Co-Located Augmented and Virtual Reality Systems

by

Connor DeFanti

a dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May, 2019

Ken Perlin

Acknowledgments

I would like to thank first and foremost my advisor, Ken Perlin, who has continuously inspired this vision of the future of technology. I would like to thank Davi Geiger and Daniele Panozzo for helping me throughout the thesis process, and David K. A. Mordecai, Win Burleson, and Denis Zorin for also being a critical part of the thesis committee.

I am thankful for the members of the Future Reality Lab, past, present, and future. They have been infinitely helpful in all of my projects, and their brilliance drives us towards the future every day.

Abstract

Augmented and Virtual Reality (AVR) systems have become increasingly popular in the worlds of entertainment and industry. However, many current systems are limited in scope to experiences that isolate a single user within a given physical space. While many such experiences allow for interactions between remotely located users, very few experiences allow for multiple users to coexist in the same physical space while interacting with a consistent world-view of shared virtual objects. Our research has found that by enabling this co-located paradigm, users are able to have rich interactions that are otherwise impossible. This thesis presents a series of experiments that demonstrate the importance of the social aspects of co-located AVR, a set of solutions that overcome the difficulties often encountered in such experiences, and directions for future scalability using forthcoming hardware and technologies.

Contents

Listing of figures

Introduction

For as long as computer graphics have been around, the idea of virtual reality (VR) has been a topic of interest. The concept of graphics becoming seamlessly integrated with our world has been the focus of research pieces and science fiction stories alike. However, in the past, exorbitant prices for hardware systems or inscalability of solutions have prevented augmented and virtual reality (AVR) systems from becoming a mainstream technology.

In recent years, advancements in technologies affecting mobile processing, screen pixel density, optics, wireless transmissions, and more have pushed AVR systems into consumer-level viability. These advancements have caused a rapid acceleration in both interest and funding for AVR systems[69]. We believe the current developmental trajectory of these systems points towards widespread use, much like how advancements a decade ago brought high-power computation and communication to our pockets in smartphones.

We envision a future where AVR devices are used in everyday communica-

tion, enabling methods of collaboration, content sharing, and communication not possible with today's technological paradigms. In our daily conversations, we will be able to enhance our discussions with immersive computer graphics that feel like they are a part of reality. Professionals in education, industry, medicine, entertainment, and more will all have access to create floating, 3D visualizations to aid the communication of their ideas.

However, for that growth to continue, we believe that AVR systems must evolve beyond the current paradigm of isolated experiences. For the past several years, my work at the Future Reality Lab (FRL) at New York University has focused on developing techniques to enable AVR systems that are specifically designed for interpersonal interactions for users within the same space. This document will discuss the techniques and design decisions our lab has developed over the past 5 years to enable these systems, as well as our evaluations of the systems.

In Chapter 1, we will discuss a few examples of systems and experiences we have developed with this paradigm, focusing on the design choices and evaluations based on user feedback. In Chapter 2, we will cover the algorithms and techniques used to overcome the problems that arise when building these systems. Finally, Chapter 3 will discuss emerging technologies and how they will benefit these systems going forward into the future.

A Brief Introduction to AVR Systems

While the focus of this dissertation will be on recent AVR systems, it is worth covering the history of the development of AVR in order to gain context on what past systems accomplished, as well as the limitations that prevented them from becoming mainstream technology. There are several devices which could be considered the first "Virtual Reality" system, depending on definition, but it is commonly accepted that the first functional Virtual Reality head-mounted display (HMD) was developed by Ivan Sutherland in 1968[79]. The device, hoisted above the user's head by a mechanical arm known as the Sword of Damocles, was able to render simple vector graphics in front of the user. The mechanical arm allowed the host computer to track the user's perspective, thus rendering the user with the appropriate virtual view. While this served as a first prototype of the HMDs that we see today, the sheer size of the device made it impossible to scale to widespread use.

Developments over the next several decades produced many different VR devices for various applications, but none became a commercially viable mainstream product. Much like Sutherland's device, these devices paved the way for future developments, but hardware limitations made them bulky, expensive, low quality, or otherwise commercially unviable. Until the 2010's, very few AVR devices were commercially available, and those that were never found mainstream success.

The last decade has seen enormous growth in terms of AVR development, at

both the consumer and industrial levels. Several generations of headsets developed by Facebook Oculus[20], Google[31], HTC/Valve[38], and others began bringing virtual reality to widespread viability in 2013 via affordable HMDs. For VR, early developments saw two branching paths:

- Untethered, *3 degrees of freedom (DOF)* headsets. 3DOF refers to tracking the three rotational axes. This sort of system is ideal for applications like *360 video*[i]. Examples of this are the Samsung GearVR[58], Google Cardboard[31], and Oculus Go[19].

- *6DOF* headsets that are tethered to an external computer. 6DOF refers to tracking that supports both position and rotation tracking. Many current VR games have moved to 6DOF headsets due to the freedom to move throughout a room. Examples of this are the Oculus Rift[22] and the HTC Vive[38].

Throughout 2016 to 2019, improvements in tracking algorithms began to allow for untethered, room-scale experiences, as with the Oculus Quest[21], Google Lenovo Mirage Solo[32], and HTC Vive Focus[39]. However, even at present day, this technology is still reaching maturation and still has its problems. Many current 6DOF systems allow a user to walk throughout a space and be correctly tracked. However, that space can be limited in size. The limitations on the size of the tracking area varies by device. For augmented reality (AR), 6DOF systems[52,48,54] have reached developmental stages that allow for experimentation

[i]360 video refers to a video recording in which every direction is recorded simultaneously. In a 360 video VR experience, a user can virtually view the video at any angle by turning their head, but they cannot move their virtual head position.

and content development, but currently are not yet widely accessible to consumers.

Despite the limitations, entertainment studios and research facilities have not hesitated to take advantage of state-of-the-art hardware, using this hardware to prototype experiences of what the future of media and interaction may look like. This is akin to the bulky devices like the kinetoscope[75] allowing for glimpses into the possibilities of motion pictures, which would eventually become a widespread cultural phenomena. Similarly, many experiences, ranging from movies in 360 video[64,10] to fully interactive room-scale games have reached critical acclaim in recent years. These experiences seek to build and evaluate the paradigms of different VR systems and their role in the entertainment industry. Similarly, accessibility to these devices has boosted the research output in fields related to AVR, including *human-computer interaction* (HCI) and *user experience* (UX), hardware, tracking and vision algorithms, and more[ii].

Many of the recent developments for AVR systems have focused on logistical problems that have been common in them since 1968, and have found solutions to these problems in both hardware and algorithm solutions. In the case of HMDs, recent innovations have allowed users to comfortably view a screen inches away from their face. Creating screens with high enough pixel denisty, which is now commonplace in smartphones, and innovations in optics[37,61,76] have allowed for these high-resolution, HMD-based VR experiences.

[ii]There are many conferences and journals that publish research focused within these areas. While many exist within the field, examples of notable ones are SIGGRAPH, SIGCHI, UIST, and IEEE VR. We recommend the reader consults the proceedings of these conferences for further reading.

VR Systems and Motion Sickness

For VR systems, poorly designed or calibrated systems can cause a phenomenon known as simulator sickness or *VR sickness*[5,45]. Humans use several different systems to coordinate balance. These systems include the proprioception of muscular movement[4], the vestibular system within the inner ear which detects accleration and rotational movements[49], and visual cues. These systems act together within the human body to create a uniform perception of the world, and therefore misinformation to one or more systems can cause nausea, vertigo, or disorientation[55]. Because many VR systems completely occlude and replace a person's vision with a virtualized view, it has been shown that misrepresenting user acceleration in a virtual environment can cause VR sickness[43].

A large amount of research in the space of VR systems has focused on combating and reducing VR sickness. Some research has found that VR sickness can be reduced with simple changes, such as rendering a shape in the user's field of view that represents their nose[86]. Other studies have found that reducing a user's field of view angle can also inhibit VR sickness[12], but this reduces immersion since the virtual rendering mismatches a person's real field of view angle. As a compromise, the work of Fernandes and Feiner[24] suggests dynamically adapting user field of view angle such that the angle is lowered while the user's head is in motion. While some research has shown that slight modifications to motion trajectories are acceptable[78], one of the most important ways to reduce VR sickness is to ensure that a user's head motion is accurately repre-

sented in the virtual world.

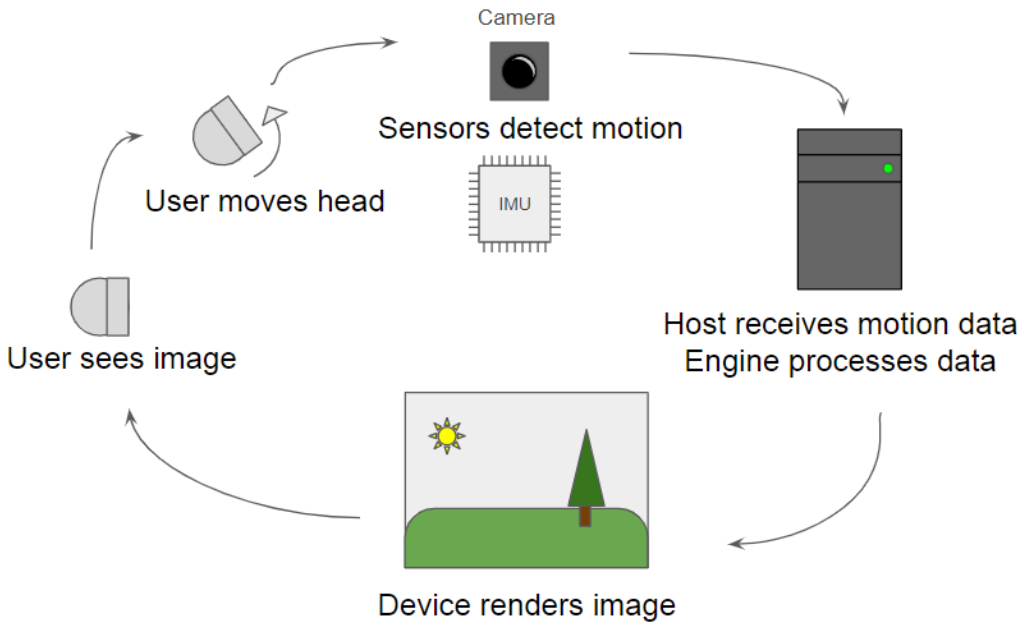

Figure 1: The motion-to-photon loop: starting with a user's head motion, sensors must capture that motion, send that data to a host processor, process that data, and have a graphics engine render the perceived motion. All of this must be done with very low latency to prevent motion sickness.

Addressing and resolving VR sickness requires several different components to be working correctly within the system. First, the graphical rendering processor must render the user's perspective at a sufficient frame rate such that their rendered perspective is not far behind the reality of their motion. This delay is known as *motion-to-photon* latency. This latency starts with the user's motion, which is then detected by a tracking system[iii], delivered to the AVR device, pro-

[iii]Tracking systems refer to both internal sensors, such as accelerometers or gyroscopes, and

cessed by a graphics engine, written into a pixel buffer, and finally displayed on the screen. This loop is visualized in Figure 1. Large amounts of research have gone into minimizing motion-to-photon latency, such as the work of Oculus TimeWarp[50,18].

While different systems have different specifications, it is generally believed that *presence*, *i.e.* the feeling of existing inside your virtual environment, requires 20 milliseconds (ms) or less[59]. Some VR systems require their applications to run at 60 frames per second (FPS), corresponding to approximately 17ms per frame, while others require 72FPS (14ms per frame)[21]. This is to ensure that the user viewpoint is accurately reflected, and the rendered viewpoint does not lag behind their head motion. Meeting this requirement comes down to having a combination of sufficiently powerful hardware coupled with optimized graphics. This is especially true for mobile VR development, where processors are much weaker than those of high-powered desktops.

Secondly, ensuring that users are smoothly tracked within the system is critical. Tracking systems that introduce large amounts of variance will also cause VR sickness due to the mismatch of perceived and real motion. In order for the rendered view to accurately reflect the user's motion, many documents recommend sub-pixel accuracy to minimize VR sickness[59,1]. Translating "sub-pixel accuracy" to rotational and translational error is useful for assessing which tracking methods are acceptable. If we take the Samsung GearVR, for example, which has a field of view angle of 101°, and a resolution of 1024x1024 pixels

external sensors, such as cameras.

