effect might vanish.

If the effect of added latency does not vanish in the interplay of parts of the application, the effects can be visible in different ways. They can be restricted to simple aspects of the application. An example would be that only the controller inputs lag behind the rest of the application. There could also be a ripple effect where delay in one part of the application causes other parts to miss their assigned time window leading the whole application performance to suffer.

With time invariant latency, the effects are better observable as there will be one pattern emerging as a result. The time variant latency, as observed e.g. in the described example application and modelled here, will exhibit complex effects and will create interference patterns when interacting in a sophisticated system.

This means that after introducing latency, especially time invariant latency, it is mandatory to measure again how the actual effects are.

Using the proposed tools to research latency jitter will allow to better understand the relationship of latency jitter to constant latency. A dejitter buffer [4] as used for packet-switched networks can reduce the jitter at the cost of added constant latency. If latency jitter proves to be worse, this is a tool to alleviate it.

In the future, it will be desirable to measure application stage latency without needing to alter the application but placing the complete benchmarking code into a middleware. For this, it is necessary to receive information from the application. Upon writing this

paper, OSVR doesn't support plugins that take input from the application. There are only plugins that sit between tracking devices and the application. There are no plugins for messages from the application to a device like a force feedback device or an audio device yet.

Once this is implemented, it will be possible to create plugins that detect which message sent by the application belongs to which tracking value. Then, the simulation latency can get determined by the middleware, allowing application stage latency measurements without the need to influence an application at all.

We proposed to use the quantile function for the inverse cumulative density function, which can be directly derived from the samples. This, however, limits the simulation to only produce random numbers that were observed before. Using a kernel density function over the data provides a smoothed probability density function that can be integrated to yield the cumulative density function, which in turn can be inverted. This way around leads to a smoothing of the observed distribution and can also generate values that are close to the observed.

The example data used for the model creation was collected from a specially designed application. As all other applications, it exhibits its own latency pattern. With repeated spawning and despawning of balls, the number of balls in the scene is constantly changing. The available room is restricted to provoke many collisions. This leads to the physics simulation working with a variable number of entities that have different amounts of collisions each frame. The measurements show a latency behaviour with repeated spikes. We did, however, not analyse if the spikes are results of the changing conditions like ball count and collision count from frame to frame or if they have another source. The many objects in our scene could also have lead to lead to the changing numbers being too small in comparison to the overall number of collisions to not be responsible for the variations.

While we proposed a latency injector that adds latency according to a distribution, its own execution adds additional latency. The described sleep to simulate a time of working yields the processor to let other processes work in the mean time, which might or might not be desirable for the application to test. Waking from sleep involves a second context switch which is costly timewise and adds to the latency. A busy loop can be used as an alternative. OSVR needs to gather the measurement from another plugin to pipe it to the latency inducing plugin to afterwards pipe it to either the application or other plugins. This causes additional overhead and therefore latency as described in the evaluation section.

The discussions so far demonstrate that it is possible to measure and induce latency into a system. It is, however, difficult to argue what the source of latency jitter in a measured dataset is as there are too many influences. On the other side, it is difficult to forsee what effects introducing latency has on the affected system and in turn on the user.

5 Conclusion

VR applications get optimised for mean and worst case behaviour, which we argue is not enough to capture latency behaviour as it does not account for different patterns of latency outliers.

This paper proposes to create latency models using empirical distributions based on measured latencies. Such latency models can then in turn be used to simulate latency behaviour at the application stage, either in the application itself or in a middleware. Usage of a latency simulator will allow to better study effects of latency and latency jitter on the user.

6 Further work

The example measurements illustrate the process of generating a latency model. The measurements shown describe the time, the physics engine runs for a test scene. The measurements seem to be similar to the measurements taken in Stauffert et al. [16], who only measured the time for single message passing. It is yet to be shown how similar latency patterns of a small algorithm are compared to a more complex system.

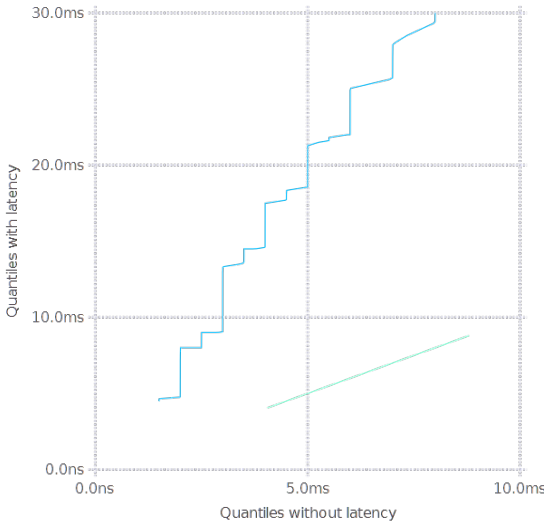


Figure 6: QQ-Plot of the spike duration of the fourth outlier category of the test setting without and with latency simulated. With latency simulated, there are more spikes visible.

Using the shown test scene, a variation of the scene in terms of available space and amount of objects will create different load on the physics system and a changed latency behaviour. Comparing applications or application parts under different load in respect to latency will uncover new insights for the system performance and how latency changes under different conditions.

The artificially injected latency has to get further researched to see if it elicits the desired effects. In any case, the evoked effects need more research.

Every machine will elicit different latency behaviour especially, different spike behaviour. It is up to future work to analyse how comparable latency models based on latency measurements of different machines are.

With a latency inducing plugin in place, effects of latency for different devices can be measured. Next steps include researching of simulator sickness when latency is injected into a head mounted display. Another is to measure performance and task load with motion controller drag and drop tasks.

References

[1] R. S. Allison, L. R. Harris, M. Jenkin, U. Jasiobedzka, and J. E. Zacher. Tolerance of temporal delay in virtual environments. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, VR '01, pages 247–, Washington, DC, USA, 2001. IEEE Computer Society.

[2] T. Arcila, J. Allard, C. Ménier, E. Boyer, and B. Raffin. FlowVR: A framework for distributed virtual reality applications. *Journees de lAFRV*, 2006.

[3] G. Bishop, H. Fuchs, L. McMillan, and E. J. S. Zagier. Frameless rendering: Double buffering considered harmful. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 175–176. ACM, 1994.

[4] R. G. Cole and J. H. Rosenbluth. Voice over IP performance monitoring. *ACM SIGCOMM Computer Communication Review*, 31(2):9–24, 2001.

[5] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence*, 19(6):569–584, 2010.

[6] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes in latency during head movement. In *Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Communication, Cooperation, and Application Design-Volume 2 - Volume 2*, pages 1129–1133, Hillsdale, NJ, USA, 1999. L. Erlbaum Associates Inc.

[7] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 43, pages 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999.

[8] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[9] S. Friston and A. Steed. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):616–625, 2014.

[10] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pages 135–144. ACM Press, 2015.

[11] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The international journal of aviation psychology*, 3(3):203–220, 1993.

[12] M. E. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, pages 9–17. IEEE, 2012.

[13] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, APGV '04, pages 39–47, New York, NY, USA, 2004. ACM.

[14] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pages 141–148, March 2003.

[15] G. Papadakis, K. Mania, and E. Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 581–584. ACM, 2011.

[16] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behavior of virtual reality systems. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pages 47–50. ACM, 2016.

[17] A. Steed. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pages 123–129, New York, NY, USA, 2008. ACM.

[18] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and S. I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pages 43–50. IEEE, 2009.

[19] T. Waltemate, I. Senna, F. Hülsmann, M. Rohde, S. Kopp, M. Ernst, and M. Botsch. The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pages 27–35. ACM, 2016.

Copyright

Author Contributions

The author wrote the software, conducted the measurements and analysis and took the lead in writing the manuscript. He provided critical feedback and helped shape the research, the analysis and the manuscript.

6.2 Conclusion

A method to simulate latency is necessary to be able to provide different latency conditions in experiments. We described an approach how to simulate not only time-invariant latency but also latency jitter, which opens up the possibility for more insight of the effects of latency on users of VR applications.

Discussed Research Questions

$R_{2.1}$ **How to simulate latency?** We show an approach how to simulate time variant latency. Latency behaviour is split into multiple intervals depending on their duration. Each part is modeled with a probability distribution describing the distribution of observed duration and another probability distribution describing the interarrival times between latencies of this duration. Latency simulation for tracking data is conducted by queuing tracking data with a delay depending on the probability distributions. Taking from the queue when the associated delay is over yields the tracking data with the simulated latency added. A schematic image is shown in the next chapter when simulated latency is used to study latency's effects.

Chapter 7

EFFECTS

We already mentioned negative effects of latency and latency jitter on users of VR applications without a detailed look at it. Cybersickness is one of the big problems that prevent virtual reality adaption. People getting sick as a result of consuming virtual reality applications will be reluctant to expose themselves again. We focused on latency effects on cybersickness because cybersickness can prevent usage of virtual reality applications. There are other metrics that are important for virtual reality applications such as, among others, performance or body ownership which are also impacted by increased latency [CMG19].

7.1 Latency Jitter provokes Cybersickness

We conducted a study to explore the connection between latency jitter and cybersickness. The study was designed to compare a condition with and without added simulated latency jitter. The latency jitter was detectable but was less disturing as similar effects of positional tracking jitter found in some other experiments conducted at the chair. Participants needed to move and look around because added latency in the tracking data is not experienced if the tracking data doesn't change. We found latency jitter to contribute to cybersickness.

The paper was published as Jan-Philipp Stauffert, F. Niebling, and M. E. Latoschik. "Effects of Latency Jitter on Simulator Sickness in a Search Task". In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2018, pages 121–127

Effects of Latency Jitter on Simulator Sickness in a Search Task

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

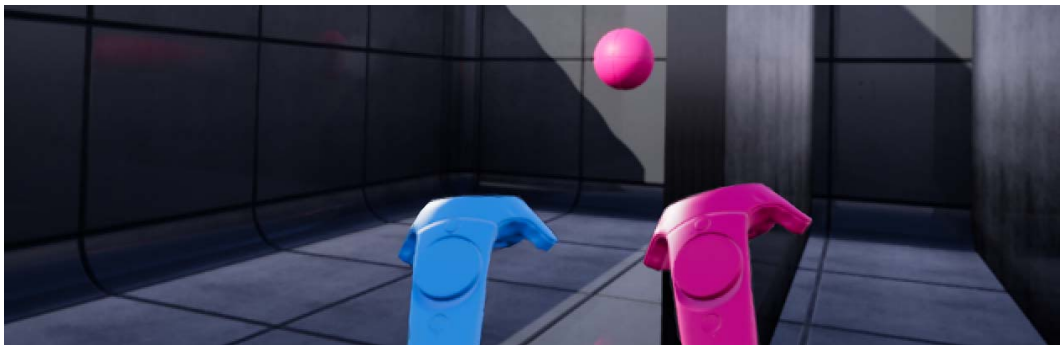


Figure 1: The employed search task. A sphere has to be found that acquires one of two colors if it is in the middle of the field of view, here magenta. Once the sphere acquired a color, the trigger button of the controller with the same color has to be pressed. Afterwards a new sphere has to be found. The task was conducted with and without latency jitter added to the HMD tracking. We found significant differences of the cybersickness with latency jitter present.

ABSTRACT

Low latency is a fundamental requirement for Virtual Reality (VR) systems to reduce the potential risks of cybersickness and to increase effectiveness, efficiency and user experience. In contrast to the effects of uniform latency degradation, the influence of latency jitter on user experience in VR is not well researched, although today's consumer VR systems are vulnerable in this respect. In this work we report on the impact of latency jitter on cybersickness in HMD-based VR environments. Test subjects are given a search task in Virtual Reality, provoking both head rotation and translation. One group experienced artificially added latency jitter in the tracking data of their head-mounted display. The introduced jitter pattern was a replication of a real-world latency behavior extracted and analyzed from an existing example VR-system. The effects of the introduced latency jitter were measured based on self-reports simulator sickness questionnaire (SSQ) and by taking physiological measurements. We found a significant increase in self-reported simulator sickness. We therefore argue that measure and control of latency based on average values taken at a few time intervals is not enough to assure a required timeliness behavior but that latency jitter needs to be considered when designing experiences for Virtual Reality.

Index Terms: D.1.3 [Programming Techniques]: Concurrent Programming—Parallel programming; D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

1 INTRODUCTION

Today's computer systems come with inherent fluctuations of performance. In many applications, especially graphical applications, programmers try to use every bit of performance the computer offers while still expecting that changed data is displayed as fast as possible. Failure to meet these demands results in delayed information which in turn affects user performance and experience. The main measurement to describe the responsitivity of an application is by measuring the end-to-end latency, the time from the user performing an action until this action's results are shown on the display. Between an action and the perception of its effects lie multiple soft- and hardware systems each running at their own pace. Delays introduced anywhere in the system might lead to inconsistencies between action and stimulus presented by the head-mounted display, leading to unpleasant jitter in the VR experience.

There are many factors that influence execution time that are not deterministic or too manifold so they appear non-deterministic. Examples for influencing factors are hardware-based aspects such as interrupts, as well as software-based aspects such as preemption by the operating system or interfering processes or threads. Communication with other devices such as external trackers or other computers in distributed systems is even more prone to fluctuations in execution time.The employment of multi- and manycore systems sacrifices determinism in the order of operations at runtime for increased performance. Collaborating threads and processes constantly compete for resources. While the overall usage of processors may be maximized, in non-hard realtime systems each thread or process is allotted time on a best effort basis. There are no guarantees that a process gets execution time at the time it needs to perform calculations, ultimately leading to missed updates in rendering, or to renderings based on obsolete inputs. Runtime behavior, especially in interactive applications, is therefore difficult to estimate. As well as these influences are included in the mean latency calculation to achieve a preferably stable frame rate, they can still accumulate to create non-deterministic latency spikes.

Most related research describes the effect on task performance and cybersickness of a constant decrease in latency [10]. There are experiments that show decreased task performance in the presence of spatial jitter [23], but few reports on the effects of temporal jitter. While it is shown that periodic jitter patterns provoke sickness [25], the quasi-random behavior of current real-time systems is yet to be

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

2018 IEEE Conference on Virtual Reality and 3D User Interfaces
18-22 March, Reutlingen, Germany
978-1-5386-3365-6/18/$31.00 ©2018 IEEE

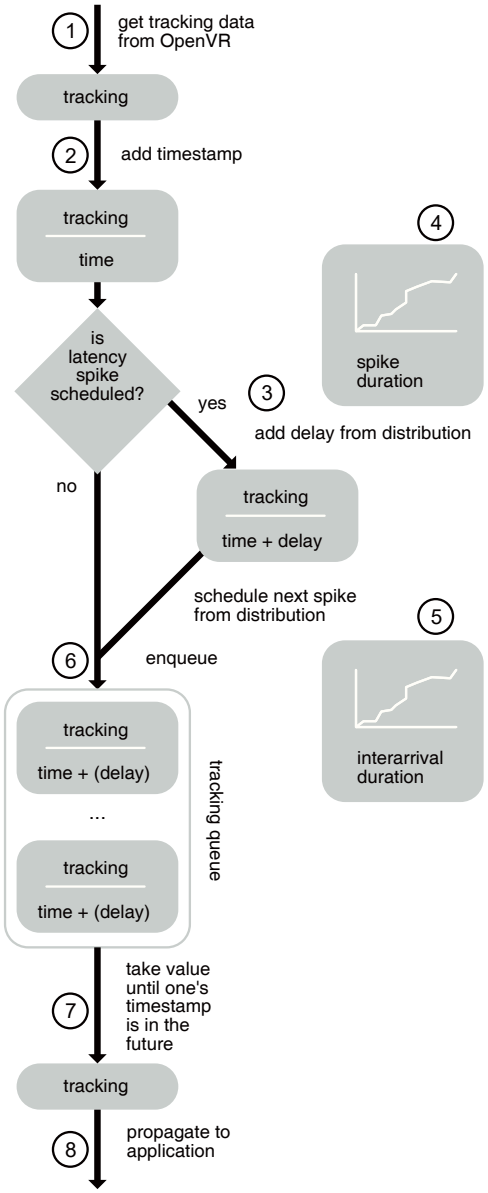


Figure 2: Latency Injection for the experiment. Tracking data is received (1) and annotated with the time of arrival (2). If a latency spike is scheduled, the timestamp is increased (3) by a delay taken from a probability distribution (4). If a spike occurred, a new spike is scheduled whose time is taken from another probability distribution (5). The tracking data is enqueued (6) and immediately dequeued until a timestamp in the future is encountered (7). The tracking data is then forwarded to the application (8).

surveyed. We assume latency jitter to cause similar issues regarding cybersickness.

Our main contributions in this paper are:

- A modification of the Unreal Engine 4 plugin for the HTC Vive to allow the introduction of latency jitter
- A user study evaluating the effect of latency jitter on cybersickness in tasks requiring both continued head movement as well as rotation, finding a significant influence of latency jitter on cybersickness

2 Related Work

Cybersickness is a problem of VR applications where users are experiencing symptoms such as nausea [11]. While some users are more sensitive, there are certain factors that make cybersickness worse for most users. Visual delay was found as a major contributing factor already in early simulators [6]. Latency also influences the performance of test subjects both if time variant latency is added [10] and if latency spikes occur [17, 23]. The assumption is consequently that latency spikes influence cybersickness with a similar impact as the better researched time invariant latency.

Latency has been injected into virtual environments to be able to repeatedly evaluate effects on task performance, presence, and other factors, in controlled experiments. Typically, latency studies delay tracker input data by a controllable amount of time units — frames or multiples of the tracker sampling rate — by employing a ring buffer or other FIFO data structures either inside the tracker itself, its software driver, or the VR application. Experiments are then performed with different, yet most often constant per experiment, amounts of latency artificially injected into the system.

Ellis et. al. tested distinguishability of changes in latency for hand [5] as well as for head [4] movements. They employ custom tracker drivers to ensure a low base latency and to provide the ability to add custom latency to their input devices. Building on this work, Mania et. al. test sensitivity to head tracking latency in virtual environments [14]. Meehan et. al. studied the effects of latency on presence in stressful virtual environments [15]. To enable user studies with different latency settings, they adapted their VRPN client implementation to delay tracker input data by a fixed amount of time to add constant end-to-end latency to their system, enabling controlled experiments with 50ms and 90ms of latency respectively. Other studies that control latency, e. g. performed by Allison et. al. [1], or more recent work on latency control by Papadakis et. al. [16] as well as by Waltemate et. al. [24], also only allow for the insertion of constant latency by delaying tracker input data using ring buffers.

Time invariant latency, however, ignores that latency in applications typically is not constant, but changing over time. This typically happens due to inhomogeneous scene complexity as well as due to different computational complexity of the simulated world in VR given different viewpoints or points in time. Previous work on time varying latency jitter focuses mostly on periodic changes [19, 26]. Periodic patterns are the result of different work cycles in cooperating systems. Here, pattern frequency is shown to play a more important part than amplitude regarding subjective sickness [13]. While periodic latency changes results in time invariant latency, we focus on non-periodic, quasi-random latency behavior. Stauffert et. al. introduced a model for non-periodic latency jitter that allows for injection of latency and latency spikes into VR applications to replicate different latency scenarios [21].

The performance of VR applications is usually assessed by measuring motion-to-photon latency which tracks the time between an input on a certain input channel and the time it takes to show its effect on a display. As with injecting latency, the measurements are used to measure a time-invariant latency. Approaches to measure

this latency are sine fitting [22], light sensing [3], and automated frame counting [7].

3 Latency Jitter Injection

We are conducting an experiment to assess the impact of latency jitter on cybersickness. To provide a controlled environment for latency studies, we first describe our method to inject latency jitter into a VR system. Subsequently, we present the user study evaluating task performance under latency scenarios, and discuss our results.

To inject latency jitter, we modify the HTC Vive plugin of the Unreal Engine 4. The built-in plugin is called every frame to update position and orientation of controllers and HMD from tracking data. The plugin queries the OpenVR interface to the device for the current values, making them accessible to the VR application. Instead of directly propagating the received values, we delay these data according to a probability distribution to inject latency jitter. See Figure 2 for a diagram describing the procedure .

After receiving the current tracking data from OpenVR (1), we annotate this acquired sample with the current time (2) . The current time is then compared to the time at which the next latency spike is scheduled to occur (3). If this time is exceeded, a spike duration is drawn from a probability distribution and added to the timestamp (4). The next time a latency spike is to occur is scheduled by drawing a value from the inter-arrival probability distribution and adding it to the current time (5). Independent whether a latency spike has occurred or not, the sample is enqueued into a buffer (6). After the last received value is added to its respective buffer, the buffer is drained until there are no more values in the buffer or a value is encountered whose availability-timestamp is in the future (7). The value taken last gets propagated to the application.

With this mechanism, one value that was received with a scheduled latency spike can block the buffer for the duration of the spike even though there would be samples from the tracker arriving afterwards that are not directly delayed by a scheduled spike.

Latency jitter can have many sources. Among them are unreliable execution times of code due to optimizations such as caching, hyperthreading and multithreading. Additionally there is other software running concurrently, needing CPU time. The plethora of influencing factors makes code execution time unpredictable. While it doesn't exhibit recurring patterns, a model representing jitter distribution and duration is needed for the simulation of latency jitter [21].

To obtain a realistic jitter profile, we base the injected latency on measurements with the VR middleware OSVR which are then scaled to resemble a more pathogenic system. The resulting latency jitter was tested by three experts who found it to realistically resemble under-performing VR systems. The assessment was a subjective feedback based on multiple years of experience with VR systems.

For the base latency distribution, we measured the time that tracking data needs to propagate through the VR middleware OSVR. With a custom device sending and a custom client receiving data, we measure the time in between the sending and receiving.

The result is a characteristic pattern where most values gather around a mean value with only a minuscule population deviating from this mean [20]. Since we only want to observe the effect of latency jitter, we use the outliers for the simulation. There, we restrict the injection to outliers of order two and above, as can be seen in Figure 3, as they represent the 0.1% of measurements that are often ignored in time-invariant latency experiments.

The chosen latency jitter pattern is noticeable for a user who has used the employed VR setup before. The system repeatedly exhibits multiple frame drops in short succession combined with times of lesser impact of latency spikes. The effect is comparable to a computer operating at the edge of its capacity or a VR system whose tracking experiences interference.

Since the scene used in the experiment has no moving parts other than user interaction, it is not discernible whether only the HMD tracking sometimes lags or if the whole VR simulation is halted.

4 Experimental Design

We conduct an experiment to survey the relation of latency jitter to cybersickness using the introduced latency jitter injection method.

The task between groups only varies in the condition if latency jitter is present or not. The elimination of other factors allows us to focus on the connection between latency jitter and cybersickness.

4.1 Hypotheses

As discussed before, latency jitter was regarded in respect to performance in virtual environments but not yet in respect to cybersickness. Cybersickness is a major factor that reduces virtual reality acceptance. Hence, we employ different indicators of cybersickness for our experiment to get a more holistic picture.

For our experiment we pose our hypotheses as follows:

H1: Latency jitter evokes cybersickness

H2: Latency jitter influences physiological measures

The evaluation of H1 is based on the established simulator sickness questionnaire [11]. On a smaller note, we ask participants during the experiment about their well-being. Since the simulator sickness questionnaire is the often employed way to measure cybersickness, it is the major tool to be able to detect if latency jitter can cause cybersickness. The mid experiment questions are less specific but may help to find further evidence of cybersickness.

H2 asks if effects can be measured from body responses. This question is based on research of Kim et. al. [12] and Meehan et. al. [15] who found a significant connection between, among others, heart rate and cybersickness. The experiment in Kim et. al. involved a driving task which is more prone to cybersickness than our method of movement which allows the user to walk freely in the tracked space. Meehan et. al. use the fear of height as a stress inducing factor. While our physiological responses are expected to be smaller, a significant find would justify to establish physiological measures as a further means to determine cybersickness along the simulator sickness questionnaire for latency experiments.

5 Method

5.1 Participants

46 participants were recruited for this study. One had to be excluded due to technical difficulties during the experiment. The final sample consisted of 45 subjects, 36 female, 9 male. The ages ranged from 18 to 31 with a mean of 21.18 and a standard deviation of 2.58. 22 had usable heart-rate data and 28 had usable skin-conductance data. The low yields were a consequence of the constant movement during the experiment impacting the measuring device. All participants gave written informed consent and got course credits for their participation.

