UNIVERSITY OF CALIFORNIA
RIVERSIDE

Quality of Experience and Security for Augmented and Virtual Reality Applications

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Carter Philip Slocum

June 2023

Dissertation Committee:

Dr. Jiasi Chen, Chairperson
Dr. Nael Abu-Ghazaleh
Dr. Craig Schroeder
Dr. Michalis Faloutsos

The Dissertation of Carter Philip Slocum is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First and foremost I like to thank and acknowledge my advisor, Professor Jiasi Chen, for her hard work and confidence in my ability to do research without which I may not have even started the PhD. I also thank her for her guidance and encouragement to investigate problems new to both of us of which the work of this dissertation is largely comprised of. Prof Chen's ability to advise myself and several other students both in person and remotely during the year of the pandemic allowed a steady progress towards becoming the academic I am today.

Next I would like to thank the other members of my committee. Professor Nael Abu-Ghazaleh has given guidance in the last few years of research at UCR and helped bring myself up-to speed on the norms of security research while remaining adaptable to the new research areas being brought to the table by our lab. Collaboration with Prof. Nael and his students were critical in the completion of this work. Professor Craig Schroeder has helped on multiple occasions provided a perspective on my work from a new point of view and helped point out the mathematical fixes that enabled progress in the engineering of the programs described in this dissertation. Professor Michalis Faloutsos has provided direct advice on the structure of this dissertation and valuable feedback on improvements to research presentations both given in the past and upcoming.

Several fellow students have helped me in the work for this dissertation. Xukan Ran (PhD) helped me cut my teeth on XR research in the first quarters of the PhD program and gave great advice on not just research but succeeding in graduate school in general. Yicheng Zhang has worked most closely with me on the last years of research projects and

been instrumental on the completion of three of these. While still early in his PhD work, I have every confidence in his future endeavors. Yi-Zhen "Angela" Tsai and Yunshu Wang as fellow advisees, took interest in my work and asked questions that helped shape not only the direction of the research itself but improve the presentations. Lastly Jingwen Huang, a summer research student had proven invaluable in reaching deadline under the stress of the pandemic and not only held clear understanding of the research, but gave helpful feedback on the direction and communication of the work to the research community.

A special thanks to Professor Ran Libeskind-Hadas for bringing me onto a research project during undergrad that lead to my initial interest in pursuing a PhD.

Financial support for this work are acknowledged in their respective chapters.

Finally I would like to acknowledge my parents Phil and Kara Slocum for their love and unwavering support. I also wish to acknowledge my sisters Jordan and Kennedy who likewise have shared their love and life experience, giving me perspective. Together my family have formed an invaluable support system to whom I dedicate this dissertation.

To Phil and Kara, loving parents

To Jordan and Kennedy, loving siblings

ABSTRACT OF THE DISSERTATION

Quality of Experience and Security for Augmented and Virtual Reality Applications

by

Carter Philip Slocum

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2023
Dr. Jiasi Chen, Chairperson

With the increasing adoption of Augmented Reality/Virtual Reality (AR/VR - XR) systems, security, privacy, and quality concerns attract attention from both academia and industry. We identify four key problems in XR applications and make contributions respectively.

For XR Quality of experience: AR applications suffer from spatial inconsistency where virtual objects "drift or "jump" in space. We develop an automated method to evaluate the spatial inconsistency of AR applications. Web-based XR suffers from long latency in downloading virtual content to be displayed on a web browser. We develop a technique to divide up and prioritize virtual objects to be downloaded in a manor to minimize the time to first correct image.

For XR Security: Head mounted displays track the user's head pose at all times in order to render content correctly. We show that these head poses can be used to help infer user hand-typed words in order to steal private information. Multi-use AR applications construct a "shared-state" using image sequences and GPS data. We show that these shared-states can be poisoned allowing for the reading and writing of information to and from arbitrary locations by a malicious user.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The broad goal of this dissertation is the improvement of Virtual Reality (VR) and Augmented Reality (AR) Applications. We focus on two specific goals in that of improving Quality of Experience (QoE) and Security. Currently, many issues unique to VR and AR (XR) applications result in inaccuracy, inefficiency, and vulnerability to attacks. While research into solving these problems have been performed for over a decade, these problems persist and new problems appear as XR expands to new devices and architectures. For Quality, we specifically identify inaccuracy in augmentations and large latency in browser based VR as open problem areas which we add contributions. For Security, we identify new threats to headset typing programs and for shared state AR applications.

1.1 Mixed Reality Overview

XR involve taking in imagery from cameras and, with the help of other sensors such as GPS and accelerometer, create a model or map of the world in which to place virtual content. Virtual content can be individual interactive objects and virtual characters such as

in AR or entire virtual worlds overlaid on the model of reality as in VR. The line between the two technologies has become increasingly blurred in recent years with devices capable of both reaching public adoption in the mobile smartphones and head mounted displays (HMD). Since both AR and VR use combinations of the same input sensors and even similar algorithms for tasks such as rendering, localization, and networking, we differentiate between the two by the output to the display. AR in this dissertation refers to applications that overlay virtual content onto imagery of the real world either in single images but more frequently as sequences of images and videos. VR in this dissertation refers to applications that display entirely virtual worlds, using the understanding of the real world to anchor the perspective of the user and to facilitate immersive interaction with the users body.

We are especially interested in "shared" or multi-user XR. Multiple devices use their own sensors to create models of the real world and place virtual content in them. These devices will then seek to enable collaboration by communicating these models and changes to one another. A core problem to Multi-User XR is that while there is one real world, these devices each make their own unique model and augmentations in it that may not be identical. In order to reconcile these models technology such as computer vision, and , importantly, computer networks are introduced to allow devices to communicate about shared content. The contributions in this dissertation come as a natural product of the research into solutions to the problems that arise from coordinating multiple devices to reconcile content correctly in an efficient manor.

1.2 A Brief History of XR Quality and Security

Artificial representations of reality have been made by humans for over 30,000 years in he form of paintings [163], but what we would consider Virtual Reality did not appear until 1968 with Ivan Sutherland's *Sword of Damocles* [157]. Despite VR's existence for many decades, research has been limited to those researchers that could afford the large costs of prototyping high performance mobile devices. Modern advances in affordable mobile devices and sufficiently powerful mobile graphics processing units (GPUs) have given rise to what Steven M. Lavalle calls the "Rebirth of Virtual Reality" [75].

These advances come from many industries and fields, video games, film, and television have driven the advances in computer graphics while robotics have provided the needed algorithms and sensors for real-time computer vision necessary to bring about modern XR. In fact, XR is by its nature a highly interdisciplinary, Combining not only Computer Graphics and Vision, but elements of mobile devices, networks, human computer interaction, multi-media, and robotics. With widespread affordable XR devices the associated increase in research interest has lead to specialization in particular problem areas of XR.

Quality of Experience (QoE) as a term, comes about as a more general outgrowth of the term Quality of Servce (QoS) from computer networks. QoS is the idea that "Quality" of a service can be measured quantitatively using things like bandwidth, latency, error frequency, ect. QoE comes from the realization that there is a difference between these measures and the end-user experience. QoE is more general but may still use QoS measures, where QoS may measure a computer vision algorithm by is run-time and reported numerical accuracy but QoE will measure it by how that result is used to improve the user's experience.

For example a virtual character drawn on an AR display may be measured by how quickly it was drawn, but it may be more important to check how stably it appears to be placed onto the images of the real world which is a combination of many factors.

XR security and privacy are simply security research applied to XR. This mix of fields when compared to XR as a whole is relatively young, with most research occurring within the last decade. This is somewhat to be expected as security issues did not have sufficient impact with so few users until the recent rebirth. With now hundreds of millions of XR capable devices world-wide, research interest has ignited in kind.

1.3 Summary of Contributions

Our Contributions are organized into four areas: AR accuracy evaluation, Web Based VR latency improvements, XR headset typing privacy, and AR shared-state security. We proceed to give an overview of each of these contributions.

RealityCheck: A tool to evaluate spatial inconsistency in augmented reality. In augmented reality, virtual objects can drift away from their original intended locations, significantly impairing a user's experience. Traditionally, a virtual object's drift is approximated by the device localization drift, which is measured using specialized hardware such as 3D scanners or laser-based positioning systems. However, with AR rapidly becoming more popular, there is a need for a lightweight, software-based approach to evaluate the drift of virtual objects. This software should be easy for researchers and developers to use, without requiring specialized hardware or extensive environment setup.

Towards this, we present RealityCheck, an open-source AR evaluation tool that reports the drift of AR virtual objects in the world coordinate system, requiring only paper

printouts and minimal modifications to the AR app. RealityCheck is designed to measure the drift of a virtual object across time of a single user, as well as the positioning differences of the same virtual objects as seen by multiple users. Our prototype is implemented on an Android smartphone running the ARCore platform, and evaluated in indoor and outdoor scenarios under a variety of user mobility patterns with traces of different lengths. We compared the results of RealityCheck with the ground truth position of the virtual object, and showed that RealityCheck matches the ground truth within 1.5 cm on average.

Visibility-aware Web-based Virtual Reality. New standards such as WebXR enable cross-platform VR experiences, relying on the ubiquity of the modern web browser. However, upon measuring performance of WebXR scenes, we found users can suffer from high latency while waiting for all 3D objects appear in their field-of-view. This is because storage and fetching of 3D objects in WebXR (and its underlying WebGL libraries) are agnostic to the user's orientation and location, leading to latency issues. Specifically, fetching of texture files in arbitrary order results in 3D objects waiting on their texture dependencies, and the storage of all objects' geometry data in one large file blocks individual objects from rendering even if their texture dependencies are satisfied. To address these issues, we propose a systematic prioritization of which 3D objects and their dependencies should be fetched first, based on the user's position and orientation in the VR scene. To improve efficiency, the geometry data belonging to each 3D object are optimally grouped together to minimize the average latency. Our experiments with various WebXR scenes under different network conditions show that our scheme can significantly reduce the time to all 3D objects appearing in the user's field-of-view, by up to 50%, compared the default WebXR behavior.

XR keylogging from user head motions. Augmented Reality/Virtual Reality are the next step in the evolution of ubiquitous computing after personal computers to mobile devices. Applications of XR continue to grow, including education and virtual workspaces, increasing opportunities for users to enter private text, such as passwords or sensitive corporate information. In this work, we show that there is a serious security risk of typed text in the foreground being inferred by a background application, without requiring any special permissions. The key insight is that a user's head moves in subtle ways as she types on a virtual keyboard, and these motion signals are sufficient for inferring the text that a user types. We develop a system, TyPose, that extracts these signals and automatically infers words or characters that a victim is typing. Once the sensor signals are collected, TyPose uses machine learning to segment the motion signals in time to determine word/character boundaries, and also perform inference on the words/characters themselves. Our experimental evaluation on commercial XR headsets demonstrate the feasibility of this attack, both in situations where multiple users' data is used for training (82% top-5 word classification accuracy) or when the attack is personalized to a particular victim (92% top-5 word classification accuracy). We also show that first-line defenses of reducing the sampling rate or precision of head tracking data are ineffective, suggesting that more sophisticated mitigation are needed.

Attacks on Shared-State Augmented Reality Applications. Augmented Reality (AR) is expected to become a foundational component in enabling shared virtual experiences. In order to facilitate collaboration among multiple users, it is crucial for multi-user AR applications to establish a consensus on the "shared state" of the virtual world and its augmentations, through which they interact within augmented reality spaces. Current

methods to create and access shared state collect sensor data from devices (camera images), process them, and integrate them into the shared state. However, this process introduces new vulnerabilities and opportunities for attacks Maliciously *writing* false data to "poison" the shared state is a major concern for the security of the downstream victims that depend on it. Another type of vulnerability arises when *reading* the shared state; by providing false inputs, an attacker can view hologram augmentations at locations they are not allowed to access. In this work, we demonstrate a series of novel attacks on multiple AR frameworks with shared states, focusing on three publicly-accessible frameworks. We show that these frameworks, while using different underlying implementations, scopes, and mechanisms to read from and write to the shared state, have shared vulnerability to a unified threat model. Our evaluation of these state-of-art AR applications demonstrates reliable attacks both on updating and accessing shared state across the different systems. To defend against such threats, we discuss a number of potential mitigation strategies that can help enhance the security of multi-user AR applications.

1.4 Related Problems from Other Fields

Computer Graphics. XR devices render 3D scenes from multiple viewpoints (such as in a head mounted display) and on a lower powered mobile device. There is, consequently, need for efficient algorithms to render these scenes and their view dependant effects quickly to minimize QoE reducing latency. The delay between a users movements and the update of the displayed image is the main source of user discomfort so any advancement in real-time rending for mobile devices has immediate application to improving XR QoE.

Computer Vision. XR devices use cameras on the device to help determine the device pose and to recognize important real world objects. These tasks fall squarely within computer vision. XR is particularly interested in recognizing not just specific objects like markers or animals, but more abstract things like flat planes and walls. A close eye is kept on advances in these problem areas.

Computer Networks. In order to enable shared XR experiences we must send information from one device to another. Computer networks research has some history with sending 3d meshes, , maps, and point clouds but remote rendering remains an open area. XR devices are typically mobile, limiting their power and requiring the use of wireless connections. Being able to store 3D information or send video, rendered using higher powered servers to devices over poor networks will provide a way forward for XR quality beyond what mobile devices alone can do. Areas such as Edge computing are of particular interest to XR as proximity to high powered devices may enable real-time remote rendering at a sufficient frame-rate to hit Quality targets.

Mobile Devices. Mobile phones are largely XR capable devices and head mounted displays have moved towards wireless operation to enable free user movement in XR. Mobile devices must get their power from batteries and be lightweight in order to be easily moved which puts hard restrictions on their compute power. Computer Graphics and Vision solutions may not be viable on mobile due to the lack of resources and require unique approaches. XR research finds solutions to computer vision and graphics tasks by necessity on mobile devices and many solutions can inform the future of mobile devices.

Robotics. Core to the ability for an XR device to find its pose relative to the world is Simultaneous Localization And Mapping (SLAM). The Robotics literature pioneered the algorithms and techniques which are used to solve this problem. XR does not have access to the same array of sensors that a heavier, more expensive robot may and therefore XR research has a unique perspective on the problem. XR researchers solve this problem using low quality cameras and inertial measurement units as opposed to higher quality sensors and LiDAR/SONAR.

Geospatial Information Systems. Global XR has need of some way to orient the virtual content to geographic features and global positions. It is often simpler to augment maps created by Geospatial Information Systems (GIS) like those from Google Earth or OpenStreetMap. When these services have issues with incorrectly located data or malicious map poisoning, the attached virtual content can also be manipulated. The Security of these two technologies are linked through usage of common imagery and GPS tags so any advancement in these may have an impact in both fields.

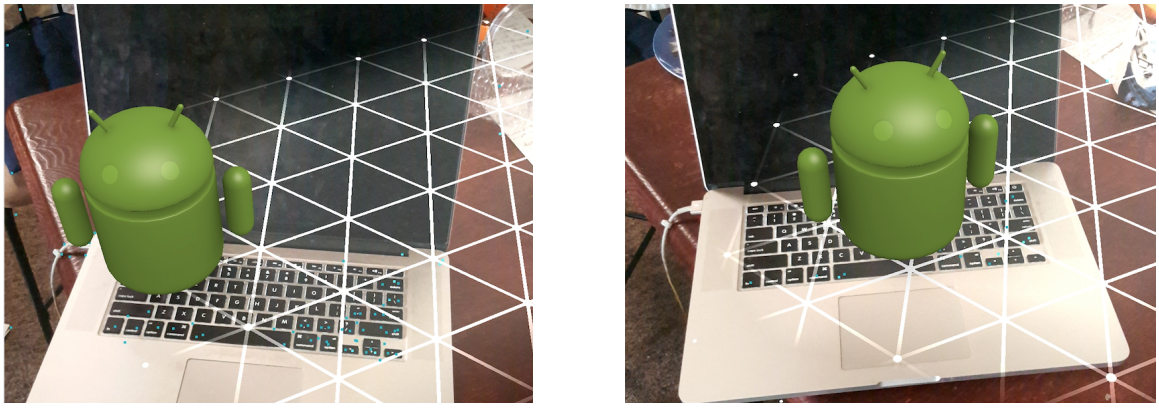
Chapter 2

RealityCheck: A Tool to Evaluate Spatial Inconsistency in Augmented Reality

2.1 RealityCheck Introduction

Mobile Augmented Reality (AR) is becoming increasingly popular, with the AR market estimated to grow to $61 billion by 2023 [49]. Many companies are developing AR platforms and integrating AR into their products. For example, Apple announced its mobile AR platform, ARKit, in 2017 and Google introduced ARCore for Android in 2018 [4, 37]. IKEA has developed a virtual furniture placement app to let users visualize virtual furniture at home [61].

In AR, virtual objects are rendered on the display and overlaid on top of a user's field of view (FoV). To provide a seamless integration with the real world, the AR app needs



(a) Virtual object at time 1.

(b) Virtual object at time 2.

Figure 2.1: Example of a virtual object (Android character) inadvertently drifting to the right over time.

to have an understanding of the surrounding real-world environment [55]; for example, in order to place a virtual cup on a real table, rather than drawing the cup unrealistically floating in the air. AR algorithms to understand the environment can be categorized into three types: (1) object detection based AR (*e.g.*, Snapchat Lenses [43]) in which machine learning or computer vision is used to classify objects in the real world and overlay on top of them; (2) SLAM based AR (*e.g.*, Just A Line [45]) in which visual-inertial sensors are used to create a 3D map of the real world; and (3) marker based AR, which relies on fiducial markers placed in the scene.

However, an AR app's understanding of the environment can be sometimes wrong or inconsistent, particularly if an AR user moves, and the virtual object may unintentionally drift away from its original position. This can significantly disrupt the connection between the virtual object and the real world and thus impair user experience. Fig. 2.1 shows an example from Android ARCore, where the virtual character drifted to the right on top

of the keyboard between time 1 and time 2. A related problem happens when multiple co-located users participate in a shared AR experience and the virtual objects drift across users, appearing to have different locations in the displays of each user [3]. In our experience, such drifts manifest in popular mobile AR platforms, such as Google ARCore and Apple ARKit.

To improve AR applications, it is necessary to measure and evaluate the spatial drift of these virtual objects. However, current methods of doing so suffer from several limitations. Subjective evaluation through user questionnaires is time-consuming and cannot provide real-time feedback or exact quantitative numbers. Objective methods to measure spatial drift are time-consuming, rely on specialized hardware, and/or only work in constrained testing environments. For example, manual labeling of virtual object's position in the scene is time-consuming, requiring multiple seconds for a human to label every single frame when the AR app is running. The most relevant comparison is probably to SLAM performance evaluation, which typically uses Absolute Trajectory Error (ATE) [156], but which suffers from the following limitations. Firstly, ATE measures the difference between the *device's* estimated location and the ground truth location, which is different from the location drift of the *virtual object* (as shown in Section 2.2). Secondly, computing ATE requires knowledge of the ground truth device location, which requires special equipment such as a 3D scanner or motion capture system that only works in a designated testing area, or offline datasets [18] that may not exemplify typical AR use cases. Thirdly, factors such as latency or graphical effects that can affect the final rendered AR object take place after the estimated device trajectory has been recorded, and thus are not accounted for by ATE. Finally, ATE is designed to measure performance of SLAM for an individual user, and does not provide

information in the multi-user scenario. Such requirements pose great inconvenience to researchers who wish to experiment with and evaluate AR.

To address these issues, in this chapter we propose an evaluation tool, called RealityCheck, for SLAM-based AR to directly and conveniently measure the drift of virtual objects in both single-user and multi-user cases. We propose a methodology that does not require specialized hardware or special lab testing environments, making AR evaluation easier and more accessible to general engineers and computer scientists. Our key idea is to temporarily replace the virtual object in the AR app with a virtual marker (*e.g.*, an ArUco marker), and use more accurate computer vision techniques (*i.e.*, PnP) to localize the virtual object/marker in 3D space. By recording the location of the virtual object over time and across users, RealityCheck can compute the spatial drift seen by a single user, or the spatial inconsistency of a virtual object seen by multiple users. We envision such a tool being used by researchers and developers to receive feedback from their AR apps on spatial drift, paving the way for corrections to be made. We focus on SLAM-based AR because it is the foundation of commercial off-the-shelf AR systems, such as Google ARCore, Apple ARKit, and Microsoft Hololens, although the methodology can be extended to other types of AR. We also focus on positioning errors as opposed to rotation errors because they tend to be larger in SLAM-based systems [66].

Overall, RealityCheck makes the following contributions:

- RealityCheck directly measures the spatial drift/inconsistency of a virtual object through a lightweight software-based approach that does not require specialized hardware or testing environments. It only needs a printed marker board, the device camera calibration parameters, and minimal changes to the AR app. This is in contrast to measuring the

ATE of the device, as typically done in SLAM evaluation, which cannot directly measure a virtual object's position.

- RealityCheck works in both single-user and multi-user scenarios. In a single-user scenario, RealityCheck evaluates the spatial drift of a virtual object over time. In a multi-user scenario, RealityCheck evaluates the spatial inconsistency of a virtual object when viewed by multiple users with different FoVs. The key idea in our methodology is to temporarily replace a virtual object with a virtual marker, and use computer vision techniques to accurately track the position of virtual object with respect to the real world.
- RealityCheck achieves 1.5 cm estimation error on average across 22 total trials, compared to the ground truth position of a virtual object. We perform these trials under various experimental conditions: indoor and outdoor environments, changing visibility of the virtual object, different user mobility patterns, and app usage length ranging from 30 seconds to 2.5 minutes.

The open-source code of RealityCheck and a video demonstration of its operation are provided through a website [130]. In the remainder of this chapter, we discuss motivation (Section 2.2), the design of RealityCheck (Section 2.3), quantitative evaluations (Section 2.4), discussion (Section 2.5), related work (Section 2.6), and finally conclude (Section 2.7).

2.2 Motivation: Issues with Manual Labeling and Absolute Trajectory Error

In this section, we provide insight into why manual labeling or ATE are insufficient for measuring drift of an AR virtual object. Note that neither manual labeling nor ATE

measurements are needed for casual users of RealityCheck, but are only needed to evaluate RealityCheck in this chapter.

Manual Labeling: In the case of manual labeling, in our experience, it took approximately 10-30 seconds to hand-annotate the position of a virtual object in each frame. For an AR app updating its display at 30 frames per second (FPS) running for 5 minutes, this could take up to 1.25 hours to measure a virtual object's drift for the duration of a user's experience. Clearly, this is infeasible and unwieldy, particularly if multiple users are participating in the AR experience and each of their frames need to be annotated.

Absolute Trajectory Error: In the case of ATE, as mentioned in Section 3.1, ATE does not provide sufficient information about the position of the *virtual object*, since the device trajectory and ATE only have information about the position of the *device*. The device position is insufficient knowledge about the virtual object, because rendering the virtual object involves projecting the virtual object onto the AR display, which requires both device position and rotation provided by SLAM. Therefore, ATE alone cannot tell the AR device where the virtual object is rendered on the display, and hence what its spatial drift is.

We next describe an experiment we conducted to illustrate why ATE cannot be used to evaluate the spatial drift of an AR virtual object; *i.e.*, why ATE or the device trajectory does not accurately capture the spatial drift of an AR virtual object. This experiment is illustrated in Fig. 2.2. We use ARCore as the AR platform. We started the experiment by creating a virtual object (the tower of shapes) on the floor, and then move backward (in the y direction), without any left/right movement (in the x direction). The height of the device is also fixed so there is also no up and down movement (z direction). In the 3D plot of Fig. 2.2, we plot the SLAM-estimated device trajectory (blue line), which

is a straight line in the XY plane. The SLAM-estimated trajectory matches well with the ground truth device trajectory (red line). Intuitively, we might expect this accurate device position estimate to mean the virtual object won't drift.