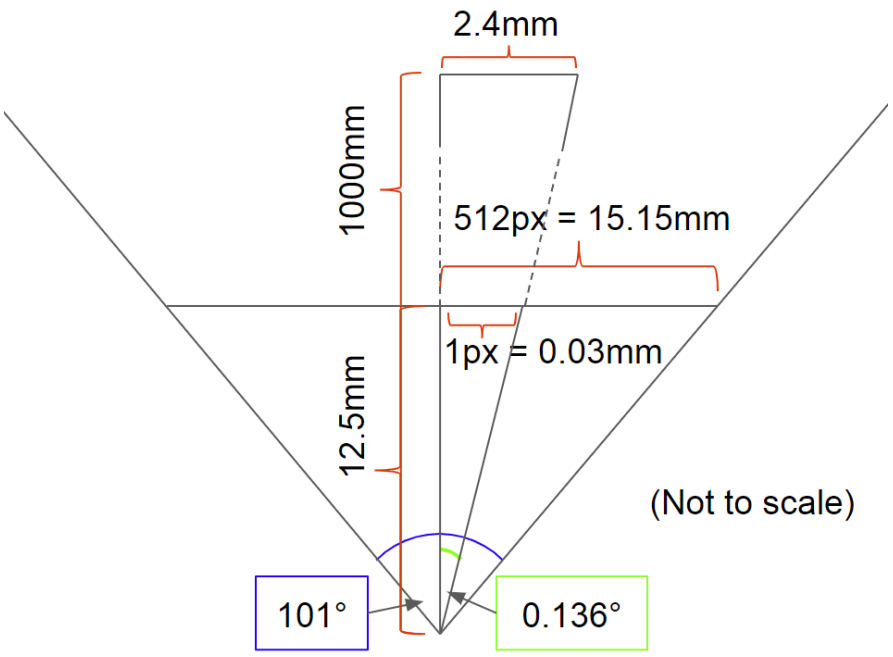

Figure 2: Understanding sub-pixel accuracy: The Samsung GearVR, with a specified field of view angle of 101°, screen resolution of 1024x1024 pixels, and a pixel density of 33.8 pixels/mm, will require an object 1 meter away from the viewer to be accurate within 2.4mm and 0.136°.

per eye at 858 pixels per inch (or approximately 33.8 pixels/mm)[72], then we can calculate exactly how precisely our rendered image must match the user's head motion in terms of angular and positional accuracy. The field of view tells us that our virtual camera is approximately 12.5mm behind the screen. Based on the pixel density, can see that 1 pixel corresponds to approximately 0.03mm on the screen. Using these numbers, we can calculate that we must be accurate to within approximately 0.136°in terms of angular precision to be within 1 pixel of accuracy. Because of the mathematics of perspective projection, calculating

positional accuracy depends on how far away we are looking from the virtual screen[iv]. For an object 1 meter away from the user, given the angular accuracy we need, we can calculate that we will need to be accurate within approximately 2.4mm in terms of positional accuracy. Figure 2 illustrates these calculations. While nearer objects require higher positional accuracy to be sub-pixel accurate, we can see that we have much more leeway in the positional tracking than rotational tracking, although both must be accurate.

While these requirements are generally less strict in AR, as those systems do not completely occlude the user's vision, they are still important to consider. Just as the idea of presence applies for transporting a user to another world in a virtual environment, AR has a concept of presence of the virtual objects; if the sense of presence is maintained, then it will feel as though those objects are actually part of the world. Maintaining their position to be accurate within the world is key to maintaining this presence[1], and thus the tracking and rendering requirements discussed above are still important.

As we discuss the work presented throughout this dissertation, it will be important to keep these limitations and requirements in mind, as they motivate many of the techniques employed to make smooth AVR systems.

[iv] Perspective projection simulates depth and parallax within a virtual environment. This means that objects that are further away appear smaller, and thus as a user moves, objects that are further away appear to move less than objects that are close to the user.

Co-Located AVR

As AVR systems become more and more common, it is only natural that more and more people will want to use these systems, and in particular, use them to interact with one another. We can see this evolving paradigm with the evolution of VR applications such as VRChat[84], or AR functionality in social applications like games[57] and communication[77].

There are two different ways one can create a multi-user AVR environment. The first is when each user is in a different physical space, and the AVR systems are rendered such that the users share their virtual space. This implementation is known as *telepresence*[v], and a large portion of research and development has been focused on enabling telepresence[92,80]. However, this dissertation largely does not concern itself with implementations for telepresence.

The alternative to users occupying different physical spaces is having multiple people use AVR systems physically within the same room. We call this *co-located AVR*. It is important to note that telepresence and co-location are not mutually exclusive; it is entirely possible to have a system with some users co-located and others telepresent. The focus of our lab, and this dissertation, is on how we can develop co-located AVR systems so that we can discover the benefits of such systems while also solving the problems that arise when designing them.

There have been many demonstrations and experiences that have implemented

[v]Telepresence was first coined by Marvin Minsky[53] as a way for a user to remotely work in another location as though they were physically present.

co-located VR. In the 1990's, the Cave Automated Virtural Environment (CAVE)[9,23] implemented this sort of environment by projecting virtual worlds onto the sides of a room. However, this does not create a unique viewpoint for each user like HMD-based solutions do. More recent efforts, such as The Void[82], Dreamscape[13], and Hologate[36] have created co-located, HMD-based experiences for a small audience, much like some of our early works presented later in Chapter 1. However, these systems encounter problems when trying to scale up to larger audiences.

One of the common problems is that each user must be aware of the other users' locations. Failure to synchronize these positions across users results in the danger of collision. If the user trusts that they will not collide into another user without visual confirmation, then they can act with greater freedom in VR. Prior work has found that many different avatar styles can work for representing a person in virtual space while giving sufficient presence for others to not collide with them[34]. In our various experiences, we have experiemented with many different avatar representations. Our standout examples will be discussed in Chapter 1.

While AR systems do not have these safety concerns, as the users can see their environment and each other, it is important for users to be able to share graphics in the same location if desired. Resolving differences in frame, or the virtual space in which the device exists, allows for users to interact with virtual objects in the same physical space. This is particularly important for virtual collaboration, where two or more co-located users may want to manipulate or

observe the same object. While some research has explored virtual collaborative manipulation without physical co-location[89,60], other use-cases require users to interact within the same physical space. In such applications, coordination of the virtual spaces is important. Synchronization techniques for AR applications have been studied and developed by Microsoft[51], Google[33], and Apple[2]. Our techniques for synchronizing difference in user frames will be discussed in Chapter 2.

Furthermore, many current visual-based tracking algorithms rely on line-of-sight from a sensor to a tracker. This is true for *outside-in* solutions where external sensors, such as motion capture or infrared cameras, are tracking VR headsets and clients, as well as with *inside-out* solutions, like common Simultaneous Localization and Mapping (SLAM)[56,16] techniques. In either case, the tracking algorithm relies on line-of-sight from some sensor to a target, whether that target is an explicit tracker, or a landmark feature. When creating a system with many users physically inhabiting the space, occlusion becomes a problem for these visual tracking algorithms.

However, when these problems are addressed, co-located AVR systems can enable rich interactions between people that would otherwise be impossible. Collaboration, communication, and entertainment are just some possibilities of interactions enhanced by co-located virtual graphics. We will explore some of these interactions in the following chapters. Additionally, we will address the above problems in the context of the experiences we developed, along with the solutions we designed to overcome them.

1

Experiments in Co-Located AVR

Introduction

Starting in 2014, a growing interest in VR coupled with innovations in optics and mobile technology led to the release of several 360 video HMDs, such as the Samsung GearVR[58] and the Google Cardboard[31]. While these were by no means the first attempts at consumer-level VR, they began a new era of VR

development due to the increase in quality compared to their predecessors.

These devices offered low-cost, untethered 3DOF tracking through the use of high-quality inertial measurement units (IMU). This meant that users could visually survey an experience without experiencing VR sickness. However, these types of experiences were extremely limited in scope due to the lack of positional tracking.

Over the next several years, our lab developed techniques to create co-located 6DOF VR experiences that used the aforementioned 3DOF headsets enhanced by external 6DOF tracking systems. As new advances allowed for better tracking systems, we adapted our own methods to take best advantage of the current technologies. The following sections detail the methods we used for each experience. Table 1.1 is a summary of the different experiences and the headsets used to run them.

Year	Title	Headset Hardware	Tracking
2015	*Holojam*	GearVR w/ Samsung Note 4	3DOF
2017	*I AM ROBOT*	GearVR w/ Samsung Galaxy S8	3DOF
2017	*Holojam in Wonderland*	GearVR w/ Samsung Galaxy S8	3DOF
2018	*CAVE*	Google Lenovo Mirage Solo	6DOF

Table 1.1: Summary of headsets used in various experiences. While our experiences all enabled 6DOF tracking, each headset aside from the Google Lenovo Mirage Solo supported only rotational, 3DOF tracking. To support full positional tracking, we had to combine the headsets with external tracking systems.

Figure 1.1: Users in the Holojam system. The left image shows users wearing the GearVR hardware with tracked markersets. The right image shows an example of how the users are rendered within the system.

1.1 Holojam

The first of our experiments to create a 6DOF, co-located VR system was named Holojam. Developed over the course of late 2014 and 2015, Holojam was first presented at the SIGGRAPH[i] 2015 VR Village, a collection of experiences that demonstrated the power of VR as an emerging technology. At the time, almost every HMD that allowed for 6DOF tracking was tethered to a host computer. This made co-location difficult, as users could not freely move around each other without the danger of interfering with cables. Some experiences, such as RealVirtuality[3], VRcade[83], and The Void[82] employed backpack-based solutions to allow users to walk around without that danger, but the weight of these backpacks reduced their practicality. Using our methods, we were able to present Holojam to 4 people at once, for a total of approximately 200 people

[i]SIGGRAPH is an annual technical conference focused on innovations in the field of computer graphics.

over the course of the day at SIGGRAPH.

The Samsung GearVR gave us a lightweight, untethered system that could be used to run our Unity-based graphics engine[81]. However, as the device only offered 3DOF tracking, we needed a way to track a user's position. To resolve this, we used a high-quality motion capture system, OptiTrack[63], coupled with a custom network protocol primarily developed by Zach Cimafonte to wirelessly transmit the data from the motion capture system to each user's headset. Once the data was received, we used a sensor fusion algorithm, which will be detailed in Chapter 2, to blend the incoming motion capture data with the internal IMU. In addition to the markers tracking the user, we also allowed the user to input button commands with a tracked wand. For this wand, we used a WiiMote[87] connected to the server computer via Bluetooth. Figure 1.2 gives an outline of the system. Below, we will detail how each component functioned within this system.

1.1.1 Motion Capture

While the GearVR offers smooth head orientation tracking, it lacks two core functions needed for a untethered experience. First, the GearVR has no form of positional tracking. If a user walks around while wearing the GearVR, no sensors can report to the application that the transformation in user position should occur, and so without any sort of controller, the user is in a fixed position. Second, there is no ground truth for the rotation. When the user puts on the headset, the software understands his or her initial orientation as the iden-

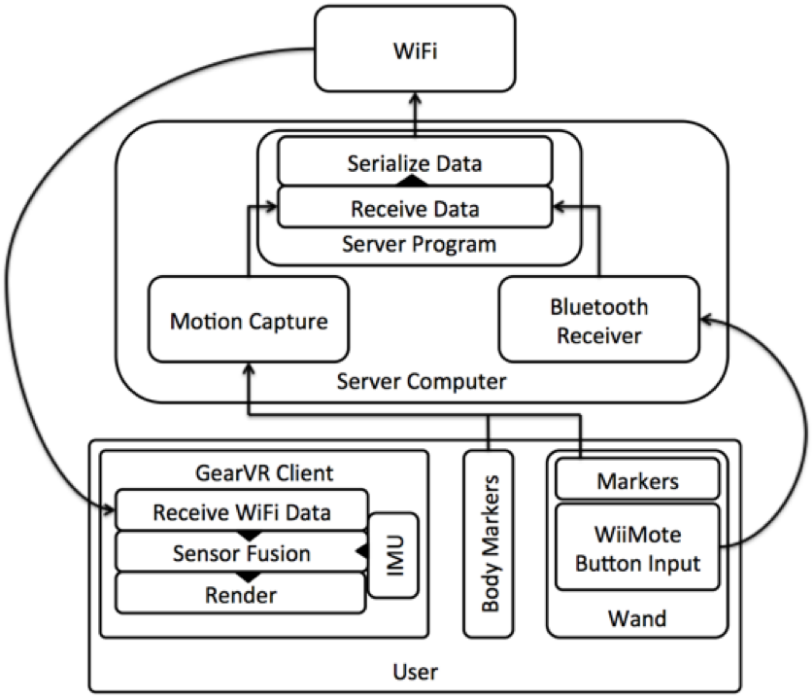

Figure 1.2: Overview of the Holojam system. Motion from the tracked markers and input from the WiiMote controller was sent to our server. That data was then processed and sent over WiFi back to the headsets. That data was combined with internal sensor data to produce a final render image.

tity rotation. Thus, if two users put on a headset running the same software, the world for one user could be backwards compared to the other user. Furthermore, if positional tracking were provided to the applications, a rotated user would observe that the virtual camera would move in a different direction than his or her physical movement.

Both of the above issues can be resolved by introducing some form of ground-truth tracking into the system. Many such tracking solutions exist today. While not the most cost-efficient, we decided that for our research purposes, motion

capture technology would provide the highest quality data and be the easiest to set up of the then available solutions. OptiTrack, a well-known motion capture system, allowed us 6DOF tracking at 240 frames per second, more than sufficient for tracking the HMD.