5.1.1 Procedure

Participants first filled out a questionnaire containing demographic questions, the motion sickness susceptibility questionnaire [8], the games motivation questionnaire [18] and the simulator sickness questionnaire [11]. They then were equipped with an Empatica E4 wristwatch measuring physiological data (galvanic skin response, heart rate) and the HTC Vive HMD with controllers to proceed with the experiment. The experiment started with five minutes of standing in the virtual scene without movement to get a baseline for the physiological measurements.After the five minute acclimatization phase, the subjects continued with a search task for 9.5 minutes. After three, six and nine minutes of the search task phase, they were asked how much fun they had on a scale from 1 to 5, how immersed they felt on a scale from 1 to 5 based on [2], and how sick they felt on

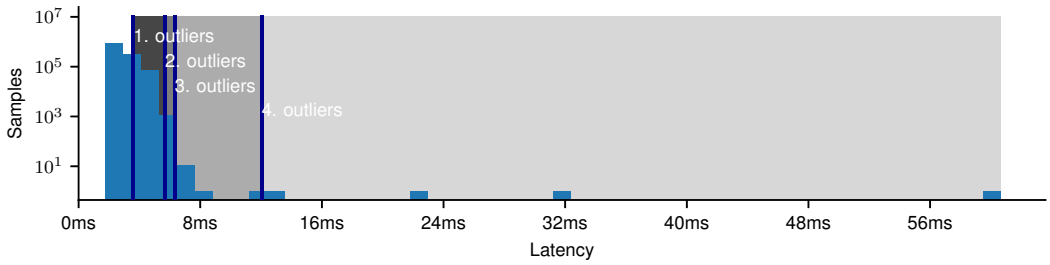


Figure 3: Histogram of the latency durations used to inject latency jitter with outlier orders adapted from [21]. Most latency values(92.72%) gather around a mean with another 7.2695% just above. The remaining 0.0105% contain the outliers that are often ignored.

Table 1: Tabular overview of the jitter data used for the jitter simulation. For the latency injection, we use the data for the outlier orders of two and above which represent less than 0.1% of the measurements.

Outlier Order	Percentage of Samples >=	Min	Max	Mean	Standard deviation
	100.0000%	1.76ms	3.61ms	2.82 ms	0.21ms
1	7.2800%	3.61ms	5.70ms	4.46ms	0.35ms
2	0.0105%	5.70ms	6.22ms	5.84ms	0.14ms
3	0.0020%	6.33ms	7.71ms	6.79ms	0.40ms
4	0.0004%	12.04ms	60.65ms	28.02ms	17.85ms

a scale from 0 to 4. The experiment ended with a one minute phase without movement to again measure physiological data. Afterwards, the simulator sickness questionnaire was filled out a second time.

The five minutes before the search task and the one minute after was chosen to be able to gather physiological data. The data during the search task is unreliable as movement may disturb the measuring process. The exposure time was chosen to resemble Kim et. al.'s procedure [12] for comparability.

5.1.2 Task

The search task was designed as to find spheres appearing in a virtual scene. The users were instructed to focus their viewing direction on the position of a sphere and press a button for the next sphere to appear. The spheres appear at random in one of four corridors which cannot be surveyed completely from a static position. Hence, movement is required to find the next sphere. The movement method was walking in the physical environment as tracked by the HMD position. This is the most natural way to express movement.

The scene was a room with an open ceiling. In front of the user were three walls, forming four corridors - two between the walls and one on each side. Confer Figure 5 for an overview of the setting. Spheres could spawn between the walls, as well as on either side. Only one sphere was present in the scene at the same time. The position was chosen at random for each sphere. The walls obstructed the view provoking a left and right movement to find a sphere.

Once a sphere was found, it had to be in the middle of the user's field of view to acquire one of two possible highlighting colors, magenta or blue. The test if the sphere was looked at was performed by an intersection test with a cone originating in the users HMD, forcing the users to focus on the sphere.

If the looked-at sphere was highlighted in either magenta or blue by user focus, the user needed to press the trigger button on the respective controller. For this, the left and right controllers were colored in blue and magenta, respectively. The controller's colors stayed the same for the whole experiment duration.

If the user pressed the trigger button of the controller with the same color as the sphere, a success message was presented. If they used the other controller's trigger button, a failure message was presented. Either way, a new sphere appeared somewhere else in the scene. The amount of right and wrong detections as well as a timer was shown above the walls.

The task as well as the feedback was chosen to suggest a competitive environment and to distract from the real intention of provoking head rotation and translation. The random position of each sphere forced a constant movement from left to right and back, spanning the whole three meters of tracked area. Users were instructed to not move forward or backward, to keep their overview of the scene comparable to that of the other participants.

Users were divided into two groups of participants. One group experienced added latency jitter on the HMD tracking, while the other did not. The latency jitter was only added in the 9.5 minutes of the search task. Both the acclimatization phase as well as the minute afterwards was free from intervention. Due to the test subjects not moving with a reduced amount of head movement in the absence of the search task, they were supposed to not detect the different conditions between the phases.

5.2 Apparatus

The experiment was conducted on a computer with an Intel i7-6700K processor and a NVidia GTX 1080.

The physiological measurements were done with an Empatica E4 wristwatch. The watch communicated with the computer via Bluetooth. The data was provided to the VR application by the Empatica Bluetooth server.

Using frame counting, we determine our system's base motion to photon latency to be 35.67ms. For this the controller and the monitor were filmed with a high speed camera at 240Hz. In the resulting video, the frame difference between the movement of the real controller and the virtual controller was counted to receive a motion to photon latency between the controller and the monitor. Multiple measurements were taken with the mean of 46.67ms. This number was adjusted by the difference in reactivity of the HMD

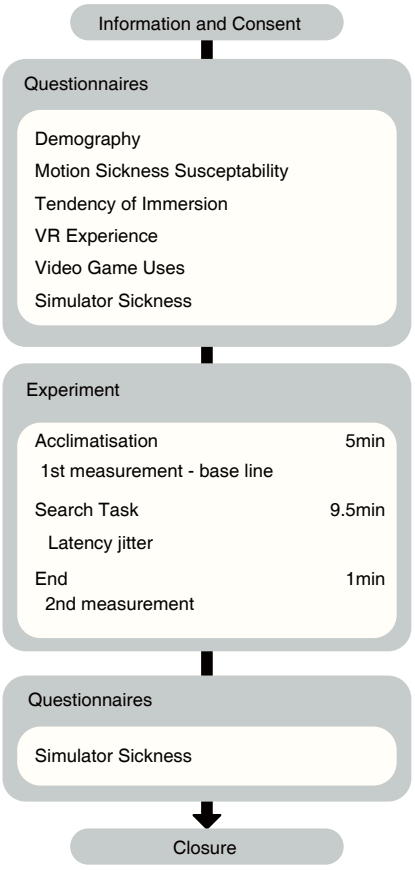


Figure 4: Illustration of the experiment procedure. Participants filled out the simulator sickness questionnaire before and after the experiment. Before and after the experiment, there is a time of standing in VR without performing any tasks, to gather physiological measurements.

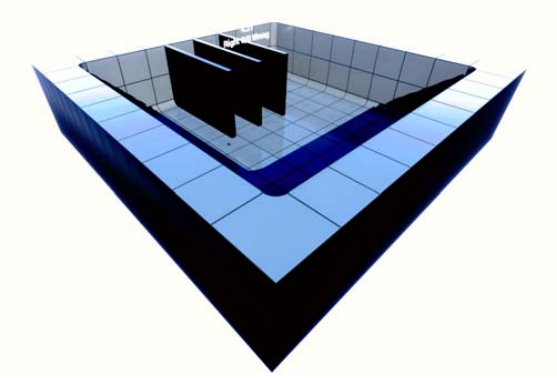


Figure 5: Illustration of the virtual scene. One sphere appears at a time in between the walls. The walls block the sight to force the user to move left and right to spot the spheres. A counter with time and successful trials distracts from the purpose of the study by suggesting a competitive environment. The walkable area is smaller than the virtual room to avoid walking through walls. The colors are adjusted for better visibility.

screen to the monitor screen again gathered with frame counting. There, colors were shown on both the monitor and the HMD and recorded with a high speed camera at 1000Hz. The frame difference between the monitor and the HMD resulted in 11.0ms, resulting in the aforementioned 35.67ms delay from motion to HMD photon latency. The detour via the monitor was necessary as it is hard and error prone to track the movement of the virtual controller on the HMD screen with a camera. The difference in frames per second of the camera video between the two measurements was chosen because for the first take, the video needed spatial resolution to determine the movement. The second video needed only to compare the time between colors shown on the displays, requiring less spatial resolution and allowing the camera to switch to a mode with increased temporal resolution.

6 LIMITATIONS

Even though we base our jitter profile on actual measurements, it is yet unclear if the scaling of values creates realistic conditions. Consulted experts report the experience concerning responsiveness to being close to experiences with unmodified, uncontrolled VR systems under heavy load.

Using the Empatica E4 wristwatch allows to gather physiological measurements. However, it can't be as accurate as arrays of electrodes used in other experiments. The gathering of measurements is influenced by movement. We tried to limit movement in the phases before and after the experiment but were not able to eliminate them.

7 RESULTS

Close to half of the test subjects ($n = 10$) in the latency jitter condition didn't notice the repeated lag in tracking of the HMD. Those who noticed reported it to be very obvious and annoying.

Each scale was analyzed by separately applying a mixed-design analysis of variance (split-plot ANOVA) with the between factor *latency jitter condition*. Generalized $\eta^2(\eta_g^2)$ is reported as a measure of effect size.

Four subjects were taken out of the analysis for their high up front sickness score. The simulator sickness questionnaire total score shows significant results for the comparison of both groups ($F_{1,39} = 4.44$, $p < 0.041$, $\eta_g^2 = 0.102$). The symptom clusters nausea ($F_{1,39} = 0.79$, $p < 0.38$, $\eta_g^2 = 0.020$) and oculomotor ($F_{1,39} = 1.02$,

Table 2: Values for the SSQ and subscales before and after the experiment. NL and L are for the condition without latency added and with latency jitter added.

		Pre		Post		Delta	
		Mean	SD	Mean	SD	Mean	SD
SSQ	NL	9.69	6.68	7.30	6.95	-2.38	6.10
	L	7.17	4.56	9.33	7.68	2.16	7.59
SSQ O	NL	25.01	15.18	17.43	14.15	-7.58	13.24
	L	19.49	10.33	16.96	16.93	-2.53	18.31
SSQ D	NL	16.70	16.03	13.22	15.95	-3.48	16.82
	L	11.27	12.15	20.55	17.94	9.28	18.33
SSQ N	NL	18.13	18.55	13.83	17.91	-4.29	12.94
	L	14.99	13.35	14.99	15.84	0.00	17.59

$p < 0.32$, $\eta_g^2 = 0.025$) did not test significant while the disorientation cluster ($F_{1,39} = 5.38$, $p < 0.026$, $\eta_g^2 = 0.121$) did.

Mid experiment questions didn't test significant for fun ($F_{1,39} = 0.21$, $p < 0.21$, $\eta_g^2 = 0.001$), immersion ($F_{1,39} = 0.90$, $p < 0.39$, $\eta_g^2 = 0.005$) and sickness ($F_{1,39} = 0.00$, $p < 0.99$, $\eta_g^2 < 0.0001$).

We analyzed the physiological data analogous to Meehan et. al. [15]. They take the delta between the mean value of the baseline and the mean value of the stressful environment. For our analysis, we discard the first minute of the acclimatization phase and the first minute of the experiment. The omission of the two minutes is to reduce the effect of the transition from the real world to the virtual world and from the relaxed phase to the active search task phase in the measurement data. The mean values are compared with a t-test between the groups with Cohen's d as an indicator of effect size. The skin-conductance values are drift corrected with a regression line derived from the base line in the first five minutes.

With the four subjects taken out due to high sickness scores in the beginning, there are 22 usable measures for heart rate (11 without latency added, 11 with latency jitter) and 28 usable measures for skin conductance (11 without latency added, 17 with latency jitter). Heart-rate did test significant ($p < .037$, $d = 0.95$) while skin-conductance ($p < 0.39$, $d = 0.34$) did not.

8 Discussion

With the significant result of the simulator sickness questionnaire comparing pre- and post-conditions between the groups with and without latency jitter, we accept hypothesis H1. Looking at the sub-scales, we find latency jitter affecting disorientation, while there is no significant result on the nausea and oculomotor sub-scales. The mid-experiment questions how sick the participants feel did not find a difference between conditions. As this question points in nausea direction, it affirms the not found significance of the simulator sickness questionnaire nausea sub-scale.

The significant finding in the simulator sickness total score forms the basis to show that latency jitter is a problem that cannot be overlooked. In addition to the base latency that is known to have implications, repeated spikes in system latency causes uneasiness even though they are only of short duration.

We tried to avoid techniques known to provoke more cybersickness such as certain travel techniques (driving, joystick movement) or other-directed camera movement. Other than the base cybersickness that VR is known to produce, we tried to have latency jitter as the only source of cybersickness. We assume that latency jitter can interact with other cybersickness inducing causes, though we leave this as future research.

We did find a significant correlation between heart rate and cybersickness. This is in line with the research of Kim et. al. [12] and Meehan et. al. [15]. Skin conductance, however wasn't conclusive. The result has to be evaluated under the small sample size. The constant movement during the experiment rendered a lot of measurement unusable. A lot of subjects didn't have any usable physiological data. The usable heart rate measurements for subjects included in the analysis were few (mean $= 97.05$, $\sigma = 57.51$ during the search-task).

It is notable how many test subjects didn't notice the repeated delays in head tracking. Pre-studies with colleagues showed that the introduced latency jitter was obvious. We assume that experience with virtual reality applications plays an important role. Although most test subjects already experienced VR before, many did so only once or twice with an exposure time of roughly an hour. Among those, some experienced systems that had a worse performance than our experiment in the latency jitter condition. They presumably see HMD-based VR systems as inherently affected by latency jitter. It is unclear if an adaptation to jitter happened, even though the random nature of the latency jitter is expected to prevent adaption. The simplicity of the task may have helped to adapt better.

Our sample contains more women than men. Graeber et. al. [9] finds that there is no gender difference with respect to cybersickness as other research has assumed. The susceptibility of cybersickness differs between individuals. Therefore, we cannot argue how much the employed latency jitter affects a person but only that there is an effect.

Qualitative discussions with test subjects after the experiment as well as the discussion above suggests that latency jitter is experienced in different ways depending on prior VR experience. Depending on the success of virtual reality as a technology to reach a wider user base, latency jitter needs to be reevaluated with users more accustomed to VR.

The task between groups only varied in the condition if latency jitter is present or not. The observation of this is that there is a connection between cybersickness and latency jitter. It will be interesting to see how latency jitter compares and interacts with other cybersickness inducing factors especially time invariant latency in future work.

9 Conclusion

This paper investigated the effect of latency jitter on cybersickness. We developed a latency extraction and injection system which allows to measure time-variant latency patterns in existing VR applications and to later inject these patterns in arbitrary target applications. This system was used to develop and execute a novel experiment to evaluate the effect of latency jitter on cybersickness in HMD-based VR. To our best knowledge, this is the first approach to assess the effect of latency jitter in VR-systems.

The implementation of the latency jitter injector for the Unreal Engine 4 allows to perform further experiments with different patterns of latency jitter and to also modulate overall latency values. The described method is easy to adapt for alternative implementations and target systems and provides a general method for further research on latency jitter and its effects.

Conducting a user study with a search task, we found a significant difference between the group with and without latency jitter added. The task was designed to provoke constant head rotations and translations. We conclude that not only time invariant latency, but also latency spikes cause unwanted implications, provoking cybersickness.

With the experimental results documenting a significant impact of latency jitter on cybersickness, we argue that more research needs to be done to understand the implications of latency jitter. Better ways to measure latency jitter need to be found, which can be used to inject latency jitter in more controlled environments to understand the effects and to possibly develop countermeasures. This research is a first step demonstrating the applicability of the developed tools to enable further research on the effects of latency jitter in VR.

REFERENCES

[1] R. S. Allison, L. R. Harris, M. Jenkin, U. Jasiobedzka, and J. E. Zacher. Tolerance of temporal delay in virtual environments. In *Proceedings of the Virtual Reality 2001 Conference (VR'01)*, VR '01, pp. 247–. IEEE Computer Society, Washington, DC, USA, 2001.

[2] S. Bouchard, J. St-Jacques, G. Robillard, and P. Renaud. Anxiety increases the feeling of presence in virtual reality. *Presence: Teleoperators and Virtual Environments*, 17(4):376–391, 2008.

[3] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence*, 19(6):569–584, 2010.

[4] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes in latency during head movement. In *Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Communication, Cooperation, and Application Design-Volume 2 - Volume 2*, pp. 1129–1133. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1999.

[5] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 43, pp. 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999.

[6] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[7] S. Friston and A. Steed. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):616–625, 2014.

[8] J. F. Golding. Motion sickness susceptibility questionnaire revised and its relationship to other forms of sickness. *Brain research bulletin*, 47(5):507–516, 1998.

[9] D. A. Graeber and K. M. Stanney. Gender differences in visually induced motion sickness. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 46, pp. 2109–2113. SAGE Publications Sage CA: Los Angeles, CA, 2002.

[10] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pp. 135–144. ACM Press, 2015. doi: 10.1145/2702123.2702432

[11] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The international journal of aviation psychology*, 3(3):203–220, 1993.

[12] Y. Y. Kim, H. J. Kim, E. N. Kim, H. D. Ko, and H. T. Kim. Characteristic changes in the physiological components of cybersickness. *Psychophysiology*, 0(0):050826083901001–???, Aug. 2005. doi: 10.1111/j.1469-8986.2005.00349.x

[13] A. Kinsella, R. Mattfeld, E. Muth, and A. Hoover. Frequency, not amplitude, of latency affects subjective sickness in a head-mounted display. *Aerospace medicine and human performance*, 87(7):604–609, 2016.

[14] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, APGV '04, pp. 39–47. ACM, New York, NY, USA, 2004. doi: 10.1145/1012551.1012559

[15] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pp. 141–148, March 2003. doi: 10.1109/VR.2003.1191132

[16] G. Papadakis, K. Mania, and E. Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pp. 581–584. ACM, 2011.

[17] K. S. Park and R. V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In *Virtual Reality, 1999. Proceedings., IEEE*, pp. 104–111. IEEE, 1999.

[18] J. L. Sherry, K. Lucas, B. S. Greenberg, and K. Lachlan. Video game uses and gratifications as predictors of use and game preference. *Playing video games: Motives, responses, and consequences*, 24(1):213–224, 2006.

[19] M. E. St. Pierre, S. Banerjee, A. W. Hoover, and E. R. Muth. The effects of 0.2hz varying latency with 20100ms varying amplitude on simulator sickness in a helmet mounted display. *Displays*, 36:1–8, Jan. 2015. doi: 10.1016/j.displa.2014.10.005

[20] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems. In *Virtual Reality (VR), 2016 IEEE*, pp. 287–288. IEEE, 2016.

[21] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behavior of virtual reality systems. In *Proceedings of the 22Nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, pp. 47–50. ACM, New York, NY, USA, 2016. doi: 10.1145/2993369.2993402

[22] A. Steed. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pp. 123–129. ACM, New York, NY, USA, 2008. doi: 10.1145/1450579.1450606

[23] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and S. I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pp. 43–50. IEEE, 2009.

[24] T. Waltemate, I. Senna, F. Hülsmann, M. Rohde, S. Kopp, M. Ernst, and M. Botsch. The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pp. 27–35. ACM, 2016.

[25] M. L. Wilson. *The effect of varying latency in a Head-Mounted Display on task performance and motion sickness*. PhD thesis, Clemson University, 2016.

[26] W. Wu, Y. Dong, and A. Hoover. Measuring Digital System Latency from Sensing to Actuation at Continuous 1-ms Resolution. *Presence: Teleoperators and Virtual Environments*, 22(1):20–35, Feb. 2013. doi: 10.1162/PRES_a_00131

Copyright

Author Contributions

The author wrote the software, conducted the experiment and analysis and took the lead in writing the manuscript. He provided critical feedback and helped shape the research, the analysis and the manuscript.

7.2 Latency and Cybersickness

Other researchers have looked at the connection between latency and cybersickness even though, they did not take latency jitter into consideration. We submitted a literature review to the Frontiers special issue on "Cybersickness in Virtual Reality Versus Augmented Reality" that looks at the connection of latency and cybersickness. There is a good part describing different approaches to measure latency and to understand latency in this paper, which was partly already discussed in previous chapters of this thesis.

The article was published as Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Latency and Cybersickness: Impact, Causes, and Measures. A Review". In: *Frontiers in Virtual Reality* 1 (2020), page 31. issn: 2673-4192. doi: `10.3389/frvir.2020.582204`. url: `https://www.frontiersin.org/article/10.3389/frvir.2020.582204`

frontiers in Virtual Reality

REVIEW
published: 26 November 2020
doi: 10.3389/frvir.2020.582204

Latency and Cybersickness: Impact, Causes, and Measures. A Review

Jan-Philipp Stauffert, Florian Niebling and Marc Erich Latoschik*

Human-Computer Interaction (HCI) Group, Informatik, University of Würzburg, Würzburg, Germany

Latency is a key characteristic inherent to any computer system. Motion-to-Photon (MTP) latency describes the time between the movement of a tracked object and its corresponding movement rendered and depicted by computer-generated images on a graphical output screen. High MTP latency can cause a loss of performance in interactive graphics applications and, even worse, can provoke cybersickness in Virtual Reality (VR) applications. Here, cybersickness can degrade VR experiences or may render the experiences completely unusable. It can confound research findings of an otherwise sound experiment. Latency as a contributing factor to cybersickness needs to be properly understood. Its effects need to be analyzed, its sources need to be identified, good measurement methods need to be developed, and proper counter measures need to be developed in order to reduce potentially harmful impacts of latency on the usability and safety of VR systems. Research shows that latency can exhibit intricate timing patterns with various spiking and periodic behavior. These timing behaviors may vary, yet most are found to provoke cybersickness. Overall, latency can differ drastically between different systems interfering with generalization of measurement results. This review article describes the causes and effects of latency with regard to cybersickness. We report on different existing approaches to measure and report latency. Hence, the article provides readers with the knowledge to understand and report latency for their own applications, evaluations, and experiments. It should also help to measure, identify, and finally control and counteract latency and hence gain confidence into the soundness of empirical data collected by VR exposures. Low latency increases the usability and safety of VR systems.

Keywords: virtual reality, latency, cybersickness, jitter, simulator sickness

OPEN ACCESS

Edited by:
Kay Marie Stanney,
Design Interactive, United States

Reviewed by:
Pierre Bourdin,
Open University of Catalonia, Spain
Wolfgang Stuerzlinger,
Simon Fraser University, Canada
Juno Kim,
University of New South Wales, Australia

****Correspondence:***
Jan-Philipp Stauffert
jan-philipp.stauffert@uni-wuerzburg.de

Specialty section:
This article was submitted to Virtual Reality and Human Behaviour, a section of the journal Frontiers in Virtual Reality

Received: *10 July 2020*
Accepted: *30 October 2020*
Published: *26 November 2020*

Citation:
Stauffert J-P, Niebling F and Latoschik ME (2020) Latency and Cybersickness: Impact, Causes, and Measures. A Review. Front. Virtual Real. 1:582204. doi: 10.3389/frvir.2020.582204

1. INTRODUCTION

Cybersickness is a severe problem for the usage and safety of VR technology. It hinders both the broader adoption of VR technology and its overall usability. Cybersickness is closely related to simulator sickness and motion sickness. Early research describes cybersickness as a motion sickness in virtual environments (McCauley and Sharkey, 1992). Cybersickness is usually defined by a set of specific adverse symptoms in combination with the use of certain technologies, such as disorientation, apathy, fatigue, dizziness, headache, increased salivation, dry mouth, difficulty focusing, eye strain, vomiting, stomach awareness, pallor, sweating, and postural instability (LaViola, 2000; Stone Ill, 2017; McHugh, 2019). These symptoms are shared with related definitions of sickness, even though their severity might vary. Stanney et al. (1997) argues that cybersickness is connected to more symptoms in the disorientation cluster of the Simulator Sickness Questionnaire

(SSQ) (Kennedy et al., 1993) than simulator sickness. The disorientation cluster contains several symptoms which do not all carry the explicit meaning of disorientation. Gavgani et al. (2018) show that motion sickness and cybersickness show the same severity of symptoms in extreme cases. Bockelman and Lingum (2017) distinguish cybersickness from other definitions of sickness by its "cyber" source. We use the term cybersickness to describe sickness with the aforementioned symptoms induced by Virtual Reality or Augmented Reality applications that do not apply external forces on the user. External forces are motion platforms or other motor actuated methods that move a user without the user's own effort. These VR or AR applications provide stimuli predominately by visual perception.