However, the SLAM-estimated device trajectory gives us no information about the position of the virtual object on the display, and the virtual object does in fact drift, despite the accurate device position estimates. In Fig. 2.2, we show screenshots of the virtual object at 3 different times. At time 1, the virtual object is directly above the piece of paper. At time 2, the virtual object drifts backward from the paper (towards the user), and at time 3 it drifts forward (away from the user). We don't know where the virtual object is or how much the virtual object drifts by looking at the device trajectory alone, until we see the screenshots of the virtual object. In other words, the accuracy of the device trajectory estimation is not tightly correlated with the drift of the virtual object.

Moreover, in the multi-user scenario, each AR app can update its virtual object position, and then our tool can compute the position difference (spatial inconsistency) of the virtual object between any two users. However, we can't compute the spatial inconsistency across multiple users just from ATE or the trajectory because of the lack of time synchronization and rotation information from ATE alone.

2.3 Design of RealityCheck

The key idea behind the design of RealityCheck is that an AR virtual object can be any object that can be rendered to a screen. Thus it is possible to render a known, easy to detect object into the scene, such as an ArUco marker board [34], and accurately determine its location using computer vision techniques. Combined with a real marker board placed in

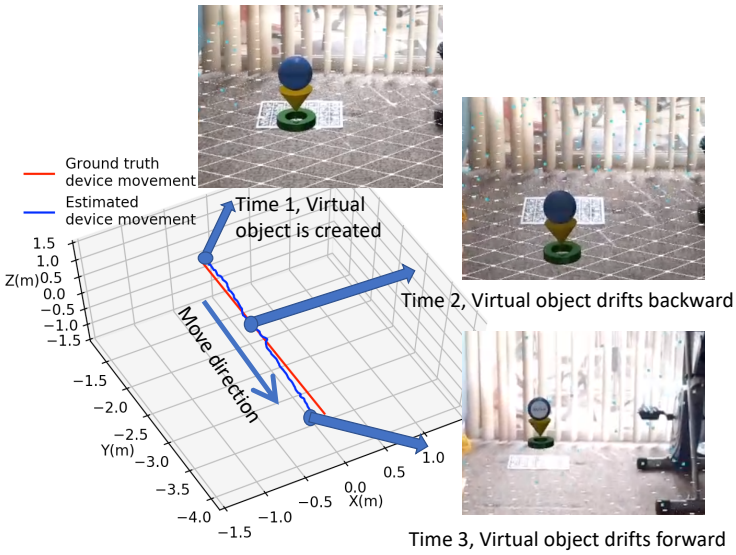
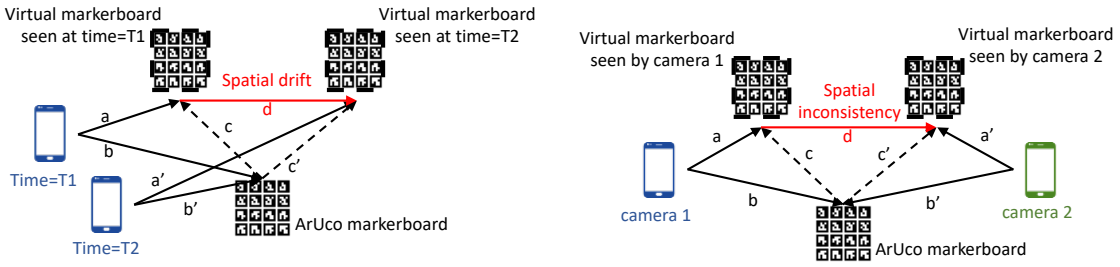


Figure 2.2: The virtual object drifts back and forth as the user moves. However, simply looking at the device trajectory/ATE alone gives little information about how the virtual object drifts.

the scene, these known objects allow measurement of the spatial consistency between the real world and the virtual objects rendered onto the screen. Fig. 2.3 illustrates the design of RealityCheck for the single- and multi-user scenarios. In both cases, we place a real marker board in the physical world and create a virtual marker board as the virtual object for rendering. We next describe this general idea in further detail for each of the scenarios.

Single user scenario:

The spatial drift in the single user scenario is determined as follows. As labeled on Fig. 2.3a, we define the vector from the device's physical position to the virtual marker board's designated physical position at time T1 as a, the vector from the device to the real marker board at time T1 as b, and the vector from the real marker board to the virtual marker board at time T1 as c. At time T2, due to registration errors in the AR



(a) Single user case: The user moved from time T1 to T2, and the virtual object also drifted.

(b) Multiple users case: Two users view a common virtual object, but see it at different positions with respect to the real world.

Figure 2.3: Design of RealityCheck.

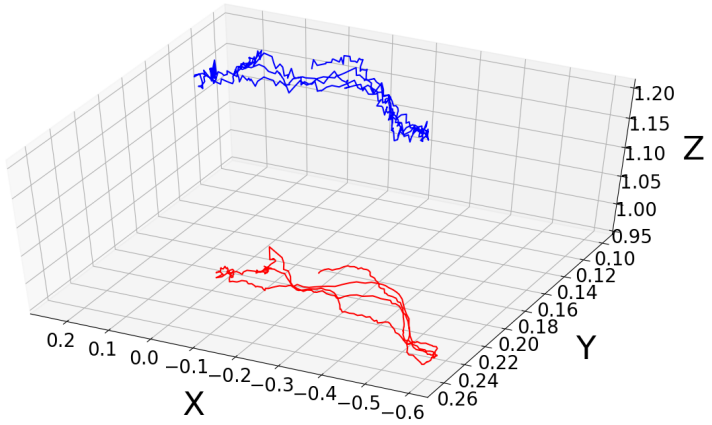


Figure 2.4: Example paths of the real/virtual marker boards over time (meters), relative to a moving camera. The red (blue) line represents the position of the real (virtual) marker board over time, vector b (a). Any change in their difference (c) is the spatial drift.

app, the virtual marker drifted to another physical location. At time T2, we label the corresponding vectors a', b', c', analogous to a, b, c. Then to compute the spatial drift (the vector d in Fig. 2.3a), we can see that $d = c' - c$. Since $c = a - b$ and $c' = a' - b'$, we have $d = a' - b' - a + b$. The vectors a, a', b, b' can be relatively easily computed using the perspective-n-point (PnP) method to determine the pose of the camera with respect to the virtual/real marker boards. This method obtains the detected corners of the marker boards in the current FoV, and along with knowledge of their size and shape and the camera calibration matrix, solves the PnP problem to obtain the desired vectors a and b for the respective marker boards. Finally, we compute $d = a' - b' - a + b$, which is the spatial drift. The complete algorithm is summarized in Alg. 1.

As an illustration of our approach, we plot the vectors a and b in Fig. 2.4 from a real trace captured by RealityCheck. The blue line represents the position of the virtual marker board (a) over time with respect to the camera. The red line represents the position of the real marker board (b) over time with respect to the camera. Their difference $c = b - a$ should ideally remain constant over time even as the camera moves, meaning that the real and virtual marker boards remain fixed with respect to each other, and there is no spatial drift. However, in practice there is spatial drift, which is reflected as changes in c. We see this in Fig. 2.4: while the general trajectory of the two lines are similar, they are not identical – their differences are the spatial drifts we are interested in.

Multi-user scenario: The multi-user scenario, shown in Fig. 2.3b, is analogous to the single user scenario, except that drift is measured across space, rather than across time. a is the defined as the vector from the phone 1's camera to the virtual object, and a' is defined as the vector from phone 2's camera to the same virtual object. b is defined as

the vector from phone 1's camera to the real marker board, and b' is defined analogously for phone 2. Then $c = a - b$ is the vector from the real marker board to the virtual marker board as seen by device 1, and $c' = a' - b'$ is the vector from the real marker board to the virtual marker board as seen by device 2. Their difference, $d = c' - c$, represents the spatial inconsistency, *i.e.*, the difference in the position of the virtual object as seen by the two devices. We follow the same computation method using PnP as in the single-user case. Thus, the same overall method is general enough to be used for the single and the multi-user scenarios.

Algorithm 1 Computation of spatial drift

Inputs: camera frame f_1 and f_2
Outputs: spatial drift d
Compute a and b in f_1 using PnP
$c \leftarrow a - b$
Compute a' and b' in f_2 using PnP
$c' \leftarrow a' - b'$
Return $d \leftarrow c' - c$

2.4 Experimental Evaluation of RealityCheck

2.4.1 Implementation and Test Methodology

Implementation

A picture of our implementation is shown in Fig. 2.5, with the major components described below.

Devices. The AR test application is built on the Android ARCore platform [37] and run on a Samsung Galaxy S20 mobile phone. RealityCheck itself runs on a separate laptop with

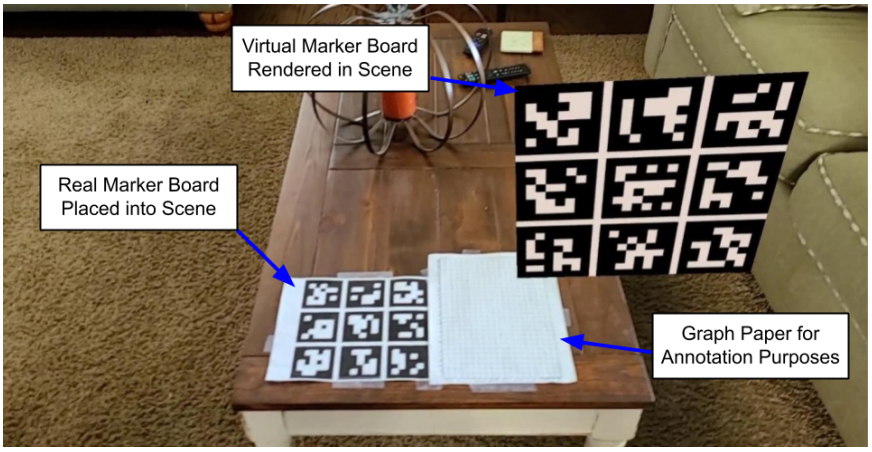


Figure 2.5: Evaluation setup. The position of the virtual object (the virtual marker board) is evaluated with respect to the real marker board on the table. The graph paper aids in hand-annotating the ground truth position of the virtual object.

an Intel Core i7 2.8Ghz CPU.

Real marker board. An ArUco marker board comprised of 9 unique markers arranged in a 3×3 grid is printed on standard printer paper. The real marker board is 0.2 m × 0.2 m in our experiments so it could fit on the paper, but in general, it may be of any size small enough to fit in the FoV of the device camera, and large enough for the marker patterns to be recognized.

Virtual object (marker board). We create our virtual marker board similarly to the real marker board. The virtual board is textured using 9 ArUco markers (different from the 9 in the real marker board) and arranged in a 3x3 grid. The virtual marker board is created in Blender [14] and imported into Android Studio. We set the size of virtual marker as 0.4 m × 0.4 m. It is important that the virtual marker board is not so large as to obstruct the real marker board, although this could be solved by simply recording the image both before and after the virtual marker board is drawn to the screen. Special attention is paid to make

sure no additional graphics effects are drawn, such as drop shadows or specular maps, as they may affect the visibility of the real or virtual markers.

Graph paper. To compare RealityCheck's measurements to the ground truth, a piece of paper of the same dimensions as the real marker board is placed adjacent to it in the scene. This paper has printed on it a grid of 1 cm $\times$ 1 cm squares for use in hand annotating the position of the virtual object offline (as described in the next subsection).

Test Methodology

AR app recording. The AR test application is run on the mobile device, and the user taps the screen to place the virtual marker board in the scene. The user then moves around the environment and captures what is shown on the display, either using screen capture software [63] or simply taking screenshots at regular intervals to obtain the data needed to run RealityCheck.

Mobility patterns. To examine the efficacy of RealityCheck with respect to user mobility, we evaluated three different walking patterns.

- *Side-to-side:* The user moved side to side a distance of approximately 1.5 meters without rotating, keeping the real marker board and the virtual object in the device's FoV.
- *Look-Away-and-Back:* The user started the trial with the real marker board and virtual object in the FoV, then walked away about 15 meters keeping them out of the FoV, and finishing the trial by returning to them.
- *Around-in-a-Circle:* The user walked a complete 1.15 meter radius circle around the real marker board and virtual object, keeping them in view.

We conducted 22 total trials. First, we conducted 5 indoor trials of each mobility pattern, for a total of 15 trials. Each trial lasts for 12-36 seconds. These indoor trials were performed without direct sunlight with the real marker board and graph paper placed on a short table. An additional 4 trials were performed in an outdoor environment in direct sunlight with one of each mobility pattern, plus an additional circular walk trial. Finally, 3 trials lasting 2.5 minutes each were performed in the same indoor environment with the 3 mobility patterns.

Running RealityCheck. To process the results, the recorded video is partitioned into its individual frames. RealityCheck performs the steps described in Section 2.3 for each frame or frame pair, and saves the resulting c vectors to a file for analysis.

Ground truth via manual labeling. Finally, in order to check the results of RealityCheck against the ground truth, it is necessary to hand annotate the position of the virtual object (specifically, the virtual marker board relative to the real marker board). The graph paper assists in this by allowing easy visual counting of the distance to the virtual marker, with a resolution of 0.5 cm. For every frame we wish to evaluate, we measure the Euclidean coordinates in terms of grid squares, then convert the grid squares to cm in order to obtain the ground truth vector between the virtual and real marker boards, which we call c_{true}. This is then compared against the estimated c vectors output by RealityCheck. We then repeat this measurement over multiple frames to compute spatial drift d_{true}. A similar approach is followed for the multi-user scenario. Note that this manual labeling is only performed by us to evaluate RealityCheck, and does not need to be performed by general users of the tool.

Using this method, we hand annotate every 30 frames ($\sim$1 s) across the 15 indoor trials, resulting in a total of 191 annotated frames, plus 77 annotated frames for the outdoor trials. Since the hand-annotated virtual object positions are in the real marker

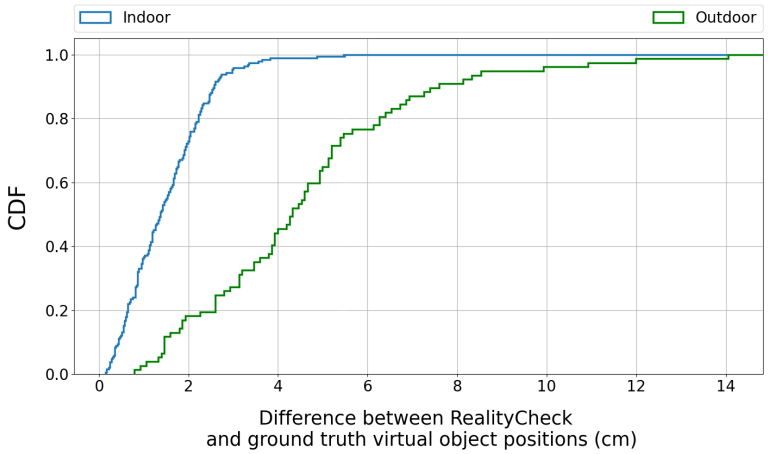


Figure 2.6: Cumulative Distribution Function of the difference between RealityCheck's estimated position of the virtual object and the ground truth ($||c - c_{\text{true}}||$) of a single user.

board coordinate system, but RealityCheck's output is in camera space, to compare their results, a transformation (obtained from the PnP method) is performed to convert RealityCheck's output from camera space to real marker board space, in order to make the vectors comparable.

2.4.2 Evaluation Results

In this section, we discuss RealityCheck's performance in terms of how accurately it reports the drift of a virtual object, compared to the ground truth.

Position of the virtual object. We first report the distance between where RealityCheck reports the virtual object is, versus the ground truth ($||c - c_{\text{true}}||$ in the notation of Section 2.3). A low distance indicates that RealityCheck matches more closely with the ground truth. The Cumulative Distribution Function (CDF) of the distances for the indoor and outdoor trials are shown in Fig. 2.6. The mean of the indoor errors is 1.5 cm,

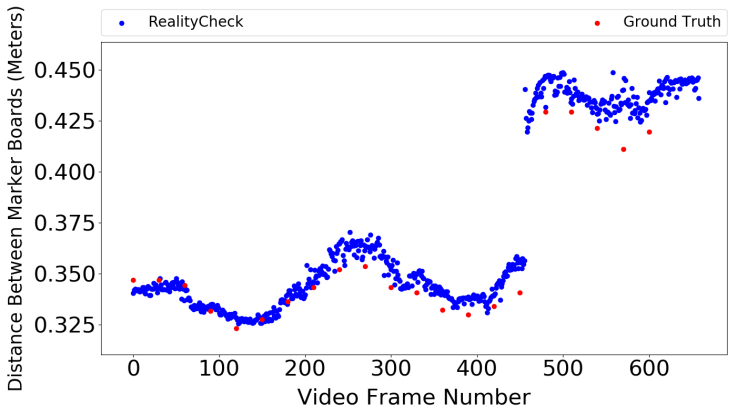


Figure 2.7: Example time series of the distance of the virtual object from the real marker board, as estimated by RealityCheck (blue, $||c||$) and the ground truth (red, $||c_{\text{true}}||$). RealityCheck is able to capture the large "jump" of the virtual object between frame 455 and 456.

whereas the outdoor errors average at 4.6 cm. The median distance in the indoor scenarios is 1.36 cm, and 90% of the samples fall within 2.5 cm. RealityCheck tends to perform slightly worse outdoors, possibly due to noise, and were biased by one particular trial that performed particularly badly. For example, outdoors, the lighting changes more frequently than indoors, or a strong wind shaking leaves on a tree can break the static environment assumption of SLAM-based AR, giving rise to larger virtual object drift and resulting in a slightly larger difference between RealityCheck's estimate and the ground truth. Overall, though, the distance between the ground truth estimates and RealityCheck's estimates are 2.4 cm on average across all indoor and outdoor scenarios, suggesting that RealityCheck is reporting accurate results.

RealityCheck can accurately report the position of a virtual object both when the virtual object drifts slightly, or when it jumps significantly. To illustrate this, in Fig. 2.7

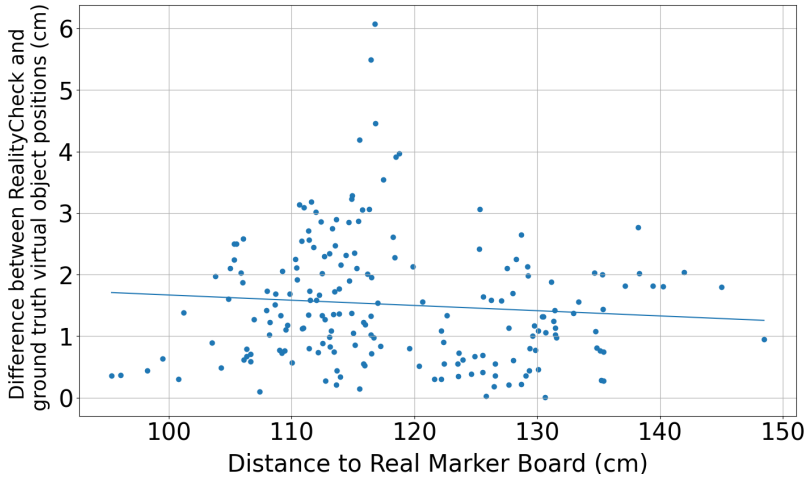


Figure 2.8: Distance error ($||c - c_{\text{true}}||$) as a function of a user/camera's distance to the real marker board. RealityCheck works even when the camera is further away.

we show an example time series of the distance between the virtual object and the real marker board, as output by RealityCheck (blue dots, $||c||$) and by the ground truth hand annotations (red dots, $||c_{\text{true}}||$). The point of interest is just past the 455th frame, where the virtual object "jumps" nearly 6 cm, as a more extreme example of spatial drift. RealityCheck is able to detect the virtual object's sudden change in position quite accurately (as the blue dots line up with the red dots).

Finally, we ask whether RealityCheck's output has low error even as the camera moves farther away from the virtual object and real marker board. In Fig. 2.8, we plot the difference between RealityCheck and the ground truth's estimate of the virtual object's position ($||c - c_{\text{true}}||$), as a function of the distance between the camera and the real marker board. The trend in Fig. 2.8 is essentially flat, indicating that RealityCheck is robust to distance from the virtual object within typical AR application ranges.

Position drift over time. We are not only interested in RealityCheck's ability

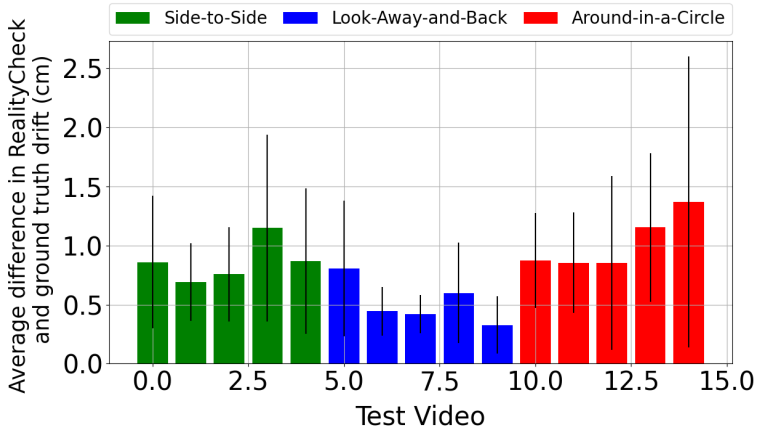


Figure 2.9: Average difference between RealityCheck and the ground truth, with standard deviation, when computing the per-second drift of a single user ($||d - d_{\text{true}}||$).

to estimate the location of a virtual object at a single point in time, but also in its ability to track the change in that position over time. To evaluate this, we define the drift as the distance that a virtual object moves during one second (d). We also compute the ground truth drift (d_{true}). Fig. 2.9 shows the difference in the drift measurement from RealityCheck and the ground truth ($||d - d_{\text{true}}||$), for each user mobility pattern. Overall, the average difference across all trials was 0.86 cm. The trials where the user moved and looked away from the virtual object and returned later (look-away-and-back) had the least drift difference (0.52 cm on average), because there was usually only one large drift as the user returned, but nearly no drift for the rest of the video, and hence little drift difference. On the other hand, in the trials where the user faced the virtual object from different angles (side-to-side), the average drift difference was higher (0.87 cm). The trials where the user moved in a complete circle around the markers (around-in-a-circle) experienced the most extreme angles and, correspondingly, the greatest drift error (1.02 cm on average).

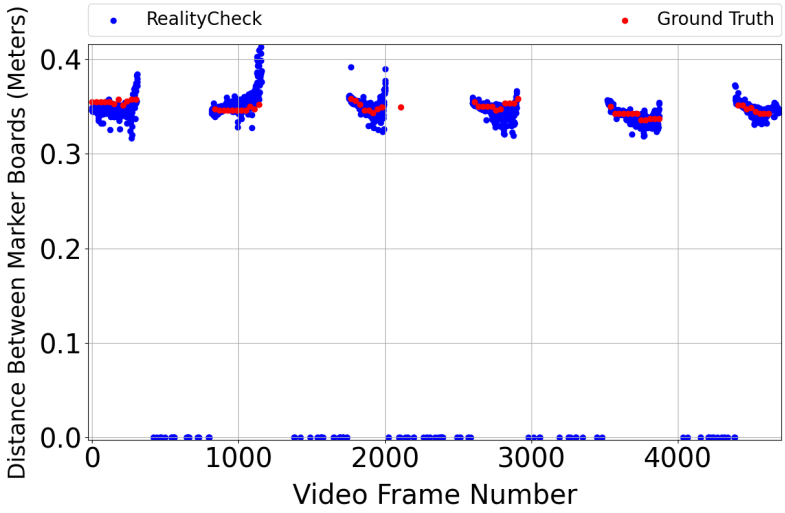


Figure 2.10: Example time series from a longer 2.5-minute trial of the distance of the virtual object from the real marker board, as estimated by RealityCheck (blue, $||c||$) and the ground truth (red, $||c_{\text{true}}||$) for a single user.