OptiTrack technology uses an array of cameras and lights to both project and receive infrared light. The projected light hits retroreflective markers and bounces back toward the cameras, making the markers appear very bright to the camera and therefore easy to identify. When sets of at least four markers are arranged in unique, 3-dimensional configurations, the software can uniquely identify these configurations and assign positions and orientations to the configurations. These configurations can be attached to the headset or other objects in the room. The tracking images for each camera are processed by the software to get object poses, and relayed through a central server to each phone client to get full 6DOF tracking data on the GearVR headsets. We broadcast this information over our custom real time wireless protocol to GearVR phones using commodity networking hardware. In order to take full advantage of OptiTrack and GearVR's capabilities, we use sensor fusion, which is detailed in Chapter 2.

Using this motion capture software for tracking has both advantages and disadvantages. One of the advantages is that the marker sets are relatively cheap to produce, allowing us to produce many such rigid tracked bodies so that multiple physical objects can appear in the virtual space with the correct pose. Holojam's networking protocol, explained below, is well equipped to transmit this information effectively. We have used this for many different features, such

as a movable table, physical controllers that act as magical wands, and for tracking joints on a user to recreate an avatar.

The biggest downside to motion capture technology is the cost. High quality motion capture cameras are widely used in academia and industry but are well outside of the consumer price range. The overall cost can be reduced to a few thousand dollars by using as few as two cameras, but fewer camera yields loss of tracking quality and capture volume. Additionally, motion capture is limited by visibility and physical interference. If a marker set is blocked, the software will not be able to report its pose, which means the user loses all positional data until the marker set is visible again. This will often cause motion sickness for the user, even if the user loses tracking for as little as half a second. Finally, because the software must process the markers as raw points and then try and solve the given positions to match marker sets, the software will have an unacceptable amount of computation latency once a large number of marker sets are created within the scene. We found that we could support up to about 30 rigid bodies with less than 3ms of latency, but with more than 30 rigid bodies, the software began to exhibit more than 10ms of latency.

Some of the above issues, particularly cost, could be solved with alternative tracking solutions. However, we wanted to primarily focus on the experience of having a lightweight shared virtual space, rather than solving the entire problem of tracking. Therefore, for our purposes, we found the OptiTrack cameras to be more than sufficient. As tracking technologies evolved, we were able to adapt the Holojam system to use cheaper methods.

Figure 1.3: An example of a set of tracked markers for a Holojam user. Pictured here are the headset, gloves, and leg straps that affixed markers to a person. The gray spheres were made of retroreflective material and assembled as unique configurations, making them easily identifiable by the OptiTrack system. The phone below the headset is the Samsung Galaxy Note 4, the device that acted as the main computer and renderer for the Samsung GearVR system.

In addition to affixing markers to the headset to enable 6DOF tracking, we also created markers that were attached to the hands and lower legs, as shown in Figure 1.3. This approach allowed users to interact with accurate hand and foot motion, enabling interactions such as dancing and hugging.

1.1.2 Networking

When delivering tracking data to wireless headsets, one of the biggest challenges is to do so with sufficiently low latency. Recall that most VR systems specify an accurate frame rate of at least 60 frames per second, which is approximately 17ms. For our data to be frame-accurate, we must deliver it within that 17ms window before the frame renders. Given that this window must also account for the OptiTrack computation latency and the headset rendering latency, both of

which were measured to take an average of 2-3ms, we had to design a protocol which could reliably transmit data in less than 10ms over WiFi.

To better understand our choices in designing the network transport protocol, it helps to review existing protocols and how they function. Two of the most commonly used protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP)[90,25]. TCP, due to its reliability in data transmission, is frequently used for control schemes and other data transfers that require that all data is received, such as email. UDP has no such guarantees, but offers greater data throughput and lower latency. Because of our needs for a low-latency data stream, UDP became a clear choice for our system.

However, there were still some problems with UDP. First, UDP does not guarantee delivery. This means that if a packet is not initially received, the client will never receive that packet. If packets are regularly dropped, a particularly prominent issue in congested networks, these missing frames can appear as lag. To combat this, we do not send one packet per render frame, but instead one packet per motion capture frame. This means that every render frame will receive up to four packets of data, and thus we will only experience a frame drop if every packet is dropped. Secondly, UDP does not guarantee delivery order. This means that in certain cases, we may see older data playing after newer data, causing artifacts that look like shaking or skipping. The fix for this is simple: we add the server timestamp to our packets and discard packets with an older timestamp than the last frame we processed.

The final problem with UDP, particularly with 802.11 WiFi communication,

is a process known as IP fragmentation. UDP packets, by design, have a maximum size of 64 kilobytes. However, many consumer-level WiFi routers have a maximum transmission unit (MTU) size of 1,500 bytes. This means that most UDP packets are broken up into smaller fragments and reassembled on the client side. If any one fragment is missing, then the entire packet is discarded by the client[15]. This causes greater packet loss, and thus the appearance of lag. To prevent this, we designed a compact protocol using Google Protocol Buffers[30] to keep our data packets, including the header, below 1,500 bytes.

With these considerations in mind, we were able to relay tracking data from the motion capture system through a C++ server to each of the headsets. This ensured that not only did each client receive their own tracking and interaction data, but the data from others as well.

1.1.3 Design Choices

Given the above systems and capabilities of those systems, we found certain design choices to be optimal for the Holojam 2015 system. These design choices related to virtual environment design, avatar design, and interaction.

To fit within the rendering budget of the Samsung Galaxy Note 4, we found that we could render approximately 30,000 triangles before we observed performance drops in the system. To meet this budget, we chose simplistic and cartoonish styles that evoked a childhood dollhouse (See Figure 1.1). Both the environment and character avatars took on this aesthetic. Additionally, we found that the stylized representations of people allowed us to avoid the trap of the

uncanny valley[ii], a common pitfall when designing avatars.

Furthermore, complex dynamic lighting solutions were completely nonviable on the hardware. However, the devices were quite proficient in efficiently rendering textures, since they were designed to run 360 video applications. Therefore, we decided to simulate lighting in the texture shaders. Designed by David Lobser, these shaders further reinforced the cartoonish stylization of the room by adding a randomly-sampled cross-hatch pattern to the shadows, as opposed to a diffuse shading model commonly seen in standard shading methods.

In order to puppeteer the avatars, we decided to track a user's head, hands, and feet. The torso, elbows, and knees were interpreted from the poses of those 5 tracked sets using a 2-link *inverse kinematics* (IK) chain. We found that this level of control was sufficient to drive our stylized avatar. Furthermore, while many IK algorithms require a calibration step to adjust for the differing heights of users and make appropriate adjustments to the arms and legs, we wished to avoid this process to minimize overhead for the user. Our IK chain continuously adjusted the limb lengths to match the height of the user. However, it also adjusted at a slow enough rate to accommodate quick gestures, like jumping or squatting.

Finally, by giving users a simple way to input commands into the system with a WiiMote, we allowed users to draw lines in 3D space to populate the scene. Our unique approach to co-located VR not only allowed users to draw with each

[ii]The uncanny valley is a phenomenon when simulations of humans are close to realistic, but not perfect[73]. Many people describe simulations where the uncanny valley is present as visually disturbing or unpleasant.

other in 3D, an experience difficult to replicate outside of VR, but they were also able to make drawings that surrounded other people, a feature made possible with our untethered solution.

1.1.4 System Evaluation

Holojam 2015 was successful in enabling people to freely move around in co-located VR, something that would otherwise have been difficult without our system. However, it was not without problems. Due to the sensitivity of the latency requirements, any periods of high latency would cause VR sickness for the users. While this was not a problem for most of the experience's duration, there were periods of heavy network traffic due to the crowded SIGGRAPH venue. Furthermore, the phone hardware was not well-equipped for long-standing VR experiences, which caused the phones to overheat quickly and have short battery life. While Holojam 2015 provided a prototype for these sorts of co-located VR experiences, the current hardware limitations made the system difficult to use.

1.2 I AM ROBOT and Holojam in Wonderland

Between 2015 and 2017, an explosion of consumer-level VR devices entered the market. This growing market created a demand for more robust VR technology. Amongst those were better mobile processors and cheaper tracking methods. Compared to the Samsung Galaxy Note 4 used in Holojam, the Samsung Galaxy S8 released in 2017 had a much more powerful processor, moving from

one quad-core 2.7GHz processor to two quad-core processors at 2.3GHz and 1.7GHz[70,71].

In 2016, the HTC Vive became publicly available, bringing room-scale VR to a consumer budget. The Lighthouse tracking technology allowed for a 20 by 20 foot tracking space for approximately $200 USD, as compared to the OptiTrack system, which cost approximately $50,000 USD in 2015. While the Lighthouse system could not expand beyond this space like the OptiTrack system could, it was large enough to enable us to create similar experiences to Holojam on a much cheaper budget. Shortly after, HTC released lightweight trackers that allowed for tracking beyond the Vive headset and controllers.

1.2.1 I AM ROBOT

Partnering with DAFFY LONDON[11] and the Superbright[67] production company, we developed I AM ROBOT. This experience was similar to Holojam 2015 but with updated technology, combining the new GearVR headsets with the HTC Vive trackers, as seen in Figure 1.4. Many of the networking considerations were kept the same as above, but differences in mobile graphical processing power and tracking technologies made us rethink those systems. I AM ROBOT was shown over the course of SIGGRAPH 2017.

While the tracking system provided by the HTC Vive allowed us to create a tracking space much more cheaply than before, the individual trackers were more expensive than our OptiTrack markers ($100 USD per Vive tracker compared to approximately $5 USD per OptiTrack tracker). Additionally, because

Figure 1.4: A user in the I AM ROBOT experience. In order to track a user, we used HTC Vive Trackers attached to the user's headset and hands. This solution was much cheaper than our previous OptiTrack method and required less overhead.

the Vive trackers were connected via Bluetooth, we found a limitation both in software and hardware that limited us to using fewer trackers. In total, we found that we could use approximately 12 tracking devices on one machine. By networking two machines together, we were able to expand that to 24. However, this additional complexity made us reconsider how many trackers we could use. Finally, the 2017 Lighthouse system was limited to only two Lighthouse stations at a time. This meant that we could not place additional stations to cover potential blindspots. Because of these factors, we opted to track fewer user points, only tracking the headset and two hand poses for each user.

This limited tracking meant that we had to create a different kind of avatar.

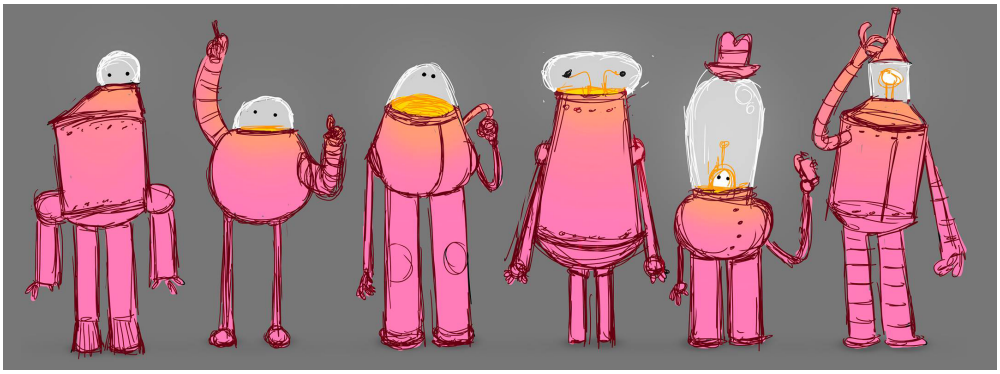

Figure 1.5: Sketches of the six original robots from I AM ROBOT. ©Jakob Schuh and DAFFY LONDON 2017[11].

Instead of explicitly tracking the feet, we were able to use IK to solve for foot positions, automatically moving the feet in a walking motion with a fixed gait. This caused initial problems with *transportation*, *i.e.* the feeling of actually being within the virtual body, as the feet would not move according to users' exact motions. In a sense, this perceptual illusion was analogous to the uncanny valley problem applied to motion.

In addition to addressing the tracking, we wanted to create avatars neutral to the participant in terms of gender, race, size, and other similar factors. To solve both of these issues, we created avatars that were very different than the human body. There were six different avatars designed to look like robots, each with a very different body shape (see Figure 1.5). Six additional avatars were designed as color variations of the originals. Upon entering the virtual world, a user was assigned one of the twelve avatars. Because the robots had signficant variance in body shape, especially compared to standard human proportions, we retargeted the trackers to map to offset locations. For example, in one avatar

seen in Figure 1.6, we remapped the head to be lower since the avatar had a much shorter torso.

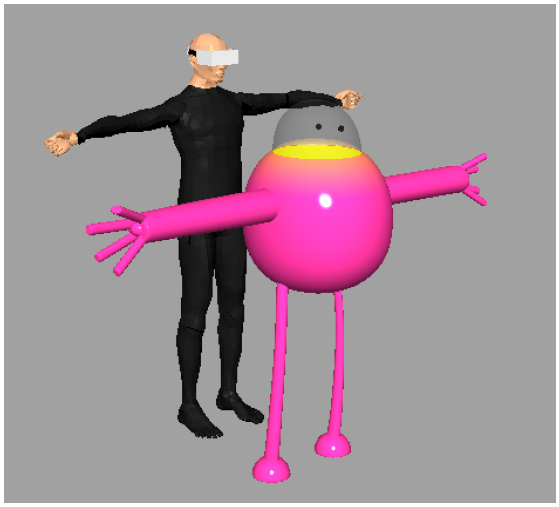

Figure 1.6: A comparison of human proportions to the I AM ROBOT avatar proportions. Note the significant difference in ratio of limb length to torso height. These differences required us to retarget some trackers to different poses.