Chang et al. (2020) review experiments that measure cybersickness. They describe the frequency of use for different subjective measurements. Out of 76 experiments, 58 ($\approx$ 76%) use the SSQ (Kennedy et al., 1993). Less often used questionnaires are the Fast Motion Sickness scale (FMS, 6 experiments $\approx$ 8%, Keshavarz and Hecht, 2011), a forced-choice question (5 experiments $\approx$ 6.5%, Chen et al., 2011), the Misery Scale (MISC, 4 experiments $\approx$ 5%, Bos et al., 2010), the Motion Sickness Assessment Questionnaire (MSAQ, 3 experiments $\approx$ 4%, Gianaros and Stern, 2010) and the Virtual Environment Performance Assessment Battery (VEPAB, 3 experiments $\approx$ 4%, Lampton et al., 1994). In contrast, Davis et al. (2014) state that the Pensacola Diagnostic Index (Graybiel et al., 1968) is the "most widely used measure in motion sickness studies"(Davis et al., 2014, p. 6). They state that another widely used questionnaire besides the SSQ is the Nausea Profile (Muth et al., 1996) and further list the Virtual Reality Symptom Questionnaire (Ames et al., 2005). Another questionnaire in use is the Motion Sickness Susceptibility Questionnaire (MSSQ) (Golding, 1998). Here again, it becomes apparent how close cybersickness is to simulator sickness and motion sickness, since questionnaires are often used for multiple sickness definitions. The listed questionnaires are in use for research on cybersickness, but care has to be taken to understand their different usage and purpose. Many, like the SSQ, report on the sickness experienced at the time of answering the questionnaire while others like the MSSQ ask for past experiences to gauge sickness susceptibility that can play into the experience. The Nausea Profile is a scale for measuring nausea due to any cause, not a motion sickness-specific scale, while the MSAQ of the same group targets motion sickness and describes subscales for further differentiating motion sickness aspects.

There are different explanations how cybersickness comes into being and there are multiple factors that influence cybersickness. Explanations for cybersickness often preceed the term cybersickness itself. They were created for motion sickness or simulator sickness and then adopted for cybersickness. Rebenitsch and Owen (2016) and LaViola (2000) list and discuss the following theories for cybersickness: the sensory mismatch theory (Reason and Brand, 1975; Oman, 1990), the poison theory (Treisman, 1977), the postural instability theory (Riccio and Stoffregen, 1991) and the rest frame theory (Virre, 1996). Oman (1990) describe their sensory mismatch theory as possibly underlying multiple different sickness definitions such as motion sickness and simulator sickness. Bles et al. (1998) adapt this statement to describe postural stability as underlying multiple different sickness definitions.

Factors that evoke cybersickness are "rendering modes, visual display systems and application design" (Rebenitsch and Owen, 2016, p. 102) as well as hardware-specific factors. Rebenitsch and Owen (2016) describe the former factors but leave hardware-specific factors such as latency open to be discussed in other publications. This review focuses on latency contributions to cybersickness. There are other hardware-specific factors such as tracking accuracy (Chang et al., 2016) that are not covered in this review. Latency describes the processing time incurred by the computer system used for the VR application. VR needs complex hard- and software to deliver the desired experience. Each part in the system contributes to the overall latency by itself and by the effects of its interaction with other parts.

Latency as an inherent property of computer system processing is easily introduced into complex architectures, and as such is subject to many evaluations. There are different angles toward research on latency in virtual environments that mutually influence each other. Effects of latency on cybersickness are found, which necessitate research into measurements and control of latency. Experiments that simulate latency are performed that gather more insight into its effects on cybersickness and user performance. And not least of all, latency in experiments performed in virtual environments needs to be reported in research articles. This review is thus organized as follows: First, we discuss experiments that show that latency contributes to cybersickness. Then, we describe ways to measure latency, which is essential for development of applications with consistent latency behavior. We then show how measured latency is reported in research articles to illustrate latency patterns in experiments.

2. EFFECTS

Table 1 shows an overview of different studies that show that latency contributes to cybersickness. The researchers conducted experiments with latency as the independent variable and cybersickness as the dependent variable. Latency is manipulated to create conditions of different motion to photon latency in the employed systems. For each condition, cybersickness is measured to compare sickness values between the conditions. Researchers measure cybersickness with subjective questionnaires or objective physiological measurements. The most often used questionnaire for the listed papers is the SSQ with six out of 11 papers (Meehan et al., 2003; Moss et al., 2011; St. Pierre et al., 2015; Kinsella et al., 2016; Stauffert et al., 2018; Caserman et al., 2019). Physiological measurements are postural stability or postural sway, heart rate, sweating and galvanic skin response. We list postural stability separate from the other physiological measurements to distinguish the different cases of usage. Frank et al. (1988) list postural stability separate from other physiological measurements. Kawamura and Kijima (2016) only observe postural stability. Postural instability

often correlates with visually-induced motion sickness (Riccio and Stoffregen, 1991) and some studies have found it to be predictive of visually-induced motion sickness (Arcioni et al., 2019). Meehan et al. (2003) and Stauffert et al. (2018) only use heart rate and galvanic skin response. Their physiological measurements showed an effect of increased latency on heart rate. Gavgani et al. (2018) argue that forehead sweating is the best physiological indicator for motion sickness which shows the same symptoms as cybersickness in extreme cases. Their rollercoaster experiment only finds minor or moderate effects for heart rate and galvanic skin response. While heart rate may not be the best indicator of latency induced cybersickness, it supports the research that evaluates cybersickness with the SSQ.

Most research for latency and cybersickness tests only the effect of static latency added (Frank et al., 1988; DiZio and Lackner, 2000; Meehan et al., 2003; Moss et al., 2011; Kawamura and Kijima, 2016; Caserman et al., 2019; Palmisano et al., 2019; Kim et al., 2020). They introduce a fixed delay into the system and test different such latencies against each other. This is based on the assumption that most observed latencies are close to one mean latency, for which one fixed added latency per condition is an approximation. This simple latency model consistently shows an increase of latency in the VR simulation, leading to increased cybersickness or a more disturbed stand equilibrium.

Movement itself is important to experience latency induced cybersickness (DiZio and Lackner, 2000). Although, Moss et al. (2011) found no influence of latency in an experiment with a lot of head movement. They report that the head movement itself evoked sickness. It may be that sickness from other sources was stronger than the latency induced sickness thereby masking it. Without movement, the user might not feel the discrepancy between real world and virtual world widened by latency. An increase of head movement can increase cybersickness (Palmisano et al., 2019; Kim et al., 2020). Studies often involve a search task that requires head movement.

Taking into account that latency in measurements often shows irregular spikes, Stauffert et al. (2018) showed that not only uniform but occasional latency spikes provoke cybersickness. St. Pierre et al. (2015) and Kinsella et al. (2016) show that periodic latency like measured by Wu et al. (2013) contributes to cybersickness. They describe latency as consisting of a time-invariant and a periodic part. Periodic latency is described to follow a sine wave. St. Pierre et al. (2015) argues that the sine's amplitude has more influence than its frequency. Kinsella et al. (2016) observes the opposite.

3. MEASURING LATENCY

The contribution of latency to cybersickness necessitates controlling latency in every VR or AR application. High latency and especially latency spikes can often only be detected by measurements, which in turn provide researchers and other developers with indications if and where interventions are needed during the development process. Approaches to measure latency are numerous and distinguish themselves in the amount of instrumentation they need, and which kind of latency they measure. Most approaches measure motion to photon latency, which is the time between a movement of some tracked object, and the effect corresponding to this movement shown on a screen, conveyed by photons emitted from the screen. Different tracked objects can be used to signify movement in the measurement of motion to photon latency, such as Motion Controllers or Head-Mounted Displays (HMD). The employed screens may be computer monitors, mobile phone screens or AR/VR HMD screens. The motion to photon latency is also called end-to-end latency. **Table 2** shows an overview of approaches.

Measurements need to compare the time difference between the motion of a tracked object and a resulting response on a screen. The observed motion can be the onset of a motion (Feldstein and Ellis, 2020), special characteristics during a motion such as the peak of acceleration (Friston and Steed, 2014), the end of a motion (Chang et al., 2016) or arrival at a predetermined position (He et al., 2000) or a predetermined motion (Di Luca, 2010). A predetermined motion is usually a sinusoidal movement of a pendulum (Steed, 2008) or the circular movement of a turntable (Swindells et al., 2000). A motion can also be the passing of time in the form of timestamps (Sielhorst et al., 2007; Billeter et al., 2016; Gruen et al., 2020).

The screen shows either a copy of the motion (Roberts et al., 2009) or an encoded version of it (Becher et al., 2018). The system needs to track the tracked object, integrate it into its simulation and show a generated image on the screen (Mine, 1993; Feldstein and Ellis, 2020). The necessary processing time leads to the image on the screen always being delayed in contrast to the original, real motion. Additional steps such as Remote Graphics Rendering (Kämäräinen et al., 2017), or using additional computers to process tracking information, leads to increased latency (Roberts et al., 2009).

A straight forward approach uses a camera to observe both the real and the virtual motion and compare the delay between their chosen motion aspect. The analysis can be done by hand (Liang et al., 1991) or automated (Friston and Steed, 2014). Tracking cameras trade spatial resolution for temporal resolution. High spatial resolution is needed to better capture the real motion, but high temporal resolution is needed to determine a high precision latency value. A way around the dilemma is to fit the mathematical function of the known movement to the tracking data (Steed, 2008). This reduces uncertainty due to restricted resolution.

Camera based measurements do not work well with HMDs, because the lenses distort the image in a way that necessitates them to be very close to the lens. This way, they cannot record the real tracked object any longer. These approaches usually use a computer monitor as the observed screen. Some researchers remove the HMD lenses (Feldstein and Ellis, 2020) or use additional lenses that reverse the distortion (Becher et al., 2018).

An alternative is to observe the real motion separately from its virtual counterpart. The obvious extension is to

TABLE 1 | Research simulating latency that tested for a connection to cybersickness.

	System	Task	Measure	Latency shape	Conditions	Result	*n*
Frank et al., 1988	Driving simulator	Driving	Rod and frame, physio, postural stability	Uniform	Added 0, 170, 340 ms transport delay	Evokes sickness visual delay more important than motion delay	54 (27f 27m)
Stauffert et al., 2018	HMD Vive	Search	SSQ, physio	Jitter	Added no latency, Added latency jitter	Jitter provokes sickness	45 (36f 9m)
Kawamura and Kijima, 2016	HMD DK2	Keep balance	Pressure plate	Uniform	Absolute 1, 26, 39, 53, 66 ms	Latency disturbs human stand equilibrium	17
Caserman et al., 2019	HMD Vive Pro full bodytracking	Search	SSQ	Uniform	Absolute 0, 50, 54, 58, 63, 69, 75, 83, 92, 104, 121, 150 ms	More latency More cybersickness	21 (6f 15m)
Moss et al., 2011	HMD No HMD	Search	SSQ	Uniform	Added 0, 200 ms Added 0, 145, 300 ms	Latency unclear connection to Simulator sickness; exposure time and active head movements Evoke simulator sickness	22 (11f 11m) 29 (12f 17m)
Kinsella et al., 2016	HMD	Search	SSQ	Periodic	2 × 2 design: Added frequency 0.2/1 Hz Amplitude 100/20–100 ms	Latency frequency with Periodic latency scenario Increases sickness 0.2 Hz sickens more	120
St. Pierre et al., 2015	HMD	search	SSQ	Periodic	0, 100 ms, 100 ms 0.2 Hz added 20–100 ms 0.2 Hz	Amplitude increases sickness frequency potentially too Periodic worse than uniform	120 (64f 56m)
DiZio and Lackner, 2000	HMD	Search	Criteria of Graybiel et al., 1968	Uniform	Absolute 67, 159, 254, 355 ms 21, 39, 80, 163 ms	Lag leads to sickness, no sickness without head movement	21 8
Meehan et al., 2003	HMD	Explore Move	SSQ, Physio	Uniform	Absolute 50, 90 ms	More latency, Increased heart rate	164 (32f 132m)
Palmisano et al., 2019	HMD	Rotate head	FMS	Uniform	Absolute 5, 46, 87, 128, 169, 212 ms	More latency, Increased cybersickness	14
Kim et al., 2020	HMD	Rotate Head	FMS	Uniform	Absolute 5, 46, 87, 128, 169, 212 ms	More latency, Increased cybersickness	30

use two synchronized cameras (Kijima and Miyajima, 2016b). More often, the real motion is observed by a photodiode that gets covered (Mine, 1993) or a rotary encoder (Seo et al., 2017) that reports the orientation of the platform that the tracked object is placed on. The screen is monitored by one (Pape et al., 2020) or more photodiodes (Becher et al., 2018; Stauffert et al., 2020a). A photodiode has a high temporal resolution but can only measure one brightness value per measurement. The application to measure needs to display its tracking information in brightness levels to use photodiodes.

The chosen method determines how many latency values are measured. Sine fitting (Steed, 2008; Teather et al., 2009; Zhao et al., 2017) and cross correlation (Di Luca, 2010; Kijima and Miyajima, 2016b; Feng et al., 2019) only report one latency for one measurement run. If the latency between an event and its reaction on the screen is measured, the number of latency measurements that can be reported depends on the approach. Some methods need to provoke an event and then wait for the result, before it is possible to measure again (Liang et al., 1991; He et al., 2000; Swindells et al., 2000; Miller and Bishop, 2002; Roberts et al., 2009; Friston and Steed, 2014; Raaen and Kjellmo, 2015; Kämäräinen et al., 2017; Seo et al., 2017; Feldstein and Ellis, 2020; Pape et al., 2020). Some approaches allow to measure the latency for every frame shown on the screen (Sielhorst et al., 2007; Papadakis et al., 2011; Wu et al., 2013; Billeter et al., 2016; Kijima and Miyajima, 2016b; Becher et al., 2018; Gruen et al., 2020; Stauffert et al., 2020a). Some approaches that only measure the latency of an event are usable to measure continuously, while others are not. We distinguish methods in **Table 2** depending on the reported usage.

4. DESCRIPTION

Looking at the approaches to measure latency, we see that latency is reported in different ways. The reported values are often not comparable, as different papers use different systems

TABLE 2 | Comparison of latency measurement approaches.

	Motion		Photon		
Paper	**Device**	**Capture**	**Device**	**Capture**	**Method**
Becher et al., 2018	HMD	Rotary encoder	HMD	Photodiode	Continuous
Di Luca, 2010	Tracked object	Photodiode	Screen	Photodiode	Cross correlation
Billeter et al., 2016	LED timestamp	Camera	AR HMD	Same camera	Continuous
Feldstein and Ellis, 2020	HMD	Camera	HMD	Same camera	Event
Feng et al., 2019	HMD	Camera	HMD	Same camera	Cross correlation
Friston and Steed, 2014	Mouse	Camera	Monitor	Same camera	Event
Gruen et al., 2020	Sub millisecond clock	Camera	HMD	Synced camera	Continuous
He et al., 2000	Wand	Camera	Monitor	Same camera	Event
Kämäräinen et al., 2017	Touch	Switch	Mobile phone	Photodiode	Event
Kijima and Miyajima, 2016a	HMD	Camera	HMD	Synced camera	Cross correlation
Kijima and Miyajima, 2016b	HMD	Camera	HMD	Synced camera	Continuous
Liang et al., 1991	Pendulum	Camera	LED display	Same camera	Event
Miller and Bishop, 2002	HMD	CCD array	Monitor	CCD array	Event
Mine, 1993	Pendulum	Photodiode	Monitor	Photodiode	Event
Papadakis et al., 2011	Tracked object	Rotary encoder	Monitor	Photodiode	Continuous
Pape et al., 2020	Rigid body	Switch	Projection	Photodiode	Event
Raaen and Kjellmo, 2015	HMD	Photodiode	HMD	Photodiode	Event
Roberts et al., 2009	Hand	Camera	Monitor	Synced camera	Event
Seo et al., 2017	HMD	Rotary encoder	HMD	Photodiode	Event
Sielhorst et al., 2007	Timestamps	Camera	AR HMD	Same camera	Continuous
Stauffert et al., 2020a	Tracked object	Motor driver	HMD	Photodiode	Continuous
Steed, 2008	Pendulum	Camera	Monitor	Same camera	Sine fitting
Swindells et al., 2000	Turntable	Camera	Half silvered mirror	Same camera	Event
Teather et al., 2009	Tracked object	Camera	Monitor	Same camera	Sine Fitting
Wu et al., 2013	Manually Moved Bar	Camera	Monitor	Same camera	Continuous
Zhao et al., 2017	HMD	Potentiometer	HMD	Photodiode	Sine fitting

Camera based measurement has a camera that observes both the real tracked object and its virtual counterpart. Photodiode based measurements read the encoded information off a screen with a photodiode. The observation of the real object is done with a different sensor. Motion to Photon latency measurements use different devices where the motion originates from and which kind of screen emits the photon. The methods column describe how often it is possible to measure latency.

with varying complexity. A less complex system is expected to show lower and more deterministic latency than a more complex system. Newer hardware often has lower latency with reduced determinism (McKenney, 2008). Some papers report multiple measurements of different systems. **Table 3** lists only a subset of the numbers reported in the respective research papers. Interested readers are referred to the original publications.

An observation is that latency is not a constant value. Latency is different with different devices (Mine, 1993), different software configurations (Friston and Steed, 2014) or different input methods (Kämäräinen et al., 2017). Different usage patterns such as a change of the movement direction can influence latency (He et al., 2000). Even small changes in the measurement setup can make a difference. Latency measured in the upper part of a screen can be lower than latency measured in the lower part, due to the scan out sequence (Papadakis et al., 2011). The problem with latency measurements is that they are often performed "under optimized and artificial conditions that may not represent latency conditions in realistic application-oriented scenarios" (Feldstein and Ellis, 2020).

The variability is usually reported by a mean value at least. Standard deviation and minimum/maximum values provide more insight. Histograms can be used to show even more information about what latencies are to be expected. We want to focus on these visualization methods here as a basis to understand the connection between latency and cybersickness. The different ways to describe cybersickness are used in the different simulated latencies of the cybersickness experiments of **Table 1**.

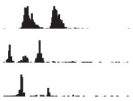
The sparklines in **Table 3** give an impression of the shape of latency. The data is stretched to take the maximum amount in x and y direction and only shows the x axis segment that contains data. Sparklines are supposed to only give a general idea of the shape (Tufte, 2001). Stauffert et al. (2016) and Stauffert et al. (2020a) use a logarithmic y axis. The other papers use a linear y axis. Every sparkline has the measured latency in x direction and its probability in y direction. We exclude Stauffert et al. (2020a) systems where there is artificial latency introduced, but include systems that have artificially high system load but mimic real world scenarios.

A key difference between representations given in publications is if they include rare outliers. Some researchers show no outliers (Wu et al., 2013; Pape et al., 2020) while others do (Sielhorst et al., 2007; Stauffert et al., 2016, 2020a). Latencies usually cluster around one or multiple values. Wu et al. (2013) system 2 and Stauffert et al. (2020a) system 1 show one cluster. Pape et al. (2020) and Sielhorst et al. (2007) system 1 and 3, Wu et al. (2013) system 1 and Stauffert et al. (2016) show two clusters. Sielhorst et al. (2007) system 2 shows 3 clusters and Stauffert et al. (2020a) system 2 shows 9 clusters, each indicated by higher probabilities surrounded by lower probabilities in the histogram.

Each cluster's distribution appears to follow a normal distribution though Sielhorst et al. (2007) system 1, Stauffert et al. (2016) and Stauffert et al. (2020a) system 2 show a more skewed distribution with a longer tail toward larger latencies, resembling more a gamma distribution. Pape et al. (2020) proposes to describe the distribution with a gaussian mixture model, i.e., an imposition of multiple normal distributions. Stauffert et al. (2018) argue to use an empirical distribution derived from the measurements. Multiple clusters presumably originate from the interplay of two or more parts running in decoupled loops in the observed system. Feldstein and Ellis (2020) list processing stages such as simulation or rendering that contribute to the final latency pattern with their runtime and communication behavior. Antoine et al. (2020) show how latency jitter emerges when input device and display sampling frequency differ.

Besides the general distribution, there may be temporal patterns. Stauffert et al. (2020a) found reoccurring latency spikes with a uniform interarrival time. Wu et al. (2013) found a sinusoidal latency pattern.

TABLE 3 | Table summarizing how latency is reported in papers that propose latency measurement approaches.

	Mean	SD	Min/Max	Histogram
Becher et al., 2018	5.1	2.7	1/10	
Billeter et al., 2016	9.8	2.1		
Di Luca, 2010	43.5	5.1		
Feldstein and Ellis, 2020	84	6.3	72/94	
Feng et al., 2019	2.3			
Friston and Steed, 2014	24		18/32	
Gruen et al., 2020	54	1.9		
He et al., 2000	58.5			
Kämäräinen et al., 2017	74.3	14.7		
Kijima and Miyajima, 2016b	16.86			
Kijima and Miyajima, 2016a	19.64			
Liang et al., 1991	85			
Miller and Bishop, 2002	100			
Mine, 1993	80.95			
Papadakis et al., 2011	50	5		
Pape et al., 2020	124.62			
Raaen and Kjellmo, 2015	4		2/5	
Roberts et al., 2009	414			
Seo et al., 2017	46.48	1.09		
Sielhorst et al., 2007				
Stauffert et al., 2020a	64.14	1.6		
Stauffert et al., 2016				
Steed, 2008	64			
Swindells et al., 2000	49			
Teather et al., 2009	73	4		
Wu et al., 2013	27.2			
Zhao et al., 2017	7.2	0.5		

All values are in milliseconds. The values are not comparable and are only for illustration because different systems or parts of systems are measured. Histograms are described with sparklines. The sparklines show only the general shape of the distribution. They are scaled to show the data range of reported values and their frequency. Some papers measure for up to three systems.

5. DISCUSSION

We have shown how latency is measured. The necessary instrumentation varies from simple observations of the VR equipment (Steed, 2008), to the need of specific software to run (Friston and Steed, 2014), to required modifications of the hardware (Stauffert et al., 2020a). The motion may be evoked manually (Wu et al., 2013) or with a pendulum (Mine, 1993) or a turntable (Chang et al., 2016). Latency is observed from one distant observer with one camera (He et al., 2000), multiple distant observers with synchronized cameras (Gruen et al., 2020)

or close observers that are attached to the moved device and the screen (Di Luca, 2010).

Most researchers that measure latency report a mean latency value with an optional standard deviation. Some report a minimum and maximum value in addition. More insight is provided by histograms and plots showing the temporal behavior (Wu et al., 2013). There is research into whether latency influences cybersickness. Most compare the effect of one latency condition with another condition that has a time invariant increased latency (Frank et al., 1988; DiZio and Lackner, 2000; Meehan et al., 2003; Moss et al., 2011; Kawamura and Kijima, 2016; Caserman et al., 2019; Palmisano et al., 2019; Kim et al., 2020). This is based on the most often reported mean latency. Latency jitter as described in latency histograms and periodic latency patterns are shown to also contribute to cybersickness (Stauffert et al., 2018). All approaches to report latency find a counterpart where latency is simulated and shown to influence cybersickness.

There is more research into latency for VR systems than for AR systems, mainly because the technology is often times easier to handle. Many AR systems are simulated with VR systems until AR technology makes the simulated features possible. While less researched, AR systems show similar problems (Sielhorst et al., 2007).

5.1. Limitations on Latency Comparability

There are many factors that can influence latency and the predictability. Kijima and Miyajima (2016a) show that HMD prediction and timewarp (van Waveren, 2016) make a difference. Asynchronous timewarp uses a shortcut to update the displayed image after it was rendered, which yields different values when measured to a system that looks at motion controller movement that is only updated in the simulation of the virtual world. A sequential scan-out process leads to the eyes getting the information at different points in time so it can make a difference which screen is taken for measurement (Papadakis et al., 2011). He et al. (2000) found different latency depending on the movement direction of the tracked object. Manufacturers optimize latency with prediction that may fail (Gach, 2019).

Latency reporting depends on the observed system. The values in **Table 3** are not comparable to one another because some do not measure certain stages of computation or use other hardware. Even though the values are not comparable, they are often reported in a similar fashion with one mean value and a standard deviation.