Longer length trials. To evaluate performance over longer runs of the AR app, to see if the tool performs correctly, 3 additional 2.5-minute trials were performed, using the same three movement strategies as in the shorter videos. Overall, the results were similar to those of the shorter-length trials discussed so far. Fig. 2.10 shows an example time series of one of the longer-length trials. The y-axis is the distance between the virtual object and the real maker board, as output by RealityCheck (blue dots, $||c||$) and the ground truth (red dots, $||c_{\text{true}}||$. Despite the virtual object coming in and out of view, RealityCheck is capable of maintaining accuracy even during longer AR sessions, as RealityCheck and the ground truth match up well. The average difference between the vectors received from RealityCheck and the ground truth was 1.6cm across all the longer-length trials.

Spatial inconsistency across multiple devices. RealityCheck is able to take in any recording of an AR app along with the board configurations and camera parameters

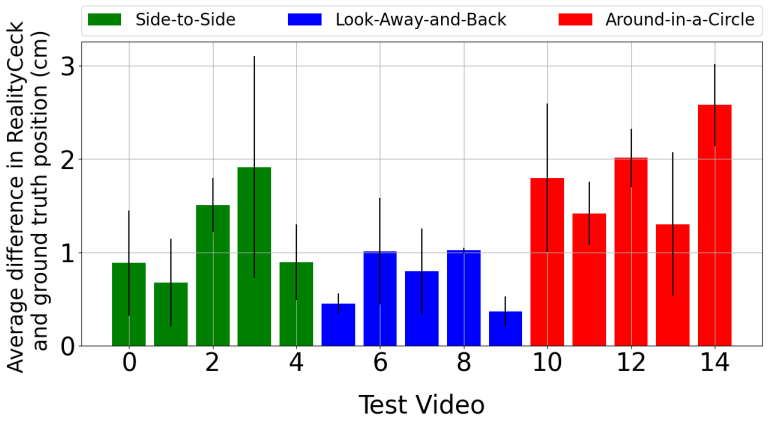


Figure 2.11: Average difference between RealityCheck and ground truth, with standard deviation, when measuring the position of a virtual object as viewed by two emulated users ($||d - d_{\text{true}}||$).

and perform the estimations, regardless of when or what device the images came from, as long as the image shows both marker boards and the camera calibrations are known. To emulate a multi-user scenario, we consider the first half of a video trace as originating from one device, and the second half as originating from a second device, then compare the drift across these two halves of the video. Fig. 2.11 shows the average difference in the position of the virtual object as measured by RealityCheck and the ground truth, as seen by each of our two emulated users. The difference is 1.3 cm on average. As the change in the virtual object's position is larger in this scenario than in the 1 second scenario previously considered (Fig. 2.9), we therefore see the average error per video increase accordingly, but stay within a reasonable 1-3 cm range.

2.5 RealityCheck Discussion

In this section, we briefly discuss several assumptions of RealityCheck. Firstly, RealityCheck relies on marker detection and pose estimation (via PnP), which have been shown to have

good accuracy [1]. We also use a marker board, which consists of 9 markers, to increase the detection accuracy [115]. RealityCheck can fail to output measurements if visibility is poor; for example due to insufficient lighting, motion blur, poor resolution, or extreme viewing angles of the virtual object (*e.g.*, 90° to the side, causing the virtual marker board to be too thin to be detected).

Secondly, the addition of a marker board to the scene, as required for RealityCheck to work, can add additional features to the scene for the AR system to use, thus impacting the spatial drift of a virtual object. However, since the amount of added features can be quite small compared to the natural visual features available elsewhere in the scene, we believe the effect on the AR application to be small, if any. Moreover, our experiments show that even with these few additional features, the AR app still experiences spatial drift (see Fig. 2.7).

Finally, while all tests were performed using SLAM-based AR on mobile devices, RealityCheck generalizes beyond SLAM-based AR, because it only needs the video of the AR app's operation, camera calibration parameters, and known markers as inputs. Recording screenshots of a non-SLAM AR framework (a machine learning-based framework, for example), will not prevent RealityCheck from measuring the spatial consistency of the virtual objects. RealityCheck is agnostic to the internals of the AR platform, and only needs the final rendered images to run successfully.

2.6 Related Work

Specialized hardware: Yagfarov *et al.* [177] evaluate SLAM performance against results from a laser tracker in a static indoor environment. However, such method requires extra

equipment(laser), and can't evaluate camera rotation, whereas our method can work in indoor, outdoor, or even dynamic environments provided that the ArUco marker in our system remain stationary. Other evaluation methods include constructing a 3D model of the real scene, requiring a 3D scanner [185].

Pixel differences: Several works [173, 184, 121] measure the pixel difference between where the virtual object is and where it should be on the device screen (*e.g.*, screen-space error). We note that RealityCheck also projects the virtual object to the device screen, and could calculate the drift in terms of pixels. However, pixel drifts can tell us how much drift the virtual objects have on the device screen, but they can have significantly different meanings in 3D space.

Statistical analysis: Faion *et al.* [31] compute camera translation and rotation multiple times using different set of markers, and then use the standard deviation of the results as the estimation reliability. However, such methods provide only camera pose but not the drift of virtual objects. MacIntyre *et al.* [89] propose a statistical method to estimate the error bounds of 3D points and then calculate the mean and covariance of the drift in the screen coordinates. However, this requires knowledge of the individual sensor errors, which can be difficult to obtain for heterogeneous AR hardware. Ran *et al.* [128] design a marker-based method using mobile devices, but require modification to the AR app, while our method requires only screenshots and camera parameters. Scargill *et al.* [141] develop an alternative methodology requiring more extensive user interaction.

User participation: Some AR evaluation methods involve user participation, rather than the objective feedback studied in this chapter. Peillard *et al.* [119] and Rosales *et al.* [134] measure the difference between the distance the user perceives from the device, and

the distance designed by the program. While RealityCheck is not evaluating the perceived position by users, the drift of virtual object we measure will affect the perception of the observer. Lehman *et al.* [80] and Bork *et al.* [16] evaluate AR performance by asking users to give feedback or fill out questionnaires. Such methods can provide direct feedback from users, which is complementary to our approach, but can be time-consuming.

2.7 Chapter Conclusion

Tools to measure spatial inconsistency and other metrics of interest are necessary for AR to improve. RealityCheck is an accurate, fast, and cheap way to check the spatial inconsistency of an AR system. Our evaluation of RealityCheck showed that it can measure the spatial drift of a virtual object with 1.5 cm error on average, compared to the ground truth. We release the code as open-source in the hope that it will be useful to other researchers and developers. In the future, RealityCheck could be extended into a suite of tools to measure additional AR-relevant metrics such as appropriate rendering of virtual objects in different real world lighting conditions, appropriate shadow placement, and proper occlusion of virtual objects.

Acknowledgements

The work in this chapter was published in the International Symposium of Multi-Media 2021 [151]. Special thanks to Xukan Ran PhD. for helping obtain the ATE results via improvised camera tripod dolly (mobile phone taped to toy scooter.) The work for this chapter was supported in part by NSF CAREER 1942700 and a U.S. Department of Education GAANN fellowship.

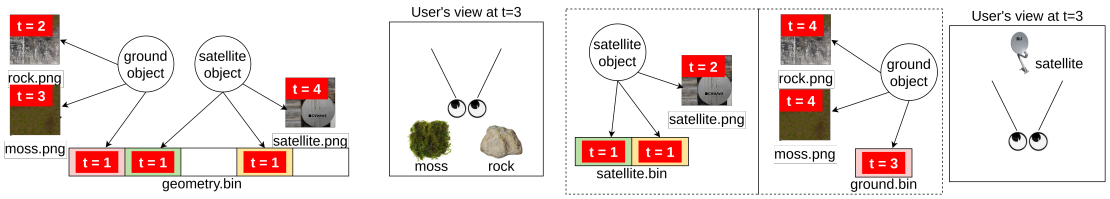
Chapter 3

VIA: Visibility-aware Web-based Virtual Reality

3.1 VIA introduction

New standards such as WebXR [168] enable cross-platform augmented and virtual reality (AR/VR) experiences by processing and displaying content through web browsers. This enables developers to write a single WebXR experience and have it work across multiple AR/VR devices, such as Oculus Quest and HTC Vive. Under the hood, WebXR works by calling WebGL Javascript libraries and retrieving the relevant assets from a remote web server. Once an object has all its dependent assets (geometry and texture data), the object is rendered on the VR display. WebXR is supported by most modern web browsers including Chrome and Edge and is standardized by the W3C.

Motivated by the current support and active development of WebXR, we experimented with various WebXR sample scenes, and observed significant performance issues in



(a) Default WebXR.

(b) Our approach, VIA.

Figure 3.1: By default, WebXR downloads the objects and their dependencies in arbitrary order (a), while VIA prioritizes downloading objects within the user's FoV so they can appear sooner (b).

terms of latency. Users loading a WebXR scene had to wait up to 9.86 seconds under a 69 Mbps connection until all objects were fully visible in the user's field-of-view (FoV), hurting user experience. The reason for these high latencies is that WebXR is agnostic to the user's current FoV, and retrieves dependencies in an arbitrary order determined simply by how they are listed in a metadata file. In the worst case, objects that are behind a user might be downloaded and rendered first, while objects that are directly in front of the user might be download and rendered last, leaving the user to view a blank screen in the meantime. Based on these observations, in this work we propose reducing the load times of WebXR scenes, by optimizing how objects are stored and retrieved from a server, as illustrated in Fig. 3.1.

However, this is not a straightforward problem to solve for the following reasons. Firstly, it is not clear in what order objects should be retrieved, as some objects are entirely contained within the FoV, while other objects may span the entire FoV, or even be out of the FoV entirely. Secondly, the geometry data associated with WebXR objects is stored in very coarse-grained format (one large binary file), preventing fine-grained resource requests. To address the first challenge, we propose a new scoring function that prioritizes which objects

to download, based on whether objects are potentially visible and their orientation from the center of the FoV. To address the second challenge, we propose grouping geometry data together into logically-meaningful chunks that minimize the average download time of any object in a scene. Implementing these techniques requires only changes to the server-side code, and works with un-modified clients and browsers.

This project is related to research in viewport optimization for web pages [19, 108] and 360° videos [26, 186, 125, 51] but differs in several major respects. Firstly, web page optimizations that prioritize resources "above the fold" rely on the DOM tree, which does not contain information about WebXR objects and their dependencies; furthermore, the implementation is very different as WebXR involves working with WebGL and Javascript, rather than mainly HTML/Javascript/CSS. Secondly, 360° video optimizations prioritize 2D tiles in the user's FoV from a single user location, whereas our solution prioritizes 3D objects and their dependencies, for any user location and orientation.

To the best of our knowledge, this is the first work to combine page load reduction time reduction techniques with VR applications, improving the user experience for browser-based VR. We call our system VIA, short for VIsibility-Aware Web-based VR. The contributions of this chapter can be summarized as:

- We showcase the latency issues of web-based VR by measuring the page load times of several WebXR sample scenes, and find that inefficient object download ordering is the root cause of the observed high latencies. For example, the time until all content is rendered in the user's FoV for a Shack scene (details in §3.7.1) is 41.26 seconds under a 10 Mbps connection. However, the objects in the FoV consumed only 64% of the total objects requested, indicating there are opportunities for significant savings.

- We propose a scoring method to determine which objects should receive priority downloads. The score is a combination of visibility and orientation from the center of the user's FoV. Given these scores, the WebGL metadata, geometry data, and and Javascript code are minimally modified to request the objects and their dependencies in the appropriate order. To enable fine-grained retrieval of object dependencies, we group relevant geometry data together on the server, enabling efficient download any combination of objects.
- To implement the above techniques, we develop a parser to determine WebXR dependencies. We experiment with various WebXR scenes under different network conditions. Our results show that our method reduces page load times by up to 50.3%, compared to the default WebXR implementation. Furthermore, our method works for different user FoVs and is robust to mis-estimation of the user FoV.

Next, we discuss related work (§3.2), a brief background on WebXR (§4.2.1), and the motivating measurements for our work (§4.2.3). Our problem setup and solutions are presented in §3.5, experimental results in §3.7, and conclusions in §3.8. The technical report and open-source code are provided on a website [149].

3.2 VIA Related Work

360° videos: Multiple papers study how to efficiently download pixels within the FoV for 360° videos (*e.g.*, [26, 186, 125, 51]). However, 360° videos are different from the true 3D scenes we consider in this work, as 360° videos only consist of 2D video data, and the decision are which tiles to request, unlike the objects, images, data buffers, and their dependencies that we have to consider in WebXR. Furthermore, typically 360° video scenes can only be viewed from a single position, whereas our method works for any initial viewing positions

and orientations.

Other VR FoV optimizations: FlashBack [15] pre-renders 3D scenes to reduce latency in a thin-client design, whereas this work follows the WebXR architecture where the client browser performs rendering. Vivo [53] optimizes fetching of point cloud data based on visibility, whereas WebXR 3D objects we consider in this work are typically stored as meshes plus textures and normal maps that cover the meshes. WebXR objects thus require different splitting techniques to enable visibility awareness. [59] has a similar approach of prioritising the download of parts of a scene, using a different scoring heuristic and a more coarse-grained object grouping. Our approach works with the recent WebXR standard, is tested on more than one indoor scene, and uses the standard HTTP client-server architecture rather than relying on P2P torrents. Other space partitioning structures such as k-d trees [8] could produce alternative object orderings than VIA's scoring method; however, we believe these gains would be incremental, as the majority of the latency savings come simply doing some form of intelligent ordering.

Page load time: Klotski [19] and Polaris [108], among others, perform dependency analysis for webpages in order re-order content delivery for faster rendering "above the fold", but cannot parse WebXR metadata for the unique dependency structure of 3D models. Their proxy-based implementation could be adapted for use in VIA, rather than the server-based mechanisms we propose. Tools such as Lighthouse [47] can record various metrics related to page load time, but cannot accurately capture when all 3D objects are visible in WebXR in our experience (discussed in Section 3.7.1), and do not provide WebXR-specific suggestions for page load time reduction.

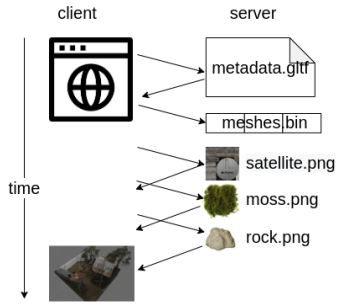
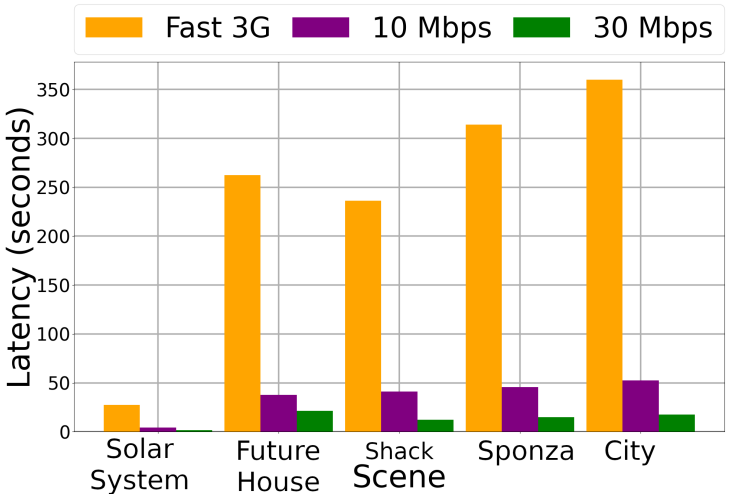


Figure 3.2: WebXR retrieves metadata, objects, and their data buffer and image dependencies from a remote server in order to render a 3D scene.

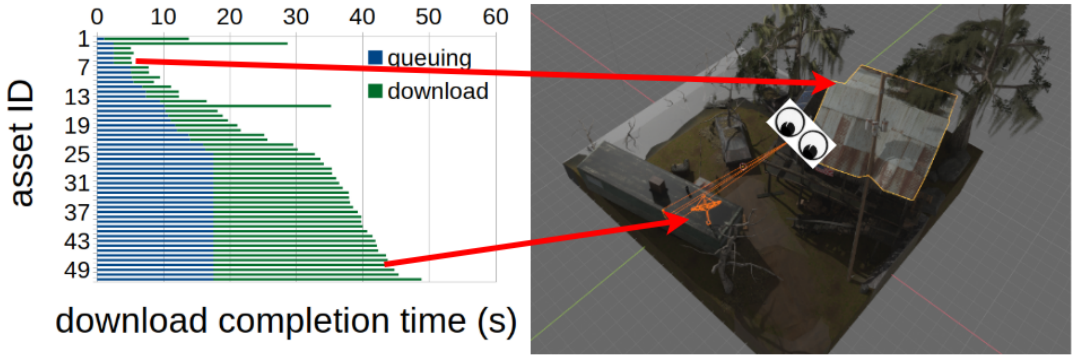
3D scenes and models: Previous work has been done to optimise page load times for streaming specific types of 3D models using the glTF [142]; however, our solution is more general as it applies to arbitrary scenes and not just cities. 3D Tiles [21] has similar goals and methods as this project, but requires converting 3D data to the 3D Tiles file format and using a custom Cesium viewer. VIA can be considered a more "lightweight" version of 3D Tiles that works directly with WebXR. Multiple authors [82, 77] suggest 3d graphic data compression and streaming standards, reducing the amount of bytes sent over the wire while requiring the browser to decompress the data before rendering, which may be complementary in addition to VIA. Several works consider 3D object streaming by modifying the underlying content, such as using "geometry images" [57] or texture/mesh compression [33, 56], which are complementary to this work, which focuses on the order in which to fetch those objects.

3.3 VIA background

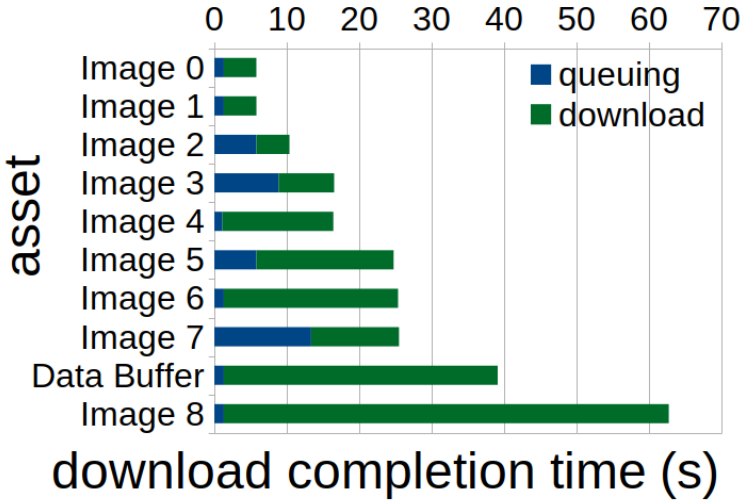
We first provide a brief background on WebXR. WebXR is an API for web-based AR/VR that enables cross-platform support for different hardware. It is the successor to WebVR,



(a) Default WebXR load latency of 38-51 seconds under 10Mbps.



(b) User-agnostic dependency fetching order. The roof tiles are fetched earlier despite being behind the user, and satellite dish is fetched later despite being within the FoV.



(c) The data buffer takes 39 s to load, blocking all objects from rendering.

Figure 3.3: High latency in default WebXR (a) is due to user-agnostic dependency fetching order (b) and coarse-grained data buffer storage (c).

with contributors including Google and Mozilla, and the latest W3C working draft was published in July 2020 [168]. WebXR operates as shown in Fig. 3.2. On the client, a WebXR scene is loaded like a regular webpage, and includes Javascript control code that handles frame updates, rendering, input devices, etc. with the help with WebGL libraries. The Javascript code first loads a metadata file (.glTF) that provides information on what objects are present in the scene, their locations and orientations, and their dependencies. These dependencies include the data buffers and image URIs to be rendered, which are stored on a remote server. In this chapter, we call the geometry data and its associated information (*e.g.*, texture mappings, vertex positions, etc.) as a "data buffer" (corresponding to a bufferView and its corresponding accessor in the glTF standard). Once the Javascript has retrieved the objects and their dependencies, they can be rendered on the display. On the server, the data buffers for multiple objects are stored by default in one large (.bin) binary file, and the texture data are stored as individual image files (typically .png or .jpg).

3.4 Motivation: High Latency in Default WebXR

In this section, we highlight the problems with the default loading process of WebXR scenes and its root causes, based on our measurements and analysis. We conducted experiments with the default WebXR implementation in Google Chrome, viewing several sample scenes and recording the latency until all the objects within the FoV appeared on the screen (further details in Section 3.7). Our main observations are that startup latency is 44.26 seconds on average on a 10 Mbps connection, and the root causes of this latency are (a) the initial view-agnostic object fetching order and (b) the coarse-grained storage of data buffers on the server, as further described below.

High latency: We first show that the time to first correct frame (*i.e.*, when all the objects appear in the FoV) which we hereafter refer to as the latency, is high. Fig. 3.3a shows the average latency for each test case, under a 10 Mbps or 3G connection. The latency is up to 51 seconds for 10 Mbps and 350 seconds for 3G. Similar multi-second load times have also been observed in 4G and 5G networks [106]. Next, we unpack the root causes of these high latencies.

User-agnostic object fetching order: We observed that when a WebXR scene is loaded, all the object dependencies are downloaded according to an arbitrary fixed order (based on the order they are listed in the .glTF metadata). This causes problems because the download order is agnostic to the user's FoV. For example, as shown in the network request trace in Fig. 3.3b, the texture files belonging to objects behind the user (*e.g.*, roof tiles) are fetched earlier, while the texture files belonging to objects in front of the user (*e.g.*, satellite dish) are fetched later. This delays the rendering of objects in front of the user. Such observations motivated our object scoring strategy, which determines what objects to prioritize in the user's FoV and fetches their dependencies first, decreasing the latency (see Section 3.5.2).

Coarse-grained data buffer storage: Besides image (texture and normal) data, objects also require data buffers containing geometry meshes and other related information in order to render the object correctly. However, we observed that the data buffers are typically stored in one large .bin file, causing issues, because a single object cannot be rendered until the entire file has been downloaded. For example, in Fig. 3.6c, the data buffer asset is the second-slowest file to download under a 10 Mbps connection due to its size; no objects can render before the data buffer finishes downloading at 39 seconds. These observations

motivated us to consider splitting the binary appropriately; however, the challenge is to determine the split granularity – a fine granularity allows object fetching flexibility but incurs an extra RTT for each request (see Section 3.5.3).