The affordances of the upgraded phone hardware allowed us to render much more complex models, allowing the avatars to take on a smoother look. Additionally, environments could be much more complex. We found our total rendering budget to work with up to approximately 500,000 triangles before we observed a performance decrease, over an order of magnitude more than the previous Holojam.

Additionally, we opted to not utilize a controller as in Holojam 2015, but instead to make intuitive interactions based on motion of the arms, head, and body. Such interactions included bouncing a ball, dancing, and moving throughout the room based on prompts. The experience cycled through several different

environments, each persisting for two minutes before switching to the next. To switch between environments, each client received a synchronization signal from the main server.

Finally, we experimented with virtual audio within the system. As our prior experiments had only dealt with virtual visuals, we believed that virtual audio would add an extra layer of immersion. However, we did not want to use headphones, as they would hinder the participants' ability to verbally interact with each other. Instead, we opted to use a sound system in which four speakers were placed in the corners of the room. The audio server, connected directly to the speaker system, was synchronized with the main server to play a different playlist of music for each environment, along with necessary sound effects for objects within the space, like the bouncing ball.

Like Holojam 2015, I AM ROBOT accommodated four concurrent users at a time. Each user stayed in the experience for approximately 6 minutes. Over the course of five days, approximately 1,000 people were able to experience I AM ROBOT. Some users reported problems with the tracking, which led to cases of VR sickness. However, aside from those reports, we received largely positive responses from the majority of participants.

In addition to learning lessons regarding the technology, user feedback helped us understand the importance of social AVR systems and their unique qualities. Some users found the feeling of being transported to be extremely powerful, especially in the physical context of co-located VR. When we asked participants about the experience, some of their responses enlightened us on this empowering

aspect:

> [1] Once I got into I AM ROBOT, I was really astonished by their very fluid concept of identity. I was immediately excited to hear that I would be dancing as a robot and that all those gender stereotypes were out the door. I wouldn't have to police my body as I always did, because I was a robot now, dancing in a room that lived and seemed to interact with us, with a group of people I had just met twenty minutes ago, that had just been faces amongst a large array of attendees previously.
>
> [2] My favorite memory is of one woman who did the experience and danced like no one was watching - enough for me to assume that she was a party person. When she came outside though, and when asked how she felt, she said that she was someone who hates being in huge and crowded parties.
>
> [3] Once inside, it quickly made me want let go of the "physical reality" tether. I AM ROBOT is easily the most immersive VR [project] I've ever experienced.

To us, these testimonials of the experience were as important as our evaluation of the technology paradigm, as they demonstrated the unique aspects of systems like ours. Our future developments built upon these concepts to continue to create enriching VR experiences.

1.2.2 Holojam in Wonderland

Shortly after I AM ROBOT, our lab produced a VR theater short named *Holojam in Wonderland*, directed by David Gochfield[29]. It used largely the same technology as I AM ROBOT, but had a few different design decisions. Instead of supporting full motion of all avatars, we opted to only track the heads of the viewing participants. In addition to the viewers, two fully-tracked users participated in the experience as live actors. These actors had tracking markers on

Figure 1.7: User viewpoint from within *Holojam in Wonderland*, viewing one of the fully-tracked actors. Character and environmental design ©Kris Layng 2017.

their hands, feet, and head, similar to Holojam 2015. Because there were a limited number of these fully-tracked avatars, we did not encounter the same issues as when designing I AM ROBOT. As live actors, these two users acted out a script based on the characters from *Alice in Wonderland*[8], taking the other participants through a live theater-like experience.

Holojam in Wonderland was shown at the Future of Storytelling Festival in 2017. Taking advantages of the affordances in VR, users were transported from an ordinary environment into a fantastical one, changing size relative to the environment, and sharing limited interactions with the characters. Figure 1.7 shows an image from the user's perspective during the experience. Feedback was overwhelmingly positive to the experience, adding to the list of experiences that demonstrate the potential of theater in VR.

However, a certain issue was common throughout Holojam, I AM ROBOT, and *Holojam in Wonderland*. The number of concurrent participants created problems within the tracking system when users occluded visibility between a tracker and a Lighthouse or camera. This was particularly problematic when the head tracker was occluded, as it meant that no perceived motion could occur, causing confusion and dizziness for the user. It was clear that for us to scale our experiences up to larger audiences, we would need to investigate different solutions and develop new paradigms.

1.3 CAVRN

Driven by the desire to push audience sizes to larger and larger scales, our lab began to think of a different approach to the co-located VR paradigm. Instead of trying to create a highly interactive experience, the lab decided to create a lean-back experience, named *CAVE*, where the audiences watched a cinema-like experience. By reducing the freedom of user motion, we mitigate both the problems of tracking and collision. Furthermore, unlike *Holojam in Wonderland*, the performance was prerecorded and loaded onto each headset to be played back individually.

Using the Google Lenovo Mirage Solo[32], an untethered headset with limited 6DOF tracking enabled users to move their heads and look around the scene, unlike a 360 video experience. This motion was mirrored in avatars which each other participant saw. Specifically, by allowing users to see other audience members and their body motions, we believed we could create a powerful, shared

experience for the participants.

In order to explore the paradigm described above, the lab created a system called the Collective Audience Virtual Reality Nexus (CAVRN), primarily designed by Sebastian Herscher. This system was in charge of synchronizing the clients so that each person saw the same virtual content rendered from their own unique viewpoint. Specifically, this required the synchronization of two components:

- First, the *CAVE* experience had to be synchronized so that each client viewed the content at the same experience state at any given moment in time. This was not only important for ensuring that the audience members would react to the content at the same time, and thus share their reactions with each other, but also to ensure that audio cues on external speakers would play at the correct moments.

- Second, we received incoming data from each client about their head pose and broadcast that back to each other client so that the head motion could be replicated for each audience member's avatar in the experience.

As each user was in charge of their own head tracking, we relaxed the requirements on the server. Instead of needing ultra-low latency to meet the tracking needs previously described, the system could send less frequent synchronization signals at 20Hz, smoothly interpolating the motions of other users. Furthermore, since the experience was preloaded on the headsets, a 20Hz signal was more than enough to keep the clients playing at the same time within the experience.

Figure 1.8: A scene from *CAVE*, where a mammoth steps into the environment between the audience members. Participants could see the reaction of other participants' avatars as the experience unfolded. Character and environmental design ©Kris Layng 2018.

We presented *CAVE* at SIGGRAPH 2018. Over the course of the week, 1,927 people were able to view the experience. Compared to many other systems previously designed, our design decisions allowed us to deliver an experience with much higher participant throughput. While it had less interactivity than our prior experiences, we believed that the content and the format would be enjoyable enough on its own without interactivity.

In order to evaluate the hypotheses of the CAVRN system, we conducted a mixed methodology study consisting of a survey and semi-structured interview. The details of this survey can be found in Herscher *et. al.*[iii] In particular, we

[iii]This research is currently under peer review and will soon be available.

sought to discover whether users found the format enjoyable, particularly the experience of being in a live, co-located audience, and if having less interactivity was acceptable within the context of VR experiences.

Overall, we found that participants enjoyed the experience, and felt that the live audience enriched the experience. Additionally, participants appreciated the unique affordances of VR that would be impossible in live theater or cinema, such as the appearance of creatures moving between the audience rows, as seen in Figure 1.8. While some participants did wish for more interactivity, many participants understood the similarities to live theater and cinema and appreciated the lean-back paradigm.

We believe that *CAVE* and the CAVRN system, along with our evaluation of the experience, make a strong argument for physical co-location in VR experiences. As it was both successful in delivering a co-located VR experience to nearly 2,000 people in less than a week, and very positively received, we believe that this work can be used as a blueprint for future designs seeking to implement a similar experience. We hope that others take inspiration in our efforts to combine the worlds of cinema and VR, and that this will open up the exploration of this new medium of entertainment.

2

Solutions for Co-located AVR

Introduction

Along with the design innovations made for the experiences described in the previous chapter, we developed several algorithms to overcome the limitations of AVR systems designed for a single user. This chapter details these algorithms, as well as why they were necessary for the systems we designed.

2.1 Sensor Fusion and Filters for 3DOF Headsets

Many of the early untethered systems came with some sort of 3DOF sensor, such as a gyroscope, which handles purely rotational tracking. The fidelity of this sensor varies from system to system. For many of the examples discussed in the previous chapter, the GearVR that we used sampled its IMU at 1000Hz. This provides for very smooth rotational tracking, as the device effectively gets 16 frames of IMU data per render frame. However, these IMUs provide angular velocities, which means that in order to calculate the orientation, the headset must perform integration on the IMU readings. This causes two problems. First, small inaccuracies in the readings, oftentimes due to precision loss or noise, can build up over time. This is known as *drift*[14]. The second problem is that as a device launches its application, it will set the forward direction of the virtual scene to be the initial orientation of the device as the application is launched. If multiple devices launch an application while facing different directions, then their virtual world is already mismatched.

To solve these two issues, we use a motion capture system to not only resolve the user's head position, but also to correct the user's head orientation to match the ground truth. Here, we detail the math behind the sensor fusion first used in the Holojam system and later applied throughout our other experiences. Let us define the following unit quaternions representing rotations. Let R_G be the rotation reported by the GearVR IMU and R_M be the rotation reported by the motion capture system. In Unity, the rendered viewport is the product of

a camera rotation offset and the reported IMU rotation. Let R_C be the camera rotation offset, and R_U be the final rotation that the user sees within the headset, so $R_U = R_C R_G$. Finally, let α be a parameter between 0 and 1, which we will explain later.

Our goal is to have R_U to approach R_M smoothly over time. Moving R_U to R_M will move the user view to the ground truth for initialization and drift correction, and the smooth movement prevents large, jumpy changes for the user perspective. We can use a *spherical linear interpolation*, or $SLERP$[74], in order to achieve this effect. For quaternions q, r and a real number $\alpha \in [0, 1]$, $SLERP(q, r, \alpha)$ is defined to be equal to $(rq^{-1})^{\alpha} q$. So, for interpolating from R_U to R_M, we have

$$\begin{aligned} R_{U_1} &= SLERP(R_{U_0}, R_M, \alpha) \\ &= (R_M R_{U_0}^{-1})^{\alpha} R_{U_0} \end{aligned}$$

Where R_{U_0} is our previous state of R_U and R_{U_1} is the updated state of R_U that we are trying to calculate. However, R_G and R_M are read-only variables, and $R_U = R_C R_G$, so the only rotation we can modify is R_C. Thus, we can modify

the above equation to be:

$$\begin{aligned} R_{C_1} R_G &= (R_M (R_{C_0} R_G)^{-1})^{\alpha} R_{C_0} R_G \\ &= (R_M R_G^{-1} R_{C_0}^{-1})^{\alpha} R_{C_0} R_G \\ \implies R_{C_1} &= (R_M R_G^{-1} R_{C_0}^{-1})^{\alpha} R_{C_0} \\ &= SLERP(R_{C_0}, R_M R_G^{-1}, \alpha) \end{aligned}$$

By doing this, we make R_C asymptotically approach $R_M R_G^{-1}$, and thus R_U will approach R_M.

This approach is known as a complementary filter[17], a common filter for both low-pass and high-pass noise filtering in signal processing, adapted to work with quaternions. While less accurate overall than the well-known Kalman filter[85], the complementary filter is much easier to implement and execute and delivers reasonably accurate results[41,35]. The parameter α defines how much the system is influenced overall by the ground truth rotation R_M. If α is close to zero, we will primarily rely on the IMU, and if it is close to 1, we will primarily rely on the motion capture system.

When we rely purely on the IMU, we encounter the integration problems described earlier. When we rely purely on the motion capture system, we encounter rotational jitter due to angular noise reported by the system as well as slight lag from the transmission time of WiFi. Thus, we want to choose an α that uses the advantages of each system.

We found the best results by varying α based on the difference between the

ground truth reports (R_M) and the rotation calculated by our algorithm (R_U), with $\alpha = 1$ at the first frame so that R_U initializes as R_M. Throughout our application runtime, if there is a large discrepancy between R_M and R_U, then we will quickly adjust to the ground truth. Otherwise, α will be very low, and thus we will primarily rely on the headset rotation. This allows us to have the smooth rotations provided by the IMU while also maintaining a known ground truth.