Spatial jitter can be similar to latency jitter by offsetting tracking positions in an unexpected way. Some measurement methods can not distinguish between latency jitter and spatial jitter by their design. 2D pointing performance suffers with spatial jitter (Teather et al., 2009). Spatial jitter is likely to evoke cybersickness as well and may partially be described in the latency jitter studies already. Some measurement methods measuring related phenomena further complicates the comparison.

5.2. Latency Variability

VR and AR applications require substantial computational power to create virtual environments. Computer systems to provide the experience are optimized for performance rather than real-time, i.e., guaranteed response times (McKenney, 2008). Some applications such as robotics and space exploration require such deterministic runtime behavior of software. Modern operating systems do not provide real-time capabilities and even the Linux PREEMPT_RT patches cannot provide reliable real-time runtimes (Mayer, 2020). Without a real-time operating system, there may be unforseeable latency spikes that can harm VR experiences, even if latency was previously acceptable.

Researchers agree that "the delays vary substantially" (Kämäräinen et al., 2017) and often try to "illustrate the variations in latency of real systems" (Friston and Steed, 2014) by reporting more than one mean latency value. As a caveat, the "latency testing on isolated virtual reality systems under optimized and artificial conditions may not represent latency conditions in realistic application-oriented scenarios" (Feldstein and Ellis, 2020). Care must be taken to measure as close to the use case as possible to best represent the expected latencies. The best case would be to measure during exposure.

Rare latency outliers show latencies much larger than the average (Stauffert et al., 2020a). Networked applications often only look at the 95th, 99th, and 99.9th percentile (Vulimiri et al., 2013) to estimate response times. Teather et al. (2009) use the 95th percentile to describe their motion-to-photon latency measurements. Stauffert et al. (2018) provide a first study with latency spiking behavior including the top one percent but more research is needed to understand if regarding only the 95th or 99th percentile is sufficient. Some web applications found the need to include the remaining one percent of latencies in their analyses (Hsu, 2015).

Latency jitter can be reduced with prediction (Jung et al., 2000). Incorporating latency jitter in the prediction model increases the prediction performance (Tumanov et al., 2007). Prediction, however, introduces its own side effects such as over anticipation (Nancel et al., 2016).

5.3. Desirable Latency Values

How much latency is tolerable for a good VR experience? Carmack (2013) says that it should be below 50 ms to feel responsive and recommends less than 20 ms. Attig et al. (2017) look at HCI experiments without VR that report no impact on usability when latency is below 100 ms. Humans can detect visual variations at 500 Hz (Davis et al., 2015) and latency below 17 ms (Ellis et al., 1999, 2004; Adelstein et al., 2003). Although, Feldstein and Ellis (2020) indicate that perceivable latency does not necessarily cause cybersickness. Jerald (2010) measures a minimum latency threshold of 3.2 ms in one of the participants, but adds that the exact perceivable latency may depend on the virtual environment.

5.4. Need to Measure Latency

Measuring latency helps to become aware of bottlenecks in employed hard- and software (Swindells et al., 2000; Di Luca, 2010). Without measuring, those problems may never be detected and may influence an otherwise sound experiment. Many researchers, however, do not report latency. The 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)

saw 104 published papers. 85 papers conducted a user study in virtual reality. Only 6 reported the latency of the employed VR system. Although a reported mean latency strengthens trust that the systems performed as expected, latency jitter may still have occurred during experiments and may have impaired individual measurements.

Which approach to use depends on the application and possibilities of the researchers. A detailed analysis helps to judge the application's performance but everything is better than not measuring at all. Every researcher should be able to do manual frame counting (He et al., 2000) as shown in Feldstein and Ellis (2020) that compare the results of different evaluators. Sine fitting (Steed, 2008) reduces imprecisions in the video analysis. Even though it is more involved than manual frame counting, software can help with the analysis (Stauffert et al., 2020b). Beyond these basic approaches, the choice of how to measure latency depends on the specific hard- and software used. Design your measurement system to fit your VR system guided by the approaches in **Table 2**. Research should strive toward measuring latency for every frame shown on the employed screen to assure validity of observations and to maximize insight. Measuring latency can hint at problems, latency values then have to be interpreted to find an intervention if need be.

6. CONCLUSION

Latency is one of the characteristics of a computer system that is often discussed to have a major impact on the system's usability. Research shows that larger latencies and latency jitter can influence well-being in a negative way in the form of cybersickness. Yet little research of VR experiences check and report the latency behavior of their employed computer system. Only 7% of the papers published at the 2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR) conducting user studies in virtual reality reported their motion to photon latency. Latency may introduce unwanted effects that are not obvious to the researchers and reviewers if a latency value is not reported.

Latency is not restricted to one value but changes over time and with the VR system usage pattern. More elaborated test setups are required to capture these dynamics. Research is only beginning to understand the implications of time-invariant latency. Even the occasional latency spike will contribute to cybersickness. Measuring latency is of importance to understand better the influence on cybersickness and to understand where latency might not be the main cause for cybersickness.

AUTHOR CONTRIBUTIONS

J-PS conducted the literature review and took the lead in writing the manuscript, he collectively discussed, and developed concepts to measure and report latency. FN worked on the manuscript and supervised the project. ML conceived the original idea, collectively discussed and developed concepts of own research on latency, and supervised the project. All authors provided critical feedback and helped shape the research, analysis, and manuscript.

REFERENCES

Adelstein, B. D., Lee, T. G., and Ellis, S. R. (2003). "Head tracking latency in virtual environments: psychophysics and a model," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 47 (Los Angeles, CA: SAGE Publications), 2083–2087. doi: 10.1177/15419312030 4702001

Ames, S. L., Wolffsohn, J. S., and McBrien, N. A. (2005). The development of a symptom questionnaire for assessing virtual reality viewing using a head-mounted display. *Optometry Vis. Sci.* 82, 168–176. doi: 10.1097/01.OPX.0000156307.95086.6

Antoine, A., Nancel, M., Ge, E., Zheng, J., Zolghadr, N., and Casiez, G. (2020). "Modeling and reducing spatial jitter caused by asynchronous input and output rates," in *UIST 2020-ACM Symposium on User Interface Software and Technology* (New York, NY). doi: 10.1145/3379337.3415833

Arcioni, B., Palmisano, S., Apthorp, D., and Kim, J. (2019). Postural stability predicts the likelihood of cybersickness in active HMD-based virtual reality. *Displays* 58, 3–11. doi: 10.1016/j.displa.2018.07.001

Attig, C., Rauh, N., Franke, T., and Krems, J. F. (2017). "System latency guidelines then and now–is zero latency really considered necessary?" in *Engineering Psychology and Cognitive Ergonomics: Cognition and Design*, ed D. Harris, Vol. 10276 (Cham: Springer International Publishing), 3–14. doi: 10.1007/978-3-319-58475-1_1

Becher, A., Angerer, J., and Grauschopf, T. (2018). Novel approach to measure motion-to-photon and mouth-to-ear latency in distributed virtual reality systems. *arXiv preprint arXiv:1809.06320*.

Billeter, M., Rothlin, G., Wezel, J., Iwai, D., and Grundhofer, A. (2016). "A LED-based IR/RGB end-to-end latency measurement device," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)* (Merida), 184–188. doi: 10.1109/ISMAR-Adjunct.20 16.0072

Bles, W., Bos, J. E., De Graaf, B., Groen, E., and Wertheim, A. H. (1998). Motion sickness: only one provocative conflict? *Brain Res. Bull.* 47, 481–487. doi: 10.1016/S0361-9230(98)00115-4

Bockelman, P., and Lingum, D. (2017). "Factors of cybersickness," in *HCI International 2017-Posters' Extended Abstracts*, ed C. Stephanidis, Vol. 714 (Cham: Springer International Publishing), 3–8. doi: 10.1007/978-3-319-58753-0_1

Bos, J. E., de Vries, S. C., van Emmerik, M. L., and Groen, E. L. (2010). The effect of internal and external fields of view on visually induced motion sickness. *Appl. Ergon.* 41, 516–521. doi: 10.1016/j.apergo.2009.11.007

Carmack, J. (2013). Latency mitigation strategies. Available online at: https://danluu.com/latency-mitigation/

Caserman, P., Martinussen, M., and Gobel, S. (2019). "Effects of end-to-end latency on user experience and performance in immersive virtual reality applications," in *Entertainment Computing and Serious Games*, eds E. van der Spek, E., S. Gobel, E. Y.-L. Do, E. Clua, and J. Baalsrud Hauge, Vol. 11863 (Cham: Springer International Publishing), 57–69. doi: 10.1007/978-3-030-34644-7_5

Chang, C.-M., Hsu, C.-H., Hsu, C.-F., and Chen, K.-T. (2016). *Performance Measurements of Virtual Reality Systems: Quantifying the Timing and Positioning Accuracy*. New York, NY: ACM Press. doi: 10.1145/2964284.2967303

Chang, E., Kim, H. T., and Yoo, B. (2020). Virtual reality sickness: a review of causes and measurements. *Int. J. Hum. Comput. Interact.* 36, 1658–1682. doi: 10.1080/10447318.2020.1778351

Chen, Y.-C., Dong, X., Hagstrom, J., and Stoffregen, T. A. (2011). Control of a virtual ambulation influences body movement and motion sickness. *BIO Web Conf.* 1:00016. doi: 10.1051/bioconf/20110100016

Davis, J., Hsieh, Y.-H., and Lee, H.-C. (2015). Humans perceive flicker artifacts at 500 Hz. *Sci. Rep.* (London) 5:7861. doi: 10.1038/srep07861

Davis, S., Nesbitt, K., and Nalivaiko, E. (2014). *A Systematic Review of Cybersickness*. ACM Press. doi: 10.1145/2677758.2677780

Di Luca, M. (2010). New method to measure end-to-end delay of virtual reality. *Presence* 19, 569–584. doi: 10.1162/pres_a_00023

DiZio, P., and Lackner, J. R. (2000). Motion sickness side effects and aftereffects of immersive virtual environments created with helmet-mounted visual displays. *Brandeis Univ Waltham ma ashton Graybiel Spatial Orientation Lab,* 5.

Ellis, S. R., Mania, K., Adelstein, B. D., and Hill, M. I. (2004). "Generalizeability of latency detection in a variety of virtual environments," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 48 (Los Angeles, CA: SAGE Publications), 2632–2636. doi: 10.1177/154193120404802306

Ellis, S. R., Young, M. J., Adelstein, B. D., and Ehrlich, S. M. (1999). "Discrimination of changes of latency during voluntary hand movement of virtual objects," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 43 (Los Angeles, CA: SAGE Publications), 1182–1186. doi: 10.1177/154193129904302203

Feldstein, I. T., and Ellis, S. R. (2020). "A simple, video-based technique for measuring latency in virtual reality or teleoperation," in *IEEE Transactions on Visualization and Computer Graphics* (New York, NY). doi: 10.1109/TVCG.2020.2980527

Feng, J., Kim, J., Luu, W., and Palmisano, S. (2019). "Method for estimating display lag in the Oculus Rift S and CV1," in *SIGGRAPH Asia 2019 Posters* (Brisbane, QLD: ACM), 1–2. doi: 10.1145/3355056.3364590

Frank, L. H., Casali, J. G., and Wierwille, W. W. (1988). Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Hum. Fact*. 30, 201–217. doi: 10.1177/001872088803000207

Friston, S., and Steed, A. (2014). Measuring latency in virtual environments. *IEEE Trans. Visual. Comput. Graph.* 20, 616–625. doi: 10.1109/TVCG.2014.30

Gach, E. (2019). *Valve Updates Steam VR Because Beat Saber Players Are Too Fast*. Library Catalog: kotaku.com

Gavgani, M. A., Walker, F. R., Hodgson, D. M., and Nalivaiko, E. (2018). A comparative study of cybersickness during exposure to virtual reality and "classic" motion sickness: are they different? *J. Appl. Physiol.* 125, 1670–1680. doi: 10.1152/japplphysiol.00338.2018

Gianaros, P. J., and Stern, R. M. (2010). A questionnaire for the assessment of the multiple dimensions of motion sickness. *Aviation, Space, and Environmental Medicine*, 72:115.

Golding, J. F. (1998). Motion sickness susceptibility questionnaire revised and its relationship to other forms of sickness. *Brain Res. Bull.* 47, 507–516. doi: 10.1016/S0361-9230(98)00091-4

Graybiel, A., Wood, C. D., and Cramer, D. B. (1968). Diagnostic criteria for grading the severity of acute motion sickness. *Aerosp. Med.* 39, 453–5.

Gruen, R., Ofek, E., Steed, A., Gal, R., Sinclair, M., and Gonzalez-Franco, M. (2020). "Measuring System Visual Latency through Cognitive Latency on Video See-Through AR devices," in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (Atlanta, GA: IEEE), 791–799.

He, D., Liu, F., Pape, D., Dawe, G., and Sandin, D. (2000). "Video-based measurement of system latency," in *International Immersive Projection Technology Workshop* (Ames, IA).

Hsu, R. (2015). *Who Moved my 99th Percentile Latency?* Available online at: https://engineering.linkedin.com/performance/who-moved-my-99th-percentile-latency

Jerald, J. J. (2010). "Relating scene-motion thresholds to latency thresholds for head-mounted displays," in *2009 IEEE Virtual Reality Conference* (Lafayette, LA), 211–218. doi: 10.1109/VR.2009.4811025

Jung, J. Y., Adelstein, B. D., and Ellis, S. R. (2000). "Discriminability of prediction artifacts in a time-delayed virtual environment," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, Vol. 44 (Los Angeles, CA: SAGE Publications), 499–502. doi: 10.1177/154193120004400504

Kämäräinen, T., Siekkinen, M., Ylä-Jääski, A., Zhang, W., and Hui, P. (2017). "Dissecting the end-to-end latency of interactive mobile video applications," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications - HotMobile '17* (Sonoma, CA: ACM Press), 61–66. doi: 10.1145/3032970.3032985

Kawamura, S., and Kijima, R. (2016). "Effect of head mounted display latency on human stability during quiescent standing on one foot," in *2016 IEEE Virtual Reality (VR)* (Greenville, SC) 199–200. doi: 10.1109/VR.2016.7504722

Kennedy, R. S., Lane, N. E., Berbaum, K. S., and Lilienthal, M. G. (1993). Simulator sickness questionnaire: an enhanced method for quantifying simulator sickness. *Int. J. Aviat. Psychol.* 3, 203–220. doi: 10.1207/s15327108ijap0303_3

Keshavarz, B., and Hecht, H. (2011). Validating an efficient method to quantify motion sickness. *Hum. Fact*. 53, 415–426. doi: 10.1177/0018720811403736

Kijima, R., and Miyajima, K. (2016a). "Looking into HMD: a method of latency measurement for head mounted display," in *2016 IEEE Symposium on 3D User Interfaces (3DUI)* (Greenville, SC), 249–250. doi: 10.1109/3DUI.2016.7460064

Kijima, R., and Miyajima, K. (2016b). "Measurement of Head Mounted Display's latency in rotation and side effect caused by lag compensation by simultaneous observation–an example result using Oculus Rift DK2," in *2016 IEEE Virtual Reality (VR)* (Greenville, SC), 203–204. doi: 10.1109/VR.2016.7504724

Kim, J., Luu, W., and Palmisano, S. (2020). Multisensory integration and the experience of scene instability, presence and cybersickness in virtual environments. *Comput. Hum. Behav*. 113:106484. doi: 10.1016/j.chb.2020.106484

Kinsella, A., Mattfeld, R., Muth, E., and Hoover, A. (2016). Frequency, not amplitude, of latency affects subjective sickness in a head-mounted display. *Aerosp. Med. Hum. Perform*. 87, 604–609. doi: 10.3357/AMHP.4351.2016

Lampton, D. R., Knerr, B. W., Goldberg, S. L., Bliss, J. P., Moshell, J. M., and Blau, B. S. (1994). The virtual environment performance assessment battery (VEPAB): development and evaluation. *Presence Teleoperat. Virt. Environ*. 3, 145–157. doi: 10.1162/pres.1994.3.2.145

LaViola, J. J. Jr. (2000). A discussion of cybersickness in virtual environments. *ACM SIGCHI Bull.* 32, 47–56. doi: 10.1145/333329.333344

Liang, J., Shaw, C., and Green, M. (1991). "On temporal-spatial realism in the virtual reality environment," in *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology* (Hilton Head, SC), 19–25. doi: 10.1145/120782.120784

Mayer, B. (2020). *SpaceX: Linux in den Rechnern, Javascript in den Touchscreens*. Library Catalog: www.golem.de

McCauley, M. E., and Sharkey, T. J. (1992). Cybersickness: perception of self-motion in virtual environments. *Presence Teleoperat. Virt. Environ*. 1, 311–318. doi: 10.1162/pres.1992.1.3.311

McHugh, N. (2019). Measuring and minimizing cybersickness in virtual reality. *Dissertation*, University of Canterbury. doi: 10.26021/1316

McKenney, P. E. (2008). ""Real time" vs. "real fast": how to choose?" in *Ottawa Linux Symposium* (Ottawa), 57–65.

Meehan, M., Razzaque, S., Whitton, M., and Brooks, F. (2003). "Effect of latency on presence in stressful virtual environments," in *IEEE Virtual Reality, 2003* (Los Angeles, CA), 141–148.

Miller, D., and Bishop, G. (2002). "Latency meter: a device end-to-end latency of VE systems," in *Stereoscopic Displays and Virtual Reality Systems IX* (San Jose, CA), Vol. 4660, 458–464. doi: 10.1117/12.468062

Mine, M. (1993). *Characterization of End-to-End Delays in Head-Mounted Display Systems*. The University of North Carolina at Chapel Hill.

Moss, J. D., Austin, J., Salley, J., Coats, J., Williams, K., and Muth, E. R. (2011). The effects of display delay on simulator sickness. *Displays* 32, 159–168. doi: 10.1016/j.displa.2011.05.010

Muth, E. R., Stern, R. M., Thayer, J. F., and Koch, K. L. (1996). Assessment of the multiple dimensions of nausea: the Nausea Profile (NP). *J. Psychosom. Res.* 40, 511–520. doi: 10.1016/0022-3999(95)00638-9

Nancel, M., Vogel, D., De Araujo, B., Jota, R., and Casiez, G. (2016). "Next-point prediction metrics for perceived spatial errors," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo), 271–285. doi: 10.1145/2984511.2984590

Oman, C. M. (1990). Motion sickness: a synthesis and evaluation of the sensory conflict theory. *Can. J. Physiol. Pharmacol.* 68, 294–303. doi: 10.1139/y90-044

Palmisano, S., Szalla, L., and Kim, J. (2019). "Monocular viewing protects against cybersickness produced by head movements in the Oculus Rift," in *25th ACM Symposium on Virtual Reality Software and Technology* (Parramatta, NSW: ACM), 1–2. doi: 10.1145/3359996.3364699

Papadakis, G., Mania, K., and Koutroulis, E. (2011). "A system to measure, control and minimize end-to-end head tracking latency in immersive simulations," in *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry - VRCAI '11*, (Hong Kong: ACM Press) 581. doi: 10.1145/2087756.2087869

Pape, S., Kruger, M., Muller, J., and Kuhlen, T. W. (2020). "Calibratio: a small, low-cost, fully automated motion-to-photon measurement device," in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)* (Atlanta, GA), 234–237. doi: 10.1109/VRW50115.2020.00050

Raaen, K., and Kjellmo, I. (2015). "Measuring latency in virtual reality systems," in *Entertainment Computing - ICEC 2015*, eds K. Chorianopoulos, M. Divitini, J. Baalsrud Hauge, L. Jaccheri, and R. Malaka, Vol. 9353 (Cham: Springer International Publishing), 457–462.

Reason, J. T., and Brand, J. J. (1975). *Motion Sickness*. Cambridge, MA: Academic Press.

Rebenitsch, L., and Owen, C. (2016). Review on cybersickness in applications and visual displays. *Virt. Real.* 20, 101–125. doi: 10.1007/s10055-016-0285-9

Riccio, G. E., and Stoffregen, T. A. (1991). An ecological theory of motion sickness and postural instability. *Ecol. Psychol.* 3, 195–240. doi: 10.1207/s15326969eco0303_2

Roberts, D., Duckworth, T., Moore, C., Wolff, R., and O'Hare, J. (2009). "Comparing the end to end latency of an immersive collaborative environment and a video conference," in *2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications* (Singapore: IEEE), 89–94. doi: 10.1109/DS-RT.2009.43

Seo, M.-W., Choi, S.-W., Lee, S.-L., Oh, E.-Y., Baek, J.-S., and Kang, S.-J. (2017). Photosensor-based latency measurement system for head-mounted displays. *Sensors* 17:1112. doi: 10.3390/s17051112

Sielhorst, T., Sa, W., Khamene, A., Sauer, F., and Navab, N. (2007). "Measurement of absolute latency for video see through augmented reality," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality* (Nara), 215–220. doi: 10.1109/ISMAR.2007.4538850

Stanney, K. M., Kennedy, R. S., and Drexler, J. M. (1997). Cybersickness is not simulator sickness. *Proc. Hum. Fact. Ergon. Soc. Annu. Meet.* 41, 1138–1142. doi: 10.1177/107118139704100292

Stauffert, J.-P., Niebling, F., and Latoschik, M. E. (2016). *Towards Comparable Evaluation Methods and Measures for Timing Behavior of Virtual Reality Systems*. Munich: ACM Press. doi: 10.1145/2993369.2993402

Stauffert, J.-P., Niebling, F., and Latoschik, M. E. (2018). "Effects of latency jitter on simulator sickness in a search task," in *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (Reutlingen: IEEE), 121–127. doi: 10.1109/VR.2018.8446195

Stauffert, J.-P., Niebling, F., and Latoschik, M. E. (2020a). "Simultaneous run-time measurement of motion-to-photon latency and latency jitter," in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* (Atlanta, GA: IEEE), 636–644. doi: 10.1109/VR46266.2020.1581339481249

Stauffert, J.-P., Niebling, F., Lugrin, J.-L., and Latoschik, M. E. (2020b). "Guided sine fitting for latency estimation in virtual reality," in *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, 707–708. doi: 10.1109/VRW50115.2020.00204

Steed, A. (2008). "A simple method for estimating the latency of interactive, real-time graphics simulations," in *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology, VRST '08* (New York, NY: ACM), 123–129. doi: 10.1145/1450579.1450606

Stone Ill, W. B. (2017). *Psychometric evaluation of the Simulator Sickness Questionnaire as a measure of cybersickness* (Doctor of Philosophy). Iowa State University, Digital Repository, Ames, IA.

St. Pierre, M. E., Banerjee, S., Hoover, A. W., and Muth, E. R. (2015). The effects of 0.2Hz varying latency with 20–100ms varying amplitude on simulator sickness in a helmet mounted display. *Displays* 36, 1–8. doi: 10.1016/j.displa.2014.10.005

Swindells, C., Dill, J. C., and Booth, K. S. (2000). "System lag tests for augmented and virtual environments," in *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (San Diego, CA: ACM), 161–170. doi: 10.1145/354401.354444

Teather, R. J., Pavlovych, A., Stuerzlinger, W., and MacKenzie, I. S. (2009). "Effects of tracking technology, latency, and spatial jitter on object movement," in *2009 IEEE Symposium on 3D User Interfaces* (Lafayette, LA), 43–50. doi: 10.1109/3DUI.2009.4811204

Treisman, M. (1977). Motion sickness: an evolutionary hypothesis. *Science* 197, 493–495. doi: 10.1126/science.301659

Tufte, E. R. (2001). *The Visual Display of Quantitative Information, Vol. 2*. Cheshire, CT: Graphics Press.