3.5 VIA Problem and Solutions

3.5.1 Overview

To quickly download and render the objects within the user's FoV, we have to solve the two aforementioned problems of object dependency fetching order, and coarse-grained asset storage. Our system system, has two modules:

- **Object scoring (Section 3.5.2):** The Object Scoring module's task is to determine which objects are within the user's FoV, and assign scores to the objects and their dependencies for later request order optimization. Here, the problem is to determine in what order to request the objects, based on where they are in the scene. The main idea is that objects within the user's FoV and directly centered in front should have higher priority. This information needed to compute this can be obtained from the glTF metadata for the scene.
- **Data buffer grouping (Section 3.5.3):** The Data Buffer Grouping module stores the data buffer dependencies in a finer-grained fashion so that individual dependencies can be fetched in the order prescribed by the Object Scoring module. The problem here is to determine which data buffers to group together, to enable fine-grained object requests, while preserving sufficient aggregation for efficient downloads. The main idea is to define an optimization problem to group data buffers in order to minimize the average download

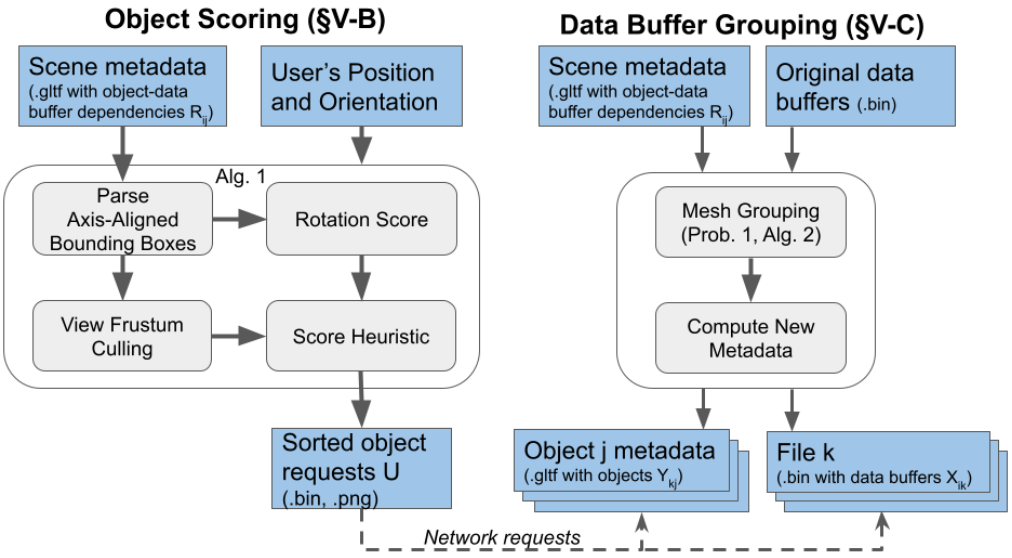


Figure 3.4: Overview of VIA.

time per object, and show that our proposed solution is optimal.

The output of the Object Scoring module are the order in which to request objects; these requests are sent to the server to retrieve the relevant objects and their dependencies, as stored by the data buffer Grouping module. A summary of the inputs, outputs, and interactions between the modules are shown in Fig. 3.4, and their details are provided next.

3.5.2 Object Scoring

The main intuitions behind our object scoring method are to de-prioritize: (a) objects that have no vertices within the FoV and (b) objects that are far from the center of the FoV in terms of angle. This requires computing two weights in our method in Alg. 2: a visibility weight, and an angle weight. Note that sometimes these two objectives may be in conflict with each other. For instance, there may be an object that is centered behind the user, but

some parts of the object are visible within the FoV (*e.g.*, a ground object); in such cases, our method returns a moderately high score due to its visibility and because the bulk of the object is behind the user.

Visibility Check. The visibility check returns whether (any part of) an object j is within the FoV, and penalizes any object that is guaranteed not to be within the camera's initial FoV by adding π to the object's score. Specifically, the visibility check relies on computing the viewing frustum (line 5), which is the area in the 3D scene that will be projected onto the user's 2D screen. Then, view frustum culling [8] (line 7) is performed to see if an object is within the viewing frustum, by computing the six planes of the viewing frustum from the initial camera view, and checking whether the axis-aligned bounding boxes (A_j intersect with the area in between the planes. The bounding box centers can be computed from the minimum and maximum vertex position co-ordinate elements given by the glTF metadata.

Angle Check. The angle weight determines how far off an object is from the center of the user's FoV. This is done by calculating the angle between the camera's default forward vector and the "look at" vector (the vector from the camera's position to the center of an object). This is performed by the `LookAtAngle` function in line 13. The weights from the visiblity check and the angle check are then added together to compute the total score for each object. Finally, the objects are sorted by their scores in ascending order (line 16). The overall algorithm has complexity $O(J \log(J))$ due to the sorting step.

Algorithm 2 Object Scoring

Inputs: Set of objects in the scene $\mathcal{U}$, axis aligned bounding box A_j for each object $j \in \mathcal{U}$, camera position C_P, orientation C_O, and FoV parameters C_F.
Variables: score $score_j$ of object j, view frustum F
Outputs: Sorted list of objects U
$\mathcal{O} \leftarrow \emptyset$
$F \leftarrow \texttt{ViewFrustum}(C_P, C_O, C_F)$
for all j in $\mathcal{U}$ **do**
    if A_j intersects F **then** ▷ visibility check
        $score_j \leftarrow 0$
        $\mathcal{O} \leftarrow \mathcal{O} \cup j$
    else
        $score_j \leftarrow \pi$
    end if
    $\theta \leftarrow \texttt{LookAtAngle}(C_P, C_O, A_j.center)$ ▷ angle check
    $score_j \leftarrow score_j + \theta$
end for
$U \leftarrow$ sorted list of $\{score_j\}$

Table 3.1: Table of notation.

Symbol	Definition
A_j	axis aligned bounding box of object j
B	Bandwidth
C_P, C_O, C_F	camera position, orientation, and FoV parameters
$\mathcal{O} \subseteq \mathcal{U}$	subset of objects in the user's current FoV
R_{ij}	Whether asset i is needed by object j
$\bar{s}_i$	size of asset i
s_k	size of chunk k
T	RTT/2
$\mathcal{U}$	complete set of objects in the scene
U	sorted list of objects in the scene
X_{ik}	Whether asset i is in chunk k
Y_{ij}	Whether object j requests chunk k

3.5.3 Data Buffer Grouping

Setup. The Data Buffer Grouping module breaks down the single data buffer file from a WebXR scene into its constituent data buffers, so that the Object Scoring module can request object dependencies at a finer granularity. However, deciding which data buffers to group together is non-trivial because there are trade-offs between RTT and propagation time. For example, grouping all data buffers into a single file would cost only one RTT to retrieve from the server; however, this would result in a longer propagation time (due to constrained bandwidth) before rendering of any object could start (as in the default WebXR). On the other hand, creating J files from J data buffers would require a fresh RTT to retrieve each data buffer, but each file would have a short propagation latency, ensuring that objects could be independently downloaded and rendered.

Problem Formulation. We formalize this problem as follows. There are N data buffers and J objects in the scene. We seek to group data buffers into files. The scene construction from the WebXR metadata tells us $R_{ij} \in \{0, 1\}$, whether data buffer i is needed for object j. The problem is to determine the integer variables X_{ik} (whether data buffer i should be included in file k) and Y_{kj} (whether object j requires file k). The objective is to minimize the download time of the set of objects $\mathcal{O}$ visible within the FoV:

$$\sum_{k} \max_{j \in \mathcal{O}} Y_{kj} \left(\frac{s_k}{B} + T \right) \tag{3.1}$$

where s_k is the size of file k, B is the current bandwidth, and T is the current RTT/2. The second term $\left(\frac{s_k}{B} + T\right)$ is the download time of file k, so $Y_{kj} \left(\frac{s_k}{B} + T\right)$ is non-zero only

if object j is dependent on file k. The $\max_{j \in \mathcal{O}}$ term accounts for browser caching within the same session; *i.e.*, we only need to count the latency of one download of file k, if file k needed by more than one object j.

If we knew $\mathcal{O}$ in advance, then we could optimize the data buffer grouping for those objects within the FoV. However, in reality, $\mathcal{O}$ could be anything, since the user's initial position and orientation could be set arbitrarily. It wouldn't be scalable to create and store files for every possible user position/orientation. Instead, we can try to optimize for a typical case, by minimizing the average time of all files, so that no matter the user's position/orientation, the relevant files can be retrieved quickly. Mathematically, we write:

$$\sum_{k} \max_{j \in \mathcal{O}} Y_{kj} (\frac{s_k}{B} + T) \leq \sum_{k} \sum_{j \in \mathcal{U}} Y_{kj} \left(\frac{s_k}{B} + T \right) \tag{3.2}$$

where the LHS is (3.1) and the RHS is an upper bound where the maximum is replaced with a summation based on the intuition above. Using the RHS as our objective, the optimization problem is as follows.

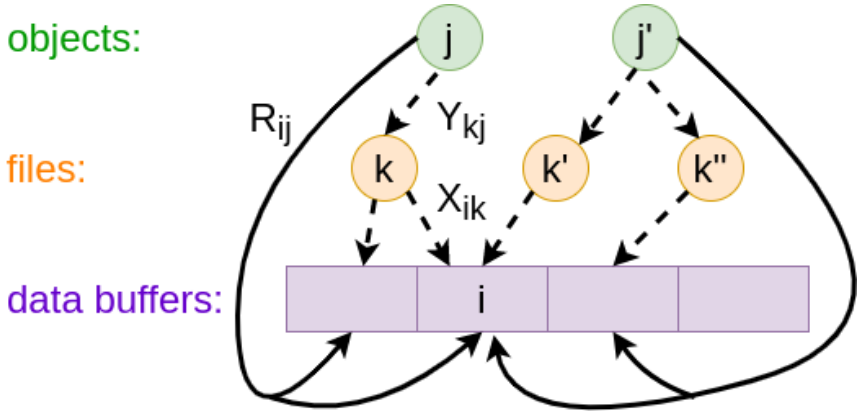


Figure 3.5: Problem 1 can be viewed as a series of set cover (Y_{kj}, mapping objects to files) and set membership (X_{ik}, mapping files to data buffers) problems.

theoremata buffer grouping

$$\underset{X_{ik},Y_{kj}}{\text{minimize}} \quad \sum_{j \in \mathcal{U}} t_j \tag{3.3}$$

$$\text{subject to} \quad t_j = \sum_k Y_{kj} \left(\frac{s_k}{B} + T \right) \quad \forall j \tag{3.4}$$

$$s_k = \sum_i \bar{s}_i X_{ik} \quad \forall k \tag{3.5}$$

$$\sum_k X_{ik} Y_{kj} \geq R_{ij} \quad \forall i, j \tag{3.6}$$

$$X_{ik}, Y_{kj} \in \{0, 1\} \quad \forall i, j, k \tag{3.7}$$

The objective (3.3) is equivalent to the RHS of (3.2), and minimizes the average time of retrieving all objects. Constraint (3.4) defines an object's download time as the sum of the transmission delay and the RTT. Constraint (3.5) defines the file size as the sum of its constituent data buffers. Constraint (3.6) states that every object must receive all its

required data buffers. X_{ik} and Y_{kj} are the integer decision variables.

Solution. This is an integer linear program, which is generally NP-hard. The problem can also be thought of as a variant of set cover, where both the subset membership and multiple set covers have to be determined. Namely, we have to determine the subset memberships (X_{ik} (which data buffers i should be included in subset k), and then solve J set cover problems, one for each object j (Y_{kj} (which subsets object j needs to cover all the data buffers in its own universe, $\{R_{ij}\}_i$). This is illustrated in Fig. 3.5.

However, it turns out that $X = R, Y = I$ is an optimal solution to the problem (where X, Y, R are the matrix versions of X_{ik}, Y_{kj}, R_{ij}, and I is the identity matrix). This solution corresponds to an easy-to-implement grouping of placing all data buffers of an object into a single file. Intuitively, it is possible to find a solution because the X_{ik} variable gives the ability to determine subset membership, actually making the set cover problem easier. The main idea behind the proof is that by requesting only one copy of each data buffer, and incurring as few RTTs as possible (one request per object), then the average latency per object is minimized. The proof is provided in the technical report [149].

$X = R, Y = I$ is an optimal solution to Problem 1.

The algorithm is shown in Alg. 3, and runs in $O(JN)$.

Image re-ordering only. We also developed a simplified version of VIA based on observations from certain test scenes where the total data buffer size was much less than the total size of the textures and images in the scene. In such cases, the data buffers did not impact the latency much, since the images consumed most of the network time. Therefore, we introduce a simplified version of VIA where only the images are re-ordered according to

Algorithm 3 Data Buffer Grouping

Inputs: Whether object i needs data buffer j R_{ij}
Outputs: Whether file k includes data buffer i X_{ik}, whether object j requests file k Y_{kj}
for all $j < J$ **do**
    for all $i < N$ **do**
        if $R_{ij} == 1$ **then** ▷ object j needs data buffer i
            $X_{ij} \leftarrow 1$ ▷ store data buffer i in file j
            $Y_{jj} \leftarrow 1$ ▷ object j requests file j
        end if
    end for
end for

the object scores, but not the data buffers (essentially, running Alg. 2 but not Alg. 3). We call this method VIA-Image.

3.6 VIA's Implementation

VIA is implemented in approximately 700 lines of Python3 and runs once per scene, when it is first placed on the server. It can be run multiple times for different initial FoV's expect from client devices. The output of the script is the new data buffer files, new metadata files. The WebGL library and a customized Javascript file (containing the new request order) are also stored on the server. A user wishing to reduce WebXR latency simply requests the new Javascript file with an unmodified web browser, without needing to make any other changes on the client side. Below, we briefly overview the key implementation steps.

Parsing object dependencies. We developed a custom parser for glTFs, which is a JSON-like object, and recorded the objects and their dependent textures, normal maps, and data buffers into a dictionary data structure. Parsing the axis-aligned bounding-boxes for Alg. 2 required finding all bufferViews associated with an object, and finding the minimum and maximum values of the vertex co-ordinates along each axis. If any transformations are

performed on the geometry data in the glTF, this also needs to be taken into account before computing the axis-aligned bounding boxes. All glTF parsing and alteration was performed based on the glTF 2.0 standard [158].

Creating new (.glTF) metadata. Alg. 2 is run to score the 3D objects, and using the scores, the images are re-ordered within the dictionary. For VIA-Image, this new dictionary is immediately written to file as a new .gltf. For the full VIA, Alg. 3 is also run and the original dictionary is split into multiple dictionaries, one for each object. Then new .gltf metadata files are written, one for each object. The glTF files are given a suffix in the file name to denote what order they should be requested in. The buffer attribute in each new glTF points to the URI of the relevant data buffer file (.bin).

We also experimented with creating one combined metadata file for all the re-ordered objects (instead of one metadata file for each object), but this resulted in all the binaries being requested first followed by all the images, due to the default behavior of the glTF loader, thus destroying our object ordering. Therefore we settled one glTF per object; however, a disadvantage was this resulted in a "flattening" of the object hierarchy stored in the original metadata file, which we plan to address in future work.

Creating new data buffer (.bin) files. The byte ranges for the data buffer associated with each object are parsed from the original glTF, and then copied over and combined into the new .bin files in order of their parent object score. Special care must be taken to maintain byte alignment (4, 8, or 16 byte-aligned) in order to allow for efficient processing of the contained data.

WebXR scene alterations: For VIA-Image, no additional changes to the Javascript code are needed. For the full VIA, a short loop in the Javascript code must be added to request each glTF in the score order. This is currently done manually in 4 lines of code, and is potentially automatable in the future.

Priority hints. An initial problem existed even with the re-ordered image requests, by default, the Chrome browser automatically tagged images with "low" priority and the .bin files containing the data buffers as "high" priority [46], thereby over-writing our careful ordering. To overcome this, we had to explicitly tag all the glTFs, data buffer binaries, and images with the same "low" priority by altering 2 lines of Javascript in the glTF loader. Note that setting these priorities alone would not enable implementation of our scheme, as there are only two priority levels, dis-allowing fine-grained re-ordering.

3.7 Experiments

We performed experiments to show the performance of VIA. The latency improvement depends on the user's viewpoint, network conditions, and characteristics of the scene itself. We also show that our methods are robust to small changes in initial orientation without needing to re-compute the object request order.

3.7.1 Setup

Experiments were performed on Google Chrome, version 91.0.4472.124, for different algorithms, test scenes, and network conditions. The three algorithms were tested were:

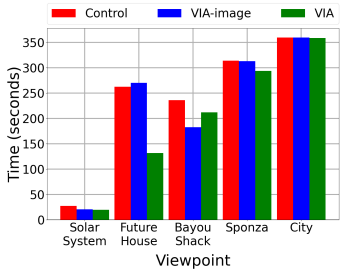
- **Control**: The default operation of WebXR.
- **VIA-Image**: VIA with Alg. 2 only, so only image requests were re-ordered, not data buffers.

- **VIA**: VIA with Algs. 2 and 3, where the scene objects and their image and data buffer dependencies were scored, sorted and requested in order.

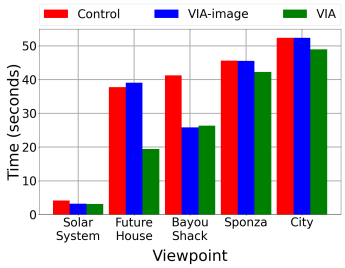
Our initial experiments were performed on four complex scenes with varying sizes, number of objects, and sizes of binaries and image files. We created three of the scenes based on publicly available 3D models, and the fourth scene is a WebXR test case.

- **Solar System** [153]: A simple, open space with an extremely high image to .bin size (15.5:1 ratio). The scene contains 29 objects with a total size of 6.72MB.
- **Future House** [35]: An enclosed indoor area with a moderate image to .bin size (2.9:1 ratio). The scene contains 27 objects with a total size of 55MB.
- **Bayou Shack** [32]: An outdoor scene with high image to .bin ratio (6:1). The scene contains 425 objects with a total size of 38.8MB.
- **Sponza** [154]: A walled-in area with a medium image to .bin ratio (4.39:1). This is a sample WebXR scene. The scene contains 103 objects with a total size of 50.2MB.
- **City** [23]: An outdoor cityscape without any images, only binaries. The scene contains 615 objects with total size 56MB.

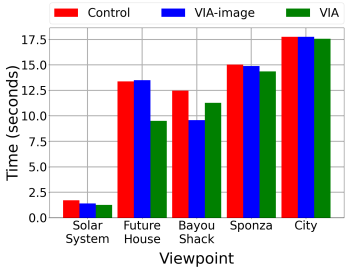
We tested on three simulated network conditions indicative of mobile networks: 30 Mbps with a 30ms RTT, 10 Mbps with a 60 ms RTT, and the "Fast3G" preset in Google Chrome DevTools, which roughly corresponds to 1.44 Mbps and 12 ms RTT based on an Internet speed test. All results were averaged across 3 trials for each Network-Scene-Algorithm combination, for a total of 135 experiments for the main results. The standard deviation in terms of latency across trials was less than 1%, and so are not shown.



(a) Time to first correct frame under Fast 3G network conditions.



(b) Time to first correct frame under 10 Mbps and 60 ms RTT.



(c) Time to first correct frame under 30 Mbps and 30 ms RTT network conditions.

Figure 3.6: Time to first correct frame for different scenes under different network conditions. VIA improves load time by up to 50%, with greater improvement in scenes where there are fewer objects in the FoV.

Measuring latency (time to first correct frame): The main evaluation metric was the latency from the page reload time to when all objects in FoV were loaded. Caching was disabled across trials. To measure this, we used the load time of the last object to show up in the FoV. By experimentally comparing with screen recordings we captured, we verified that the last object's load time corresponds very closely to its actual display time, with the rendering latency being negligible on the order of a few ms. We developed this methodology because existing page load time tools such as Lighthouse [47] do not capture the desired latency. For example, for the Sponza scene, Lighthouse reported a Largest Contentful Paint of 0.7 s, which only corresponded a system menu appearing, while in reality the entire scene was not view-able for 9.86 s.

3.7.2 Overall performance

Performance of VIA Figure 3.6b shows the latency of each scene with a 10 Mbps connection. While VIA had strictly lower latency than the control in all scenarios, the

largest gains occurred in scene with the fewest bytes, with Solar System, Future House and Shack saving 26.08%, 48.52% and 36.17% of time compared to Control, respectively. Sponza and City saw the least improvement due to a large number of visible objects in the FoV, as well as the floor beneath the camera having a moderate object score and thus loading near the end of the trace, delaying the time to correct frame. In fact, Sponza, due to its scene structure, required 88% of all bytes to be loaded before a correct frame could be rendered, and thus is a challenging test case.

Performance of VIA-Image. VIA-Image showed some gains as well, significantly beating the control (going from ˜40 to ˜25 seconds) for Bayou Shack , the scene with the second highest image size to binary size ratio. This is because fetching Bayou Shack's large images in the correct order saved significant amounts of time. However, for all scenes besides Bayou Shack, the last resource to finish downloading was typically the large binary file (for Control and VIA-Image), meaning that all the objects had to wait for the binary to finish downloading before they were rendered all at once. In other words, the binary was the bottleneck, and hence the full VIA with data buffer grouping was needed. The City scene was particularly challenging, because the majority of the objects were within view (only 50 objects were culled by the visibility check).

Example screenshots. Figure 3.7 shows screenshots of the Sponza scene loading for the 10Mbps results discussed above. VIA is able to render in the first 3D objects at the five second mark, while the Control and VIA-Image are unable to render a single 3D object until the 45 second mark. VIA was able to render a partial scene after 20 s. This is due to the binary file splitting allowing rendering of individual objects earlier in VIA, rather than

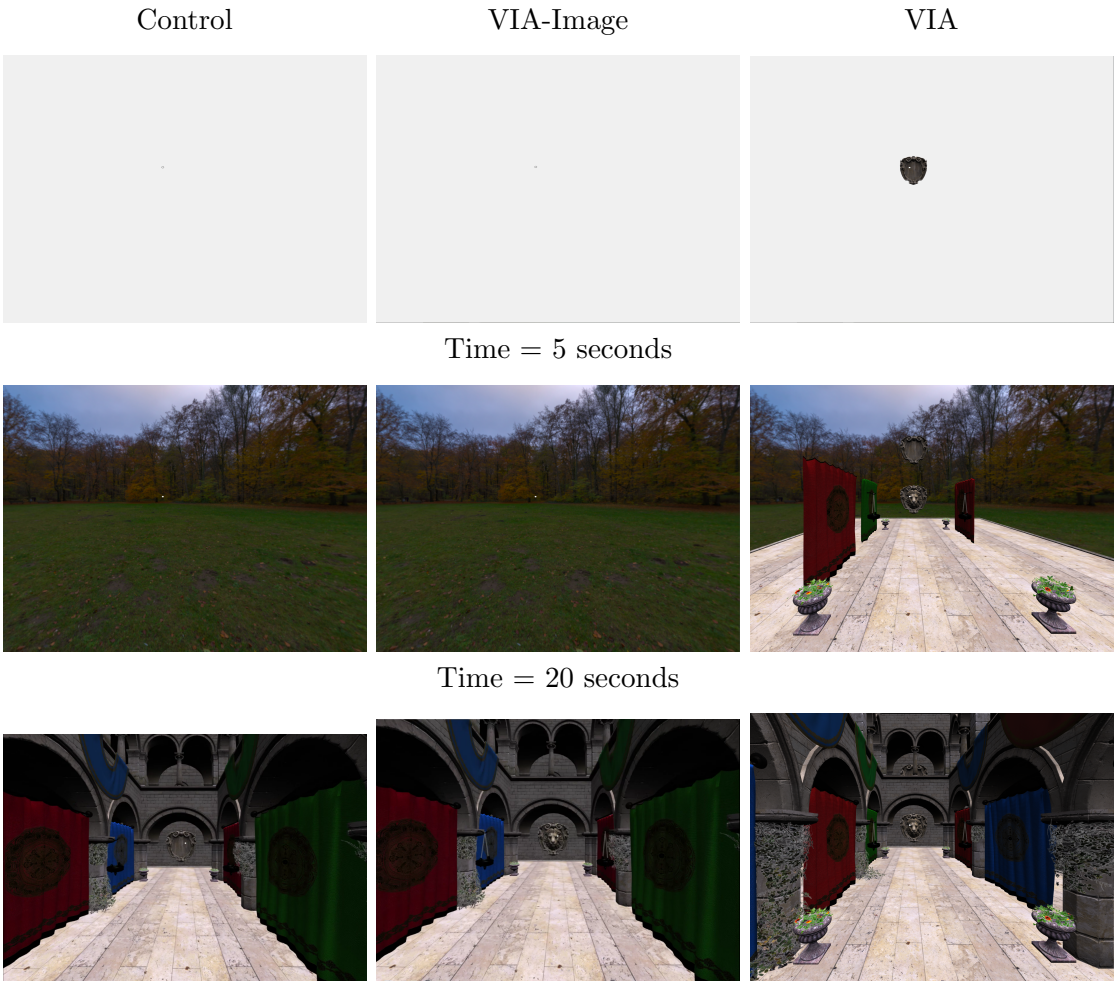


Figure 3.7: Screenshots of the Sponza WebXR scene at three different time instances, for the Control, VIA-Image, and VIA methods. VIA loads objects into the user FoV the fastest, while VIA-Image provides some benefits over Control.

waiting the original single, large binary file. Control finishes a fraction of a second later than VIA-Image (bottom row), due to it downloading the lion's head textures (at the end of the corridor) last, whereas VIA-Image downloads it early on and thus is able to render a complete frame as soon as the binary finishes downloading.