Later, we discovered a simplification of the algorithm due to an inherent property of the IMU. Because many IMUs can detect the downward direction based on the pull of gravity, the device can utilize that information to accurately solve pitch and roll. However, as yaw is not affected by gravity, this is the one axial rotation we must correct. Thus, instead of doing a spherical LERP on the quaternions, we can do an *angular LERP* ($ALERP$) on the yaw angle. $ALERP$ is defined as a *linear interpolation* that correctly accounts for the change from 0° to 360° radians[i]. If we define θ_C as the camera yaw rotation in a given frame, and then θ_D as the yaw rotation of the quaternion $R_M R_G^{-1}$, then we get a formula similar to the one above:

$$\theta_{C_1} = ALERP(\theta_{C_0}, \theta_D, \alpha)$$

[i]For real values a, b and real value $t \in [0, 1]$, we define linear interpolation as $LERP = a(t-1)+bt$. However, when considering angles, using this interpolation can produce undesired results. For example, linearly interpolating between 5° and 355° would give a result between 5° and 355°. However, frequently we would want the smaller angle that exists between 355° and 360° or 0° and 5°. Angular linear interpolation can give us this desired result. We first assume the two angles are between 0° and 360°. Then, if the difference between the two is greater than 180°, we subtract 360° from the larger value, then linearly interpolate as normal.

This simplified version allows us to perform effectively the same filter by only operating on one dimension, as opposed to four dimensions with the quaternion blending method.

2.2 Synchronizing Independent Co-Located Clients

In prior sections, we have discussed the importance of maintaining the same sense of where each virtual object is within the room, including other users. The placement of objects within the virtual scene is defined by the frame, or the reference position and rotation that the client understands to be the zero vector and identity quaternion. Every virtual object pose is then defined relative to that origin. Thus, synchronizing the virtual objects between the different clients is just a matter of resolving the difference in frame poses.

When the clients are synchronized with a centralized tracking system, that system is responsible for synchronizing the frames between the clients. Frequently, the tracking system will define its own frame and then communicate the clients' poses relative to that frame. The OptiTrack motion capture system discussed in Chapter 1 for the Holojam 2015 system is one example of this sort of paradigm.

However, having this sort of centralized tracking server requires a large amount of overhead. As mentioned in Chapter 1, the OptiTrack system had a large financial cost of $50,000. In addition to this, physically setting up the system within a room is demanding. The room must be appropriately equipped to allow for cameras to be mounted with all necessary wiring, and mounting and cal-

ibrating the cameras takes time. In our experience, setting up a 16-camera system in a 25x25 foot room takes between 1 and 2 hours. While this is acceptable for large productions, it makes it difficult to use in arbitrary locations or within short time frames. This makes it less than ideal for our vision of ubiquitous co-located AVR. Thus, it is within our interests to find methods to synchronize frame poses without an external system.

Without a centralized tracking system, clients typically have their own internal tracking system. In recent years, inside-out techniques have become more prevalent amongst AVR systems, frequently using SLAM-based algorithms. For these systems, it is common to have a calibration step that defines the frame of the space, as is the case with the Oculus Quest. From that initialized frame, the clients are free to move about the room, using landmarks and internal sensors to update their pose. However, for multiple devices, it is necessary to synchonize their frame since that is calibrated independently per frame.

The simplest way to do this synchronization is to calibrate the clients such that they have the same initial pose. This typically requires manually placing each device in the same pose one at a time and performing the necessary software calibration steps. While this method theoretically works, it has problems in practice. Primarily, it can be difficult to align the devices such that each has the same exact pose. While it is not difficult to coordinate the origin position with reasonable accuracy, the clients must have their rotation calibrated extremely precisely. Simple calculations yield that even 5° of error will cause objects to have almost 10cm of error per 1m they are positioned away from the

frame origin.

Given that it is relatively easy to maintain positional accuracy compared to rotational accuracy while performing this calibration step, we propose a method that relies on positional calibration to synchronize frame poses amongst clients. In our calibration step, we create 2 physical markers at fixed, measured points around the room to form a set P. Then, a user places the headset at each of the 2 points, pressing a button each time to confirm registration. As these positions are registered, the application creates a set of points Q which represent the virtual locations of the points. Given P and Q, there exists a rotation quaternion R and a translation vector T such that the difference between P and $T + RQ$ is minimized. If R and T are found for each client and applied to their frame, then assuming the systems do not significantly drift from their initial frame, then each client will have the same frame.

To find R and T, we can make one simplification. Given our prior discussions, we know that R only needs to be a yaw rotation. Thus, we have four degrees of freedom that must be solved: the x, y, and z position along with the yaw rotation.

Therefore, we must find the optimal yaw angle θ that rotates each point of Q to the matching point in P. Known algorithms, such as the Kabsch algorithm[42], can solve this rotation using singular value decomposition, but because we are only solving for one axis of rotation, we can use a simpler method. For the two vectors $Q[1] - Q[0]$ and $P[1] - P[0]$, we calculate the signed angle θ_Q and θ_P that represents those vectors' rotations around the y (yaw) axis. The signed

difference $\theta = \theta_P - \theta_Q$ will tell us the rotation we need to bring the points of Q into P:

$$R = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix}$$

To calculate T, and thus solve the remaining 3 degrees of freedom, we calculate the difference in centroids between P and RQ:

$$T = \frac{P[1] + P[0]}{2} - R(\frac{Q[1] + Q[0]}{2})$$

If the two points form the diameter of the boundary space, then the calibration error of the space will be bounded by the positional error of the calibrated points. In other words, if a user can accurately place the headset within 1cm of each of the fixed points, then there will be no more than 1cm of error throughout the tracked space. This is because by construction, we choose two points that are further away from each other than any other two points in the tracked space. Positional errors from angular differences between the calibrated frame and the real frame scale linearly with distance from the center of the space, and positional errors will be constant throughout the space. Thus, the calibration error of every point closer to the center of the space, and therefore every point within the space, must be smaller than the error at the two calibration points.

2.3 A Distributed Tracking System for Co-Located AVR

One common problem we observed throughout our experiences was the lack of scalability for 6DOF tracking. For different systems, different problems arise. When using any sort of outside-in tracking method where an external sensor with a known position is used to track headsets in some way, as with the OptiTrack system for Holojam 2015, the camera must be able to see the headset tracker. Likewise, for many inside-out systems where a headset-mounted sensor detects landmarks with known positions, line-of-sight is required between the sensor and the landmark. For robust inside-out systems that use room-mapping techniques such as SLAM, this problem is reduced, but nonetheless the sensor must be able to see several static features to function. Furthermore, many current SLAM-based AVR systems do not share discovered landmarks with each other, thus making co-located tracking difficult. It is noteworthy that some systems are beginning to do this, such as Microsoft's Azure-based spatial anchors[51] and Google Cloud Anchors[33], but this practice is not yet widespread. Other works, like that of SynchronizAR[40], have sought to do similar frame synchronizations, but use an additional ultra-wide bandwidth module attached to the device. Instead, we look to utilize built-in devices to solve our problem, such as the camera already required to do SLAM tracking.

We propose a solution that takes advantage of the idea of having multiple clients in the same physical space instead of being hindered by them. We do this by combining outside-in and inside-out tracking methods, using each device

as a method to track other users. In this way, we distribute the tracking procedure instead of relying on one centralized system to coordinate every user.

2.3.1 Resolving Frame Differences

To test our distributed tracking method, we use ARCore[33], a SLAM-based tracking library for Android phones, and OpenCV[62], a well-known computer vision library. ARCore maintains the client's localized position and creates a frame within which the client exists. This frame is established upon application launch, and is set to the client's forward facing direction and initial position. ARCore then adjusts the client's known position and rotation starting from that initialized pose using SLAM techniques.

Since each client establishes their frame independently, they cannot easily share information about the poses of their virtual objects. For example, if one client decides to create an object at their origin and then communicate this, then *a priori* the other clients will believe that an object has been created at their origin. This would be incorrect, unless each client is registered to understand that the same physical location is the world space origin.

We resolve this by allowing clients to track each other, similar to traditional outside-in approaches. By establishing a relative difference between the two clients, they can calculate the difference in frame poses. From there, coordination of virtual object placement is easy.

Let us define (P_A^A, Q_A^A) as the position and rotation of client A within its own frame, (P_B^A, Q_B^A) as the position and rotation of client B as observed by client

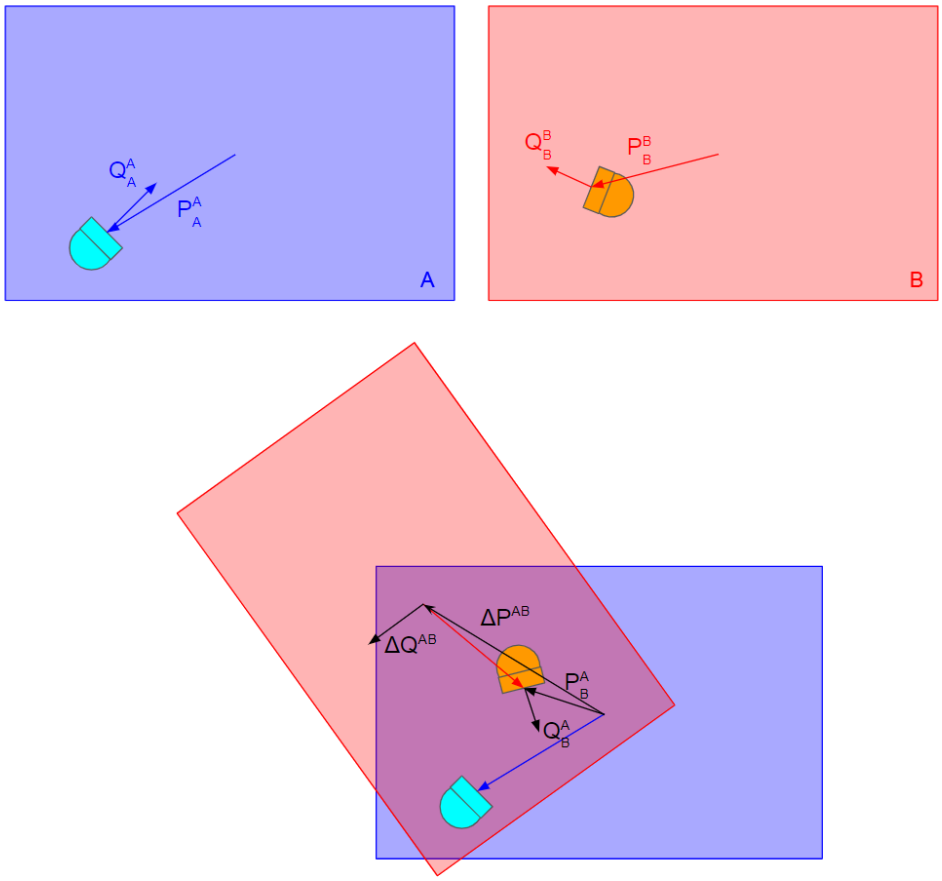

Figure 2.1: Transforming one frame to another. The top left shows Client A's frame along with Client A's pose within their own frame. The top right shows the same for Client B. At the bottom, we see Client B's pose in Client A's frame, along with the transformations necessary to adjust frame B into frame A.

A, and let (P_A^B, Q_A^B) and (P_B^B, Q_B^B) be defined similarly within client B's frame. Client A can calculate the difference between their frame and client B's frame by first calculating the difference in rotation between Q_B^A and Q_B^B as:

$$\Delta Q^{AB} = Q_B^A (Q_B^B)^{-1}$$

This quaternion, ΔQ^{AB}, represents the quaternion that rotates client B's frame into client A's frame. Hence, if we rotate any rotation or position reported by client B by ΔQ^{AB}, it will be correctly oriented within client A's frame. Using this information, we can calculate the difference in frame position as:

$$\Delta P^{AB} = P_B^A - (\Delta Q^{AB} P_B^B)$$

By applying this positional difference to any object that has been rotated from client B's frame into client A's frame, we correctly position the object within client A's frame. This is illustrated in Figure 2.1.

If we have an object created by client B with position and rotation (P_O^B, Q_O^B), then client A can receive that data and put it into its own frame with the following transformations:

$$Q_O^A = \Delta Q^{AB} Q_O^B$$

$$P_O^A = \Delta P^{AB} + \Delta Q^{AB} P_O^B$$

We note that similar to our discussions above for rotational correction, we do not need to run these computations with full quaternion data, as only the yaw rotation will differ. Therefore, as opposed to multiplying and inverting quaternions, we can simplify our algorithm by calculating angular differences instead.

Finally, it is important to note that this algorithm assumes that the incoming pose from the other client, (P_B^B, Q_B^B), is globally consistent with our view of the client, (P_A^B, Q_A^B). However, due to network latency, the incoming pose will be received by our client several frames later, and thus the data should be matched

with an older pose. To address this, we use a time synchronization server to gain accurate timestamps throughout the clients and record a history of observed poses for each client. By doing this, we can accurately match the incoming data with an older observed pose based on the incoming data's timestamp, and thus have a more accurate frame match.

2.3.2 Tracking Algorithm Implementation

As previously mentioned, we use OpenCV as our primary vision library. To track other users, we use the ArUco extension[27], a library which quickly tracks the pose of simple markers. ArUco markers have the benefit of providing ID and pose, while also not encoding excess information that might make tracking more computationally expensive, as is the case with QR codes[46]. ArUco markers are extracted from the image, identified, and located within the camera frame in a fast and efficient manner, which allows us to use it within our program in real-time. Full implementation details can be found in the original paper[27].