Tumanov, A., Allison, R., and Stuerzlinger, W. (2007). "Variability-aware latency amelioration in distributed environments," in *2007 IEEE Virtual Reality Conference* (Charlotte, NC), 123–130. doi: 10.1109/VR.2007.352472

van Waveren, J. M. P. (2016). "The asynchronous time warp for virtual reality on consumer hardware," in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16* (Munich: ACM Press), 37–46. doi: 10.1145/2993369.2993375

Virre, E. (1996). Virtual reality and the vestibular apparatus. *IEEE Eng. Med. Biol. Mag.* 15, 41–43. doi: 10.1109/51.486717

Vulimiri, A., Godfrey, P. B., Mittal, R., Sherry, J., Ratnasamy, S., and Shenker, S. (2013). Low latency via redundancy. *arXiv preprint arXiv:1306.3707*. doi: 10.1145/2535372.2535392

Wu, W., Dong, Y., and Hoover, A. (2013). Measuring digital system latency from sensing to actuation at continuous 1-ms resolution. *Presence Teleoperat. Virt. Environ.* 22, 20–35. doi: 10.1162/PRES_a_00131

Zhao, J., Allison, R. S., Vinnikov, M., and Jennings, S. (2017). "Estimating the motion-to-photon latency in head mounted displays," in *2017 IEEE Virtual Reality (VR)* (Los Angeles, CA), 313–314. doi: 10.1109/VR.2017.7892302

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Author Contributions

The author conducted the literature review and took the lead in writing the manuscript, he collectively discussed and developed concepts to measure and report latency. He provided critical feedback and helped shape the research, the analysis and the manuscript.

7.3 Multi User Experience

Virtual reality is not only restricted to single user applications but will be able to play its strength in multi user experiences. We published a journal article that explains the technical challenges when implementing multi user experiences. Performance degrades the more users are in a virtual space. The experience suffers both from the latency introduced in the network exchange and from latency due to increased simulation and rendering demand with many avatars.

The article was published as Marc Erich Latoschik, Florian Kern, Jan-Philipp Stauffert, Andrea Bartl, Mario Botsch, and Jean-Luc Lugrin. “Not Alone Here?! Scalability and User Experience of Embodied Ambient Crowds in Distributed Social Virtual Reality”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.5 (2019), pages 2134–2144

2134 IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 25, NO. 5, MAY 2019

Not Alone Here?! Scalability and User Experience of Embodied Ambient Crowds in Distributed Social Virtual Reality

Marc Erich Latoschik, Florian Kern, Jan-Philipp Stauffert, Andrea Bartl, Mario Botsch, and Jean-Luc Lugrin

Fig. 1. An immersive Social Virtual Reality (SVR) with multiple avatars and agents co-located in the same virtual space as seen from an immersed participant's point of view. Our SVR system supports large crowds of distributed avatar/agent participants of variable appearances (see an abstract virtual body in the front left, and a photorealistic virtual body from photogrammetry scans on the right).

Abstract—This article investigates performance and user experience in Social Virtual Reality (SVR) targeting distributed, embodied, and immersive, face-to-face encounters. We demonstrate the close relationship between scalability, reproduction accuracy, and the resulting performance characteristics, as well as the impact of these characteristics on users co-located with larger groups of embodied virtual others. System scalability provides a variable number of co-located avatars and AI-controlled agents with a variety of different appearances, including realistic-looking virtual humans generated from photogrammetry scans. The article reports on how to meet the requirements of embodied SVR with today's technical off-the-shelf solutions and what to expect regarding features, performance, and potential limitations. Special care has been taken to achieve low latencies and sufficient frame rates necessary for reliable communication of embodied social signals. We propose a hybrid evaluation approach which coherently relates results from technical benchmarks to subjective ratings and which confirms required performance characteristics for the target scenario of larger distributed groups. A user-study reveals positive effects of an increasing number of co-located social companions on the quality of experience of virtual worlds, i.e., on presence, possibility of interaction, and co-presence. It also shows that variety in avatar/agent appearance might increase eeriness but might also stimulate an increased interest of participants about the environment.

Index Terms—Social Virtual Reality, quality of experience, performance characteristics, co-location, co-presence, possibility of interaction, ambient crowds, avatars and agents, computer-mediated communication, multi-user virtual environment

1 Introduction

Embodied Social Virtual Reality (SVR) exploits the rich social signals and behavior patterns humans use in the physical world [44]. These signals significantly originate from our paraverbal and non-verbal expressions in face-to-face encounters. Body movements, gestures, mimics, and eye movements play crucial roles in social behavioral phenomena like joint attention, grouping, eye contact, or mutual synchronization and coordination [38]. Embodiment technologies provide the necessary means to realize virtual face-to-face encounters enabling social signals via so-called avatars, our digital alter egos in the virtual realm.

- *Marc Erich Latoschik, Florian Kern, Jan-Philipp Stauffert, Andrea Bartl, and Jean-Luc Lugrin are with the HCI group of the University of Würzburg. E-mail: marc.latoschik@uni-wuerzburg.de.*
- *Mario Botsch is with Bielefeld University. E-mail: botsch@techfak.uni-bielefeld.de.*

Manuscript received 10 Sept. 2018; accepted 7 Feb. 2019.
Date of publication 17 Feb. 2019; date of current version 27 Mar. 2019.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2019.2899250

SVRs have gained much interest in both, academia and industry. Companies like Second Life[1], AltspaceVR[2], Pixo VR[3], or NVIDIA with its Holodeck[4] project already developed real-world applications or impressive demonstrations. Although, to some extent, these developments suggest a solid maturation of the overall field, in fact, the existing approaches differ in significant aspects. Schroeder [40] separates such Multi-User Virtual Environments (MUVEs) into *immersive environments*, e.g., NVIDIA's Holodeck project, and *online worlds*, e.g., Second Life or typical networked multi-user computer games. We argue that this distinction is drawn from the availability or absence of embodiment features and its qualities in terms of (1) completeness of represented and controllable body parts, (2) the avatars' appearances or looks, and (3) direct control of the avatars' bodies with a sufficient sensory coverage of the controlling users' movements in real-time.

Overall, embodied SVR promises novel forms of computer-mediated communication but it is particularly challenging regarding sensory

[1] https://secondlife.com
[2] https://altvr.com
[3] https://pixogroup.com
[4] https://www.nvidia.com/en-us/design-visualization/technologies/holodeck/

coverage and temporal and precision requirements, which is considered one of the grand challenges for VR [41], specifically if it has to be realized with distributed systems. Table 1 derives hypothetical data rates while scaling up sensory coverage and the number of co-located avatars for a selection of fidelities. Typically, setups with just two embodied avatars in 1:1 dyadic social avatar-avatar encounters [4, 5, 25, 39, 44] do not need to utilize distribution. Similarly, work on the effect of larger crowds of virtual humans also either use computer-controlled agents [33] or utilize pre-recorded material presented as videos [2, 9]. Hence, little is known about the experience of users fully immersed and embodied in a simulated Virtual Environment (VE) with larger groups of co-located avatars with variable appearance, and potential performance characteristics as given by a real-world distribution of these companions.

Contribution

This article investigates the current state-of-the-art of distributed Social Virtual Realities with consumer VR technology, and the effects of larger embodied ambient crowds on participating users. The approach deliberately utilizes today's consumer VR technology, i.e., the Unreal Engine 4 (UE4), for two reasons: (1) It targets out-of-lab real-world applications. (2) To show potential benefits and limitations of today's consumer technology when realizing distributed SVRs. A scalable system architecture supports various types and fidelities of control schemes and avatar appearances, up to high-quality, realistic looking individualized avatars captured by photogrammetry. We propose a hybrid evaluation combining technical benchmarks with user-centered tests. We demonstrate the hybrid evaluation for up to 125 participants and show the close relation between objective measures and subjective ratings. The evaluation confirms sufficient performance for 25 participants in a typical real-world distribution scenario. This upper bound is applied to and verified by a user study on the experience of users immersed in an SVR with co-located ambient crowds. To the best of our knowledge, this evaluation reveals two novel effects: (1) An increasing number of co-located social companions has a positive effect on the quality of experience of virtual worlds, i.e., on presence, possibility of interaction, and co-presence. (2) It also shows that variety in avatar/agent appearance might increase eeriness but might also stimulate an increased interest of participants about the environment.

2 Related Work

Schroeder gives a comprehensive overview of the fundamental aspects of social interaction in virtual worlds [40]. Steed and Schroeder also highlight some fundamental concepts as well as technical aspects and requirements for SVRs [47]. They classify current embodiment approaches along a scale representing the degree of user modeling and identify three main types of current approaches: (1) puppeteered, (2) reconstructed, and (3) tracked. Typical online games, as well as multi-user worlds similar to Second Life, are basically puppeteered. Avatars here are controlled by some non-direct animation scheme, e.g., by pressing a button to trigger an animation. With an immersive first-person perspective from inside one's avatar, this control scheme seems detrimental. Still, even puppeteered systems motivate research in SVR [20, 32], specifically about alternative social mechanics.

Real-time reconstruction of dynamic scenes and users promises a faithful dynamic replication of real physical appearances and environments [3, 13, 35]. It certainly is appropriate for many use-cases, e.g., teleconferencing. However, it does not allow to easily modify avatar appearance, as, e.g., is required by work on the *illusion of virtual body ownership (IVBO)* [18, 30, 42] or the *Proteus* effect [51], specifically in avatar-avatar encounters [25] or dyadic social avatar-avatar interactions [4, 5, 37, 39]. A deliberate change of the appearance of avatars could also be desirable to avoid stigmatizing in SVR. Additionally, concerning scalability, dynamic 3D reconstruction is characterized by high bandwidth requirements. These exceed typical requirements of tracked approaches (see Table 1) by orders of magnitudes even with appropriate compression [26], which often will also increase the latency.

Tracked avatar embodiment for SVRs requires direct control schemes of as many degrees of freedom as the human body has, and hence, a) elaborated sensor technology like full-body motion tracking [21, 45] and/or face tracking [24], and b) an appropriate model of a virtual human body matching the sensory input. Such models typically consist of a properly rigged body mesh for skeletal animations together with blend shapes for facial animations where applicable. They are either generated manually via 3D-modeling, or via off-line reconstruction from real humans [1, 10] (or a combination of both), effectively combining dynamic tracking with static reconstruction.

The extent of sensory coverage depends on the reproduction accuracy between the controlling user and the controlled avatar and its body and animation model. A full embodiment of self-avatars increases presence but full as well as partial embodiment increases co-presence [15], although recent work could not substantiate these findings [28], which might be caused by strong contextual distractors. Recently, initially single-user embodiment studies also started to explore the effect of the appearance and the behavior of an other's avatar co-located in SVR [25, 39]. All agree on the necessity of low latencies for contingencies [49], e.g., convincing visuomotor synchrony. Additional requirements are a first-person perspective, a sufficiently realistic avatar appearance [2, 31, 48], and a high degree of immersion [48]. Work which includes larger groups of others so far either used non-distributed settings of up to eight virtual agents with comparable looks [6, 43], or even non-immersive and non-interactive pre-recorded videos [9].

Table 1. Estimations of net data transfer rates required to communicate tracked non-verbal behavior of avatars of different fidelities. Accurate numbers are subject to a variety of design choices and optimizations: (1) applied body model, (2) sensory coverage, (3) model-based optimizations, (4) resolution, i.e., number of bits per value, and (5) compression. Model-based optimizations refer to potential uses of forward kinematics (FK), or inverse kinematics (IK), for shared models. IK is potentially applicable, i.e., for linked models (supporting a proper skeleton with chained joints and links) based on B1-B3, but could also be applied to partial models. FK is necessary for linked models based on B4-B6. Results reflect a coarse upper bound estimation: no potential overhead, no compression applied, single precision float values yielding 4 bytes, representation of **p**(osition) and **r**(otation) each by 3 floats, **f**(lexion) and **a**(bduction) each by one float. For the face we include the seven basic emotions of Ekman and Friesen in F1 and a selection of 44 Action Units of the Facial Action Coding System [11] encoded with either 3 bit (F2) or 1 float (F3).

			Bytes / avatar	Data rate in KB/s per number of clients (1 avatar/client) at 90 Hz							
				2	5	10	25	50	75	100	125
Fidelity of body model											
B1	head	$1 \times \mathbf{pr}$	24.0	4.3	10.8	21.6	54.0	108.0	162.0	216.0	270.0
B2	+ 2 hands	$3 \times \mathbf{pr}$	72.0	13.0	32.4	64.8	162.0	324.0	486.0	648.0	810.0
B3	+ 2 feet and spine	$6 \times \mathbf{pr}$	144.0	25.9	64.8	129.6	324.0	648.0	972.0	1296.0	1620.0
B4	skeleton medium	$1 \times \mathbf{p} + 17 \times \mathbf{r}$	216.0	38.9	97.2	194.4	486.0	972.0	1458.0	1944.0	2430.0
B5	hand low	$5 \times \mathbf{f} + 5 \times \mathbf{a} + \mathbf{r}$	52.0	9.4	23.4	46.8	117.0	234.0	351.0	468.0	585.0
B6	hand high	$15 \times \mathbf{f} + 5 \times \mathbf{a} + \mathbf{r}$	92.0	16.6	41.4	82.8	207.0	414.0	621.0	828.0	1035.0
Fidelity of face model											
F1	2 eyes + 7 emotions	$2 \times \mathbf{r}$ + 7 float	52.0	9.4	23.4	46.8	117.0	234.0	351.0	468.0	585.0
F2	2 eyes + 44 FACS bit	$2 \times \mathbf{r}$ + 44×3 bit	40.5	7.3	18.2	36.5	91.1	182.3	273.4	364.5	455.6
F3	2 eyes + 44 FACS float	$2 \times \mathbf{r}$ + 44 float	200.0	36.0	90.0	180.0	450.0	900.0	1350.0	1800.0	2250.0

2.1 Discussion and Requirements

We currently have little knowledge about the experience of users co-located with larger groups of avatars realized with distributed immersive and embodied SVR. How does it feel to be surrounded by virtual others? Current game technology provides sophisticated rendering and networking features. SVRs are sensitive to performance characteristics due to the close temporal patterns of non-verbal social signals. How does current technology cope with the extensive embodiment requirements and how do the specific performance characteristics impact user experience regarding scalability? To answer these questions, we chose a hybrid approach combining tracked user motions and avatar models (up to avatars reconstructed by photogrammetry) for three reasons: (1) To support applications requiring scalability in terms of modified avatar appearances, (2) to scale up the number of distributed avatars and/or sensory coverage, and (3) to be compatible with animation principles of current game engines.

Following the theoretical estimates in Table 1, we chose a medium fidelity B3 (324.0 KB/s) for the current evaluation. Note that B3 potentially either requires IK to be in effect at the distributed clients, or it has to do without any linked models at all. However, its resulting bandwidth is roughly comparable to B4, which transmits all data necessary to replicate complete linked models. Hence, B3 is a suitable candidate for our upcoming performance evaluation, which are mainly targeting impacts from potential bandwidth and latency bottlenecks caused by a real-world distribution.

UE4 promotes a multi-user client-server distribution architecture. We assume a standard 1 Gbit network link from the server to the internet backbone, and clients connected with potentially much less bandwidth, e.g., from private homes. A proper multicast infrastructure can in general not be expected for the given distribution scenario. Hence, an increasing number of clients would certainly increase the load on the server, specifically for outgoing replication. For example, given 25 clients and fidelity B3, we require a total bandwidth of 324 KB/s to the server, and 8100 KB/s from the server (who has to replicate all data back to all clients) at 90 Hz. The resulting client bandwidth here is much lower (ca. 13 KB/s up; 324 KB/s down). Overall, the final requirements for the developed system are as follows:

- R1 SVR supporting a scalable number of physically distributed users.
- R2 Real-world application with potentially novice end-users.
- R3 High immersion with first-person perspective.
- R4 Full embodiment with sufficient sensory coverage.
- R5 Variable, realistic avatar appearance.
- R6 High visuomotor synchrony and responsiveness.
  - (a) Sufficient data throughput.
  - (b) Low latencies and low latency jitter.
  - (c) High data fidelity (accuracy and precision).

Similarly to work in [23, 29], our system supports mixed virtual crowds of user-controlled avatars with AI-controlled virtual agents for various application-specific tasks (e.g., as role-models or troublemakers). However, this article does not focus on any agent-specific research questions which are the topic of an alternative publication [27].

Benchmarking the non-functional requirements R6 for VR systems often uses two general approaches in combination. Intrusive benchmarking like for real-time systems, in general, requires instrumentation of the code itself. Elaborate VR frameworks and game engines usually support intrusive benchmarking and provide additional tools for profiling. The intrusive approach also allows pinpointing sources of problems inside the code. On the downside, it requires full-blown code access, measurements potentially interfere with the results, and the outcomes do not necessarily correlate to end-user experiences. Hence, VR-benchmarking often applies non-intrusive black-box benchmarking and end-to-end measurements. Non-intrusive benchmarks include camera-based latency measures by phase shift analysis of sine curve movements [46] or body movements [49], automated frame counting [12], or formal simulation of systems [36]. Chang et al. found interesting system behaviors with a non-intrusive high-speed camera-based approach [8]. They identified sensitivity-precision tradeoffs of the underlying engines, which are of high relevance for our approach. Such tradeoffs often result from internal optimizations of the underlying engine potentially out-of-reach for application developers.

Overall, we will use a combination of intrusive and non-intrusive benchmarks to evaluate the specified non-functional performance requirements concerning the scalability features of the system. We will complement the technical benchmarks with subjective ratings of the user experience concerning the performance characteristics (fluidity, synchrony, annoyment, and simulator sickness) to identify potential correlations between both evaluation methods. Finally, we shed some light onto the subjective effects of being inside an SVR populated by a variable number of co-located avatars with different appearances. Here, we use the system to investigate the resulting user experience based on a selection of adequate factors for evaluating an SVR with co-located avatars, e.g., attractiveness, humanness and eerieness, presence, co-presence, and the possibility of interaction.

3 System Description

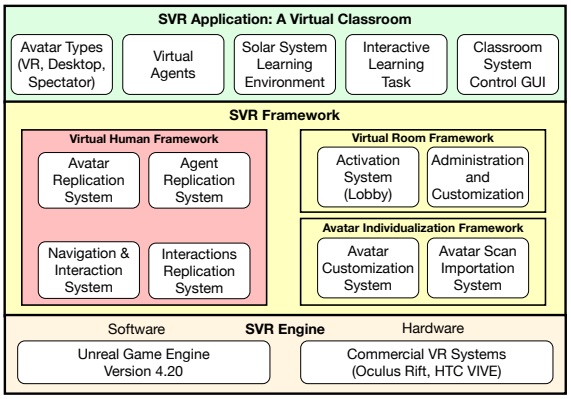


Fig. 2. Overview of the three main layers of the software architecture, which separates specific system functions according to their specificity with respect to the required application domain.

Fig. 2 illustrates the overall software parts of the system organized into three main abstraction layers:

1. SVR Application Layer: Specific functions for customized SVR content, e.g., for distributed embodied learning or training.
2. SVR Framework Layer: Generic functions for distributed embodied SVR that provide the basic avatar and agent representation and animation capabilities and the environment specification.
3. SVR Engine Layer: Underlying hardware and software functions supporting device Input/Output (I/O), basic interaction schemes, core visualization and simulation capabilities, network communication facilities, and software component integration schemes.

The following sections describe the SVR Engine and Framework layers in detail and encompass the relevant functionality for general SVR support. We take a closer look at the distribution and networking architecture which follows common practices proposed by the Unreal Engine development community and builds upon the provided UE4 network replication methods.

3.1 SVR Engine Layer: Hard- and Software

Requirements R2 and R3 are the determining factors for the use of consumer VR hard- and software. End-users have to operate the system from their homes without the help of a technician or trained personnel. To also support R3, the system uses Oculus Rift as well as HTC VIVE and HTC VIVE pro head-mounted displays. UE4 provides application packaging and distribution necessary to support R2. The

engine has proven to be beneficial in related work [23, 29]. It has a well-known reputation for rendering high-quality virtual humans. It is used in combination with the photogrammetry-based method to capture high-quality avatars following [1, 48] to fulfill R5. Elaborated software tools like the UE4 usually already support several essential features, which are necessary to implement the functional requirements. Notably, they often also predefine how to satisfy a specific functionality, and they enforce a particular development model. This guidance is helpful for novices but also restrictive for experienced programmers. Additionally, such tools often incorporate idiosyncratic terminology for their programming primitives, which complicates comprehension of existing correlations to important software engineering concepts. In the following section, we try to pinpoint differences where appropriate, but most often adhere to the provided programming model and the resulting terminology and naming. We deliberately made this choice since one goal of this work is to identify where we stand concerning the realization of SVR with the given technology. It should also foster replicability since this specific terminology is used throughout the UE4 documentation.

3.2 SVR Framework Layer

3.2.1 Virtual Human Framework

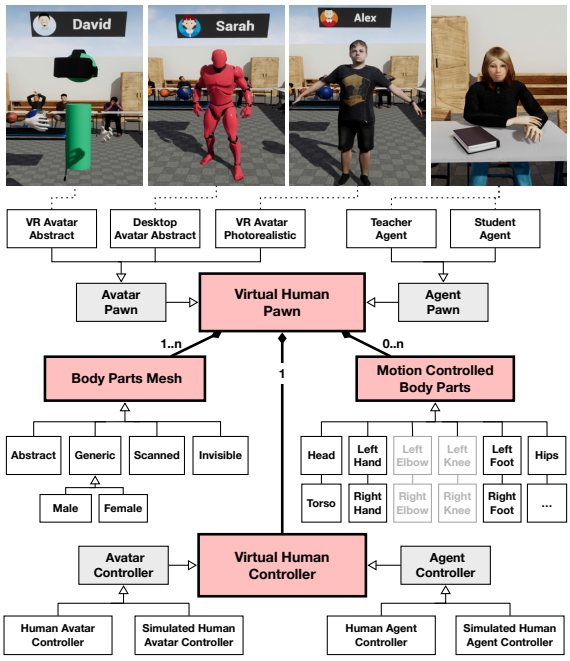


Fig. 3. Class diagram of the Virtual Human Framework (bottom). It supports variable body models (top) in accordance with the *Pawn* and *Controller* abstractions proposed by UE4.

Fig. 3 illustrates the architecture of the *Virtual Human Framework*, which collectively supports virtual humans controlled by a user (i.e., avatar) as well as controlled by the system (i.e., agent). The framework models each virtual human as *Virtual Human Pawn*, which is composed of a *Virtual Human Controller* and a combination of *Body Part Mesh* and *Motion Controlled Body Part*. A flexible combination of different body models and motion-controlled body parts provides various specializations of *Virtual Human Pawn*, such as our *VR Abstract Avatar*, *VR Photorealistic Avatar*, or the *Desktop Avatar*. The *Controller* classes contain the logic responsible for driving the animations of the pawn. Any pawn can be controlled in real-time either (1) puppeteered using users' keyboard, mouse, or controller inputs, (2) tracked using sensory input of users' movements and expressions, or (3) algorithmically animated based on artificial intelligence techniques (e.g., behavior tree or scripted scenario). The classes *Simulated Human Avatar Controller* and *Simulated Human Agent Controller* provide valuable features for testing and benchmarking. They provide pre-recorded tracking data or random movement sequences for any pawn type. They also permit to control the input frequency and amplitudes in order to create more controllable and realistic variations during benchmarking.

3.2.2 Virtual Room Framework

A lobby menu provides an application's starting point. The lobby supports the selection and administration of the target virtual environment and the user's configuration. The menu allows customizing the avatar of the user: name, image, color, and types (e.g., VR Abstract, Photorealistic, or Desktop). By default, the user who creates the server instance is the administrator. She/He can ban other users and select different types of virtual rooms. Shortcut buttons provide a faster setup of multiple parameters like in the benchmark scenario and allow the administrator to set the user's configuration to default values.

3.2.3 Avatar Individualization Framework

Fig. 4 shows the photogrammetry rig we use to generate photorealistic and individualized avatars. It includes 106 Canon DSLR cameras, model EOS1300D. 96 cameras focus the body, 10 cameras focus the face. The 3D model is generated with the photogrammetry software CapturingReality, and post-processed and cleaned with Autodesk Mudbox. Retopology and polycount reduction, as well as UV mapping, is achieved with R3dS Wrap. The resulting avatars have a polycount of around 40k triangles. For comparison, the standard UE4 mannequin has a polycount of 41k triangles. The UV-mapped textures exported from R3dS Wrap have a resolution of 4096×4096. We use Maya to rig our avatars and to export the results as an FBX file into UE4. Current work integrates pre-processing speed-ups for the avatar models as motivated by [1].

3.3 Distribution Architecture

The distribution architecture and multi-user support follow a client-server model as promoted by UE4. Clients can control avatars as well as agents. The *Virtual Human Pawn* class (see Fig. 3) centralizes the replication semantics per virtual human (avatar or agent) as illustrated in Fig. 5 for two clients. The client packs the updated state of all body parts (e.g., head, hands, and feet) into an array for efficiency, and sends it in bulk to the server via remote procedure calls (RPCs). The server locally stores and replicates this data to all clients. Receiving clients apply the replicated data to the corresponding body parts.