Large objects. One particularly challenging type of scene is those with large objects whose bounding boxes cover nearly the entirety of the scene. In such cases, poor object scoring may cause bottlenecks in the latency for VIA, such as in Sponza or City. For example, objects may pass the view frustum check yet the center of their bounding boxes are located behind the user (*e.g.*, floors or building walls), resulting in a low to medium request priority from the Object Scoring module, despite them being visible in the FoV. On the other hand, other objects may pass the view frustum check with their centers directly in front of the user, thus receiving a higher priority from the Object Scoring module, despite them not being visible in the FoV because they are discontinuous (*e.g.*, two windows on both sides of a hallway) or non-convex more generally. Scenes without large overlapping bounding boxes, such as Solar System, do not have this issue and avoid these potential bottlenecks.

System overheads. Here we briefly discuss the system overhead of running VIA. One overhead is in terms of storage – a side effect of splitting the large .bin of the original scene into many smaller .bins results in a small storage overhead. For the four test scenes, the average space overhead was an additional 3.3%, which is minimal. In terms of runtime, the main Python script executed in at most 16 seconds on an Intel i7-10700K CPU @ 3.80 Ghz, with over 15 seconds of that consumed by file I/O (*e.g.*, reading and writing the new data buffer files). For extremely large scenes in the future, view frustum culling may be

accelerated with common space partitioning structures like octrees [174] or K-d trees. Since the script only has to run once per WebXR scene, when it is first saved to the server, we consider this runtime acceptable and did not implement acceleration structures.

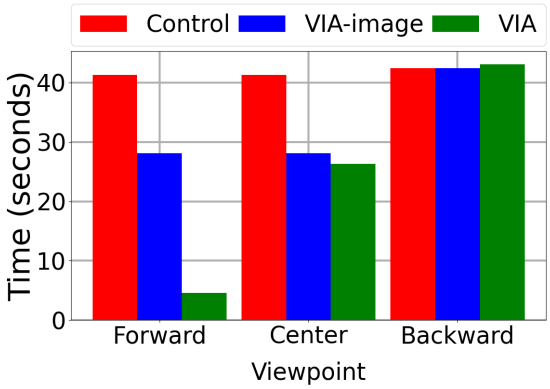


Figure 3.8: Time to first correct frame from different viewpoints, for Bayou Shack. The improvement of VIA over Control depends on the viewpoint.

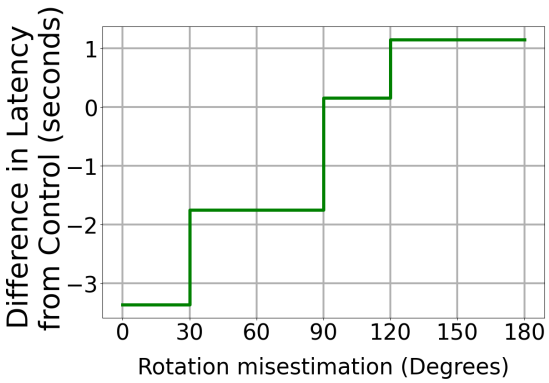


Figure 3.9: VIA Improvement of VIA over Control for the Sponza scene, if the system mis-estimates the user's head orientation.

3.7.3 Impact of different viewpoints

The above experiments were conducted at the default initial user viewpoint. In this set of experiments, we examined the impact of different user viewpoints on performance. In particular, we studied the Bayou Shack scene at 10 Mbps in greater detail to show the extreme impact that different initial viewpoints can have on time to first correct frame. Three viewpoints were chosen: a viewpoint at the edge of the scene with only two objects in view (labeled as "forward"), a viewpoint in the center of the scene, similar to the earlier experiments (labeled as "center"), and a viewpoint near the edge of the scene looking inwards with nearly all objects visible (labeled as "backward").

Figure 3.8 show the latency of Bayou Shack at these viewpoints. The greatest gains are achieved by VIA in the "forward" viewpoint, because there are only two objects in the FoV that need to be downloaded, and thus the remaining 423 objects can be loaded afterwards. The "backward" viewpoint is the worst case scenario, as very little latency savings are possible due to all objects in the scene being visible. The results show that if all the resources are needed to render a frame correctly, VIA can actually perform slightly worse than Control, due to the overhead incurred by extra RTTs to the server for each additional object (metadata and binary file) request. However, when few resources are needed, the gains can be substantial, around a 90% reduction in latency.

3.7.4 Impact of network conditions

The latency savings with the various methods depends on the network conditions, as shown in Figures 3.6b, 3.6a, and 3.6. When increasing the bandwidth from Fast 3G to 10Mbps to 30Mbps, VIA and VIA-Image were able to outperform Control, with the magnitude of those improvements varying greatly; for example, going from approximately 125 to 20 to 9 seconds as network speeds increased for the Future House scene.

In general, the latency improvements of VIA over Control are inversely proportional to the network bandwidth (*i.e.*, greater improvement at slower speeds, which is exactly where we need the most improvement). This is because as network bandwidth increases, the overhead from extra RTTs (from the individual object requests) gradually dominates the total latency, while the propagation time shrinks. Note that as network speeds change, the relative improvement of VIA over Control may change (such as in Future House from 10Mbps to 30Mbps). Based on our observation that the last object to appear in the FOV

depends on the network conditions, we hypothesize this is because modern browsers typically make parallel resource requests, so while the total latency tends to reflect the network speed, it may not be perfectly proportional due to changes in download completion order and other overheads.

We also examined performance under a fast wired network connection (250 Mbps with 30ms RTT). In those experiments, the fast data transfer speeds resulted in the scenes loading very quickly (2.5 s on average for Control), so the extra RTTs from VIA method dwarfed the overall download time for the scene (approximately 4 times as long on average). Consequently, we do not recommend using VIA for extremely high speed connections (250Mbps+), as the gains from the algorithms are minimized and the extra overhead from the additional object requests could result in worse performance than Control. However, in practical use cases, the VR devices are wireless, and so the network speeds would not be as fast.

3.7.5 Impact of FoV orientation mis-estimate

Our last set of experiments examined the impact of user orientation changes. Due to user movement during the page load time or incorrect orientation estimates from the VR device, it is possible for the viewpoint at the time of first correct frame to be different from the original viewpoint input to VIA. We performed experiments to show that our methods are robust to slight changes in the user's viewpoint. We loaded the Sponza scene and recorded the time to first correct frame with a user making a yaw rotation (turning left to right) from 0° to 180°, at intervals of 15°. However, the objects downloaded by VIA still used the object scores from the mis-estimated 0° rotation.

On the x-axis of Fig. 3.9, we plot the amount of yaw rotation, and on the y-axis, we plot the latency differences between VIA and Control (a negative number indicates an improvement over Control). The resulting plot has discrete jumps because the latency only changes when the user has rotated enough that an object that was not previously visible comes into view, or vice versa. The main observation is that the VIA method continues to outperform the Control, unless the user looks more than 90° away during the page load time. This indicates that slight deviations from the user's initial viewpoint don't significantly affect the performance of VIA.

3.8 Conclusions

This chapter is the first to study page load times for WebXR-based VR scenes. Upon measuring the performance of the default WebXR, we observed that the startup latency until all the 3D objects was quite high, around 10 seconds on a 60 Mbps connection. Motivated by this, we developed methods of fine-grained splitting and requests of the objects within the user's FoV. Experiments performed on a working prototype indicated savings of more than 90% on some test scenes, depending on the scene structure (number of objects in the FoV). Future work includes generalizing our techniques to other WebGL-based applications, and working with scenes with more complex object hierarchies.

Parts of this chapter appeared in ACM Web3D 2021 [150]. Special thanks to Jingwen Huang for helping perform additional trials to show robustness. This work has been supported in part by NSF CAREER 1942700 and a Google exploreCSR grant.

Chapter 4

Going through the motions: AR/VR keylogging from user head motions

4.1 Typose Introduction

While Augmented and Virtual Reality (AR/VR) headsets have existed for many years [157], they have not been widely available to consumers until recently [70]. There has been widespread and growing adoption commercially with an estimated 26 million devices already in circulation [25]. Applications of AR/VR continue to grow beyond entertainment, to areas such as education, training, social media, and remote work [98, 100]. Such applications may require the entry of sensitive data in the form of text, entered by interacting with a virtual keyboard rendered in the headset. Private messages, passwords, and PIN codes might be entered in this manner. At the same time, AR/VR platforms now provide the ability to

run multiple applications simultaneously [99, 97]. Examples include multiple virtual web browser windows open and visible at the same time, or a chatting application overlaid on top of a virtual game.

This chapter studies the security risks posed by the confluence of these two trends – AR/VR users typing sensitive information in virtual spaces, and support for multi-app scenarios. In particular, we find that without any special permissions, background apps can infer the typed words and characters of a foreground app. The key insight is that a user's head moves subtly as she types on a virtual keyboard, and these head motions are enough to accurately infer what she is typing. Moreover, this attack is feasible because the sensor data about these head motions are freely available to a malicious background app. Fundamentally, this is because all applications, even those running in the background, need to continuously estimate the user's head pose in order to update their rendering in response to a user's head motions.

Recent works have shown that it is possible to infer the sensitive typed information through side-channels, but only through the same modality (*e.g.*, hand or head motions). For example, Meteriz-Yildiran et al. [101] show hand tracking data can be used to reconstruct hand-typed characters. Holologger [87] shows head tracking data can be used to reconstruct characters typed using "head gaze commit," where the user points her head at the desired keys. In contrast, we show that *head* tracking alone is sufficient to steal *hand*-entered text. This was initially surprising to us because qualitatively, we did not observe much head motions when AR/VR users were typing text.

To infer hand-entered text accurately from head motions alone, we faced several challenges. (1) There is an unclear relationship between a user's head motions and what

they are typing. The idea is that some character sequences could result in larger head motions (*e.g.*, "a" followed by "p"), while other sequences could result in smaller head motions (*e.g.*, "a" followed by "s"), helping differentiate between the two cases. (2) From the head motions alone, it is unclear when the user is actively typing characters or words. The idea is that text entry applications force the user to make larger movements of the head corresponding to the start and end of the text entry, and certain key presses (such as the space bar) are strong indicators of a break between words. (3) How to mitigate such attacks, without degrading user experience, is unclear. The issue is that blocking an app's access to headset tracking data would cause the app's rendering to freeze as the user moves around, breaking the immersion with the virtual world [75].

We call our system TyPose, since we estimate hand *typed* characters/words from the victim's head *pose*. Overall, we make the following contributions:

- To the best of our knowledge, TyPose is the first attack that infers the private hand-entered text of an AR/VR user using only head motion tracking information, requiring zero permissions by a malicious background app (Section 4.2).
- We design machine learning techniques to automatically infer words and characters typed by a user, including a Segmenter to divide a stream of sensor readings into the corresponding words/characters and a Classifier to infer the text corresponding to those segments (Section 4.3).
- We collect traces of AR/VR typing behavior from 21 users and evaluate our attack on these traces. The results show that TyPose can detect segments and identify words with high accuracy; for example, a top-5 classification accuracy of 82% for inferring words

(Section 4.4). We also explore the feasibility of an end-to-end attack (Section 4.5).

- We evaluate first-line mitigation strategies, including down-sampling the head tracking sensor stream or reducing their floating point precision. We find that the proposed attacks are generally robust to such mitigations, and thus these sensor streams may need to be further blocked to potentially malicious background apps (Section 4.6).

4.2 Background and Threat Model

In this section, we first introduce the relevant background with respect to AR/VR headsets (Section 4.2.1). We then define the threat model (Section 4.2.2), and discuss some of the intuitions and challenges behind the proposed attack (Section 4.2.3).

4.2.1 Background on VR

Head motion tracking. Standalone AR/VR headsets track their position and orientation in the real world using a combination of camera and inertial measurement unit (IMU) sensor readings. This allows the AR/VR platform to render believable and immersive scenes, updating the display as the user moves. Without headset tracking, the scene would appear "frozen in place" even as the user moved her head. Laggy or inaccurate headset tracking are key causes of motion sickness, particularly in VR [75]. Thus, headset tracking is a fundamental aspect of AR/VR, and it is standard practice to allow access to head tracking data to all applications to make sure they can continue rendering.

The Intertial Measurement Unit (IMU) in an AR/VR headset contains an accelerometer and gyroscope. The IMU tracks 6 Degrees of Freedom (DoF) as shown in Fig. 4.1: 3 DoF correspond to linear acceleration along the x, y, z axes, measured by the

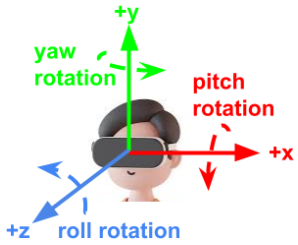


Figure 4.1: Illustration of the x, y, z axes on the VR headset. The accelerometer measures the linear acceleration along these axes, and gyroscope measures the angular velocity around them.

accelerometer in m/s^2, and 3 DoF correspond to angular velocity along the x, y, z axes, measured by the gyroscope in rad/s. These raw readings from the IMU are passed to the AR/VR software so it can update the display appropriately. The raw readings are integrated to obtain the position and orientation of the headset (also known as "pose").

Multiple AR/VR applications. Recent improvements to AR/VR platforms allow multiple applications to run simultaneously. These apps can run simultaneously in the foreground [97], or some apps can be suspended (moved to the background) in order for other apps to run (in the foreground) [99]. An example of the former is opening up a virtual web browser and a TV show app simultaneously. An example of the latter shown in Fig. 4.2, where the user pauses her virtual drumming game ② to message a friend ④ using a social media service that may require a PIN or password to be entered in order to log in. While the background app is considered paused, it is not completely suspended in its execution. Users are able to interact with the foreground application while the view of the background app continues to update if the user moves her head, and animations continue to play. In other words, the background application still receives 6 DoF headset tracking data in order

Figure 4.2: Attack scenario. (1) The foreground app (Facebook Messenger) displays on top of the (2) background app (Beat Saber), which continues to render using real-time head motion tracking. (3) The victim types messages using the system keyboard into the (4) text entry field. (5) The controller inputs are not available to background app (2) while the foreground app is open.

to render correctly and preserve the immersion of the user.

4.2.2 Threat Model

Attack overview. Our threat model assumes that the user has installed a VR application with malicious code. The attack proceeds as follows. The user switches to a new foreground application ①, suspends the malicious application to the background ②, and begins entering sensitive text ④ in the foreground application (for example, a password or work emails) using the system keyboard ③ and VR controller ⑤. The malicious application receives a signal from the VR platform that it is not in focus and logs all headset tracking data. Specifically, the malicious app logs the 6 DoF accelerometer and gyroscope data every frame for later transmission to, or pickup by, the attacker. The attacker's goal is to reconstruct portions of the sensitive text with a reasonable degree of accuracy.

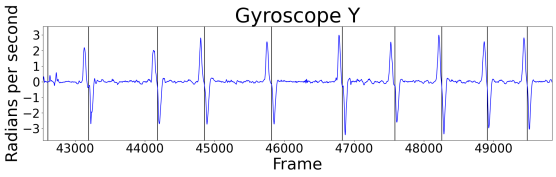
Assumptions. While a malicious app could be installed through physical access to the device by the attacker and then handed to the victim, this assumption is not necessary. The malicious app could simply be developed as a benign app or game with hidden malicious code, and be released through the VR app store (*e.g.*, the Oculus Quest Store). The user would then install the malicious application themselves unknowingly. To retrieve the data from the malicious app, they could implement benign network functionality (*e.g.*, an online leader-board) in order to obtain network access permissions from the user, then abuse those permissions to send the headset tracking logs to a remote attacker [189, 190]. Using the network in this manner is unlikely to raise concerns from the user, as prior studies have shown that over 70% of Oculus VR app dataflows were not properly disclosed by the privacy policy [162].

Our threat model assumes that the default VR operating system (in our case, the Meta Quest 2) is active and un-altered, and developer options and privileges are disabled. The sensitive text entered by the user is in the foreground app and therefore not available directly to the background app, nor is the headset or controller position/orientation available (access to this is disabled once the foreground app launches). Headset tracking is enabled to background applications without special permissions. We experimentally verified this by running a real foreground application (Facebook Messenger) and recording the headset tracking data in a custom background app with standard permissions. The above settings are the default in the Meta Quest 2. The attacker does not need access to other sensors such as eye tracking or cameras, network diagnostics, or performance counters. The attacker also does not need information about the system keyboard that the user is using, such as its

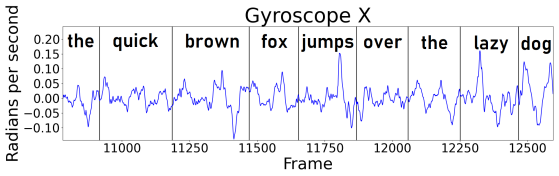
	Train with all users' data	**Train with one victim's data**
Infer words	Scenario 1A	Scenario 1B
Infer characters	Scenario 2A	Scenario 2B

Table 4.1: Attack taxonomy. TyPose can infer words or characters, and can be trained with multiple users' data or just a single victim's data. For scenario A, the victims' data is excluded from training.

position, orientation, or size, or the timing of key presses.



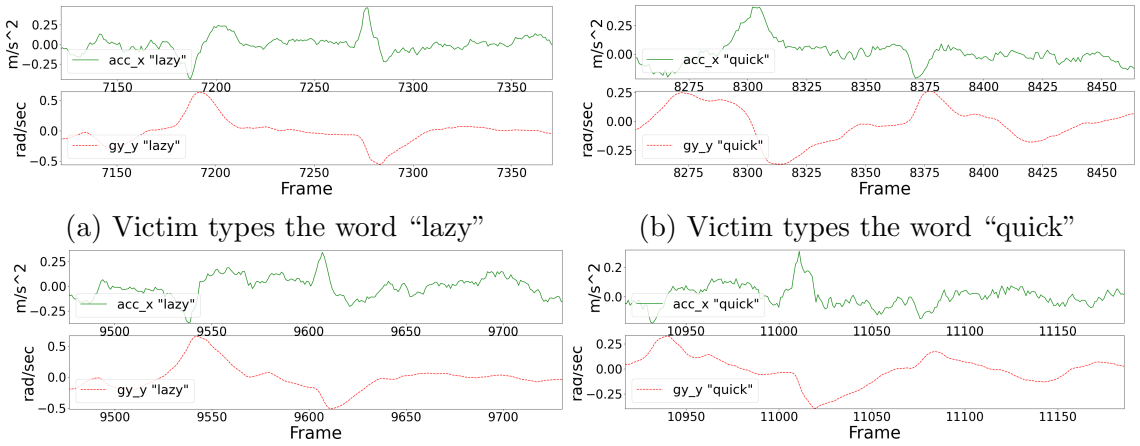
(a) Segmenting sentences is easier when the user must press a distant "submit" button in between words, resulting in large head rotations and spikes in the plot.



(b) Segmenting sentences is harder when the user only presses the space bar between words, as the word boundaries are not visually distinct.

Figure 4.3: Examples traces of a user's head rotation when typing sentences. The vertical black lines are the ground truth word boundaries. The goal is to segment the sentence into words or characters.

Attack taxonomy. We further divide the general attack described above into multiple scenarios, as summarized in Table 5.1. The attacker may be interested in inferring words typed by the victim (Scenario) 1) or individual characters (Scenario 2). Word inference is useful if the victim is typing full English sentences, while character inference is useful if the victim is typing in a random character sequence (*e.g.*, in a password). We also consider how much ground truth data the attacker has access to. In Scenario type A, the attacker has



(a) Victim types the word "lazy"

(b) Victim types the word "quick"

(c) Victim types the word "lazy"

(d) Victim types the word "quick"

Figure 4.4: Example of a user's head rotation and linear acceleration when typing the words "lazy" and "quick", twice each. The traces are quite dissimilar across words, and somewhat dissimilar across trials of the same word.

ground truth data from all users (excluding the victims), and attempts to infer the typing of a particular victim. In Scenario type B, the attacker only has ground truth data from a particular victim, and attempts to infer further typing by the victim.

The ground truth data could be collected from the attacker(s) themselves or from by willing experiment participants. Ground truth data from the victim could be collected by for example, a phishing attempt where the attacker sends chat messages to a victim. As the victim responds, the background app records the victim's head movements, so the attacker has both the victim's head movements and the ground truth text to train the models. Successful attack samples can also be used to expand the ground truth dataset.

4.2.3 Illustration of Challenges and Intuition

In this section, we present several motivating examples to illustrate the intuition as well as the challenges in inferring user typing from headset tracking data.

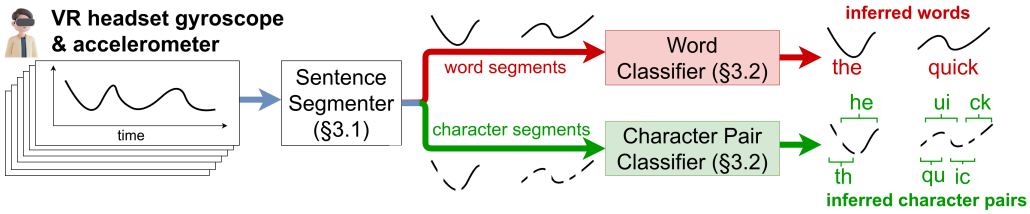


Figure 4.5: System overview. TyPose takes the 6 DoF VR headset gyroscope and accelerometer sensor readings as input, segments the time series into words (characters), and classifies the words (character pairs).