ArUco tracking reports the marker as a set of its four corner locations in camera frame space. With these four corners, we use the well-known Perspective-n-Point (PNP) solver provided by OpenCV to find the center of the tracker along with the orientation of the tracker. Since this pose is in the camera frame space, we move it into world space by adjusting the position by the camera's world position and then rotating the position and rotation by the camera's world rotation. When these trackers are placed on a headset or other client, in our case an Android phone, our other clients can use the above algorithms to deter-

mine where the clients are within our frame.

To create more robust trackers for a client, we can designate a group of markers with a known, rigid configuration as a tracker group. This practice gives greater tracking fidelity for two major reasons:

1. Multiple trackers create redundancy. As ArUco markers are only visible from the front, additional markers can allow the device to be tracked from other angles.

2. Multiple trackers stabilize the tracking. ArUco markers are tracked with a small amount of noise, both in position and rotation. This can be improved with good lighting conditions and larger markers, but the markers will still not be precisely tracked each frame. Adding multiple trackers can stabilize the tracking overall by averaging the errors in centroid calculation. Other solutions in tracking have utilized multiple trackers at different angles for this same stability[88].

When a tracker group is assembled, giving each tracker a known position and rotation relative to the client, we can find the disparity between the known and observed centroids of the visible trackers. For position, calculating this difference is simply equal to the average of all trackers' positions. Because quaternions cannot be averaged in the same way, we use the Kabsch algorithm[42] to find a rotation that minimizes the positional difference between each trackers' known and observed position. In this algorithm, we calculate the singular value decomposition of the cross-covariance matrix of the trackers' known and ob-

served positions. If we define P to be the set of known positions and Q to the be set of observed positions, then the cross-covariance matrix is defined as:

$$H_{ij} = \sum P[i] \cdot Q[j]$$

We then calculate the singular value decomposition of H as:

$$H = USV^T$$

Since we do not expect the transformation to include skew transformations, we are primarily interested in U and V. Due to the nature of the algorithm, there can be ambiguity between right-handed and left-handed systems. To solve this, we correct the rotation by flipping the z-axis to ensure we are using a right-handed system:

$$d = \det(VU^T)$$

Finally, we calculate the optimal rotation R:

$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} U^T$$

To verify that multiple trackers stabilize the tracked object, we created a few different tracker sets and measured the reported error. We created a 2-tracker system where the trackers were connected at the edge at a 135° angle, and a

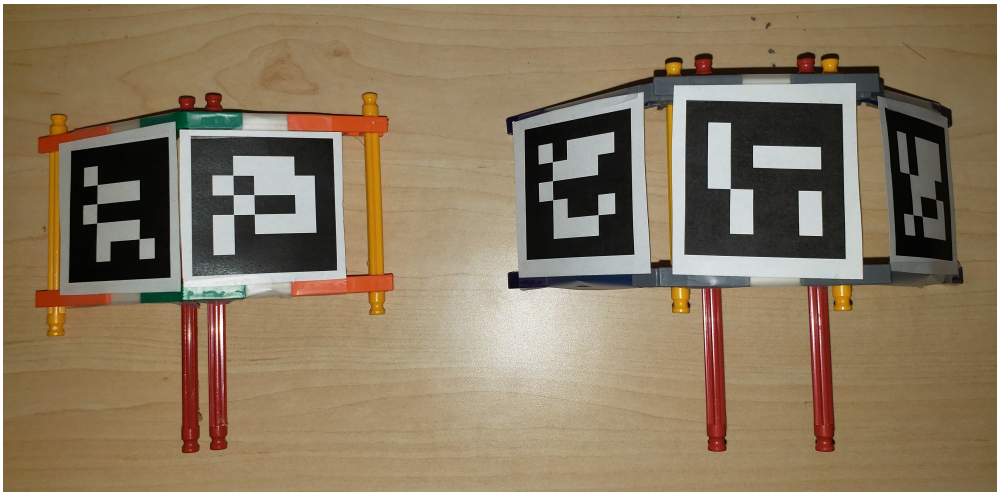

Figure 2.2: Examples of multiple trackers assembled. The left shows a set with 2 trackers, the right shows a set with 3 trackers. Both of these constructions were observed to have less overall error than a single tracker.

3-tracker set where a center tracker was bordered by two trackers on the left and right at a 135° angle. These sets are pictured in Figure 2.2. In our tests, we placed a camera with resolution 1280x720 pixels approximately 1.5 meters away from the tracker sets and recorded the error of the trackers' position. We found that the 1-tracker had a median error of 33mm (variance of 0.91mm), the 2-tracker system had a median error of 8.0mm (variance of 0.038mm), and the 3-tracker system had a median error of 5.2mm (variance of 0.055mm). Figure 2.3 shows scatter plots of the reported error. Most notably, we can see that one tracker has a large amount of jitter, whereas 2 and 3 trackers are much more stable. We can also see that the difference between 2 and 3 trackers is largely a stabilization of the Z-axis, or the depth axis.

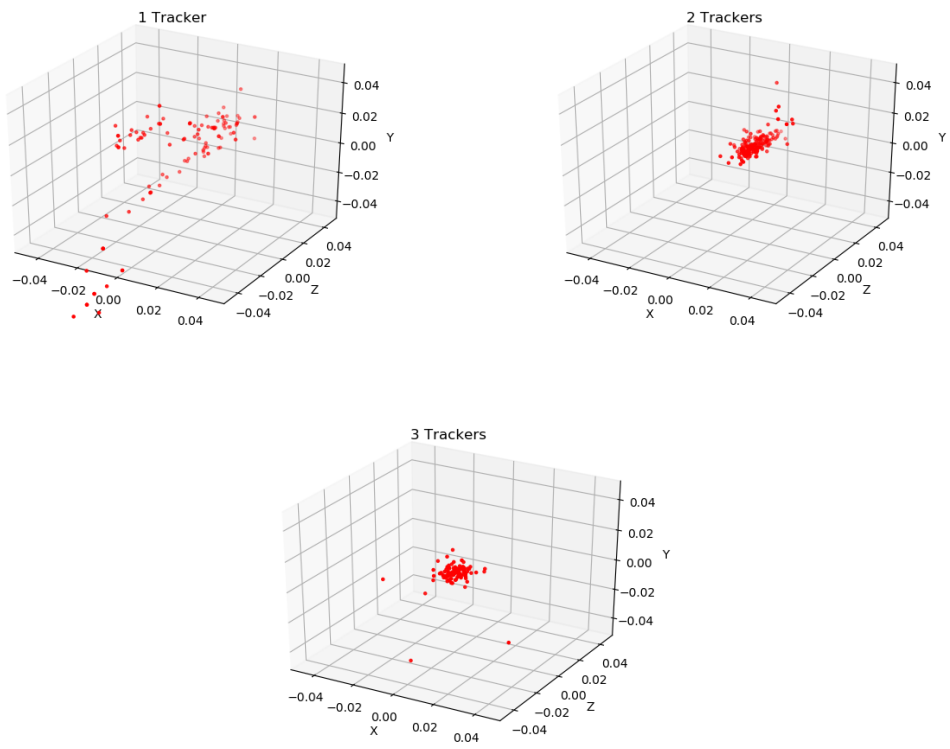

Figure 2.3: Error of the three tracker sets we used, measured in meters. Each scatter plot is a sampling of data points reported from the tracker sets, centered around the average position. The single tracker reported a large amount of instability along all axes. With two trackers, the error is reduced, but still persists around the Z-axis, or the axis pointing out of the camera frame. The three tracker set saw a great reduction in error in all three axes.

2.3.3 Networking Algorithm Implementation

Because we do not use a centralized server to synchronize our clients, each client is responsible for communicating their data to each other client. To model the communications, we take inspiration from ad hoc decentralized networks[7]. Due to a variety of open problems with ad hoc networks[47,6], we did not want to in-

tegrate them into our system, but we found some of the core concepts useful. Instead of directly sending messages from client to client, we still use a router to maintain our connections, but we establish a network topology to determine to which clients a given client should send and forward data.

Our network topology is determined by client visibility. In other words, each client will send data to the clients it can see. We make this choice based on our algorithm for frame synchronization: since the clients' data is only valid when the frames are synchronized, and the frames are only synchronized through visibility, we only want to send data to visible clients. Because we use ARCore, which uses a variant of the SLAM algorithm to maintain its pose as it moves, we can slightly relax our requirement of strict visibility. Instead, we can send data to clients when we are confident our last frame synchronization is still correct. In systems with little drift, we can maintain this confidence for a long period of time after losing tracking, but systems with greater drift must re-establish visibility more often to maintain confidence. Overall, each system must be evaluated individually to measure drift over time.

We can further enhance the overall architecture by allowing clients to forward data they receive, propagating that data throughout the network. Each packet maintains a list of clients C that has seen this data, starting with the original sender. When a client receives a packet, it first transforms the data into its frame using the algorithm described above. Then, it forwards it to all of its edges, or visible clients, as long as that client is not included in C. This forwarding method propagates the data in a manner similar to breadth-first search

algorithms, so it is known that the maximum number of hops required is linear with the number of edges and clients. By doing this, clients within the connected graph structure can communicate with each other throughout a room, even if they are not directly visible to one another.

2.3.4 Applications of this Method

This method has several applications in the context of co-located AVR. First, it allows for two or more clients to collaborate and communicate shared graphics with consistent poses. As described before, this method makes it simple to broadcast object data throughout our network and have each client observe the object in its correct position within their frame.

Additionally, this method can be used to combat the problem of occlusion that has been previously mentioned. If we use this method coupled with another tracking method, such as the ones described in Chapter 1, then the ground truth established by devices such as the OptiTrack cameras or Lighthouse stations only need to see some of our clients. Those clients can then distribute the ground truth frame throughout the network. In doing this, a client that may be occluded from the ground truth devices can still obtain the ground truth data.

Finally, we believe this distributed tracking method can be combined with other tracking algorithms to improve their performance. Many tracking algorithms rely on room-mapping, where the device generates a set of trackable feature points throughout the room. As mentioned earlier, some systems resolve frame differences between clients by independently calculating and com-

paring feature points. However, this requires each client to scan the room for feature points individually, and upload a large data set to a cloud server. Instead, if we can resolve frame differences ahead of time, separate clients can map different parts of the room and easily combine the feature sets, thus distributing the problem. In this way, our method does not seek to replace other established methods, but instead augment them for the context of multiple, co-located users. In future work, we plan to look into integrating our methods with these other systems.

3

Future Development of Co-Located AVR

Introduction

Now that we have discussed the capabilities of co-located AVR and solutions to make these systems work, it is important to consider how we can adapt them to utilize future technologies on the horizon. Throughout our previous chapters, we have discussed problems brought about by network connectivity, optical track-

ing, and mobile rendering. While our solutions and paradigms have been able to create co-located AVR environments despite these limitations, new technologies may allow us to scale to larger systems with greater ease.

Many of these improvements will change the fundamental paradigms of AVR. However, the insights and algorithms from the previous chapters will not be obsoleted by these developments. Instead, it is important to understand how our work will fit into the evolving AVR paradigms with each of these advancements. We will discuss some of the upcoming technologies, how they will impact AVR systems, and how our work can integrate with them.

3.1 Robust Tracking Methods

The next generation of AVR devices will arrive with improvements to tracking algorithms within the following years. In particular, many devices are beginning to use robust inside-out tracked methods, like the SLAM-based algorithms mentioned in earlier chapters. As opposed to relying on external tracking systems to track and relay pose data to them, these devices calculate their own local pose internally. This sort of tracking has already been seen in certain devices, such as the forthcoming Oculus Quest[21] and Microsoft Hololens[52]. While *CAVE* was run using an inside-out tracked system, it had limited capability and did not allow the user to move around a room, unlike some of the newer systems.

Many of our experiences described in Chapter 1, particularly the active participation experiences, would benefit from this model of tracking. As we discussed, those experiences saw issues when trying to relay tracking data over a

busy network. Allowing the local tracking to be computed within the device would reduce instances of VR sickness, even in high latency situations. While we would use the networks to communicate ground truth data such that the clients all exist within the same virtual frame, this synchronization does not need to be computed every render frame, especially if the devices do not incur large amounts of drift.

Furthermore, many of the algorithms discussed in Chapter 2 were designed with these sorts of inside-out tracking methods in mind. The algorithms for client frame synchronization are most useful when clients can track themselves without an external system, and only need to solve the difference between each others' frames.

Finally, as discussed at the end of Chapter 2, the distributed tracking method works well within these sorts of methods. Many of these methods use landmarks throughout the room to understand their position. Through our method, separate devices could combine their landmark sets to create even more robust room mappings.

3.2 5G Networks

Within the next few years, we can expect to see 5G wireless networks, the successor of the 4G networks that are common today[68]. These networks promise higher bandwidth, lower latency, and higher reliability than existing generations. For comparison, 4G networks offer a data rate of approximately 20 megabits per second (Mbps) at 60ms of latency, while 5G networks can practically of-

fer up to about 1 gigabit per second (Gbps) at 1ms latency[26]. These enormous jumps will enable paradigms previously impossible. In the context of the AVR systems we have discussed, 5G can provide several considerable benefits.