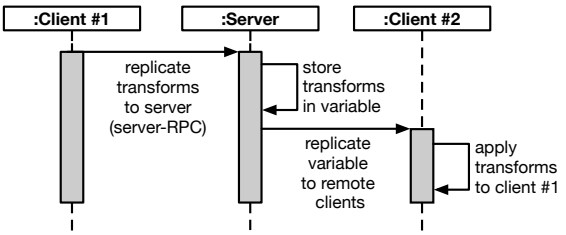


Fig. 5. A client replicates the movements of body parts by executing an RPC on the server. The server replicates this data to the remote client. The remote client applies the replicated movement data to body parts.

UE4 provides several options to realize and parameterize networking. Our current system uses unreliable client-server communication and server-client replication without notification, both to reduce potential latencies. A pre-study did not reveal any significant drop-outs in a comparison of unreliable to reliable client-server communication. We set the replication rate for client-server communication to 60 Hz (desktop clients) and 90 Hz (VR clients) to limit the maximum data

2138 IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 25, NO. 5, MAY 2019

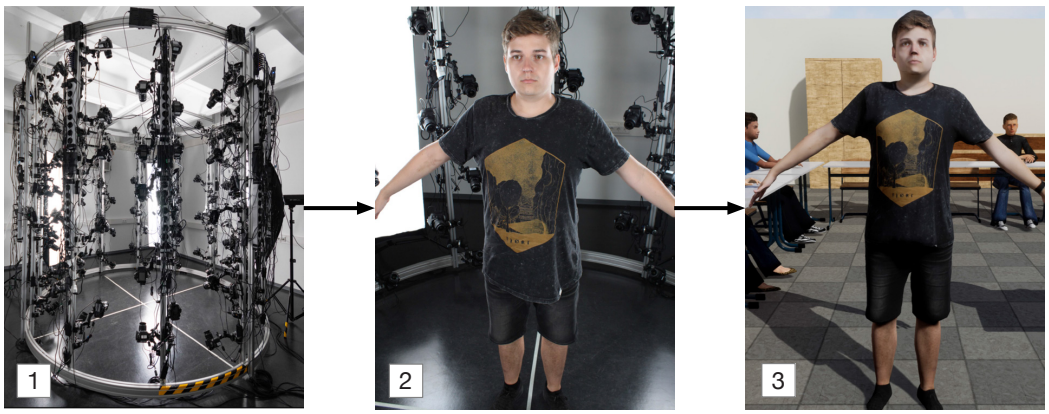


Fig. 4. The photogrammetry rig used to scan individuals (1), a person during the scan (2), and the resulting high quality avatar (3).

rate while still providing smooth animations. Such technical replication parameters will certainly affect the overall characteristics of the system performance and potentially will, in consequence, also affect user-perception in some unpredicted way [8]. Hence, we test the system concerning objective and subjective measures in combination.

4 Evaluation Design, Scenario, and System

The evaluation consisted of three consecutive evaluation phases (EPs):

EP1 **Performance Benchmarking** tested the scalability concerning objective performance characteristics of latency, frame rate, and data rate with an increasing number of clients (2, 5, 10, 25, 50, 75, 100, 125), i.e., avatars participating in the SVR. EP1 recorded the stimulus material for the upcoming evaluation phases: video recordings of the two screens (see Fig. 6) for EP2 and movement recordings, i.e., position and rotation for chosen fidelity, of the collaborating partner and the three planets with an extended movement sequence for EP3. EP1 finally identified 25 as a first potential upper bound for the number of avatars for EP2 and EP3.

EP2 **Performance Perception** tested the scalability concerning the subjective impact of the technical performance characteristics measured in EP1 on perceived fluidity, synchrony, and annoyment on a non-immersed user. EP2 used the video recordings from EP1 with the same scaling conditions. This phase validated and confirmed the upper bound for the number of avatars for EP3.

EP3 **Subjective Experience of Co-Location and Scalability** tested the subjective user experience inside a distributed ambient crowd and the impact of the technical performance characteristics of the recorded movements of the interactions from EP1 with an increasing number of simulated avatars of different avatar appearances (uniformly human-like as in Fig. 3, upper right, and mixed human-like and abstract as in Fig. 3, upper right and left) on an immersed user. EP3 used a reduced number of avatars (2, 10, 25, 100), with the condition 100 explicitly exceeding 25 as the target maximum number of avatars, and confirmed this maximum. These numbers are reported here already as a lookahead to some of the results from EP1 and EP2 for clarity.

The consecutive phases EP2 and EP3 deliberately used prerecorded animations to not induce any confounds by changed stimuli throughout the experiments and to ensure comparability. However, these recordings retained all visibly perceivable performance characteristics resulting from an increased number of clients and avatars, i.e., latencies and stuttering of the interactive animations. The virtual environment for all evaluation phases resembled a classroom with a teamwork-oriented layout, with the participating avatars seemingly collaborating in distributed groups around tables (see Fig. 1). The user and one participant who apparently was directly interacting with her/him are seated vis-a-vis at one table. A virtual solar system is visualized floating in the air in-between them. All executables for the three phases were initially developed using the blueprint visual scripting system of UE4. Phases EP1 and EP2 were nativized, i.e., automatically translated to C++ and compiled as stand-alone executables. This approach is in line with our initial assumption to show what we can expect from today's consumer systems without any further close-to-metal optimizations. Only the recording and playback functions were natively implemented in C++ to reduce any performance overhead from these intrusive functions. The server application in EP3 was not nativized due to an incompatibility with a required plugin but did not require any of the potentially performance-critical networking capabilities.

Table 2. Specifications of the hardware used during the study.

Computers	CPU	RAM	GPU
1× Server	i7-8700K	16GB	NVIDIA GTX 1080 Ti
2× VR Clients	i7-8700K	16GB	NVIDIA GTX 1080 Ti
Load Test Clients			
5× Computers	i7-8700k	16GB	NVIDIA GTX 1080 Ti
9× Computers	i7-7700k	16GB	NVIDIA GTX 1080
18× Computers	i5-6600	16GB	NVIDIA GTX 1080

All phases were implemented using the hardware specified in Table 2. Hosts used 1 Gbit ethernet connected via a switch infrastructure. The VR clients used Oculus Rift HMDs with Oculus Touch controllers. The system was implemented using the Unreal Engine 4.20 and Microsoft Windows 10. Up to 4 instances of the client systems had to share one of the load test hosts for scaling conditions beyond 25 live clients. We took care to distribute client systems to the load test hosts uniformly and to reduce performance impact by the graphics stages as much as possible. Still, multiple clients per host potentially result in additional bottlenecks. However, all results identified to satisfy R1 are not affected by this. The distribution of client and server systems on the available hosts and the control of the avatars and agents were as follows:

EP1: 1 server per server host; 2 VR clients, each with dedicated VR host; uniform distribution of load test clients to the load test hosts following the required scalability conditions. The non-interacting clients simulate simple avatar movements which do not stress the clients but which produce the appropriate data rates.

EP2: No live systems needed. Initially, the setup is the same as for EP1 since it is a recorded video of all animations for the given scalability conditions from EP1.

EP3: One server per server host to play back the recordings of the interaction, and to integrate the participant's avatar inspecting

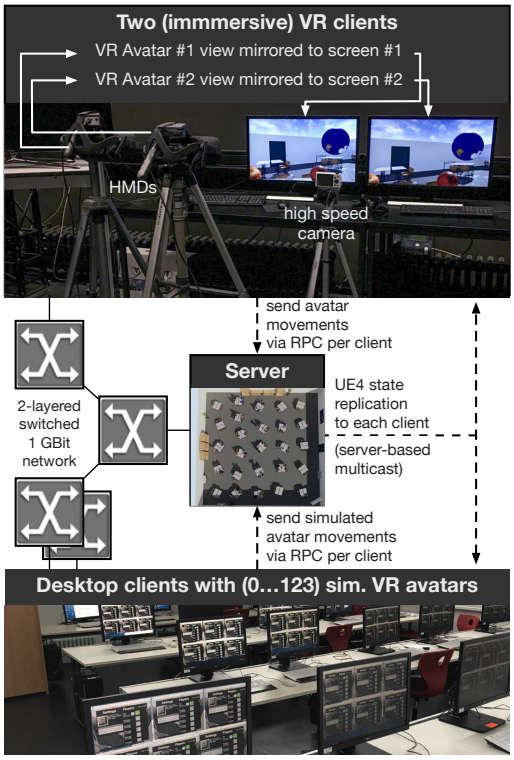


Fig. 6. Physical layout, interconnect scheme (mid left), and logical distribution architecture (mid right) of the benchmarking scenario. On 32 computers, up to four application instances are running for load tests.

the scene. Notably, the play-back did not cause any significant additional load, and the recordings of the movement data of the interactions retained all visually perceivable performance characteristics generated by EP1.

5 EP1 – Performance Benchmarking

Fig. 6 illustrates the physical setup for the performance benchmarking. The displayed scene of the two VR clients is mirrored to the two monitors with a refresh rate of 60 Hz. A high-speed camera is placed in front of the two VR clients to record both screens at 240 Hz. One user continuously performs a smooth drag-and-drop operation of a planet from left to right and back.

5.1 Measuring Latency by Frame Counting

We conducted manual frame counting on the high-speed video following [14] for all scaling conditions (2, 5, 10, 25, 50, 75, 100, 125). We counted how many frames passed between observing updates of the object's location between the two VR clients. Movement smoothing techniques were disabled to see the raw updates. Additionally, we measured the time elapsed between an initiated grab of an object and the reception of this event by the other VR client.

Execution of the triggered event of grabbing a planet completed within a mean of at most 12 ms between the two clients for all test conditions. This delay is equal or below the screen refresh rate of the benchmark monitors (60 Hz $\approx$ 16.6 ms), hence clients receive updates with a delay that is less than the smallest measurable unit. The delay between position updates of an initially smooth movement determines how choppy the movement looks to users. This value increases with an increasing number of connected clients. Fig. 7 visualizes the results of this measurement. Table 3 shows the averaged numbers for all frame counting measurements. As can be seen, there is a notable latency increase detectable beyond 25 clients.

Table 3. Latency as determined by counting how much time passes between initiating a movement on one computer and seeing the movement on a network-connected computer's screen (left) and between two updates of a moving object (right). Higher latency means bigger and more discernable jumps in the movement.

Number of Clients	Latency in ms as means (SD) for Movement Begin	Update Rate
2	8(3.65)	18(06.26)
5	7(2.81)	21(08.94)
10	8(2.64)	19(06.99)
25	8(2.64)	30(06.67)
50	12(7.3)	40(16.60)
75	8(1.96)	57(13.22)
100	8(0)	95(14.82)
125	10(1.3)	158(10.65)

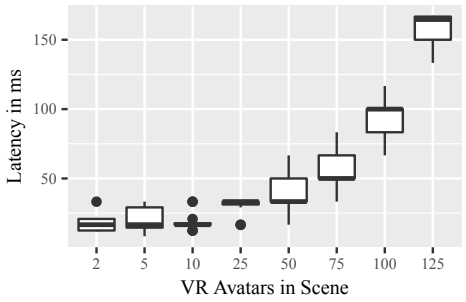


Fig. 7. Average latency and standard deviation between two updates of a movement for a varying number of clients/avatars connected.

5.2 Measuring Performance by Network Statistics

We measured network performance on the server using the *Stat Net* command and the *Network Profiler* of UE4 for the scaling conditions (2, 5, 10, 25, 50, 75, 100, 125) to find out, whether and how an increasing number of clients and avatars increases the latency and impacts the frame rate. Table 4 reports the results for the benchmarking scenario, i.e., the FPS for server and VR client, the VR client ping for the latency, the network I/O rates on the server and on average per client, the latter four ordered in the sequence of the replication.

The FPS on server and clients decrease with more than 25 connected clients. Also, the clients' latencies increase with an increasing number of connected clients. First, this confirms our results from the frame counting in Sect. 5.1. Second, the results for the data rate from the clients to the server (before the performance drop) are in general estimated by Table 1 when scaled to a 60 Hz replication rate. Observable differences are caused by an enabled compression and a detected basic load of UE4's network layer, which is in effect even without additional payload. Third, the measurements for conditions with 50+ clients illustrate typical challenges in the context of black-box testing, i.e., to precisely pinpoint the source of the bottleneck. With 50+ clients most data rates decrease, or slow-down their increase as for the server out-rate. This certainly is due to the decreased FPS at the server and the clients, since replication between the hosts is bound to the simulation rates. Still, measured data rates are well inside the available bandwidth, rendering a network bottleneck unlikely. Inspecting CPU system performance revealed a high load on one core of the server for the critical conditions. Since the simulation loop is responsible for all replication I/O, our current analysis strongly suggests the performance bottleneck likely to be located somewhere within the server's network I/O capabilities.

Table 4. The server and client performance and network statistics during the benchmarking scenario. The results visualize a decreasing number of frames per second (FPS) at the server and an increasing latency of the clients (Client Ping) with an increasing number of connected clients for measurements beyond 25 clients. Client FPS and data rates decrease slightly time-delayed. See text for further discussions.

Number of Clients	Server FPS	VR Client FPS	Client Ping (ms)	Out Rate Client ⟶ Avg. (KB/s)	In Rate Server ⟶ (KB/s)	Out Rate Server ⟶ (KB/s)	In Rate Client Avg. (KB/s)
2	120	90	9	11.5	23.0	21.5	10.8
5	120	90	9	11.0	54.5	146.9	29.4
10	120	90	11	10.8	107.5	571.8	57.3
25	120	90	13	11.6	288.7	3653.7	143.1
50	40	90	35	10.5	534.7	7081.1	136.4
75	17	60	65	6.4	474.8	9377.3	124.4
100	11	60	117	4.2	1038.8	10163.3	98.7
125	6	35	182	2.8	342.7	11386.2	80.9

Table 5. Descriptive statistics for the three items. For fluidity and synchrony, high values mean high approval. For annoyment low values mean low annoyment. Scales range from 1 to 5.

Number of Clients	Fluidity $M(SD)$	Synchrony $M(SD)$	Annoyment $M(SD)$
2	3.88(1.21)	4.48(.67)	1.67(.82)
5	3.95(1.23)	4.40(.83)	1.57(.97)
10	3.90(1.08)	4.52(.67)	1.57(.86)
25	3.88(1.02)	4.19(.97)	1.93(.92)
50	2.90(1.28)	3.88(.89)	2.48(1.27)
75	2.88(1.11)	3.12(1.13)	2.76(.91)
100	1.81(.97)	2.12(.94)	3.88(.94)
125	1.33(.72)	1.48(.74)	4.64(.58)

6 EP2 – Performance Perception

We recorded eight short video clips of 30 seconds of the interaction described in Sect. 5 for each scaling condition from 2 to 125 clients as before. These videos were provided via an online survey to collect the user feedback about the perceived latency. Each participant watched all eight videos. The order of the videos was randomized. We included three items for each video. Participants rated their approval to the statements "The movement of the ball on the right screen is fluid." and "The movement of the two balls is synchronous." on a 5-point Likert scale. Additionally, we included an adapted version of the ITU-R impairment scale [19]: Participants stated if they perceived a difference between both movements. The 5-point Likert scale ranged from "Imperceptible" to "Perceptible, but not annoying", "Slightly annoying", "Annoying" up to "Very annoying".

$N = 42$ people (19 female, 23 male) with a mean age of $M(SD) = 28.86(9.55)$ participated in the subjective evaluation. On average they reported playing video games $M(SD) = 6.7(10.24)$ hours a week with values ranging between 0 hours and 40 hours. 40 participants answered the employment question. 15 participants were students, 24 participants were employees, 1 participant was self-employed.

To analyze the data, we calculated a repeated-measures ANOVA for each item. For all three items, Mauchly's test indicated a violation of the assumption of sphericity (all $ps < .01$). Therefore, we report Greenhouse-Geisser-corrected tests for Fluidity ($= .74$), Synchrony ($= .66$), and Annoyment ($= .76$). All post-hoc tests were pairwise comparisons with Bonferroni adjustments. We used IBM SPSS Statistics 25 for the analysis of the quantitative data.

6.1 Results

Table 5 displays the descriptive statistics for the three items. Fig. 8 shows the means, standard errors, and significant differences. The ratings regarding the fluidity of the movement of the right ball differed significantly, $F(5.20, 213.05) = 43.74$, $p < .001$, partial $\eta^2 = .52$. Post-hoc tests showed that 2, 5, 10 and 25 avatars differed significantly from all higher numbers. 50 and 75 avatars differed from 100 avatars and higher ($p \leq .01$). No significant differences occurred between 2, 5, 10 and 25 avatars, between 50 and 75 avatars, and between 100 and 125 avatars. Participants' approval to the synchrony statement also differed significantly, $F(4.61, 188.89) = 101.15$, $p < .001$, partial $\eta^2 = .71$. Post-hoc tests revealed that 2, 5 and 10 avatars differed significantly from 50 and more avatars, 25 differed significantly from 75 and more, 50 and 75 avatars differed significantly from 100 and more, and 100 differed significantly from 125 (all $ps < .05$). Finally, the ratings on the ITU-R impairment scale (annoyment of the perceived difference) differed significantly, $F(5.31, 217.82) = 79.51$, $p < .001$, partial $\eta^2 = .66$. Post-hoc tests showed that the ratings differed significantly between the same numbers of avatars as for the synchrony ratings ($p \leq .01$). Overall, the subjective ratings were very much in line with the objective measures and did confirm the still acceptable limit of 25 avatars.

7 EP3 – Subjective Experience of Co-Location and Scalability

The aim of the final phase of the evaluation was three-fold: to get insights into (1) the subjective experiences of users immersed inside of an SVR with an increasing number of co-located embodied others, (2) the potential effects of different avatar appearances of the co-located others in such an environment, and (3) the impact of potential technical characteristics hampering the overall experience.

The user study followed a mixed-methods design. As the within-subjects factor, each participant experienced four conditions with a varying number of co-located avatars (2, 10, 25, 100) in randomized order. These numbers resulted from the first phases choosing 100 as a value certainly impacting the experience. As the between-groups factor, we manipulated the appearance of the other avatars. In the *Human* condition, all other avatars looked human (Fig. 3, right). In the *Mixed* condition, half of the avatars looked human, and the other half had an artificial appearance (Fig. 3, left). We assessed quantitative as well as qualitative data.

7.1 Procedure

Fig. 9 illustrates the experimental procedure. The first step introduced the participants to the procedure and the HMD and controller. Then they gave their informed consent to take part in the study and answered the pre-questionnaire. They put on the HMD and adjusted the head straps and lens distance according to their personal preferences.

Now participants experienced the first SVR scene consisting of 24 other avatars sitting around tables. Participants could inspect the surrounding for 20 seconds and then gave oral qualitative feedback on their impression of the scene without leaving the VR. Next, the experimenter showed an example question floating in front of the participant to explain how to interact with such in-vitro text questions in VR and to assure readability. The following experimental phase iterated through the four within-subject conditions, randomly varying the numbers of avatars. One abstract VR avatar sat at the same table as the participant throughout this phase. He moved the planets according to the recordings taken under the respective load condition. The participant answered questions in VR after each exposure. Fig. 10 shows screenshots of the scene (1) and a VR question afterward (2). In the end, participants removed the HMD and answered the post-questionnaire.

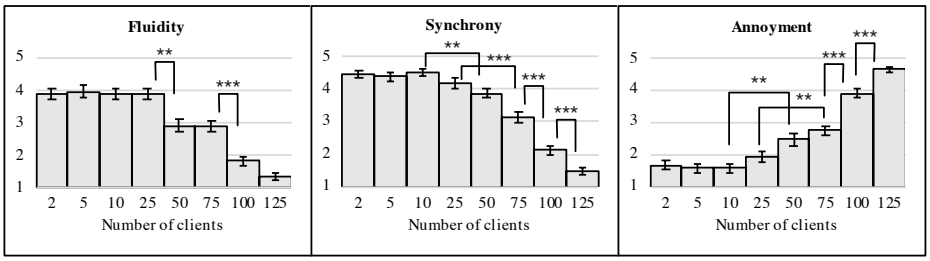


Fig. 8. Means and according standard errors for the items regarding the fluidity and synchrony of the movements and the annoyment. Low values mean low fluidity, synchrony, and annoyment. Significances are marked as follows: $^{*} < .05$, $^{**} < .01$, $^{***} < .001$.

The experiment simultaneously took place in three rooms with identical setups but different experimenters (2 male, 1 female). All experimenters followed a strict study protocol to ensure comparable results.

7.2 Measures

Participants filled in a pre- and post-questionnaire on a dedicated computer using the online questionnaire tool LimeSurvey and answered in-vitro questionnaires while immersed in the virtual environment.

1. Pre-Questionnaire: Participants answered the *Immersive Tendency Questionnaire* (ITQ) [50]. The ITQ consists of 18 items with 7-point Likert scales and values ranging from 1 to 7. The second part of the pre-questionnaire was the *Simulator Sickness Questionnaire* (SSQ) [22]. The questionnaire consists of 16 4-point scales ranging from 0 to 3.

2. Qualitative Feedback: To assess qualitative feedback, we asked the following questions:

- "How does it feel to be in this virtual environment?"
- "How do you feel about the presence of the others?"
- "Do you think you would interact with the others in the same way as you would in the real world?"

3. In-Vitro Questions: Presentation and answering of the in-vitro questions directly took place in the virtual environment after each condition. We measured the subjective presence of the participants with a single item as proposed in [7]. Participants answered the question "How present do you feel in the virtual environment right now?" on a rating scale ranging from 0 to 10. After that, participants stated their agreement on 7-point Likert scales ranging from 1 to 7 for the following five items:

- "The movement of the {ball / person at my table / other people in the room} was fluid."
- "The movement of the {person at my table / other people in the room} was natural."

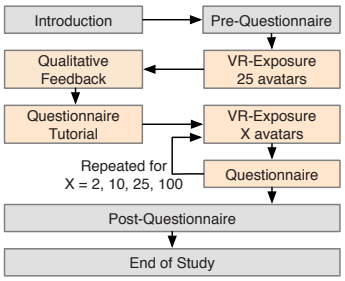


Fig. 9. General procedure of the experiment. Stages performed within VR are colored in orange.

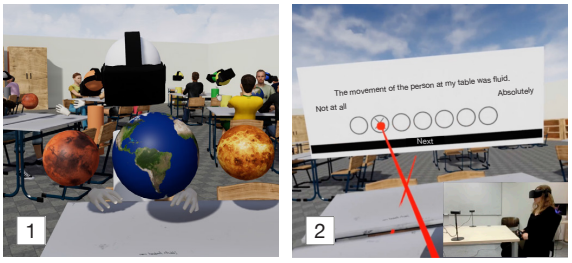


Fig. 10. The participants observed the movements of the planets for 30 seconds (1). They were asked to answer the questions directly in VR using a ray-cast pointing method (2). The participant selected an answer by pointing at a circular option field and pressing the forefinger button.

For the condition with only one additional avatar in the room, we excluded the items asking about the other people in the room. To obtain quantitative information about the experience of being in the virtual environment co-located with other avatars, we included the two subscales *co-presence* and *impression of interaction possibilities* of [34]. The co-presence subscale consists of three items, and the possible interactions subscale consists of four items. All scales are 7-point Likert scales ranging from 1 to 7.

4. Post-Questionnaire: To gather information about possible uncanny valley effects we included the four subscales *Attractiveness*, *Humanness*, and *Eeriness* [16, 17]. We divided Eeriness into the subscales *eerie* and *spine-tingling* [16]. The last stimulus-related question was another open question: "How did you feel about the different numbers of avatars in the room?"

Participants answered the SSQ [22] a second time to assure that our application did not induce any unwanted effects regarding simulator sickness. Each participant answered the following demographic items: age, gender, occupation, highest educational achievement, debility of sight, years of experience with the location's language, and experience in playing video games and with VR environments.

7.3 Participants

$N = 47$ people participated in the study. We excluded two data sets from the analysis due to technical problems (1 disconnected controller, 1 false position calibration) during the experiment. The remaining sample consisted of $N = 45$ people (71.7 % female) with a mean age of $M(SD) = 20.96(1.75)$. All participants gave written informed consent and got course credits for their participation. Assignment of participants to one of the two conditions Human ($n = 23$, 56.5% female) or Mixed ($n = 22$, 86.4% female) was randomized. In both groups, most people reported playing video games less than 1 hour a day (Human: 14, Mixed: 18 people). In the Mixed condition, four people stated that they had never experienced Virtual Reality with a head-mounted

Table 6. Descriptive statistics for the Pre- and Post-Questionnaire Scales. ITQ-scales range from 1 to 7. SSQ scores are total scores calculated according to [22]. For the uncanny valley subscales (value range -3 to +3) low scores mean low humanness, low eeriness, and low attractiveness.