Segmenting sentences into words or characters. A first challenge is to infer when the victim is actively typing when the keyboard is open. Only if we know *when* a user is typing can we then begin to infer *what* is being typed. In certain special cases, finding when the user is typing can be relatively easy, such as when a user types words into a search bar and then has to make a large head movement to press the "search" button. These large head movements provide a strong signal that a word has been entered. In Fig. 4.3a, we show an example trace of such a case, where the user types a sequence of words and presses a button after every word. The button is located approximately 60° horizontally from the text entry field. This causes distinct spikes in the yaw angular velocity between every word, as shown in Fig. 4.3a (the black line is the ground truth word boundary). In this situation, an attacker could simply "eyeball" the raw gyroscope data to find word boundaries, or use a simple threshold policy.

The general case is when words boundaries are marked by presses of the space-bar. This is a harder problem because the space-bar is much closer to the other keys being typed and consequently results in far less distinct patterns in the head tracking data. An example trace of a user typing a sentence is shown in Fig. 4.3b, in terms of the pitch

(*i.e.*, headset rotation up and down). Eye-balling the data proves unsatisfactory to find these space-bar-marked word boundaries. However, there are generally large changes in the pitch near the word boundaries, since the user looks down slightly to press the space bar. This suggests that perhaps some patterns can be learned, and motivates our learning-based approach (Section 4.3.2).

Inferring what word or characters are typed. Even if the word/character segments are given, TyPose still needs to determine what words/characters are being typed. In Fig. 4.4, we show an example of a victim typing the same word ("lazy" and "quick"). It can be seen that the traces contain similarities as well as dissimilarities across trials. We were surprised because the head movements of the user were barely discernible visually during the experiment, but the gyroscope and accelerometer were able to pick up enough signal to differentiate the various cases. These experiments provide motivation that TyPose can successfully infer VR user typing (Section 4.3.3).

4.3 TyPose's Design

4.3.1 System Overview

TyPose consists of the following two main modules, as summarized in Fig. 4.5:

- **Sentence Segmenter:** TyPose determines when the user is actively entering text on the keyboard, as opposed to pausing in between words or in between key entries. We adopt a machine learning approach in order to segment a sentence into words (Scenario 1) or characters (Scenario 2), based solely on the gyroscope and accelerometer readings. TyPose trains two convolutional neural networks (CNNs) for these purposes,

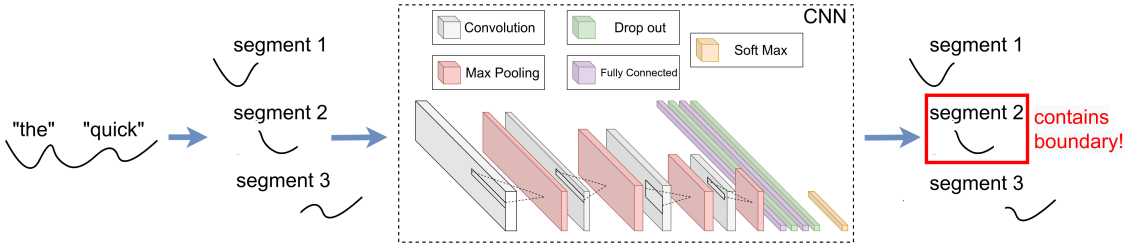


Figure 4.6: Sentence segmenter design. The example shows finding the word boundaries in a sentence. The same design is used to find character boundaries in a word.

respectively.

- **Word Classifier:** The output of the Segmenter is the word boundaries, which are converted into time series segments representing probable words. The Classifier analyzes these segments to infer the typed words. We also experiment with classifying character pairs, given character boundaries. A separate CNN is trained for each task.

In the subsequent sections, we explain the details of each of these modules.

4.3.2 Sentence Segmenter

Overview. The first problem, since the attacker only has access to the head pose and not key press timings, is to determine when the victim is starting and ending a typed word (Scenario 1), or a typed pair of characters (Scenario 2). In other words, we need to segment a sentence into words, or a sentence into characters. Our first attempt to do this was using conventional auto-segmentation techniques from the time series literature, which unfortunately proved inadequate. These techniques start with a small segment of the data, find a line of best fit, and then grow the segment until the mean squared error from the line passes a threshold [69]. Intuitively, these methods suffer from the inability to recognize

common motifs in the time series, and instead seek to segment a time series at points where a nebulous "state change" occurs. While useful for certain tasks, these methods do not leverage the fact that in our application domain, there is a clearly defined motif (*e.g.*, space bar presses in Scenario 1) that is correlated with the start/end of a segment.

TyPose leverages this intuition that there are specific motifs in the data, and that segments occur where the motifs are found. The main issue is that we do not know what exactly these motifs look like in the time series data. Instead, these motifs need to be learned. To solve this, we treat the segmentation problem as a binary classification problem: given a short window of the time series, is it a boundary of a segment? In this way, we transform the problem of finding segments to one of finding boundaries. For sentence segmentation into individual words, the boundaries are the presses of the space key. For sentence segmentation into individual characters, the boundaries are any character press, including spaces.

Beyond accuracy (as we will show later in Section 4.4), the classifier-based approach has several advantages: there is no need to know the exact dimensions of the keyboard, its position in the victim's field-of-view, or whether the keyboard moves during the typing process. This is because the model is trained on the raw accelerometer and gyroscope data and does not need semantic information about the keyboard or VR scene. For example, even if the keyboard drifts to follow the user's head orientation (as it does in our experiments with the Meta Quest 2), our classifier can still perform well. We believe this is because the sensors measure change, and hence the readings for a given head movement are similar no matter the keyboard's absolute position/orientation.

Model design. At a high level, the segmenter takes windowed samples from the 6 DoF time series as input and outputs the probability that a sample contains a boundary or

not. From this, TyPose considers the sample as potentially containing a boundary if the probability is above a threshold T. This is illustrated in Fig. 4.6. The length of the window is parametrized by W.

Our specific classification method is based on CNNs, which have excellent predictive ability in image classification and time series classification problems [129, 183]. Later in Section 4.4, we also compare against classical k-nearest neighbors (KNN) and Random Convolutional Kernel (ROCKET) time series methods. Essentially, TyPose treats the 6DoF time series window as a 2D "image" of size $6 \times W$, where the value of each "pixel" in the input "image" is the floating point linear acceleration or angular velocity of the headset at each time. The specific model architecture comprises 4 convolutional layers: four 1D kernels with kernel size 3 and 32, 64, 128, 256 units respectively. Finally, the data is fed into two fully connected layers and a soft-max layer. The output dimension of the last fully connected layer is equal to the size of the number of classes (*i.e.*, 2 classes – space or not – in the sentence segmenter, or 2 classes –any character press or not– in the word segmenter). After every convolutional layer, there is a 1D max pooling layer of size 4, and in between each fully connected layer, there is a drop out layer with a chance of 0.5 in order to reduce over-fitting. The CNN uses ReLU activation functions and the sparse categorical cross entropy loss function.

The intuition behind this CNN architecture was to initially convolve the data points from the gyroscope and accelerometer individually along the time axis, rather than jointly convolving gyroscope and accelerometer readings together. In other words, we used 1D kernels in the early layers so that the matrix entries corresponding to gyroscope or accelerometer readings were mostly kept separate. We found empirically that such separation

resulted in better performance. Otherwise, the model architecture was close to those found in previous literature [84, 124, 68].

4.3.3 Word Classifier

Overview. Given the word segments, (either the ground truth or those predicted by the Segmenter from Section 4.3.2, the next problem is to determine what words are being typed. To the best of our knowledge, no models for predicting typed VR characters based solely on head pose exist. We model the attacker's problem as a classification problem, with the head pose 6 DoF time series data as input, and the typed text as the output classes. In Scenario 1 for word inference, the output classes are the words being typed. We also experiment with Scenario 2 for character inference, where the output classes are the character pairs being typed.

Model design. The word or character pair classifier use a similar CNN model to the segmenter (Section 4.3.2), with the main difference being the size of the input and the number of possible output classes. Since the word/character segments output from the Segmenter can be of variable length, depending on the user, TyPose considers the maximum length word/character, pads shorter samples with 0's, and inputs them to the word/character pair classifier. Changing the CNN design to predict from a different number of possible output classes simply requires changing the output size of the final fully connected layer. In Scenario 1, the number of output classes is equal to the size of the word corpus being trained on. In Scenario 2, the number of output classes is equal to the number of unique character pairs under consideration. In this work, we consider common words and character pairs in the English language (details in Section 4.4).

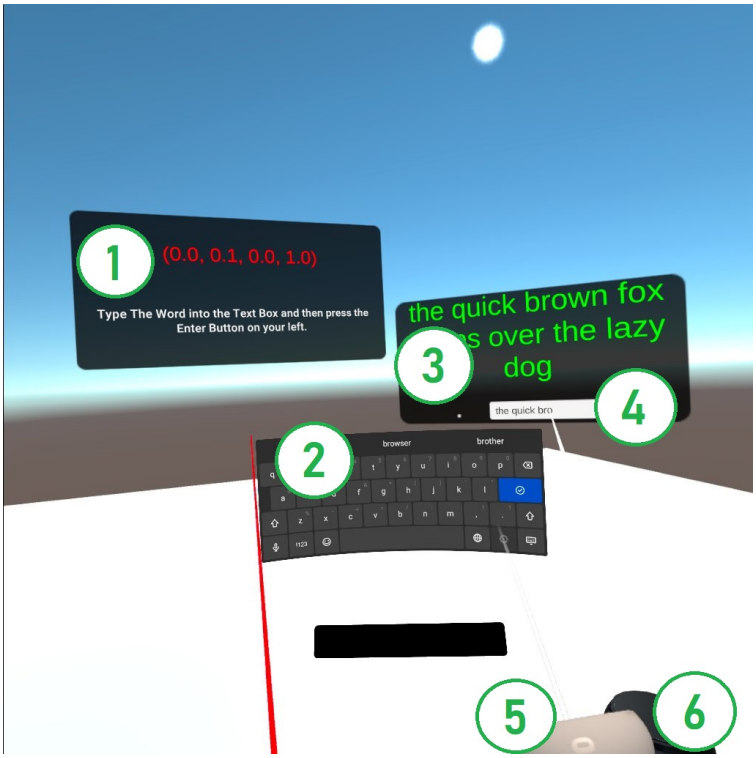


Figure 4.7: Data collection app. (1) The background app records the sensor readings even it is not in focus. The user types into the foreground keyboard (2), following the sentence prompt (3), in the text field (4). (5) The system hand controller is used to type, but the background app only has access to the (6) application hand controller, which is frozen while the keyboard is in focus.

4.4 TyPose Evaluation

4.4.1 Data Collection

Data collection application. In order to show the viability of an end-to-end attack on current AR/VR systems, we created a malicious background VR application to log the headset's accelerometer and gyroscope readings. The application was created in Unity version 2020.3.26f1 [166] and deployed on the Meta Quest 2. A screenshot of the application

is shown in Fig. 4.7. The app prompts the user to type the specified words in the text field using the default Meta Quest 2 system keyboard. The app records the headset's accelerometer and gyroscope data (velocity and angular velocity) at 72 Hz [165]. If users glance up at the text prompt while typing, this adds noise to the collected data, although we did not notice such movements when visually inspecting the data. During the training phase, our data collection app has ground truth access to the typed characters for analysis and training the machine learning models [42]. During a real attack, the text input is to the foreground application, and the malicious background app only has access to the headset tracking readings and the pre-trained machine learning models.

User study recruitment and warm-up phase. With approval from our institution's IRB, we collected typing data from volunteers wearing an AR/VR headset. The user study was performed with 21 participants of varying age, height, weight, gender, and amount of experience with VR. This is in line with prior AR/VR human subjects research [87, 146, 101], which have 2-25 volunteers per experiment. Users were recruited through Slack or personal contacts, requesting volunteers for an AR/VR typing experiment, and participants were entered into a raffle for a $50 Amazon gift card. Before starting the study, users were informed that "the headset will record signals from your behavior while you are interacting with it" and that they could stop the study at any time. Users were initially not informed of the exact purpose of the study in order to avoid "participant bias" [17], an effect where participants who know the hypothesized outcome of a study may act to achieve (or confound) the outcome. Users were also instructed on how to operate the headset and controllers, how to adjust the headset to fit comfortably, and not to touch the headset with one's hands or anything else that would interfere with the headset's tracking. Each trial took up to 30

minutes, varying based on the individual participant's natural typing speed.

User study experiment phases. Volunteers participated in two phases of data collection: *word typing* and *sentence typing*. Each phase lasted approximately 15 minutes with a break in between. The former was used to train and test the word and character pair classifiers. The latter was used to train and test the sentence segmenter as well as provide additional words and character pairs for the classifiers.

- *Word typing phase:* A participant wore the VR headset and held the right hand controller while typing each prompted word and pressing the submit button between each word. The prompted words were from a list of 40 different words of 2 or 6 characters in length, selected at random from the top 5000 most frequent words in the Corpus of Contemporary American English [27]. For ease of ground truth labeling in this experiment, the submit button was placed far away to the side from the keyboard and text field, in order to require the user to move their head a large amount in between words and create a large signal change on the gyroscope readings. Each participant repeated this process for all 40 words, 3 times each (120 words per participant). In total, 21 users participated with 2520 total typed words.

- *Sentence typing phase:* Participants typed full sentences, with words separated using the space bar (rather than the submit button in the previous experiment, in order to be more realistic). The sentences were 5 words long and were randomly generated permutations from the 2-letter words from the word typing phase plus 20 new 6-letter words. The participant typed the full sentence, cleared the text field with the submit button, and each word was represented 3 times in the total sentences for each

participant. 21 participants participated in this experiment for a total of 610 sentences typed.

We combined the unique words from the word and sentence typing phases when performing word classification, for a total of 60 unique words and 5040 samples. Over all experiments, about 6% of words contained typographical errors and were therefore not classifiable. These errors were not evenly distributed between participants and varied from 2 to 12% depending on the participant. Typos were included in the character pair classifier if they appeared at least once per participant.

User post-study disclosure. Several weeks after the study, users were sent a debrief email disclosing the purpose of the study, the specific nature of the data collected (*i.e.*, headset gyroscope and accelerometer data), and researcher contact information in case of questions, concerns, or complaints. We acknowledge that the debrief should have taken place in person immediately after after the study. Users did not express or appear to experience any discomfort during the study, and we did not receive any complaints or requests to exclude a participant's data after the debrief messages were sent.

4.4.2 Comparison Methods

We compared our Classifier against several other methods: a KNN-based approach [73] and a Random Convolution Kernels transform (ROCKET) with logistic regression [29].

- **KNN:** We experimented with multiple distance metrics for the KNN, including including Dynamic Time Warping [12] and first order statistics of the time series sample (*e.g.*, mean, variance, etc.); however, neither of these metrics gave prediction

results significantly better than random. Instead, we element-wise multiply the head tracking vector (6 elements) by the inter-frame time and sum them. This essentially performs a coarse integration of the acceleration and angular velocity into a single velocity-pose vector. The number of clusters was set to the number of classes in each problem.

- **ROCKET:** The main idea of ROCKET is to convolve the time series with 10,000 randomly generated kernels to produce features, and train a logistic regression on these features. There has been some success using ROCKET in classifying 6 DoF data into activities [137]. We chose ROCKET as a baseline due to its past success and its ability to handle multivariate time series data.

- **Random:** We compute the probability of a random guess, *i.e.*, $1/N$, where N is the number of classes to predict (*e.g.*, the corpus size for word classification).

We use the ROCKET implementation in the "sktime" package [148] and KNN from "scikit-learn" [143]. The CNN model was implemented using Tensorflow 2.8.0 and the Adam optimizer [44]. All methods were coded in Python 3.7.0 [123] and run on a PC with a 2.8 GHz Intel i7 processor and 32 GB of RAM, taking up to 1 hour to train a CNN. Unless otherwise noted, 75% of the data was used for training, and 25% for test. Results conducted with our full dataset (5040 typed words, 60 unique words) are denoted by the method name appended with a "+" (*e.g.*, "CNN+"). Results obtained from an initial, smaller dataset (1200 typed words, 40 unique words) lack the suffix (*e.g.*, "CNN").

Data pre-processing. Before feeding in the gyroscope and accelerometer data into the Segmenter, we perform several pre-processing steps. Consumer-grade accelerometer and

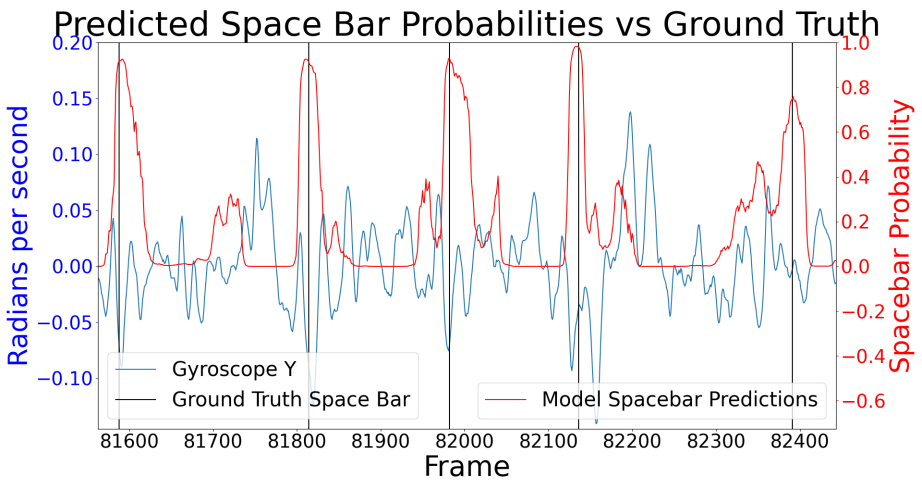


Figure 4.8: Example of TyPose's predicted word boundary probabilities (red) plotted against the ground truth (black). The raw gyroscope trace is shown in blue.

gyroscope data can be noisy [76]. An optional pre-processing smoothing pass may be warranted, which is performed by setting the accelerometer and gyroscope values at a specific frame to the average of the surrounding values (windowed averaging). TyPose performs this pre-processing step for the Segmenter, but not for the Classifier, as it was not found to have a significant impact there.

4.4.3 Evaluation of Segmenter

Sentence Segmentation into Words

Setup. We use the sentence typing dataset (Section 4.4.1) to train and evaluate our boundary classifier. For Scenario 1A, 487 sentences were used to train and 54 were used for evaluation. For Scenario 1B with a single victim, 23 sentences were used for training and 6 were used to evaluate. To handle the class imbalance, since there were few examples of space bar presses, we oversampled the minority class and added class weights.

		Boundary Predicted?	
		Yes	No
Actual	Yes	TP = 69	FP = 154
	No	FN = 135	TN = 83, 067

Table 4.2: Sentences segmented into words (by space bar presses) across multiple users in Scenario 1A.

To evaluate word classification, we walk a window of size $W = 200$ frames across an un-labeled time series trace, including all 6 DoF. Fig 4.8 shows an example trace with the probability of each window being a boundary marked in red, and the vertical black lines indicating ground truth boundaries. The windows are overlapping in the test set, so ideally only those windows with a boundary near the center will be classified as highly likely to contain a word boundary. If several adjacent windows have softmax probability $> T = 0.8$, the local maximum is predicted as the boundary. A predicted boundary is considered as a true positive if its center is within 10 frames of the true boundary. This definition is used to compute the true positives, false positives, false negatives in the following results.

Results. For Scenario 1A, where multiple users' data is used to train a model, the CNN-based sentence segmenter results in 69 true positives, 154 false positives, and 135 false negatives out of 223 true space bar presses, as shown in Table 4.2. When trained on sentences from multiple participants, TyPose is able to find word boundaries for many words, albeit with a fairly high false positive rate. We account for this later in the end-to-end attack (§4.5) using data augmentation strategies and information about the average word length.)

For Scenario 1B, where only data from a specific victim is used for training, and later test, the CNN classifier gave 21 false positives and 10 false negatives, as shown in Table 4.3.

		Boundary Predicted?	
		Yes	No
Actual	Yes	TP = 14	FP = 21
	No	FN = 10	TN = 9633

Table 4.3: Sentences segmented into words (by space bar presses) for 1 participant in Scenario 2A.

The performance seems better when using data from just one participant, out-performing training and evaluation on data on many individuals, which makes sense intuitively, since the segmenter becomes personalized to an individual user.

Sentence Segmentation into Characters

We also utilize the sentence typing data to train and evaluate Scenarios 2A and 2B. Instead of training on windows centered on just space characters, we train on all character entries. The window size was shrunk to 32 frames due to the samples being closer together in time. When training with 19 users and predicting the remaining 2 users, the classification accuracy is 71%. When training and testing on a single user (with an 80/20 train/test split), the classification accuracy is 85%. Since the focus of this chapter is on word classification, we describe those results next.

4.4.4 Evaluation of Classifier

Word Classification

Setup. For Scenario 1A, we used the *word typing* dataset described earlier (Section 4.4.1), which contains both two-letter and six-letter words. For ROCKET and CNN, 75% of the data were used for training and 25% for evaluation for both word lengths (corresponding to 408

Method	Top-1 accuracy	Top-5 accuracy
Random	0.025	0.125
ROCKET	0.289	0.675
CNN	0.353	0.710
CNN+	0.400	0.820

(a) Classify from 2-letter or 6-letter words

Method	Top-1 accuracy	Top-5 accuracy
Random	0.05	0.25
ROCKET	0.390	0.796
CNN	0.654	0.932
CNN+	0.659	0.929

(b) Classify from 6-letter words only

Figure 4.9: Word classification accuracy in Scenario 1A, when training and testing on data from all participants.

two-letter words and 398 six-letter words for training, and 136 and 132 for test respectively). Additional data were gathered for CNN+ where 19 users were used for training and 2 users excluded for testing, then 5-fold cross validated (5 different train/test splits) for the final averaged accuracy results. For Scenario 1B, we form a third word classification data set from additional trials performed by a random participant to evaluate ROCKET and CNN. The participant repeated the experiment three times on three separate days for a total of 180 six-letter words. For CNN+, five participants typed a total of 120 six letter words each out of 40 possible instead of 20 words and were cross validated.

Results. For Scenario 1A where all users' data is used for training, Fig. 4.9a shows the top-1 and top-5 accuracy for the across all 2-letter and 6-letter words, for all the different methods (CNN+, CNN, ROCKET, and Random). Top-k accuracy is a common evaluation metric for speech and keyboard inference [101, 87, 146]. While both CNN and ROCKET drastically out-perform random guessing, the CNN proves the most accurate overall. Focusing in on the

Method	Top-1 accuracy	Top-5 accuracy
Random	0.05	0.25
ROCKET	0.18	0.66
CNN	0.75	0.99
CNN+	0.65	0.92

Table 4.4: Word classification accuracy in Scenario 1B, when training and testing from 1 participant.

prediction accuracy of the 6-letter words along (Fig, 4.9b), we see the accuracy can be even better than the general case. This may be because the 6-letter words have longer duration, providing more information to the classifier. Adding additional users' data and additional words (CNN+) has a slightly increased accuracy. These results are also in line with other top-k accuracies reported by other cross-modality AR/VR inference attacks [146]. Note that we do not include KNN results, which were very poor. This is because the KNN input features only represented the aggregate head movement during the word, in order to have a low-dimensional (6×1) input size and a reasonable run time; however, higher-dimensional input features would be needed to accurately differentiate words from each other.