3.2.1 Reliable Low Latency Networks

A large portion of our earlier discussion on AVR systems focused on limitations imposed by latency and network reliability. Throughout several of our SIGGRAPH experiences, we encountered problems with WiFi connectivity due to congested WiFi bands at the busy venue. Higher reliability and performance in dense networks granted by 5G[28] would solve some of these problems, making the use of networked AVR systems much more viable in busy areas. Additionally, some of the tracking paradigms we discussed in earlier systems were dependent on network connectivity. Having this reliability in those sorts of systems will lead to smoother tracking experiences.

Finally, while our systems have been able to function on dedicated WiFi networks, the current latency of 4G means that they could not function properly over 4G. The ability to function outside of dedicated WiFi networks is necessary for AVR systems to be ubiquitously used.

3.2.2 Edge Compute Nodes

With greater bandwidth, entirely new paradigms could be imagined for AVR tracking, processing, and rendering. Current HMDs must accommodate mobile processors, batteries, and other such hardware, making them bulky. A common

concept discussed with 5G networking is the idea of an edge compute node[91]. This node would serve as a powerful data processor for a low-powered device, with data being streamed between the device and server over the 5G network.

We could imagine the above applying to an AVR device. Suppose the AVR device functions with a camera-based inside-out tracking solution, such as the SLAM methods previously discussed. Then, instead of processing the costly SLAM algorithm on the device itself, the device could send its camera stream to an edge compute node, which can process the data much quicker than a mobile device, and send the tracking results back to render. Even further, we could imagine that the edge compute node could fully render the proper image itself and send it back to the device. This sort of paradigm of offsite rendering has been explored for video streaming contexts[66], but could easily apply to AVR. In this system, the AVR device would need barely any processing power, and thus the size and weight of the device could be greatly reduced.

Finally, edge compute nodes can act as cloud storage for large databases, such as AR landmark anchors. These landmark databases are important for storing persistent tracking models of rooms. In current AR paradigms, it is common to establish these anchors to maintain persistence of tracked spaces and share these persistent models with other users. However, storing these anchors often requires large amounts of data transfer and storage. Dedicated 5G-enabled edge compute nodes that interface with the cloud could reduce some of the upload and download times of these anchors, allowing spaces to be reestablished and shared faster, especially in locations without dedicated WiFi networks.

3.3 The NYU Holodeck

While much of the work we have discussed has focused on co-located AVR systems, telepresence is undeniably an important consideration when discussing the powers of AVR. However, co-located AVR and telepresent AVR are not mutually exclusive. Analogously to how we commonly see video conferencing where some members of the conference are physically seated in one room, we could imagine an AVR experience where several remotely located clusters of co-located users are joined together through high speed networks.

The NYU Holodeck is an ongoing development to create a platform that allows users to combine the paradigms of co-located and telepresent AVR. We seek to apply the lessons we learned from the experiences developed throughout Chapters 1 and 2 to enable an AVR system that enables entirely new forms of interaction, collaboration, and learning. This platform will be used as a supercomputing infrastructure for NYU students, faculty, and researchers to develop new ideas through our techniques. As the system develops, we envision it as a powerful tool that could be distributed out to other communities as field kits, allowing them to utilize the same interactions.

Developing these kinds of collaborative interactions and learning techniques has been studied in other AVR systems, such as the data visualization and manipulation tools available in the CAVE systems[9]. We seek to create similar capabilities within the context of our own AVR systems, as well as to further augment them with the infrastructure for research in audio, human cognitive and

affective states, and other fields.

3.3.1 High Speed Research Network

Communications throughout the Holodeck are driven by a high-speed research network enabled by NYU Research Technology (NYU RT). This network offers up to 1Gbps network speeds to endpoints with a 10ms wireless latency[44]. Similar to the discussion regarding 5G above, this bandwidth and latency greatly improves the possibility of interaction compared to many current network hardware architectures.

In order to facilitate connections throughout the Holodeck system, we use a relay server named CoreLink. This server uses a TCP control stream to initiate connections with various clients, allowing them to log into the system and designate which types of data they intend to send and receive. These data streams can use UDP, TCP, or WebSocket protocols. From there, the relay can connect multiple nodes by matching sending data streams to the clients that have subscribed to that type of data stream.

Furthermore, the relay can forward data from clients into custom plugins that process the data. For instance, a client could send an audio stream to a specified plugin to process voice commands. The result of that processing is then distributed out to the appropriate subscribers. In doing this, we can offload expensive computations to a dedicated server. To facilitate large transmissions, communications between the internal servers of the network have 100Gbps network speeds, which allows the relay to quickly distribute data across several compute

nodes if needed.

3.3.2 Applications and Modules

While the Holodeck seeks to be flexible enough to accommodate any number of different applications, it is useful to talk about the ones we have implemented and the experiences we have developed. Our current software currently focuses on sending 3D data, such as avatar tracking data, audio data, such as voice chat, and affective data, such as heart rate sensors. Using these capabilities, we have constructed a virtual conference room. Users wearing tracked markers, like Vive trackers, can send their 3D data to the relay, which then sends the data to a plugin to construct an avatar using an IK system similar to Holojam. This avatar data is forwarded to all other members of the virtual conference room, and the avatar is rendered. Additionally, we forward audio data in a similar matter, combining it with the head pose to correctly position the spatialized audio source at the user's head. In this way, multiple people can sit in the same virtual space with limited motion and speak with each other. As this system is developed, we plan to add capabilities for more interactivity, such as Holojam-like line drawing or Chalktalk[65].

Additionally, the Holodeck software has been used to create two distributed concerts, where musicians in separate physical locations performed together. This sort of performance required extremely low latency for musical coordination, which was enabled by our network. Additionally, dancers on the stage performed with motion captured dancers at another location. For the remote

dancers, we used a similar software paradigm to the one described above to render the avatars, except we used full motion capture suits as opposed to a few tracked points. The avatars were rendered onto a projection screen on stage, which allowed for the physically present and remote dancers to synchronize a performance.

3.3.3 Future Work

As the Holodeck develops, we will seek to add more interactivity into the system, as well as enabling many of the paradigms we discussed in previous chapters within the system. Development of software applications will allow for users to perform richer interactions with each other, view and analyze complex visualizations, and complete collaborative simulations. Users will be able to join in as passive observers, and will be able to spectate the interactions live without the need for tracking setups. While our initial groundwork has built a few simple demonstrations, we see boundless possibilities for the Holodeck system.

Conclusion

Co-located AVR systems have the potential to revolutionize how we think about interactions and communication. Over the past 5 years, we have sought to demonstrate the power of co-located AVR and prototype the new forms of interaction enabled by these systems. To do this, we have designed a set of co-located VR experiences to test different forms of interaction, evaluated the successes of each of these experiences, and resolved issues that have made co-located AVR difficult to implement.

Overall, we found that people responded very positively to our experiences, whether they were doodling 3D art, dancing as colorful robots, or watching a lean-back experience together. In our observations and collection of user feedback, we found that people appreciated the co-located format, being able to interact with strangers and friends alike in VR. Most of the negative feedback was in response to failures of the tracking systems, typically either due to networking failure or occlusion.

However, the future is bright. Promising solutions will provide clients with room-scale tracking without the need for external software or network-based tracking. While many of these systems do not account for client frame synchronization, we have developed solutions that allow users to coexist in the same physical and virtual space. Our solutions are lightweight, simple, and do not require hardware other than the cameras and sensors frequently found on the devices.

We look forward to the coming years of AVR, where the lessons we learned and algorithms we have developed can empower these future systems. Within the next decade, we expect AVR to become a powerful upgrade to human interaction, just as our prototypes have demonstrated. The work presented here is a toolkit for implementing compelling co-located AVR systems and experiences. We hope that the ideas we have documented will be used in further research to push co-located AVR forward as a world-changing paradigm.

While this research document concludes here, we believe that we are just at the beginning of a new technological era.

A

Glossary of Terms

In this appendix, we define and detail some of the terms commonly used throughout this dissertation.

360 Video - Video recordings where an omnidirectional camera captures a 360° view of its surroundings. These recordings are mapped to the inside of a sphere and rendered such that the user is placed in the middle of the sphere.

In these experiences, users can rotate their head to view any angle of the video capture, but the user cannot move their head position through the virtual space. Also known as *360° video*.

Augmented Reality (AR) - A system that allows a real-world environment to be augmented with virtual graphics. Similar experiences are also commonly referred to as *mixed reality*. Examples of augmented reality include the Microsoft Hololens[52] and Mira Prism[54].

Co-location - Users of an augmented or virtual reality system existing within both the same physical and virtual space.

Degrees of Freedom (DOF) - While this has many definitions depending on context, within this document we use it to describe tracking capabilities of a system. A 3DOF system supports tracking of a user's head rotation, and a 6DOF system supports tracking of a user's head rotation and position.

Drift - Inaccuracies in pose reporting due to integration or numeric precision errors.

Frame - This term can have many definitions, but within Chapter 2, it is most commonly used to describe the origin and forward vector of a virtual world.

Frames Per Second (FPS) - The frequency of render updates made within a virtual system.

Head-mounted Display (HMD) - A display device that is mounted on a user's head that places a screen in front of one or both of the user's eyes. These devices are common for powering current augmented and virtual reality systems.

Inertial Measurement Unit (IMU) - An electronic sensor that measures a va-

riety of forces, frequently to calculate acceleration and angular velocity. These sensors are common within augmented and virtual reality headsets to calculate a user's head pose.

Inside-out/Outside-in Tracking - A pair of terms that categorize tracking systems. Inside-out tracking describe systems in which sensors are placed on an augmented or virtual reality device, and these sensors track features within the environment. Outside-in tracking describe systems in which trackable features are placed on the device, and external sensors, such as cameras, detect these features and report the device pose to the device. Outside-in systems are currently used when ground-truth tracking or highly accurate tracking systems are desired. However, inside-out tracking solutions are becoming more and more common as the technology evolves, as they allow for larger tracking volumes and do not require external infrastructure.

Inverse Kinematics - A method for simulating the motion of a digital character based on a set of target positions, such as a character's hands, head, and feet. These methods are useful in virtual reality systems for creating an avatar without full motion tracking.

Motion-to-photon Latency - A term used to describe the delay between moving and seeing that motion rendered within a virtual environment. Minimizing this latency is important in virtual reality systems.

Pose - A position and orientation. A pose is frequently defined using a three-dimensional vector and a quaternion.

Simultaneous Localization and Mapping (SLAM) - An inside-out tracking

algorithm that combines data from internal sensors with room-mapping techniques to resolve a user's head pose. Some examples of current SLAM algorithms are ORB-SLAM[56] and LSD-SLAM[16].

Telepresence - A term used to describe a system that allows for a user to interact within a remote location as though they were physically present.

Virtual Reality (VR) - A system where a user is placed within a virtual environment. In many contemporary VR systems, these environments are completely immersive, such as the headsets produced by Oculus[22], HTC[38], and Google[32].

Virtual Reality Sickness (VR Sickness) - A term used to describe the motion sickness or vertigo experienced within virtual reality systems. Also known as *simulator sickness*.

References

[1] Abrash, M. (2014). What VR could, should, and almost certainly will be within two years. *Steam Dev Days, Seattle*, 4.

[2] Apple (2019). ARKit - Apple Developer. https://developer.apple.com/arkit/.

[3] Artanim Interactive (2015). Real Virtuality. http://artaniminteractive.com/real-virtuality/.

[4] Blackburn, T., Guskiewicz, K. M., Petschauer, M. A., & Prentice, W. E. (2000). Balance and joint stability: The relative contributions of proprioception and muscular strength. *Journal of Sport Rehabilitation*, 9(4), 315–328.

[5] Bonato, F., Bubka, A., Palmisano, S., Phillip, D., & Moreno, G. (2008). Vection change exacerbates simulator sickness in virtual environments. *Presence: Teleoperators and Virtual Environments*, 17(3), 283–292.

[6] Bouckaert, S., Naudts, D., Moerman, I., & Demeester, P. (2008). Making ad hoc networking a reality: Problems and solutions. 1.

[7] Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y.-C., & Jetcheva, J. G. (1998). A performance comparison of multi-hop wireless ad hoc network routing protocols. In *MobiCom*, volume 98 (pp. 85–97).

[8] Carroll, L. (1872). *Alice's adventures in wonderland / by Lewis Carroll.* MacMillan London.

[9] Cruz-Neira, C., Sandin, D. J., DeFanti, T. A., Kenyon, R. V., & Hart, J. C. (1992). The CAVE: Audio Visual Experience Automatic Virtual Environment. *Commun. ACM*, 35(6), 64–72.

[10] Curtis, C., Eisenmann, D., El Guerrab, R., & Stafford, S. (2016). The Making of Pearl, a 360° Google Spotlight Story. In *ACM SIGGRAPH 2016 VR Village*, SIGGRAPH '16 (pp. 21:1–21:1). New York, NY, USA: ACM.