Questionnaire	**Human** $M(SD)$	**Mixed** $M(SD)$
ITQ	4.35(.67)	4.66(.49)
SSQ Pre	12.68(14.56)	14.96(15.31)
SSQ Post	12.52(14.82)	13.60(15.68)
Uncanny Valley	**Human**	**Mixed**
Attractiveness	.52(.91)	1.07(.71)
Humanness	-.39(1.37)	.30(.98)
Overall Eeriness	-1.07(.65)	-.58(.68)
Eerie	-.88(.94)	-.44(.65)
Spine-Tingling	-1.22(.60)	-.69(.89)

display before. In both groups, about 50 % stated that they have 1 to 5 hours of VR experience. In the Human condition, more people (7) stated to have more than 10 hours of VR experience than in the Mixed Condition (3 people). The two groups did not differ significantly regarding their Immersive Tendency ($t(43) = -1.75, p = .09$) or their Simulator Sickness Ratings before ($t(43) = -.51, p = .61$) or after ($t(43) = -.24, p = .81$) the experiment. Overall, simulator sickness did not change significantly throughout the experiment, $t(44) = .53, p = .60$. Table 6 displays the descriptive statistics for the ITQ and SSQ.

7.4 Results

We used IBM SPSS Statistics 25 for the analysis of the quantitative data. Table 7 reports descriptive statistics for all dependent variables assessed inside of the virtual environment.

7.4.1 In-Vitro Measurements

We calculated mixed design ANOVAs for all measurements assessed inside of the virtual environment. Mauchly's test indicated a violation of the assumption of sphericity for the fluidity of the ball (= .61), the fluidity (= .53) and naturalness (= .64) of the person at the participant's table, as well as for the perceived possibility of interaction (= .71), with all $ps < .001$. Therefore, we report Greenhouse-Geisser-corrected tests for these variables. For the fluidity and naturalness of the crowd as well as for presence and co-presence Mauchly's test indicated that sphericity could be assumed.

The analysis reveals a significant main effect of the number of clients on the perceived fluidity of the ball ($F(1.82, 78.23) = 33.30$, partial $\eta^2 = .44$) and the perceived fluidity ($F(1.60, 68.94) = 42.61$, partial $\eta^2 = .50$) and naturalness ($F(1.92, 82.48) = 27.83$, partial $\eta^2 = .39$) of the person at the participant's table, all $ps < .001$. Post-hoc tests for these main effects show that the condition with 100 clients differs significantly from all other conditions (all $ps < .001$). Participants rated this condition the least fluid (ball and person at the table) and natural. We found no significant main effects of the number of avatars on the perceived fluidity or naturalness of the crowd. In all tests regarding the fluidity and naturalness, we found no significant main effect of the avatar appearance (Human vs. Mixed).

The main effect of the number of avatars on the presence rating was not significant, $F(3, 129) < 1$, $p > .05$, partial $\eta^2 = .02$. Presence ratings were similar across conditions with means ranging between $M = 6.22$ and $M = 7.09$. Presence was rated highest in the condition with 25 avatars. There was a significant main effect of the number of avatars on the perceived co-presence, $F(3, 129) = 22.84$, $p < .001$, partial $\eta^2 = .35$. In the condition with only two avatars (the participant plus the person at the table) co-presence was rated lowest, differing significantly (all $ps < .001$) from all other conditions. The co-presence ratings for 10, 25, and 100 avatars do not differ significantly and are similarly high. We found a significant interaction between the number and the appearance of the clients regarding co-presence, $F(3, 129) = 2.85$, $p < .05$, partial $\eta^2 = .06$. The difference between the condition with two avatars and the other three conditions was smaller in the group that saw the human avatar crowd. Compared to this group, the group with the mixed avatar crowd gave a lower co-presence rating for the condition with only one additional avatar and higher ratings for the other three numbers of avatars. We also found a significant main effect of the numbers of clients on the perceived possibility of interactions, $F(2.13, 91.57) = 6.19$, $p = .002$, partial $\eta^2 = .13$. Post-hoc tests show that the condition with 25 clients differs significantly from the condition with two clients ($p = .002$) and the conditions with 100 clients ($p = .046$). The 25 clients condition shows the highest rating regarding the perceived possibility of interaction. For presence, co-presence, and the perceived possibility of interactions nearly all ratings were higher in the group that saw the mixed avatar appearances. However, we found no significant main effect of the avatar appearance (Human vs. Mixed).

7.4.2 Uncanny Valley

We compared the ratings for attractiveness, humanness, and eerieness between both groups calculating independent t-tests. Levene-tests for all subscales were non-significant. The groups did not differ significantly regarding the perceived humanness of the avatars, $t(43) = -1.95, p = .06, r = .29$. Nevertheless, it is noteworthy that the perception of the humanness of the human-looking avatar crowd was lower than of the mixed avatar crowd.

The avatar crowd with mixed appearances appeared as significantly more attractive than the human avatar crowd $t(43) = -2.23, p < .05$, $r = .32$. The two groups differed significantly regarding the perceived eeriness. The human looking avatar crowd appeared to be less eerie than the mixed avatar crowd, $t(43) = -2.47, p < .05$, $r = .35$. As proposed in [16], we split eeriness into its two subscales *eerie* and *spine-tingling*. As a result, we found no significant difference regarding the eerie subscale, $t(43) = -1.81, p > .05$, $r = .27$, but the human looking avatar-crowd was significantly less spine-tingling than the mixed avatar crowd, $t(43) = -2.34, p < .05, r = .34$.

7.4.3 Qualitative Feedback

Qualitative feedback was mixed. Some users were surprised by how real the whole scenario felt and how real the avatars appeared to be. Other users made exactly opposite remarks. Comments about the realness of the avatars were usually restricted to the humanlike avatars. The abstract avatars were described as "things" or robots. Participants almost consistently denied them human status. Many participants commented on the movements of the avatars. Some noticed the repetition of movements. Many stated that the movements seem unnatural.

Some participants reported to feel alone or as being excluded: The avatars shared their tables with others while the participant was the silent observer in the middle. Some said the other avatars ignored them while others interpreted the avatars as staring at them. When getting accustomed to the situations, they reported that more avatars decreased the feeling of loneliness. The situation was reported to be overwhelming when too many avatars were present. Some participants would have liked to interact with the avatars if it were possible because they looked real. Others rejected a potential interaction because they did not experience the avatars as real enough. In both cases, the stated decision factor was how humanlike the avatar is perceived.

8 Discussion and Conclusion

We defined specific requirements for SVRs to evaluate how scalability would affect overall subjective and objective system performance. We developed a software system with consumer soft- and hardware to identify the current state-of-the-art with such an approach. The system design includes all the functional requirements initially defined by R1 to R6. The system supports scalability in the number of distributed co-located avatars, the sensory coverage, as well as in the variable avatar appearance up to photorealistic avatars created by photogrammetry.

Benchmarking confirmed the non-functional performance requirements using objective performance characteristics. We demonstrated how the latter coherently matches the user experience measured by

Table 7. Descriptive statistics for all dependent variables assessed inside the virtual environment (in-vitro). Item scales for fluidity, naturalness, co-presence and interaction range from 1 to 7. The mid-immersion presence item ranges from 0 to 10.

In-Vitro Questions	Condition	2 clients $M(SD)$	10 clients $M(SD)$	25 clients $M(SD)$	100 clients $M(SD)$
Fluid Ball	Human	5.48(1.34)	5.87(1.06)	5.57(1.34)	3.30(1.82)
	Mixed	5.36(1.29)	5.18(1.33)	5.45(1.34)	3.68(2.10)
Fluid Person Table	Human	5.48(1.34)	5.74(.96)	5.61(1.23)	3.22(1.54)
	Mixed	5.82(1.05)	5.59(.85)	5.59(1.01)	4.00(2.00)
Fluid Others	Human	-	4.04(1.55)	4.13(1.58)	4.43(1.44)
	Mixed	-	4.77(1.15)	4.64(1.29)	4.82(1.18)
Natural Person Table	Human	5.13(1.29)	5.48(.90)	5.26(1.21)	3.96(1.55)
	Mixed	5.14(1.25)	5.27 (1.03)	5.18(1.47)	3.68(1.46)
Natural Others	Human	-	3.52(1.56)	3.70(1.58)	3.35(1.61)
	Mixed	-	4.73(1.03)	4.36(1.36)	4.55(1.22)
Presence	Human	6.39(2.13)	6.39(2.23)	6.57(2.79)	6.22(2.73)
	Mixed	6.73(1.32)	6.82(1.30)	7.09(1.77)	6.64(1.99)
Co-Presence	Human	3.81(1.87)	4.90(1.52)	4.87(1.66)	4.88(1.61)
	Mixed	3.26(1.63)	5.46(.71)	5.46(.99)	5.58(1.09)
Interaction	Human	2.52(1.38)	2.74(1.18)	2.99(1.31)	2.58(1.20)
	Mixed	2.55(1.40)	3.32(1.26)	3.52(1.29)	3.16(1.19)

subjective user reports on perceived system characteristics in the non-immersive as well as in the immersive setup. Here, it only affected the perception of the modified movements of the ball and the interaction partner as assumed. Our evaluations also confirm the theoretical estimates of the bandwidth requirements for the client-server system, and the chosen interconnect and fidelity B3 (324.0 KB/s) with a maximum of 25 concurrently co-located distributed avatars, leaving enough bandwidth to distribute whole bodies as specified by B4 and to avoid client-side IK, if required.

We investigated the experience of users immersed inside an embodied SVR with a variable number of participants and appearances of avatars. The condition with 25 avatars significantly resulted in the highest perceived possibility of interaction and had the highest presence ratings. Co-presence was significantly lower for the two avatar condition, and there was a significant interaction between number and appearance of the crowds. Here, potential inconsistencies and incoherences with participants' expectations may cause them to more intensely focus on the surrounding avatars, which would be in line with the significantly higher attractiveness of the mixed avatar crowd.

In general, the vivid surrounding with active companions not hampered by any technical limitation (as emerging here for 25+ avatars) seems to imply a dynamic and stimulating environment despite the used canned animations. Also in line with the reported significant results, presence, the possibility of interaction, and co-presence were consistently higher for the mixed crowd.

All results confirm the positive effects of co-located social companions as well as detrimental effects of suboptimal system performance (here illustrated for 25+ avatars) on the quality of experience of virtual worlds. Finally and notably, the human avatars were rated less human but also significantly less eerie than the mixed crowd. We explain this discrepancy by the incoherence between static appearance and behavior appearance. Overall, these results also inspire an interesting design finding: If we want to manipulate users' interest into a given SVR we can do so by providing mixed avatar appearances, but we have to consider that we are also increasing an inherent eeriness, which might or might not be advisable for a given application context.

8.1 Future Work

Future work will test performance and user experience for extended avatar fidelities and appearances including speech interaction and will experiment with various optimization schemes. These evaluations will be followed by studying application-specific effectiveness for training and learning inside an SVR.

ACKNOWLEDGMENTS

This work was supported in part by a grant from the German Federal Ministry of Education and Research (BMBF project *ViLeArn*).

REFERENCES

[1] J. Achenbach, T. Waltemate, M. E. Latoschik, and M. Botsch. Fast generation of realistic virtual humans. In *23rd ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 12:1–12:10, 2017.

[2] J. N. Bailenson, N. Yee, D. Merget, and R. Schroeder. The effect of behavioral realism and form realism of real-time avatar faces on verbal disclosure, nonverbal disclosure, emotion recognition, and copresence in dyadic interaction. *Presence: Teleoperators and Virtual Environments*, 15(4):359–372, 2006.

[3] S. Beck, A. Kunert, A. Kulik, and B. Froehlich. Immersive group-to-group telepresence. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):616–625, 2013.

[4] G. Bente, S. Rüggenberg, N. C. Krämer, and F. Eschenburg. Avatar-mediated networking: Increasing social presence and interpersonal trust in net-based collaborations. *Human communication research*, 34(2):287–318, 2008.

[5] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality built for two: a virtual reality tool. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 35–36. ACM, 1990.

[6] A. Bönsch, S. Radke, H. Overath, L. M. Asché, J. Wendt, T. Vierjahn, U. Habel, and T. W. Kuhlen. Social VR: How personal space is affected by virtual agents' emotions. In *IEEE Conference on Virtual Reality and 3D User Interfaces*, pages 199–206, 2018.

[7] S. Bouchard, J. St-Jacques, G. Robillard, and P. Renaud. Anxiety increases the feeling of presence in virtual reality. *Presence: Teleoperators and Virtual Environments*, 17(4):376–391, 2008.

[8] C.-M. Chang, C.-H. Hsu, C.-F. Hsu, and K.-T. Chen. Performance measurements of virtual reality systems: Quantifying the timing and positioning accuracy. In *Proceedings of the 24th ACM international conference on Multimedia*, MM '16, pages 655–659, New York, NY, USA, 2016. ACM.

[9] M. Chollet and S. Scherer. Perception of virtual audiences. *IEEE Computer Graphics and Applications*, 37(4):50–59, 2017.

[10] M. Dou, H. Fuchs, and J.-M. Frahm. Scanning and tracking dynamic objects with commodity depth cameras. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 99–106. IEEE, 2013.

[11] P. Ekman and W. V. Friesen. *Manual for the facial action coding system*, 1978.

[12] S. Friston and A. Steed. Measuring latency in virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):616–625, 2014.

[13] M. Gross, M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. Van Gool, S. Lang, K. Strehlke, A. V. Moere, and O. Staadt. Blue-c: A spatially immersive display and 3d video portal for telepresence. *ACM Transactions on Graphics (TOG)*, 22(3):819–827, 2003.

[14] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin. Video-based measurement of system latency. In *International Immersive Projection Technology Workshop*, 2000.

[15] P. Heidicker, E. Langbehn, and F. Steinicke. Influence of avatar appearance

on presence in social VR. In *2017 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 233–234, 2017.

[16] C.-C. Ho and K. F. MacDorman. Measuring the uncanny valley effect. *International Journal of Social Robotics*, 9(1):129–139, 2017.

[17] C.-C. Ho, K. F. MacDorman, and Z. A. D. Pramono. Human emotion and the uncanny valley: a GLM, MDS, and Isomap analysis of robot video ratings. In *3rd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 169–176. IEEE, 2008.

[18] W. A. IJsselsteijn, Y. A. W. de Kort, and A. Haans. Is this my hand I see before me? The rubber hand illusion in reality, virtual reality, and mixed reality. *Presence: Teleoperators and Virtual Environments*, 15(4):455–464, 2006.

[19] ITU. Methodology for the subjective assessment of the quality of television pictures, recommendation ITU-R BT. 500-11. Technical report, ITU Telecom. Standardization Sector of ITU, 2002.

[20] D. Jeffers. Is there a second life in your future? In *Proceedings of the 36th Annual ACM SIGUCCS Fall Conference: Moving Mountains, Blazing Trails*, pages 187–190, New York, NY, USA, 2008. ACM.

[21] V. Kasapakis, E. Dzardanova, and C. Paschalidis. Conceptual and technical aspects of full-body motion support in virtual and mixed reality. In L. T. De Paolis and P. Bourdot, editors, *International Conference on Augmented Reality, Virtual Reality, and Computer Graphics*, pages 668–682. Springer International Publishing, 2018.

[22] R. S. Kennedy, N. E. Lane, K. S. Berbaum, and M. G. Lilienthal. Simulator sickness questionnaire: An enhanced method for quantifying simulator sickness. *The International Journal of Aviation Psychology*, 3(3):203–220, 1993.

[23] M. E. Latoschik, J.-L. Lugrin, M. Habel, D. Roth, C. Seufert, and S. Grafe. Breaking bad behavior: Immersive training of class room management. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (VRST)*, pages 317–318, New York, NY, USA, 2016. ACM.

[24] M. E. Latoschik, J.-L. Lugrin, and D. Roth. FakeMi: A Fake Mirror System for Avatar Embodiment Studies. In *Proceeding of the 22nd ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 73–76, 2016.

[25] M. E. Latoschik, D. Roth, D. Gall, J. Achenbach, T. Waltemate, and M. Botsch. The effect of avatar realism in immersive social virtual realities. In *23rd ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 39:1–39:10, 2017.

[26] Y. Liu, S. Beck, R. Wang, J. Li, H. Xu, S. Yao, X. Tong, and B. Froehlich. Hybrid lossless-lossy compression for real-time depth-sensor streams in 3D telepresence applications. In Y.-S. Ho, J. Sang, Y. M. Ro, J. Kim, and F. Wu, editors, *Advances in Multimedia Information Processing – PCM 2015*, pages 442–452, Cham, 2015. Springer International Publishing.

[27] J.-L. Lugrin, F. Charles, M. Habel, H. Dudaczy, S. Oberdörfer, J. Matthews, J. Porteous, A. Wittmann, C. Seufert, S. Grafe, and M. E. Latoschik. Benchmark framework for virtual students' behaviours. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2236–2238, 2018.

[28] J.-L. Lugrin, M. Ertl, P. Krop, R. Klüpfel, S. Stierstorfer, B. Weisz, M. Rück, J. Schmitt, N. Schmidt, and M. E. Latoschik. Any "body" there? Avatar visibility effects in a virtual reality game. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 17–24. IEEE, 2018.

[29] J.-L. Lugrin, M. E. Latoschik, M. Habel, D. Roth, C. Seufert, and S. Grafe. Breaking bad behaviours: A new tool for learning classroom management using virtual reality. *Frontiers in ICT*, 3:26, 2016.

[30] J.-L. Lugrin, J. Latt, and M. E. Latoschik. Anthropomorphism and illusion of virtual body ownership. In *Proceedings of the 25th International Conference on Artificial Reality and Telexistence and 20th Eurographics Symposium on Virtual Environments*, pages 1–8. Eurographics Association, 2015.

[31] A. Maselli and M. Slater. The building blocks of the full body ownership illusion. *Frontiers in human neuroscience*, 7:83, 2013.

[32] J. McVeigh-Schultz, E. Márquez Segura, N. Merrill, and K. Isbister. What's it mean to "Be Social" in VR?: Mapping the social VR design ecology. In *Proceedings of the 2018 ACM Conference Companion Publication on Designing Interactive Systems (DIS)*, pages 289–294. ACM, 2018.

[33] D.-P. Pertaub, M. Slater, and C. Barker. An experiment on public speaking anxiety in response to three different types of virtual audience. *Presence: Teleoperators & Virtual Environments*, 11(1):68–78, 2002.

[34] S. Poeschl and N. Doering. Measuring co-presence and social presence in virtual environments–psychometric construction of a German scale for a fear of public speaking scenario. *Annual Review of Cybertherapy and Telemedicine*, pages 58–63, 2015.

[35] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling and spatially immersive displays. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 179–188. ACM, 1998.

[36] S. Rehfeld, M. E. Latoschik, and H. Tramberend. Estimating latency and concurrency of asynchronous real-time interactive systems using model checking. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 57–66. IEEE, 2016.

[37] D. Roberts, R. Wolff, J. Rae, A. Steed, R. Aspin, M. McIntyre, A. Pena, O. Oyekoya, and W. Steptoe. Communicating eye-gaze across a distance: Comparing an eye-gaze enabled immersive collaborative virtual environment, aligned video conferencing, and being together. In *IEEE Virtual Reality Conference*, pages 135–142, 2009.

[38] D. Roth, C. Kleinbeck, T. Feigl, C. Mutschler, and M. E. Latoschik. Beyond Replication: Augmenting Social Behaviors in Multi-User Social Virtual Realities. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 215–222, 2018.

[39] D. Roth, K. Waldow, F. Stetter, G. Bente, M. E. Latoschik, and A. Fuhrmann. SIAMC – A socially immersive avatar mediated communication platform. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (VRST)*, pages 357–358. ACM, 2016.

[40] R. Schroeder. *Being There Together: Social interaction in shared virtual environments*. Oxford University Press, 2010.

[41] M. Slater. Grand challenges in virtual environments. *Frontiers in Robotics and AI*, 1:3, 2014.

[42] M. Slater, D. Perez-Marcos, H. Ehrsson, and M. V. Sánchez-Vives. Towards a digital body: the virtual arm illusion. *Frontiers in Human Neuroscience*, 2(6), 2008.

[43] M. Slater, D.-P. Pertaub, and A. Steed. Public speaking in virtual reality: Facing an audience of avatars. *IEEE Computer Graphics and Applications*, 19(2):6–9, 1999.

[44] H. J. Smith and M. Neff. Communication behavior in embodied virtual reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 289:1–289:12. ACM, 2018.

[45] B. Spanlang, J.-M. Normand, D. Borland, K. Kilteni, E. Giannopoulos, A. Pomés, M. González-Franco, D. Perez-Marcos, J. Arroyo-Palacios, X. N. Muncunill, and M. Slater. How to build an embodiment lab: Achieving body representation illusions in virtual reality. *Frontiers in Robotics and AI*, 1:9, 2014.

[46] A. Steed. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 123–129, New York, NY, USA, 2008. ACM.

[47] A. Steed and R. Schroeder. Collaboration in Immersive and Non-immersive Virtual Environments. In *Immersed in Media*, pages 263–282. Springer, 2015.

[48] T. Waltemate, D. Gall, D. Roth, M. Botsch, and M. E. Latoschik. The impact of avatar personalization and immersion on virtual body ownership, presence, and emotional response. *IEEE Transactions on Visualization and Computer Graphics*, 24(4):1643–1652, 2018.

[49] T. Waltemate, F. Hülsmann, T. Pfeiffer, S. Kopp, and M. Botsch. Realizing a low-latency virtual reality environment for motor learning. In *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 139–147. ACM, 2015.

[50] B. G. Witmer and M. J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoperators and virtual environments*, 7(3):225–240, 1998.

[51] N. Yee and J. Bailenson. The Proteus effect: The effect of transformed self-representation on behavior. *Human communication research*, 33(3):271–290, 2007.

Copyright

Author Contributions

The author provided a recording functionality to the system, conducted the latency measurements and helped conducting the study. He provided critical feedback and helped shape the research, the analysis and the manuscript.

7.4 Conclusion

We have shown that latency jitter provokes cybersickness and reviewed literature that finds effects of different latency patterns on cybersickness. An application providing a networked multi-user experience was found to suffer from latency with increased numbers of users. Test subjects were negatively impacted by increased latency. All papers in this chapter show that latency can be a problem when experiencing virtual reality applications.

Discussed Research Questions

R_2 **What are the effects of latency?** We focus on cybersickness as the effect observed in virtual reality systems that is most destructive to the experience. Latency and latency jitter can cause other effects on the experience that were not further investigated.

We show that occasional latency spikes evoke cybersickness. The experiment uses a scaled latency distribution gathered in experiments described in Chapter 3 and Chapter 4. The experience was designed to provoke head movement to make the latency noticeable. The simulated latency spikes were large and results may vary for different severities of spike durations. The experiment, however, is the first to show that random latency spikes can contribute to cybersickness.

A literature review shows that other forms of time invariant latency also contribute to cybersickness. The paper summarises the contents of this thesis and shows how the described research fits into other research in this field: Most research is on time invariant latency with little research on time variant latency. Most research on time variant latency discusses periodic latency behaviour where we describe irregular latency jitter.

Multi user virtual reality applications are prone to impacts of latency. Latency can reduce the enjoyment of these applications.

$R_{2.1}$ **How to simulate latency?** We take up the latency simulation description of the previous chapter and provide a detailled description how to implement it in a game engine instead of in virtual reality middleware.

Chapter 8

DISCUSSION

Latency is an inherent property of processing systems. We showed how to measure it and found that latency behaviour can differ between computer systems. Latency behaviour shows outliers of varying duration. We proposed a way to describe such latency behaviour with a stacked z test. Simulation of latency provides the opportunity to conduct experiments that research effects of latency. We tested the effect of latency spikes on cybersickness and found it to increase sickness symptoms.

We have described our approach to answer the research questions in the last chapters. Most of the discussion has already happened in the reprinted papers. The discussion here is to be seen as an extension to the papers' discussions. The discussion is structured to address the research questions.

$R_{1.1}$ How to measure latency?