For Scenario 1B, the top-1 and top-5 accuracy for the different methods are shown in Table 4.4. The classification accuracy is similar to that of Scenario 1A (Fig. 4.9a), again with the CNN-based methods performing the best. Adding 20 additional words and cross validating the single user results brings the accuracy in line with the multi user scenario. We did note, however, that some users have results far above (or below) the average, suggesting that some participants have more predictable head movements than others. Hence models trained on a single participant may be able to learn participant-specific behaviors and exploit that learning for improved classification accuracy. On the downside, Scenario 1B may be

Method	Top 1 accuracy	Top 5 accuracy
Random	0.022	0.111
KNN	0.18	0.48
ROCKET	0.20	0.54
CNN	0.23	0.58
CNN+	0.33	0.72

Table 4.5: Character pair classification accuracy in Scenario 2A, across all participants.

less practical because it requires ground truth training data from a target victim.

Character Pair Classification

Setup. For Scenario 2A, we further subdivide the words from the *word typing* dataset in their constituent character pairs. For example, the word "fox" contains two character pairs, "fo" and "ox". In total, 1852 character pairs were used for training and 618 were used for test when evaluating KNN, ROCKET, and CNN. This resulted in 45 possible character pairs. For CNN+, 121 unique character pairs were used for a total of 9448 training and 2362 test sample character pairs. For Scenario 2B, similar to word classification (Section 4.4.4), we formed a character pair dataset from a single user. The number of possible character pairs (62) is larger than in Scenario 2A dataset, due to a large number of frequently repeated typographical errors.

Results. For Scenario 2A, we show the top-1 and top-5 accuracy in Table 4.5. The CNN-based outperforms other methods and is far better than random, but further accuracy improvements are desirable. Adding additional users and samples to the training set improves the accuracy by 25-50% (CNN+ compared to CNN). We believe that even noisy predictions could still be useful in reducing the search space of possible passwords, for example by

Method	Top-1 accuracy	Top-5 accuracy
Random	0.016	0.08
KNN	0.21	0.50
ROCKET	0.24	0.52
CNN	0.28	0.58
CNN+	0.31	0.77

Table 4.6: Character pair classification accuracy in Scenario 2B, out of 62 (121 for CNN+) possible character pairs, from 1 participant.

passing in these character pair probabilities to password cracking software such as "John the ripper" [116]. Reducing the search space would reduce the password cracking time and show headset accelerometer and gyroscope as a useful side-channel for password stealing.

For Scenario 2B that zooms on a single user, we show the top-1 and top-5 accuracy in Table 4.6. Compared to Scenario 2A that uses all users' data for training, the personalized attack has a better accuracy. This is in line with the results in previous subsections.

4.5 Demonstration of end-to-end attack

In the event that the attacker does not have access to word or character pair entry times, an end-to-end attack on the entire unmarked time series of the victim's head pose is needed. Towards this, we combine the models from Sections 4.3.2 and 4.3.3 to explore the feasibility of an end-to-end attack.

Setup. We utilize the sentence typing dataset from Section 4.4.1. Following the top branch in Fig. 4.5, the raw sensor streams are fed into the segmenter, which finds the word boundaries that are then fed into the Word Classifier. To measure the performance of the end-to-end attack, we compute several metrics:

- The **edit (Levenshtein) distance** $e(A, B)$ assigns a penalty of 1 to every add, delete, or swap made to transform the ground truth sentence A into the predicted sentence B [101, 58]. The edit distance is the minimum number of such operations needed, and lower is better. The minimum edit distance is 0 and the maximum is unbounded.

- The **normalized edit distance**, $\frac{e(A,B)}{\max(|A|,|B|)}$, normalizes the edit distance by the maximum sentence length, since sentence lengths are variable.

- The **discounted edit distance** assigns a lesser penalty to swaps when the true word is within the top 5 predictions. The weight is [0.2, 0.4, 0.6, 0.8] if the true word is predicted as the [2nd, 3rd, 4th, 5th] most likely word, respectively.

To account for the noisy segmenter, we employed two strategies. First, we performed data augmentation when training the Word Classifier; specifically, we add a random amount (up to 1 second) of extra data to the beginning and end of the true word segment. Second, we provided the average word and sentence duration as side information; with an average word length L and sentence typing duration D, the segmenter picks the $\lfloor \frac{D}{L} \rfloor - 1$ most likely word boundaries in a sentence. The results reported below are the average of 5-fold cross-validation. Each test has about 50-60 sentences from never before seen users, so the scenario is quite challenging.

Results. The edit distance is 4.6, the discounted edit distance is 4.4, and the normalized edit distance is 0.93. A naive attacker could guess $\lfloor \frac{D}{L} \rfloor \approx 5$ words in a sentence with a 1/40 chance of getting each word correct, giving a naive edit distance at 4.875. Thus compared to a random guess, our approach has better performance. We observe that the end-to-end

attack is sensitive to the segmenter's performance, since it is the first step of the end-to-end pipeline. Upon decomposing the edit distance, we find that 16% are from insertions, 7% are from deletions, and 77% are from swaps. The insertions and deletions can be attributed to segmenter errors, while the swaps could be attributed to the segmenter or classifier.

The end-to-end attack results could likely be improved by adding priors on English grammar semantics. However, our scenario is a particularly challenging one with users typing random sequences of words to form sentences, so we could not experiment with such priors. Our approach could also be combined with other sensor modalities to improve segmentation, such as performance counters [182] or WiFi signals [5]. More sophisticated post-processing, such as a feedback loop between the word classifier and sentence segmenter, could also help. Overall, we believe that this end-to-end attack is a good starting point to demonstrate the feasibility of head pose as a source of information leakage for hand-typed sentences. Moreover, each of the individual components of the end-to-end attack can be used independently; for example, if the attacker desires only a single word to be detected (*e.g.*, the answer to a secret question password challenge), then the word classifier alone can be used with good accuracy. Further discussion is provided in the Limitations section.

4.6 TyPose Attack Mitigation

The simplest way to prevent the system's attack is to disallow background apps' access to the VR headset accelerometer and gyroscope readings, when the background app is out of focus. However, this may not be desirable as it also prevents the background app from updating the rendered image in response to the user's movements (in other words, causing a "freeze"), leading to poor user experience [118, 75]. Therefore, we first investigate mitigation

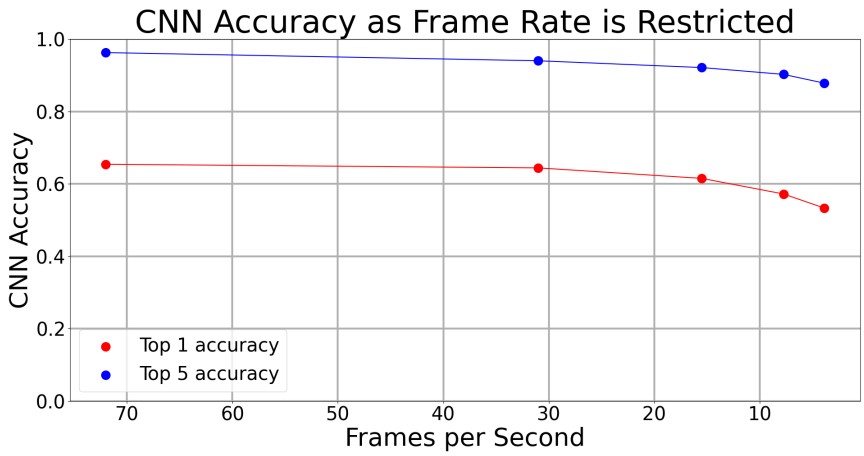


Figure 4.10: Reducing the rate that head tracking information is given to the background application does not have significant effects on Classifier accuracy until < 10 Hz.

strategies that try to avoid harming the user experience by still allowing background app rendering. We experiment with two methods: (a) reducing the frequency that the sensor streams are given to the background app, and (b) reducing the precision of the floating point values provided to the background app.

Reduced IMU sampling rate. We re-train and evaluate the CNN from Section 4.3.2 on word classification using the same dataset as Fig. 4.9b, but down-sample the IMU reading provided to the background app. We plot the classification accuracy in Fig. 4.10 for various sampling frequencies, ranging from 72 Hz (the default) to 5 Hz. The top-1 and top-5 accuracies do not drop much as the sampling frequency decreases to 5 Hz. This suggests that 5 Hz still supplies sufficient information for classification. Alternatively, the robustness to sampling rate may suggest that an important predictor of what a user is typing is the length in time of the sample (although we experimented with length alone as a classifier, and found it insufficient). In any case, we conclude that the frequency reduction needed (to

less than 5 Hz) would also inhibit the background app's ability to update the display at an acceptable frame-rate [193].

Reduced precision. We also re-train and evaluate the CNN with the same data as above, but during pre-processing, we round off the floating point IMU values to different decimal precision. Plotting the results in Fig. 4.11, we see minimal accuracy reduction even at as low as two significant figures. This is likely because even though information from the significand of the floating point number is reduced, the remaining information along with the exponent, sign, and in the length of the samples provides enough for classification. Since reduction of significant digits in head tracking information can lead to "judder" or drift of the rendered image, without significantly reducing the classification accuracy, we do not recommend this as a viable mitigation.

Overall, given that the classification can still be performed at better than random accuracy with both of these mitigation techniques, more elaborate defense mechanisms are needed. One possibility is for the AR platform to move the user to a default "system room" in the background whenever a keyboard is opened in the foreground, instead of keeping the malicious app in the background. This comes at the expense of reducing user immersion and possibly losing state in the background app. Another possibility is to randomize the keyboard location or size each time it is opened, or even randomize digit key locations in the case of PIN code entering. Alternatively, the AR platform may employ a modified version of "time warping" [75] where the background app is rendered with a wider field-of-view, and the AR platform later crops the final rendered image using real-time head tracking data not provided to the background app. The above mitigation strategies may degrade user experience, and further human subjects research may be needed to understand their

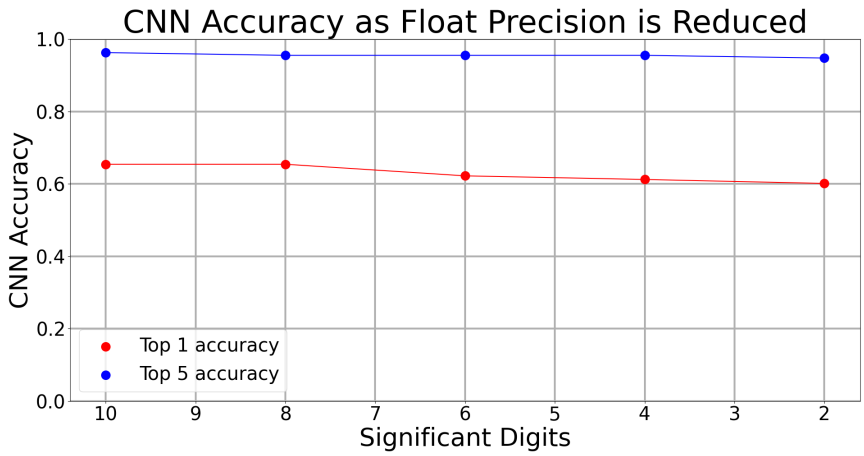


Figure 4.11: Rounding off the floating point values of the head tracking information given to the background application does not have significant effects on CNN accuracy.

perceptual impacts.

4.7 TyPose Limitations

While participating in the user study, some volunteers who were not able to find a comfortable way to wear the headset or did not visibly move their heads at all while typing, possibly due to discomfort. The data from these users were not discarded but did show less head motions than others. This suggests that if a user were aware of the possibility of malicious head tracking, it is possible to hold one's head nearly still while typing in order to confuse the classification model and limit its accuracy. However, this would require conscious behavioral change on the part of the users.

A second limitation is that any change to the text entry method may require the collection of a new training dataset and model in order to maintain TyPose's accuracy. These changes could include operating system updates that replace the system keyboard,

the addition of a numberpad, or future novel AR/VR text entry mechanisms. Related to this, the machine learning models we experimented with may be suboptimal, as the primary goal of this project was to demonstrate the attack's feasibility. We would be interested to see other learning-based approaches towards these types of classification problems. Further, the models are trained to detect only 60 unique words, and it could fail if users type a more diverse set of words. The attack could be generalized by expanding this dictionary through additional user data collection and model training. The set of words could also be carefully selected so that the attack could still recognize key words in the sentence to extract the main meaning, even if every word is not perfectly recognized.

Finally, a key lesson learned is the cascading effects of errors and thus the importance of the first segmentation step on end-to-end performance. The classifier accuracy dropped when noisy segmentation boundaries were provided by the segmenter, despite our best data augmentation techniques to mitigate this. Since segmentation has shown success in single-modality attacks [87, 101], we are optimistic that improved segmentation techniques or additional side channels [182, 5], combined with the already high-performing classifier, can help the end-to-end attack.

4.8 TyPose Related Work

AR/VR key-logging attacks. AR and VR devices open the door to new application spaces with unique threat models. As a result, a number of new attacks that target these devices have emerged. Arafat et al. [5] developed a key logging side channel attack leveraging fluctuations in wireless signal around a VR user. Different head and hand motions cause different fluctuations in the Wi-Fi signal which can be correlated to key inputs. Similarly,

TyPose uses changes in head movements and timings to infer user text input, but does not require external wireless sensors. Face-Mic [146] is an example of another type of a cross modality attack (head tracking to infer spoken words), but focuses on audio input whereas text-based input is currently more common.

Meteriz-Yildiran et al. [101] and Holologger [87] demonstrate single modality attacks capable of stealing sensitive information on AR devices (*i.e.*, hand tracking to infer hand-based typing, head tracking to infer head-based typing), rather than cross modality (head tracking to infer hand-based typing) as we do. Our problem has different threat models: We did not use hand tracking data as in [101] because it is blocked from background apps by popular AR/VR development engines like Unity, in our experience; and head-based typing as in [87] is less common and results in higher typing error rates for users compared to hand-based typing [54]. Our problem is significantly more challenging than these single-modality attacks because we have to infer both when and what words are typed, causing errors to accumulate in the end-to-end attack. Such single-modality attacks get the latter for free (*e.g.*, once you can correctly estimate when a hand presses a key, identifying which key is relatively straightforward from the hand pose).

AR/VR authentication mechanisms. AR/VR introduces unique password entry methods [72, 172, 155, 96]. These works use unique 3D hand motion paths, head gaze, or eye tracking to enter passwords, suggesting that if the sensor data used for these input methods were logged, the sensitive information could be inferred. These are outside the scope of this work, as TyPose focuses on text entry using virtual keyboards. There have also been efforts to design shoulder surfing resistant authentication methods [181, 114]. Since TyPose uses head tracking information freely given to AR/VR applications, shoulder surfing is not

necessary and mitigating it would not reduce the efficacy of our attack.

Other AR/VR security issues. Other security and privacy threats on AR/VR devices and applications are covered by several excellent surveys [133, 132, 112]. Our work falls at the intersection of data access, input security, and multiple applications mentioned by these surveys. Cheng et al. [22] investigate the human impacts of AR/VR perceptual manipulation attacks. Shang et al. [144] use network traffic analysis in a multi-user app to infer user location through custom built malicious applications. Our attack focuses on inferring sensitive text rather than location. Designing sharing techniques to enforce permissions or prevent 3rd parties from accessing private virtual content is another problem [138, 126], as is malicious output of the AR display [78]. TyPose is orthogonal to such works in that it focuses on AR/VR input modalities, rather than rendering and sharing of virtual content.

Conventional key-logging side-channel attacks. Our work investigates head movement information to build up side-channel for information leakage attacks. Prior works have explored using motion sensor to steal keylogging secrets on mobile phones [176] and smart watches [90, 85, 170, 176, 171]. Similar to the insights in our attack, the user motion as they type correlates with the location of the keys on a soft keyboard, enabling these attacks. Other works exploited acoustically [7, 52] or EM [167, 9] side-channels to extract keylogging input. However, these attacks require physically connecting a probe or microphone close to the keyboard. In more conventional computing settings, several works have deployed keystroke inference attacks based on CPU-based [131], cache-based [50] or GPU-based [105] side-channel. More generally, the use of machine learning in keylogging attacks using data from seemingly-unrelated sensor inputs has some precedent [107, 192]. In particular, prior

knowledge of the English language and grammar [192] could be used to improve on our results.

4.9 TyPose Conclusions

As AR/VR devices become prevalent, there is a pressing need for research into their security and privacy risks. In this work, we show an attacker may freely obtain a stream head tracking data from an AR/VR device, segment it, and classify it in order to obtain sensitive text information. The attack is shown to be especially accurate when trained on specific targeted victims. While a simple mitigation tactic – blocking access to the head tracking data – exists, it breaks desired functionality in a background application. The attack is also resilient to less extreme mitigation strategies, such as reducing the frequency and precision of the sensor readings, to the point that the background app would have to be visually compromised before successful attack mitigation. In future work, we plan to characterize user sensitivity to the background app's visual display, in order to develop new mitigation strategies, as well as other text entry methods.

Acknowledgements

The work for this chapter is published in part in USENIX 2023. We thank the anonymous reviewers and shepherd of that publication for their valuable comments, from which this chapter greatly benefited. We are also grateful to the user study participants who generously volunteered their time. This work was partially supported by the NSF grants CNS-1942700, CNS-2053383, CCF-2212426, and a Meta faculty research award.

Chapter 5

Attacks on Shared-State Augmented Reality Applications

5.1 Shared-State Introduction

AR technologies have made possible a wide range of applications that involve using real-world data to create imagery with overlaid virtual objects. These virtual objects take the form of anything from face-filters to virtual characters but are always placed relative to some point in the real world, such as a table, face, or recognizable landmark. AR has been around for some decades [20] but recent ubiquity of mobile devices and even more recent introduction of AR headsets [103] has set the foundation for AR applications to hit the mass market [175]. AR sees use in entertainment, engineering, education, and more.

Most of these applications did not allow for multiple users to interact or collaborate with the same augmented imagery until the last few years. For example, in 2019, Pokémon Go enabled users to view the same virtual creatures at the same time in some shared space

using a "Buddy Adventure" System [111]. In order for these multi-user interactions to take place, some information about the state of the world, such as nearby flat planes, landmarks, and virtual objects, must be shared. As security researchers, a natural question arises: What possible security threats exist for these "shared-state" applications?

Interactions between users and this shared-state can be thought of as read and write operations. One can "write" a new virtual character to the shared space and "read" others virtual characters in order to render them to the device. Are there undesirable or potentially harmful uses of these operations? We seek to find these threats and demonstrate how they work in detail so as to inform and suggest ways of mitigation.

Directly manipulating the shared-state is not normally possible, as it is typically stored on some server under control of a third party which are well secured and would involve attacks not unique to AR. This provides a novel challenge: How to manipulate the shared-state using the unique inputs that a basic AR user has available to them. These inputs take the form of Global Positioning System (GPS) coordinates, mobile camera images, and Inertial Measurement Unit (IMU) vectors.

We are primarily interested in threats that cause issues unique to AR rather than simply preventing applications from functioning. For example, a malfunctioning construction marker application could cause confusion and destruction of property if demolition signs are maliciously placed in incorrect areas, not just missing. These issues are related to one of the main problems in AR research: How do we get a device to place augmentations, accurately in the real world?

In this chapter, we make the following contributions:

- We create a taxonomy of shared-state Augmented applications categorized by input sensors,

shared-state storage time span, and user permissions.

- We form a unified threat model covering many current and prospective AR applications.
- We demonstrate multiple AR-specific attacks on three different augmented reality applications using mobile phones and personal computers and document the results. To the best of our knowledge, these attacks are the first formally documented attacks of their kind on these applications.
- We discuss mitigation strategies for these attacks for all scenarios.

First, we cover the relevant background for shared state AR attacks, then define our threat model. We then describe the methods used to perform and show evaluations for three different attack scenarios using three different AR applications using separate APIs. Then we discuss a wide array of mitigation for these threats. Finally, we wrap up with a discussion of the related works and conclude the chapter.

5.2 Shared-State AR Background

In these next few subsections, we first introduce the relevant background with respect to shared state in AR (Section 5.2.1). Next, we describe the current landscape of shared state in commercial AR systems (Section 5.2.2). Finally, we define the general threat model (Section 5.2.3).

5.2.1 Shared State in Augmented Reality

In order to facilitate interactions between multiple users in AR, a mutually agreed-upon model of the reality to augment, and the augmentations within, is needed between users [127, 169]. Ideally, this model should be consistent across devices and thus is typically stored in

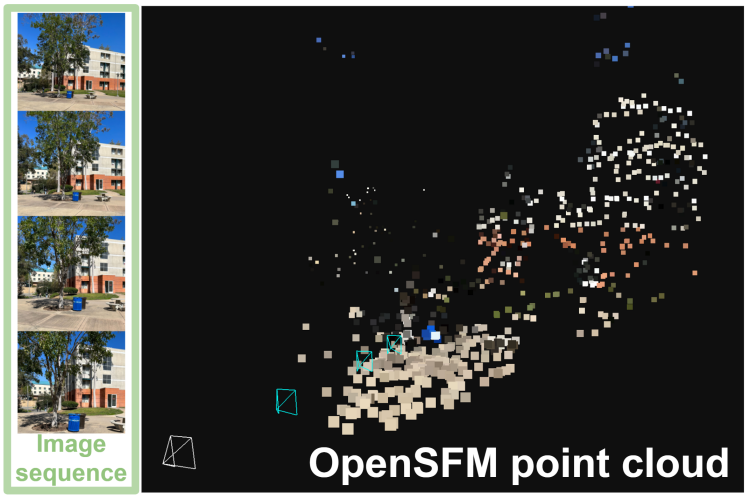


Figure 5.1: Example point cloud map for an AR shared state. Visual features are found in multiple images of an image sequence and placed into 3d space via multiple view geometry.

the cloud, providing a central access point. In such a model, multiple users interact with the shared augmentations, such as in a remote meeting app (*e.g.*, MeetinVR [95]). They also fuse spatial information about the real environment, using sensor data, to construct an immersive experience for the users. We call this shared model of reality the *shared state*.

In greater detail, this shared state commonly contains a "map" of 3D points (an example is shown in Fig. 5.1). The points in this map are features extracted from camera images (*e.g.*, SIFT [110], ORB [136], and BRISK [81]). Each feature contains an estimate of its 3D position and a descriptor of its visual neighborhood, for use in finding and correlating the same feature in other images. The exact feature description and matching algorithm varies from application to application. The map may contain additional points that are added artificially to "anchor" augmentations. This is needed to give the augmentation the appearance of blending in with the real world. To give augmentations the appearance of blending in with the real world, virtual objects are also described by their 3D coordinates.

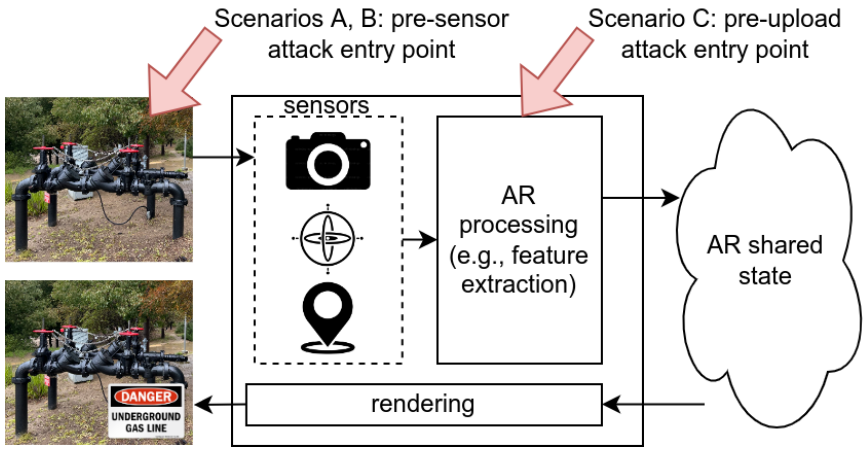


Figure 5.2: AR processing pipeline. AR devices sense the environment, process the sensed data (including feature extraction), and upload information to the shared state. The shared state can return augmentations which are overlaid onto the user's display.