[11] DAFFY LONDON (2019). DAFFY LONDON. http://www.daffy.london.

[12] DiZio, P. & Lackner, J. R. (1997). Circumventing side effects of immersive virtual environments. In *HCI (2)* (pp. 893–896).

[13] Dreamscape Immersive (2019). Dreamscape. A VR experience like no other. https://dreamscapeimmersive.com/.

[14] Dushman, A. (1962). On gyro drift models and their evaluation. *IRE Transactions on Aerospace and Navigational Electronics*, ANE-9(4), 230–234.

[15] Eggert, L. & Fairhurst, G. (2008). *Unicast UDP Usage Guidelines for Application Designers*. Technical report.

[16] Engel, J., Schöps, T., & Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. In D. Fleet, T. Pajdla, B. Schiele, & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (pp. 834–849). Cham: Springer International Publishing.

[17] Euston, M., Coote, P., Mahony, R., Kim, J., & Hamel, T. (2008). A complementary filter for attitude estimation of a fixed-wing uav. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 340–345).

[18] Evangelakos, D. & Mara, M. (2016). Extended timewarp latency compensation for virtual reality. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (pp. 193–194).: ACM.

[19] Facebook Technologies, LLC (2018). Oculus Go: Standalone VR Headset — Oculus. https://www.oculus.com/rift/.

[20] Facebook Technologies, LLC (2019a). Oculus. https://www.oculus.com/.

[21] Facebook Technologies, LLC (2019b). Oculus Quest — Oculus. https://www.oculus.com/quest/.

[22] Facebook Technologies, LLC (2019c). Oculus Rift: VR Headset for VR Ready PCs — Oculus. https://www.oculus.com/rift/.

[23] Febretti, A., Nishimoto, A., Thigpen, T., Talandis, J., Long, L., Pirtle, J. D., Peterka, T., Verlo, A., Brown, M., Plepys, D., Sandin, D., Renambot, L., Johnson, A., & Leigh, J. (2013). CAVE2: a hybrid reality environment for immersive simulation and information analysis. *Proc.SPIE*, 8649, 8649 – 8649 – 12.

[24] Fernandes, A. S. & Feiner, S. K. (2016). Combating vr sickness through subtle dynamic field-of-view modification. In *2016 IEEE Symposium on 3D User Interfaces (3DUI)* (pp. 201–210).

[25] FIEDLER, G. (2009). UDP vs. TCP. http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/.

[26] Ford, R., Zhang, M., Mezzavilla, M., Dutta, S., Rangan, S., & Zorzi, M. (2017). Achieving ultra-low latency in 5g millimeter wave cellular networks. *IEEE Communications Magazine*, 55(3), 196–203.

[27] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., & Marín-Jiménez, M. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6), 2280 – 2292.

[28] Ge, X., Tu, S., Mao, G., Wang, C.-X., & Han, T. (2015). 5g ultra-dense cellular networks. *arXiv preprint arXiv:1512.03143*.

[29] Gochfeld, D., Brenner, C., Layng, K., Herscher, S., DeFanti, C., Olko, M., Shinn, D., Riggs, S., Fernández-Vara, C., & Perlin, K. (2018). Holojam in wonderland: Immersive mixed reality theater. In *ACM SIGGRAPH 2018 Art Gallery*, SIGGRAPH '18 (pp. 362–367). New York, NY, USA: ACM.

[30] Google (2010). Protocol Buffers. https://developers.google.com/protocol-buffers/?hl=en.

[31] Google (2015). Google Cardboard. https://vr.google.com/cardboard/. Online; accessed 20-September-2018.

[32] Google (2018). Daydream - Standalone VR. https://vr.google.com/daydream/standalonevr/.

[33] Google (2019). Fundamental Concepts — ARCore — Google Developers. https://developers.google.com/ar/discover/concepts.

[34] He, Z., Zhu, F., Perlin, K., & Ma, X. (2018). Manifest the Invisible: Design for Situational Awareness of Physical Environments in Virtual Reality. *CoRR*, abs/1809.05837.

[35] Higgins, W. T. (1975). A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, AES-11(3), 321–325.

[36] Hologate (2019). HOLOGATE - VR Virtual Reality + XR Extended Reality Solutions. http://hologate.com/.

[37] Howlett, E. M. (1990). Wide-angle orthostereo. In *Stereoscopic Displays and Applications*, volume 1256 (pp. 210–224).: International Society for Optics and Photonics.

[38] HTC Corporation (2019a). VIVE — Discover Virtual Reality Beyond Imagination. https://www.vive.com/us/.

[39] HTC Corporation (2019b). VIVE — Vive Focus Developer Kit. https://developer.vive.com/eu/vive-focus-for-developer/.

[40] Huo, K., Wang, T., Paredes, L., Villanueva, A. M., Cao, Y., & Ramani, K. (2018). Synchronizar: Instant synchronization for spontaneous and spatial collaborations in augmented reality. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18 (pp. 19–30). New York, NY, USA: ACM.

[41] Islam, T., Islam, M. S., Shajid-Ul-Mahmud, M., & Hossam-E-Haider, M. (2017). Comparison of complementary and kalman filter based data fusion for attitude heading reference system. *AIP Conference Proceedings*, 1919(1), 020002.

[42] Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5), 922–923.

[43] Kolasinski, E. M. (1995). *Simulator Sickness in Virtual Environments.* Technical report, Army research Inst for the behavioral and social sciences Alexandria VA.

[44] Kyriannis, J. (2004). NYU-NET^3 our third-generation campus network.

[45] Lee, J., Kim, M., & Kim, J. (2017). A Study on Immersion and VR Sickness in Walking Interaction for Immersive Virtual Reality Applications. *Symmetry*, 9(5).

[46] Liu, Y., Yang, J., & Liu, M. (2008). Recognition of qr code with mobile phones. In *2008 Chinese control and decision conference* (pp. 203–206).: IEEE.

[47] Macker, J. (1999). Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations.

[48] Magic Leap, Inc. (2019). Home — Magic Leap. https://www.magicleap.com/.

[49] Manchester, D., Woollacott, M., Zederbauer-Hylton, N., & Marin, O. (1989). Visual, Vestibular and Somatosensory Contributions to Balance Control in the Older Adult. *Journal of Gerontology*, 44(4), M118–M127.

[50] Michael Antonov (2015). Asynchronous Timewarp Examined — Oculus. https://developer.oculus.com/blog/asynchronous-timewarp-examined/.

[51] Microsoft (2019a). Azure Spatial Anchors — Microsoft Azure. https://azure.microsoft.com/en-us/services/spatial-anchors/.

[52] Microsoft (2019b). Microsoft hololens — mixed reality technology for business. https://www.microsoft.com/en-us/hololens.

[53] Minsky, M. (1980). Telepresence.

[54] Mira Labs, Inc. (2019). Mira Augmented Reality. https://www.mirareality.com/.

[55] Money, K. E. (1970). Motion sickness. *Physiological Reviews*, 50(1), 1–39. PMID: 4904269.

[56] Mur-Artal, R., Montiel, J. M. M., & Tardós, J. D. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5), 1147–1163.

[57] Niantic, Inc. (2016). Pokémon go. https://www.pokemongo.com/en-us/.

[58] Oculus VR, LLC (2016). Samsung Gear VR. https://www.oculus.com/en-us/gear-vr/.

[59] Oculus VR LLC (2019). Oculus Developer Center. https://developer.oculus.com/.

[60] Oda, O., Elvezio, C., Sukan, M., Feiner, S., & Tversky, B. (2015). Virtual replicas for remote assistance in virtual and augmented reality. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, UIST '15 (pp. 405–415). New York, NY, USA: ACM.

[61] Olson, J. L., Krum, D. M., Suma, E. A., & Bolas, M. (2011). A design for a smartphone-based head mounted display. In *2011 IEEE Virtual Reality Conference* (pp. 233–234).: IEEE.

[62] OpenCV Team (2019). OpenCV library. https://opencv.org/.

[63] OptiTrack (2016). OptiTrack. http://www.optitrack.com/.

[64] Penney, D. & Chung, E. (2017). Arden's Wake. In *ACM SIGGRAPH 2017 Computer Animation Festival*, SIGGRAPH '17 (pp. 28–28). New York, NY, USA: ACM.

[65] Perlin, K., He, Z., & Zhu, F. (2018). Chalktalk vr/ar. *International SERIES on Information Systems and Management in Creative eMedia (CreMedia)*, (2017/2), 30–31.

[66] Qiao, J., He, Y., & Shen, X. S. (2016). Proactive caching for mobile video streaming in millimeter wave 5g networks. *IEEE Transactions on Wireless Communications*, 15(10), 7187–7198.

[67] Quantize LLC (2019). Superbright. http://www.superbright.me/.

[68] Rappaport, T. S., Xing, Y., MacCartney, G. R., Molisch, A. F., Mellios, E., & Zhang, J. (2017). Overview of millimeter wave communications for fifth-generation (5g) wireless networks—with a focus on propagation models. *IEEE Transactions on Antennas and Propagation*, 65(12), 6213–6230.

[69] Research, G. S. G. I. (2016). *Virtual and augmented reality: Understanding the race for the next computing platform*. Technical report.

[70] Samsung (2015a). Samsung Galaxy Note 4 (Black) - Full Specs — Samsung UK. https://www.samsung.com/uk/smartphones/galaxy-note-4-n910f/SM-N910FZKEBTU/.

[71] Samsung (2015b). Specifications — Samsung Galaxy S8 and S8+. https://www.samsung.com/global/galaxy/galaxy-s8/specs/.

[72] Samsung (2016). Specifcations — Samsung GearVR with Controller. https://www.samsung.com/global/galaxy/gear-vr/specs/.

[73] Seyama, J. & Nagayama, R. S. (2007). The uncanny valley: Effect of realism on the impression of artificial human faces. *Presence: Teleoperators and Virtual Environments*, 16(4), 337–351.

[74] Shoemake, K. (1985). Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3), 245–254.

[75] Singer, B. (1988). Early home cinema and the edison home projecting kinetoscope. *Film History*, (pp. 37–69).

[76] Smith, D. A., Wiese, G. E., Cuddihy, G. C., & Harrison, G. A. (2017). Head-mounted display apparatus employing one or more fresnel lenses. US Patent 9,632,315.

[77] Snap Inc. (2019). Snapchat - The fastest way to share a moment! https://www.snapchat.com/.

[78] Suma, E. A., Azmandian, M., Grechkin, T., Phan, T., & Bolas, M. (2015). Making Small Spaces Feel Large: Infinite Walking in Virtual Reality, booktitle = ACM SIGGRAPH 2015 Emerging Technologies. SIGGRAPH '15 (pp. 16:1–16:1). New York, NY, USA: ACM.

[79] Sutherland, I. E. (1968). A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (pp. 757–764).: ACM.

[80] Tang, J. C., Wei, C., & Kawal, R. (2012). Social Telepresence Bakeoff: Skype Group Video Calling, Google+ Hangouts, and Microsoft Avatar Kinect. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work Companion*, CSCW '12 (pp. 37–40). New York, NY, USA: ACM.

[81] Technologies, U. (2018). Unity. =https://unity3d.com/.

[82] The Void (2016). The Void: The Vision Of Infinite Dimensions. https://thevoid.com/.

[83] VRcade, Inc (2015). The VRcade: A Virtual Reality Platform for Truly Immersive Gaming. http://vrcade.com/.

[84] VRChat Inc. (2019). Vrchat. https://www.vrchat.net/.

[85] Welch, G., Bishop, G., et al. (1995). An introduction to the kalman filter.

[86] Whittinghill, D. M., Ziegler, B., Case, T., & Moore, B. (2015). Nasum virtualis: A simple technique for reducing simulator sickness. In *Games Developers Conference (GDC)*.

[87] Wingrave, C. A., Williamson, B., Varcholik, P. D., Rose, J., Miller, A., Charbonneau, E., Bott, J., & LaViola, J. J. (2010). The Wiimote and Beyond: Spatially Convenient Devices for 3D User Interfaces. *IEEE Computer Graphics and Applications*, 30(2), 71–85.

[88] Wu, P.-C., Wang, R., Kin, K., Twigg, C., Han, S., Yang, M.-H., & Chien, S.-Y. (2017). Dodecapen: Accurate 6dof tracking of a passive stylus. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17 (pp. 365–374). New York, NY, USA: ACM.

[89] Xia, H., Herscher, S., Perlin, K., & Wigdor, D. (2018). Spacetime: Enabling Fluid Individual and Collaborative Editing in Virtual Reality. *UIST*.

[90] Xylomenos, G. & Polyzos, G. C. (1999). TCP and UDP performance over a wireless LAN. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 2 (pp. 439–446 vol.2).

[91] Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., & Zhang, Y. (2016). Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE access*, 4, 5896–5907.

[92] Zhang, Z. (2012). Microsoft Kinect Sensor and Its Effect. *IEEE MultiMedia*, 19(2), 4–10.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, "Science Experiment 02", was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (X11) license, and can be found online at github.com/suchow/Dissertate or from its author, Jordan Suchow, at suchow@post.harvard.edu.