In addition to latency jitter, there is also tracking jitter [Tea+09]. Our latency measurement [SNL20b] only interpreted the observed angular difference in terms of motion to photon latency. The angular difference between the Vive tracker and the reported angle on screen may as well be a result of tracking jitter. While the origin is different, we expect this tracking jitter to have a similar effect on users as it leads to a comparable deviation between the expected transform of a controller or HMD to the real transform.

It makes a difference which input and output hardware is chosen to measure motion to photon latency. Input devices use models of their assumed usage to optimise tracking information. Motion controllers expect human hand movements to be bound in acceleration, speed and orientation [Gac19]. They are optimised for this usage. Latency values may not be transferable to other input devices even with the rest of the computer system staying the same.

Warping of the image post rendering allows to adjust the image orientation according to the most recent HMD tracking information [MMB97]. Users react very sensitive to a mismatch between real and virtual head orientation. This approach can reduce the disparity. The shown simulation will still use older tracking information as only the final image is warped. There are multiple latencies in play: the latency from tracking information until it is shown in the image orientation and until it is shown in the simulation. Latency can be different depending not only on in- and output modality but also between different parts in the output image.

$R_{1.2}$ How to describe latency?

Reporting latency needs to happen in a concise and understandable way. We showed that most researchers report only a mean and an optional standard deviation [SNL20a]. This is a big improvement over not reporting latency at all but fails to describe the latency dynamism. We provide a way to describe this behaviour better [SNL16b] and show how other researchers approach this problem [SNL20a]. Neither our, nor any other solution is able to convey latency behaviour in a concise way that can be quickly understood by fellow researchers. Our proposed measuring system [SNL20b] may uncover common latency patterns in the future or determine that the measured systems are too different in their latency behaviour to show common patterns. If it is possible to find common patterns, there may be a latency representation that is both more concise and more expressive than our suggestion. This representation will likely include parts of our description such as the mean latency value, some descriptions of latency value clusters that are larger than the mean and a description of rare, extreme outliers. Our measurements, however, already indicate that there may be no shared patterns. This would necessitate our stacked-z-test visualisation [SNL16b] or a similar approach to properly describe latency behaviour. This is presumably too complex to be used by many other researchers. An alternative could be to turn towards effects. A latency description might not describe the complex latency behaviour anymore but express the amount of cybersickness or the loss in performance it provokes.

R_1 How does latency behave in real-time interactive systems?

We described our measurement data and found different latency behaviour in different systems. Latency reported by other researchers as described in Chapter 7 shares characteristics such as normally distributed outlier clusters and rare extreme outliers. These experiments also show behaviour we did not observe in this way like periodic processes. There needs to be more detailed measurements of varying virtual reality applications to get even more insight. If measuring can get easier and is done more often in a comparable way, it will be possible to determine which of our findings are always present and which are specific to certain systems.

$R_{2.1}$ How to simulate latency?

We described how to simulate latency and applied this simulated latency to conduct an experiment researching latency effects. The problem of the simulation is that it is conducted on top of a system that already shows time variant latencies with spikes. Our experiment used simulated latencies that were much larger than the already existing processing latencies. This justified ignoring the base spiking latency behaviour. If latency is to be simulated in a similar magnitude than the base latency, careful measure-

ment of the system with and without the simulated latency is necessary make sure there are no unexpected effects when a latency spike in the base system amplifies a simulated latency spike.

R_2 What are the effects of latency?

We focused on the effect of latency on cybersickness. There is an effect but it is questionable how problematic it is. Our experiment evoked only minor cybersickness judging from the oral feedback of the test subjects. The effects of latency are most likely to cause concern for long term use or use cases that are sensible to any unforseen interrutions such as psychological therapy. Long term use of virtual reality can allow the effect to increase but this is not yet a common use case in virtual reality. Scientific experiments may get impacted by latency problems but applications for the general commoner may care less. A short exposition to virtual reality in the context of a house demonstration by an architect or the demonstration of a car by a car salesman may not suffer much from increased latency as long as the application is still usable. Many of today's virtual reality applications may ignore the effect of latency by instructing the participants to rest after the short exposure to virtual reality. Playing a game in virtual reality is often done in the free time before going to bed therefore having a natural rest time afterwards. A small uneasiness due to increased latency may be tolerable by many users. If virtual reality devices and applications find their way out of today's niche, the problem may become more pressing again.

A different view is described by [Adh+20] who argue that "due to recent advances, tracking inaccuracy and rendering latency are no longer significant causes of visually induced VR sickness on consumer VR platforms"[Adh+20, p. 2]. Advances in hard and software indeed reduce the possible minimum latency. The increased performance, however, is often used to present more elaborated experiences whose computational demand drive up the systems' latencies. Generous buffers alleviate the impact of latency spikes but those buffers are often sacrificed to increase the impressivity of virtual scenes or drive the simulation power of the engine up. Our research describes that counting on modern hard- and software alone doesn't need to solve the latency problem by itself. The educated researcher needs to still make sure that latency stays small.

How small is a question we only have touched briefly. Caserman, Martinussen, and Göbel [CMG19] recommend an end-to-end latency below 58 ms to keep sickness symptoms low but a latency below 101 ms to not reduce the sense of body ownership. The question for a recommended latency threshold therefore depends on the chosen metric. The thresholds reported in research papers are time invariant latency values. We have shown that latency jitter can have an impact as well but do not yet know how this impacts the recommended latency thresholds.

Chapter 9

CONCLUSION

This thesis showed that latency in virtual reality systems can have negative effects on the users' experience. We have shown that latency can be one factor contributing to cybersickness, a sickness caused by exposition to virtual reality applications. Most research describes only the effect of a time invariant latency but latency behaves with outliers of varying size. We showed how to measure latency and how the measurements look like: the majority of latency measurements gather close to the mean value with multiple clusters of larger latencies and occasional outliers. Latency is an inherent problem as a result of evermore complexity in our computing machines. The only solution is to be careful in designing systems and applications and making them fast enough so the occasional latency spike doesn't impact the system as much. Measuring latency can make the developer aware that there are problems in the application that need to be fixed.

The thesis consists of multiple peer reviewed and published papers that demonstrate how latency can behave, how to measure it, how to simulate it and what effects can be. The latency problem of virtual reality systems is not solved yet and probably never will. It is a problem that is inherent to the used systems and always needs consideration. The thesis furthers the scientific understanding and provides a base on which to build upon. The proposed approaches need to get developed further to derive more insight which then will result in more refinement.

The new surge in popularity of virtual reality systems fires interest in latency research. Our review of latency measurement approaches saw the majority of research papers appear in the recent past [SNL20a]. Future work will build upon our work to further the insight into latency and its effects. The most obvious further work is to provide easier access to detailed measurement systems as described in Stauffert, Niebling, and Latoschik [SNL20b]. New developments and use cases of virtual reality will open new possibilities but also new ways that latency can interfere with the new experiences. The most obvious upcoming use case is prolonged use of virtual reality that will see issues not yet observed in the current day short exposures.

Bibliography

[Adh+20] Isayas Berhe Adhanom, Nathan Navarro Griffin, Paul MacNeilage, and Eelke Folmer. "The Effect of a Foveated Field-of-view Restrictor on VR Sickness". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE. 2020, pages 645–652.

[Ans73] Francis J Anscombe. "Graphs in statistical analysis". In: *The american statistician* 27.1 (1973), pages 17–21.

[Bac00] Alexander Backlund. "The definition of system". en. In: *Kybernetes* 29.4 (June 2000), pages 444–451. issn: 0368-492X. doi: 10.1108/03684920010322055. url: https://www.emerald.com/insight/content/doi/10.1108/03684920010322055/full/html (visited on 08/11/2020).

[BAG18] Armin Becher, Jens Angerer, and Thomas Grauschopf. "Novel Approach to Measure Motion-To-Photon and Mouth-To-Ear Latency in Distributed Virtual Reality Systems". en. In: *arXiv:1809.06320 [cs]* (Sept. 2018). arXiv: 1809.06320. url: http://arxiv.org/abs/1809.06320 (visited on 10/07/2019).

[BBL01] Guillem Bernat, Alan Burns, and Albert Liamosi. "Weakly hard real-time systems". In: *IEEE transactions on Computers* 50.4 (2001), pages 308–321.

[BLM91] Asghar Bashteen, Ivy Lui, and Jill Mullan. "A superpipeline approach to the MIPS architecture". In: *COMPCON Spring'91*. IEEE Computer Society. 1991, pages 8–9.

[Blu+94] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. "The minimum latency problem". en. In: *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing - STOC '94*. Montreal, Quebec, Canada: ACM Press, 1994, pages 163–171. isbn: 978-0-89791-663-9. doi: 10.1145/195058.195125. url: http://portal.acm.org/citation.cfm?doid=195058.195125 (visited on 01/07/2020).

[BRC96] Grigore Burdea, Paul Richard, and Philippe Coiffet. "Multimodal virtual reality: Input-output devices, system integration, and human factors". In: *International Journal of Human-Computer Interaction* 8.1 (1996), pages 5–24.

[Bur91] Alan Burns. "Scheduling hard real-time systems: a review". In: *Software Engineering Journal* 6.3 (1991), pages 116–128.

[Cho+18] Song-Woo Choi, Siyeong Lee, Min-Woo Seo, and Suk-Ju Kang. "Time Sequential Motion-to-Photon Latency Measurement System for Virtual Reality Head-Mounted Displays". en. In: *Electronics* 7.9 (Sept. 2018), page 171. issn: 2079-9292. doi: 10.3390/electronics7090171. url: http://www.mdpi.com/2079-9292/7/9/171 (visited on 08/19/2020).

[CMG19] Polona Caserman, Michelle Martinussen, and Stefan Göbel. "Effects of End--to-end Latency on User Experience and Performance in Immersive Virtual Reality Applications". In: *Joint International Conference on Entertainment Computing and Serious Games*. Springer. 2019, pages 57–69.

[CS06] Jichuan Chang and Gurindar S Sohi. "Cooperative caching for chip multiprocessors". In: *ACM SIGARCH Computer Architecture News* 34.2 (2006), pages 264–276.

[CSA00] Neal Cardwell, Stefan Savage, and Thomas Anderson. "Modeling TCP latency". en. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Volume 3. Tel Aviv, Israel: IEEE, 2000, pages 1742–1751. isbn: 978-0-7803-5880-5. doi: 10.1109/INFCOM.2000.832574. url: http://ieeexplore.ieee.org/document/832574/ (visited on 08/06/2020).

[Cze+97] Marek Czernuszenko, Dave Pape, Daniel Sandin, Tom DeFanti, Gregory L Dawe, and Maxine D Brown. "The ImmersaDesk and Infinity Wall projection-based virtual reality displays". In: *ACM SIGGRAPH Computer Graphics* 31.2 (1997). Publisher: ACM New York, NY, USA, pages 46–49.

[DL10] Massimiliano Di Luca. "New method to measure end-to-end delay of virtual reality". In: *Presence* 19.6 (2010), pages 569–584. url: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6797715 (visited on 02/04/2016).

[Eng09] Irv Englander. "The Architecture of Computer Hardware, System Software, and Networking". In: *An Information Technology Approach* 11 (2009).

[Far18] Matteo Farinella. "The potential of comics in science communication". en. In: *Journal of Science Communication* 17.01 (Jan. 2018). issn: 1824-2049. doi: 10.22323/2.17010401. url: https://jcom.sissa.it/archive/17/01/JCOM_1701_2018_Y01 (visited on 08/25/2020).

[FE20] Ilja T. Feldstein and Stephen R. Ellis. "A Simple, Video-Based Technique for Measuring Latency in Virtual Reality or Teleoperation". In: *IEEE Transactions on Visualization and Computer Graphics* (2020). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pages 1–1. issn: 1941-0506. doi: 10.1109/TVCG.2020.2980527.

[Gac19] Ethan Gach. *Valve Updates Steam VR Because Beat Saber Players Are Too Fast*. en-us. Library Catalog: kotaku.com. Nov. 2019. url: https://kotaku.com/valve-updates-steam-vr-because-beat-saber-players-are-t-1832536574 (visited on 06/09/2020).

[He+00] Ding He, Fuhu Liu, Dave Pape, Greg Dawe, and Dan Sandin. "Video-based measurement of system latency". In: *International Immersive Projection Technology Workshop*. 2000. url: http://www.researchgate.net/profile/Dave_Pape/publication/2628137_Video-Based_Measurement_of_System_Latency/links/542166dc0cf203f155c66ad6.pdf (visited on 12/11/2015).

[HVCT10] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-TIN. "How much anonymity does network latency leak?" en. In: *ACM Transactions on Information and System Security* 13.2 (Feb. 2010), pages 1–28. issn: 1094-9224, 1557-7406. doi: 10.1145/1698750.1698753. url: https://dl.acm.org/doi/10.1145/1698750.1698753 (visited on 08/06/2020).

[JW89] Norman P Jouppi and David W Wall. "Available instruction-level parallelism for superscalar and superpipelined machines". In: *ACM SIGARCH Computer Architecture News* 17.2 (1989), pages 272–282.

[Kä+17] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. "Dissecting the End-to-end Latency of Interactive Mobile Video Applications". en. In: *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications - HotMobile '17*. Sonoma, CA, USA: ACM Press, 2017, pages 61–66. isbn: 978-1-4503-4907-9. doi: 10.1145/3032970.3032985. url: http://dl.acm.org/citation.cfm?doid=3032970.3032985 (visited on 01/30/2020).

[Lat+19] Marc Erich Latoschik, Florian Kern, Jan-Philipp Stauffert, Andrea Bartl, Mario Botsch, and Jean-Luc Lugrin. "Not Alone Here?! Scalability and User Experience of Embodied Ambient Crowds in Distributed Social Virtual Reality". In: *IEEE Transactions on Visualization and Computer Graphics* 25.5 (2019), pages 2134–2144.

[Lim98] Patricia Nelson Limerick. "Dancing With Professors: The Trouble With Academic Prose". In: *Negotiating academic literacies: Teaching and learning across languages and cultures* (1998), page 199.

[Mac] *LATENCY (noun) definition and synonyms | Macmillan Dictionary*. en. url: https://www.macmillandictionary.com/dictionary/british/latency (visited on 08/06/2020).

[McK08] Paul E McKenney. "'Real Time'vs.'Real Fast': How to Choose?" In: *Ottawa Linux Symposium (July 2008), pp. v2*. 2008, pages 57–65.

[MF17] Justin Matejka and George Fitzmaurice. “Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pages 1290–1294.

[Min93] Mark Mine. “Characterization of end-to-end delays in head-mounted display systems”. In: *The University of North Carolina at Chapel Hill, TR93-001* (1993). url: http://www0.cs.ucl.ac.uk/teaching/VE/Papers/93-001.pdf (visited on 12/11/2015).

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. “Post-rendering 3D warping”. In: *Proceedings of the 1997 symposium on Interactive 3D graphics*. ACM, 1997, 7–ff. url: http://dl.acm.org/citation.cfm?id=253292 (visited on 02/04/2016).

[Nic97] Serge Nicolas. “On the speed of different senses and nerve transmission by Hirsch (1862)”. In: *Psychological Research* 59.4 (1997), pages 261–268.

[Oxf] *latency noun - Definition, pictures, pronunciation and usage notes | Oxford Advanced Learner’s Dictionary at OxfordLearnersDictionaries.com*. url: https://www.oxfordlearnersdictionaries.com/definition/english/latency?q=latency (visited on 08/06/2020).

[Pap+20] Sebastian Pape, Marcel Krüger, Jan Müller, and Torsten W. Kuhlen. “Calibratio: A small, low-cost, fully automated Motion-to-Photon Measurement Device”. In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. Mar. 2020, pages 234–237. doi: 10.1109/VRW50115.2020.00050.

[Pat04] David A. Patterson. “Latency lags bandwith”. en. In: *Communications of the ACM* 47.10 (Oct. 2004), pages 71–75. issn: 0001-0782, 1557-7317. doi: 10.1145/1022594.1022596. url: https://dl.acm.org/doi/10.1145/1022594.1022596 (visited on 08/06/2020).

[Pat+95] R Hugo Patterson, Garth A Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. “Informed prefetching and caching”. In: *Proceedings of the fifteenth ACM symposium on Operating systems principles*. 1995, pages 79–95.

[Pin+02] Axel Pinz, Markus Brandner, Harald Ganster, Albert Kusej, Peter Lang, and Miguel Ribo. “Hybrid tracking for augmented reality”. In: *Ögai Journal* 21.1 (2002), pages 17–24.

[PMK11] Giorgos Papadakis, Katerina Mania, and Eftichios Koutroulis. “A system to measure, control and minimize end-to-end head tracking latency in immersive simulations”. en. In: *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry - VRCAI ’11*. Hong Kong, China: ACM Press, 2011, page 581. isbn: 978-1-4503-1060-4. doi: 10.

1145/2087756.2087869. url: http://dl.acm.org/citation.cfm?doid=2087756.2087869 (visited on 01/23/2020).

[Pur+98] Gopathy Purushothaman, Saumil S. Patel, Harold E. Bedell, and Haluk Ogmen. "Moving ahead through differential visual latency". en. In: *Nature* 396.6710 (Dec. 1998), pages 424–424. issn: 0028-0836, 1476-4687. doi: 10.1038/24766. url: http://www.nature.com/articles/24766 (visited on 08/06/2020).

[Qua+18] Nhon Quach, Moinul Khan, Maurice Ribble, Martin Renschler, Mehrad Tavakoli, Rashmi Kulkarni, Ricky Wai Kit Yuen, and Todd Lemoine. *Systems and methods for reducing motion-to-photon latency and memory bandwidth in a virtual reality system*. US Patent 9,858,637. 2018.

[RSL19] Daniel Roth, Jan-Philipp Stauffert, and Marc Erich Latoschik. "Avatar Embodiment, Behavior Replication, and Kinematics in Virtual Reality". In: *VR Developer Gems, 1st Edition*. Edited by William R. Sherman. A K Peters/CRC Press, 2019, pages 321–346. url: https://www.taylorfrancis.com/books/e/9781315157764/chapters/10.1201/b21598-17.

[Rum+11] Stephen M Rumble, Diego Ongaro, Ryan Stutsman, Mendel Rosenblum, and John K Ousterhout. "It's Time for Low Latency". en. In: (2011), page 5.

[Sie+07] Tobias Sielhorst, Wu Sa, Ali Khamene, Frank Sauer, and Nassir Navab. "Measurement of absolute latency for video see through augmented reality". In: *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. ISSN: null. Nov. 2007, pages 215–220. doi: 10.1109/ISMAR.2007.4538850.

[SNL16a] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems". In: *2016 IEEE Virtual Reality (VR)*. IEEE. 2016, pages 287–288.

[SNL16b] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Towards comparable evaluation methods and measures for timing behavior of virtual reality systems". In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. 2016, pages 47–50.

[SNL16c] Jan-Pilipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Reducing application-stage latencies for real-time interactive systems". In: *2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE. 2016, pages 1–7.

[SNL17] Jan-Philipp Stauffert, F. Niebling, and M. E. Latoschik. "A Latency and Latency Jitter Simulation Framework with OSVR". In: *2017 IEEE 10th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. 2017, pages 1–7.

[SNL18] Jan-Philipp Stauffert, F. Niebling, and M. E. Latoschik. "Effects of Latency Jitter on Simulator Sickness in a Search Task". In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2018, pages 121–127.

[SNL20a] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Latency and Cybersickness: Impact, Causes, and Measures. A Review". In: *Frontiers in Virtual Reality* 1 (2020), page 31. issn: 2673-4192. doi: 10.3389/frvir.2020.582204. url: https://www.frontiersin.org/article/10.3389/frvir.2020.582204.

[SNL20b] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. "Simultaneous Run-Time Measurement of Motion-to-Photon Latency and Latency Jitter". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2020, pages 636–644.

[SP+15] Matthew E. St. Pierre, Salil Banerjee, Adam W. Hoover, and Eric R. Muth. "The effects of 0.2Hz varying latency with 20–100ms varying amplitude on simulator sickness in a helmet mounted display". en. In: *Displays* 36 (Jan. 2015), pages 1–8. issn: 01419382. doi: 10.1016/j.displa.2014.10.005. url: http://linkinghub.elsevier.com/retrieve/pii/S0141938214000791 (visited on 01/15/2018).

[Sta+20] Jan-Philipp Stauffert, Florian Niebling, Jean-Luc Lugrin, and Marc Erich Latoschik. "Guided Sine Fitting for Latency Estimation in Virtual Reality". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, pages 706–707.

[Sta+21] Jan-Philipp Stauffert, Kristof Korwisi, Florian Niebling, and Marc Erich Latoschik. "Ka-Boom!!! Visually Exploring Latency Measurements for XR". In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI EA '21. Yokohama, Japan: Association for Computing Machinery, 2021. isbn: 9781450380959210 5. doi: 10.1145/3411763.3450379. url: https://doi.org/10.1145/3411763.3450379.

[Sta+95] John A Stankovic, Marco Spuri, Marco Di Natale, and Giorgio C Buttazzo. "Implications of classical scheduling results for real-time systems". In: *Computer* 28.6 (1995), pages 16–25.

[Ste08] Anthony Steed. "A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations". In: *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*. VRST '08. New York, NY, USA: ACM, 2008, pages 123–129. isbn: 978-1-59593-951-7. doi: 10.1145/1450579.1450606. url: http://doi.acm.org/10.1145/1450579.1450606.

[Sut65] Ivan E Sutherland. "The ultimate display". In: *Multimedia: From Wagner to virtual reality* 1 (1965).

[Tea+09] Robert J Teather, Andriy Pavlovych, Wolfgang Stuerzlinger, and I Scott MacKenzie. "Effects of tracking technology, latency, and spatial jitter on object movement". In: *2009 IEEE Symposium on 3D User Interfaces*. 2009, pages 43–50.

[Ten+11] Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. "How to Grow a Mind: Statistics, Structure, and Abstraction". en. In: *Science* 331.6022 (Mar. 2011), pages 1279–1285. issn: 0036-8075, 1095-9203. doi: 10.1126/science.1192788. url: https://www.sciencemag.org/lookup/doi/10.1126/science.1192788 (visited on 08/14/2020).

[Wal+16] Thomas Waltemate, Irene Senna, Felix Hülsmann, Marieke Rohde, Stefan Kopp, Marc Ernst, and Mario Botsch. "The impact of latency on perceptual judgments and motor performance in closed-loop interaction in virtual reality". en. In: ACM Press, 2016, pages 27–35. isbn: 978-1-4503-4491-3. doi: 10.1145/2993369.2993381. url: http://dl.acm.org/citation.cfm?doid=2993369.2993381 (visited on 03/17/2018).

[Wav16] J. M. P. van Waveren. "The asynchronous time warp for virtual reality on consumer hardware". en. In: *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology - VRST '16*. Munich, Germany: ACM Press, 2016, pages 37–46. isbn: 978-1-4503-4491-3. doi: 10.1145/2993369.2993375. url: http://dl.acm.org/citation.cfm?doid=2993369.2993375 (visited on 06/09/2020).

[WDH13] Weixin Wu, Yujie Dong, and Adam Hoover. "Measuring Digital System Latency from Sensing to Actuation at Continuous 1-ms Resolution". en. In: *Presence: Teleoperators and Virtual Environments* 22.1 (Feb. 2013), pages 20–35. issn: 1054-7460, 1531-3263. doi: 10.1162/PRES_a_00131. url: http://www.mitpressjournals.org/doi/10.1162/PRES_a_00131 (visited on 03/10/2018).

[Wen+20] Stephan Wenninger, Jascha Achenbach, Andrea Bartl, Marc Erich Latoschik, and Mario Botsch. "Realistic Virtual Humans from Smartphone Videos." In: *VRST*. Edited by Robert J. Teather, Chris Joslin, Wolfgang Stuerzlinger, Pablo Figueroa, Yaoping Hu, Anil Ufuk Batmaz, Wonsook Lee, and Francisco Ortega. ACM, 2020, 29:1–29:11. url: http://dblp.uni-trier.de/db/conf/vrst/vrst2020.html#WenningerABLB20.

[Wik20] Wikipedia. *System*. en. Page Version ID: 971111227. Aug. 2020. url: https://en.wikipedia.org/w/index.php?title=System&oldid=971111227 (visited on 08/14/2020).

[WM98] David Whitney and Ikuya Murakami. "Latency difference, not spatial extrapolation". en. In: *Nature Neuroscience* 1.8 (Dec. 1998), pages 656–657. issn: 1097-6256, 1546-1726. doi: 10.1038/3659. url: http://www.nature.com/articles/nn1298_656 (visited on 08/06/2020).