These augmentations may be a virtual character or a simple image to be rendered onto real-world imagery. Thus, the AR shared state is the map of visual features combined with the augmentations placed in the map. Fig. 5.2 shows the processing pipeline of an AR device accessing the shared state in the cloud, starting from sensing the environment, processing the sensor data to extract features or other information, and communicating with the shared state to receive augmentations and finally render them onto the display.

Communication with the shared state. For a user to view or place shared augmentations, communication with the shared state is needed. Abstractly, we can think of viewing or placing the shared augmentations as reading or writing, respectively, from key-value pairs stored in the shared state. The *key* is some piece of information relating to the user's physical location that a user provides (details later), and the *value* is the associated augmentation's coordinates (and optionally its visual appearance). The cloud processes these key-value pairs

and updates the shared state accordingly. There are two operations for users to communicate with shared state: Read and Write, as follows.

- **Read:** A user may read the shared state in order to determine where she is in the map and render the appropriate augmentations. For instance, a user may go to a park where virtual characters are located and upload an image *key* of the park to the cloud, captured from the phone's camera, receive back the *value* of the augmentation's coordinates, then render the virtual characters on the display.

- **Write:** Users may write augmentations at specific locations in the map in the shared state. For instance, a user may place virtual treasure for other users to find in the future as part of an AR scavenger hunt, by uploading a *key* consisting of a short video sequence near the treasure and the associated GPS coordinates, alongside a *value* of the virtual treasure's coordinates.

Regarding keys, there are three main types of sensors commonly used in AR applications to generate appropriate keys to shared state: GPS, camera, and IMU. GPS data provides information about the user's geographical location and typically consists of numerical values representing latitude, longitude, altitude, and time. Camera data in AR applications can take the form of video or a sequence of timestamped images. IMU data refers to the measurements collected by sensors such as accelerometers, gyroscopes, and magnetometers. This data provides information about the device's orientation, acceleration, and rotation. The IMU may not be strictly necessary for these applications to work but is often included to assist in speed and accuracy [67].

	Non-curated	**Curated**
Local	**Scenario A: Cloud Anchor** *Keys*: camera, IMU *Attacks*: Read, Write	**Commercial scenario not found.** *Keys*: camera, IMU *Attacks*: Read
Global	**Scenario C: Mapillary** *Keys*: camera, IMU, GPS *Attacks*: Write	**Scenario B: Geospatial Anchor** *Keys*: camera, IMU, GPS *Attacks*: Read

Table 5.1: Taxonomy of AR shared states.

5.2.2 AR Shared State Taxonomy

We studied the current landscape of multi-player AR and found three major examples of shared state: CloudAnchors [36], and Geospatial Anchors [39], and Mapillary [92], which we primarily focus on in this chapter. Cloud Anchors and Geospatial Anchors are part of Google ARCore, which is Google's AR SDK for Android devices. Mapillary is an crowd-sourced mapping service purchased by Meta in 2020. The design of these frameworks can be dissected along two dimensions: global/local, and curated/non-curated, as summarized in Table 5.1. Next, we describe each of these dimensions.

Global vs. local shared state. AR applications can run in local or global geographic areas; for example, a treasure hunt may take place locally within a building, while Pokemon

Go takes place globally. Consequently they can have larger or smaller maps in their shared state, respectively, which we categorize as local or global shared state. AR frameworks with global shared state tend to utilize GPS coordinates plus camera images as the key to write into the shared state. Specifically, each writer uploads local images tagged with GPS coordinates to the shared state, where all data is merged by the cloud to create the global shared state. Users seeking to read from the shared state may use a combination of GPS, camera, and optionally IMU data as a key into the database. Global shared states tend to be persistent in that they have no clear expiry time, typically persisting for years.

AR frameworks with local shared states are typically smaller in geographic scope and lack global positioning (GPS). The key typically consists of just camera images and optional IMU, without GPS. Local shared states tend to be ephemeral in that they have a configurable lifetime, typically of less than one year [38]. It is possible to combine multiple different local maps into a larger shared map if they overlap significantly [194]. However, for our purposes, we consider local shared states to use one local map per user "reconciling" the maps temporarily to allow augmentations to appear in the same visual location on separate maps.

Curated vs. non-curated shared state. The maps contained in the shared state can be either curated or non-curated. Curated maps are constructed by "high trust" users or "curators". These curators have elevated write permissions to the shared state and usually have an incentive to avoid malicious behavior. Most commonly, these curators are paid employees, contract workers, or trusted research groups. An example is the Street View car [41] where company employees drive a car around and capture camera images to upload to the cloud, which processes them and inserts them into the shared state's map.

Non-curators can still read the curated shared state but cannot otherwise manipulate it.

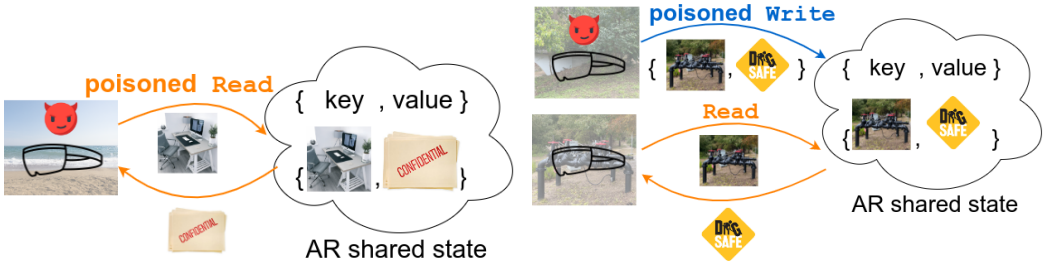
AR frameworks with non-curated shared states allow all users to read and write to the map in the shared state. These users are low trust but come with the advantage of increased numbers, allowing rapid construction and updating of the shared state compared to curators. An example is Mapillary's crowd-sourced street mapping model, where public users can camera images to upload to the cloud, which processes them and inserts them into the map.

The write permissions for these shared state *maps* and the shared state *augmentations* may be separate. For example, a user may be able to add a virtual character to a shared state but not be able to add a new map area of visual features to the shared state. For our purposes a curator has permission to write both shared state map and augmentation data to the shared state while a non-curator can only read map data from the shared state but may be able to both read and write augmentation data. In the future, applications that use more granular permissions may become more common [24].

5.2.3 Threat model

In our threat model, we assume that an attacker engages in AR experiences with shared states using an unmodified AR application. The attacker does not require any specialized software or hardware permissions, and they possess the same read/write permissions as any non-curated user. The primary objective of the attacker is to compromise the integrity or confidentiality of the multi-AR shared state. We identify two types of attacks within this context, as depicted in Fig. 5.3: a *1) Read attack* and *2) Write attack*.

Read Attack. Such an attack focuses on extracting sensitive information stored within the



(a) Read attack (b) Write attack

Figure 5.3: Read and Write attacks on AR shared state. In the read attack an augmentation is read using a poisoned key (false image and/or tagged data). In the write attack an augmentation is written into shared state using a poisoned key. Both allow an attack to be performed from an area other than the key-implied location.

shared state created by other users. In this attack, a victim user has created an augmentation containing sensitive data, which is only supposed to be viewable from her private office, and uploaded it to the shared state. Thus in Fig. 5.3, the shared state contains the {key=office image, value=confidential document} entry. The objective of the attacker is to retrieve and access this private document, thereby breaching confidentiality, by providing a false {key=office image} to retrieve the associated value (the private augmentation).

Write Attack. The attacker aims to manipulate the shared state in order to deceive subsequent victim AR users. To accomplish this, the attacker creates and uploads manipulated images or falsified sensor readings as keys in the shared state. Thus in Fig. 5.3, the shared state contains the {key=pipe image, value="dig safe" sign} entry. Subsequently, when a victim attempts to read from the shared state, they may encounter misleading or false information, leading to inaccurate perceptions or actions within the AR environment. For example in Fig. 5.3, the victim uses a legitimate {key=pipe image} and retrieves an augmentation telling her it is safe to dig there. This attack undermines the integrity and

reliability of the shared state, potentially compromising the experiences and interactions of unsuspecting users.

The fundamental issue with the shared state that enables these attacks is that the ingest pipelines of these AR frameworks accept most keys as inputs, and they do not have any way of verifying that users are uploading legitimate information. Furthermore, even if the attacker fails to generate perfect keys that look exactly like legitimate inputs, the shared state still accepts them because it attributes their imperfections as noise. We speculate that these weaknesses are due to the fledgling nature of multi-user AR frameworks; because AR frameworks want to encourage user participation, they want to make it easy to users to participate in the AR ecosystem, and thus have not yet built in safeguards against the types of attacks we propose and study in this chapter.

Attacker's Goal in Each Scenario. As various multi-AR platforms rely on different combinations of sensor inputs to generate these keys, our investigation focuses on three attack scenarios outlined in Table 5.1. In Scenario A, the attacker's goal is indeed to perform both *Read* and *Write* attacks on the shared state. She aims to read or write augmentations to locations they are not physically present in. By doing so, they deceive other users by providing false or manipulated information. Both camera and IMU data are needed as keys to read or write data from the shared state.

In Scenario B, the attacker's goal is to perform a *Read* attack only. She attempts to read an augmentation from a location where the augmentation does not exist, effectively lying about her location and reaping the benefits. In addition to the camera and IMU data needed as keys in Scenario B, GPS data is additionally needed. We do not investigate *Write* attacks in Scenario B due to the curated shared state. In other words, since threat model

assumes the attacker is an ordinary user, only *Read* attacks can be performed on a curated shared state with the appropriate key. These keys are used by all users freely with no need for special permissions.

Finally, the Scenario C attacker *Writes* augmentations to false locations. This would allow an attacker to manipulate the augmentations other users view, potentially leading to sabotage and safety issues. We do not investigate *Read* attacks in Scenario C because this API does not yet exist in the commercial framework we studied.

5.3 Scenario A: Local (ARCore CloudAnchor)

CloudAnchor refers to anchors that are available on the ARCore Cloud Anchor API [36], allowing users to share experiences within a single app. When these anchors are hosted for a duration exceeding 24 hours, they are referred to as Persistent Cloud Anchors. Using an app that integrates the ARCore Cloud Anchor API, a user (User A) can easily create and host a cloud anchor in a specific location within their environment, such as their office desk. The cloud anchor serves as a reference point in the physical space. Another user (User B), who has been granted access credentials by User A, can then augment and interact with the anchor within the same physical space. This functionality enables multiple users to collaborate and engage in shared AR experiences, providing a platform for interactive and collaborative virtual content within a real-world setting.

However, we have identified a potential attack vector related to this feature. In the following methodology section, the attack and its implications will be thoroughly explained and explored.

5.3.1 Methodology

Imagine a user who wants to host an anchor on their office desk. To do so, they should point their phone to the desk and hover around the desk a few times as usually guided by apps. During this process, the camera extracts a 3D point cloud of the environment, assigns it to the hosted anchor, and sends it to the cloud. Later, if someone wants to resolve this anchor, they should point their phone to that specific desk, allowing the camera to extract 2D features of the environment, and through a 3D-2D feature comparison, if the 3D features of the previously extracted point cloud match the current 2D features, the anchor is returned by the cloud and the user can resolve it successfully. As you may have noted so far, the anchor gets resolved if and only if the 3D-2D feature comparison meets a certain threshold. This means that in order to resolve successfully, normally, the user must be physically present in the same environment where the anchor had been hosted in. Our attack lets a malicious user perform either remote-host or remote-resolve, meaning that there's no need for the malicious user to be physically present in the target environment. Specifically, we show that the attacker can either host or resolve using only a photograph of that environment. By pointing the camera at the photograph, we trick it into extracting the features of the target environment even though we are not actually there. This has serious privacy and security implications as described in the following sections:

Remote-host: An attacker, with access to some photographs, can host anchors in locations that they are not either authorized to enter or to host AR content, e.g., museums, private places, kindergartens, etc. The situation can become more severe if the anchor contains

Figure 5.4: We have remotely hosted an anchor using the photograph of someone's desk.

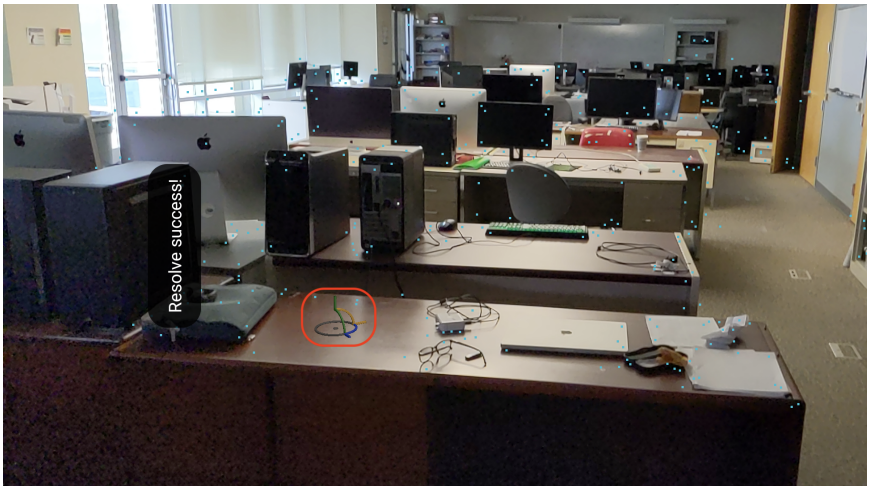


Figure 5.5: We can resolve the remotely-hosted anchor when we are physically present in the actual environment, even in a different lighting condition and also the desk is much more cluttered.

inappropriate or obscene material. As figure 5.4 shows, we display a photograph of someone's desk on a laptop monitor and remotely host an anchor on his desk using the photograph. Even though there are some other objects in the camera's view, like the laptop's keyboard and monitor border, the total extracted features have enough of the features in the photograph, that later when we actually go to the lab and point the phone at the desk, we can successfully resolve the remotely-hosted anchor as figure 5.5 shows.

Remote-resolve: In this scenario, the attacker can resolve the personal private anchors hosted in the environment again by having a photograph of that place. For instance, notes, passwords, or even voices kept as personal anchors of the victim can be resolved by the attacker.

Experiment Setup. We execute the remote-host attack using a photograph of different locations including both indoor and outdoor environments and were able to succeed. In addition, we performed the remote resolve attacks in the same environments with successes also. All of the experiments have been done with a Samsung Galaxy S20 mobile phone to perform hosting and resolving and an Apple MacBook Pro served as the monitor that shows the photograph of the environment.

5.4 Scenario B: Global, Non-curated (Geospatial Anchors)

Geospatial Anchors. After more than fifteen years of collecting street view images, Google has recently introduced the ARCore Geospatial API [39]. This API allows users to attach AR holograms to any location within Google Street View, creating a compelling AR experience on a global scale. In this section, we have demonstrated a practical attack in which the attacker remotely *read* to steal a private hologram *written* by the victim.

Write hologram at physical location

Read hologram at remote location

Figure 5.6: View attack example. An attacker is able to view the hologram without being physically present at the hologram's real-world location.

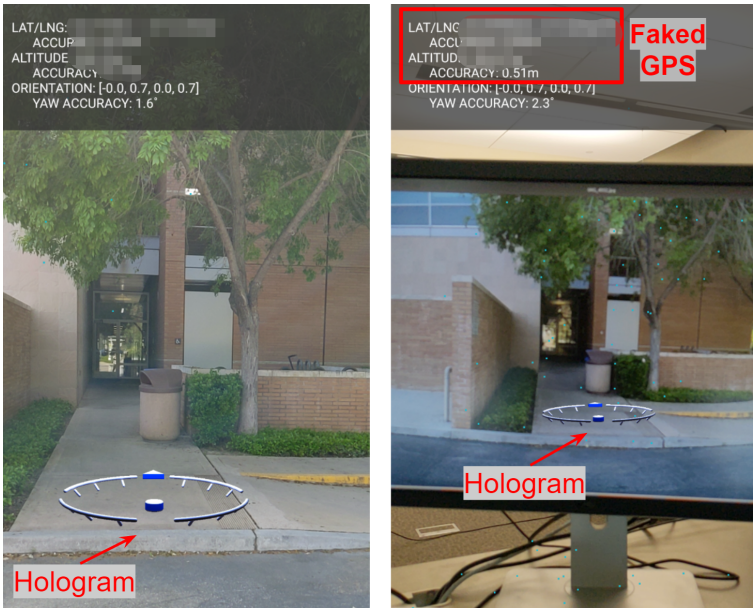


Figure 5.7: Remotely *read* attack example on Geospatial Anchors. The attacker can view the hologram with faked GPS and photograph.

5.4.1 Methodology

The ARCore Geospatial API presents users with the capability to seamlessly integrate holograms into their physical surroundings by leveraging spatial data obtained from Google's Visual Positioning System (VPS) [40]. Through the utilization of computer vision algorithms, the API facilitates the accurate determination of the device's location and orientation, surpassing the capabilities of GPS in isolation. However, this technology also introduces potential security vulnerabilities that can be exploited by malicious actors.

Remote Read Attack. By employing GPS spoofing applications, attackers can remotely *read* holograms by altering the GPS location of the targeted device. By utilizing a GPS emulator, the attacker can redirect the device toward printed photographs to impose the

augmentation of holograms onto the desired location. These photographs can be sourced from various online platforms, including Google Street View [2] or Mapillary [109]. Fig. 5.7 demonstrates the process, illustrating how the attacker successfully manipulates the device's GPS location using a GPS emulator and achieves the remote resolution of holograms onto their monitor.

Additionally, since the Geospatial API grants all users the ability to write anchors at any outdoor location, we have chosen not to consider remote *write* attacks on Geospatial Anchors in this chapter.

5.4.2 Evaluation

In this section, we conduct an evaluation of remotely *read* attacks in global, non-curated scenarios, with a specific focus on the ARCore Geospatial API. The evaluation involves several key steps. To begin, we place 23 geospatial anchors across various locations within our university campus using the Geospatial API. The selection of these anchor locations is carefully designed to encompass a range of environmental differences and varying light conditions. Subsequently, we capture photographs of the areas where the anchors were placed. In the remotely *read* attack, we employ a GPS emulator application[135] to generate fake GPS locations on the Android phones utilized for testing. By manipulating the GPS coordinates, we aim to deceive the Geospatial API into incorrectly placing the geospatial anchors on a monitor that displays the previously captured photographs.

Experiment Setup. We conduct the remotely *read* attack from [0.25, 0.5, 0.75, 1, 1.5, 2] meters away from the monitor, as illustrated in Fig. 5.8. To assess the effectiveness of these attacks, we utilize the attack success rate as the primary metric. Our testing involved two

Figure 5.8: Experiment Setup.

Android phones, namely the Samsung Galaxy S8 and the Samsung Galaxy S21. The former refers to the application utilized for writing geospatial anchors, while the latter refers to the application employed for conducting remote *read* attacks. The *read* and *write* application was developed using Android Studio version 2022.2.1. We utilize the Samsung Galaxy S21 to capture the images. The size of the images is fixed at 3024 x 4032 pixels.

Results. Fig. 5.9 provides our findings derived from the evaluation of remote *read* attacks on geospatial anchors. It is worth noting that when the distance between the attacker's device and the monitor is too close, such as at 0.25 meters, the camera on the phone may struggle to focus properly. This can result in blurred images, making conducting successful remote *read* attacks challenging. Notably, we achieved a 100% success rate for remote *read* attacks conducted at a distance of 0.5 meters. This distance proves to be optimal for the camera on the device to focus properly, resulting in clear and discernible images. However, as the distance between the attacker's device and the monitor increases, the success rate of the remote *read* attacks declines significantly. We speculate that this decline in success rate may be influenced by several factors. Firstly, as the distance increases, the photos displayed on the monitor become smaller, making it more challenging for the Geospatial API to accurately place the geospatial anchors. Additionally, when the camera is positioned at a greater distance from the monitor, there is an increased likelihood of capturing unrelated objects in the field of view. This can significantly impact the success rate of remote *read* attacks on geospatial anchors.

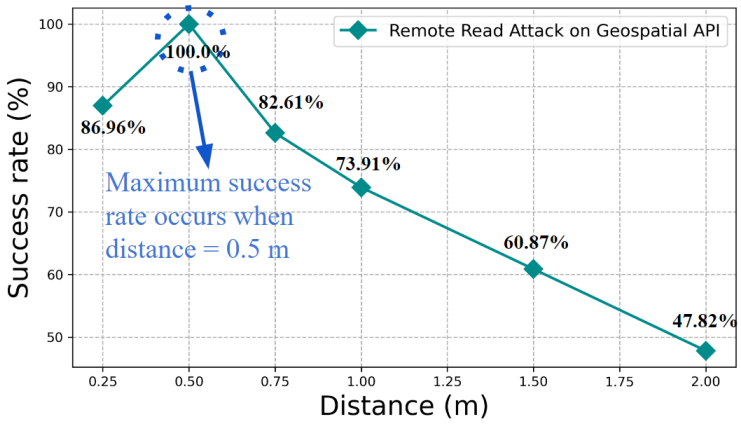


Figure 5.9: Results of Remote *read* and *write* attacks at variant distances.

5.5 Scenario C: Global, crowd-sourced (Mapillary)

While Geospatial app services necessary for Global AR exist in the form of Google geospatial anchors, these services shared states are curated in the sense that only trusted individuals (paid contractors) are able to gather and submit data for building the map. More recent services like Mapillary [92] allow users to use the map but also submit new data in order to expand and update the map. Mapillary is non-curated in that all users can *read* and *write* to the shared state. This includes the raw map data as well as virtual representations of real objects like traffic signs, fire hydrants, and light poles among others.

Non-curated applications that rely on GPS and cameras input like Mapillary provide for novel attack vectors in that an attacker with the least permissions has more capabilities available to them. In this section, we explore several attacks using the web service Mapillary and it's underlying open-source Structure From Motion (SFM) library OpenSFM [93]. We Specifically demonstrate the ability for an attacker to upload imagery

with spoofed Global Positioning (Section 5.5.1) and use this capability to swap the positions of augmented reality objects in the shared map (Section 5.5.1). Finally, we show the ability to add fake objects that should otherwise be real to the shared state map (Section 5.5.1).

All experiments were performed with permission from Mapillary, in a "Geo-fenced" shared state section specifically for experiments where our experiments uploaded data can be segregated from data uploaded and viewed by normal Mapillary users.

5.5.1 Methodology and Evaluation

GPS spoofing, Mapillary

In order to upload street imagery to the Mapillary shared state, a sequence of images each with latitude, longitude, altitude, and time are needed. Image sequences can be as short as three images but are often longer, stretching into the low hundreds of images. All of these data are in the image file in the form of exchangeable image file format (EXIF) [30]. EXIF data are freely manipulable using scripts and an attacker may set the images to have been captured at any arbitrary location and time (within a certain range, one cannot submit imagery captured with timestamps in the far future for example). We demonstrate this in figure 5.10. Two Sequences of five images captured using an iPhone 12 were uploaded with the latitude, longitude, altitude, and time swapped using python scripts. the images were uploaded using Mapillary's own desktop uploader utility [94]. Mapillary allows these sequences to be uploaded, processed, and displayed at the swapped locations for any user to view.

We repeated this experiment a total of 8 times for 15 total image sequences with false GPS data (one sequence was a duplicate, not a swap). These swaps occurred using

Figure 5.10: Two uploaded Image Sequences with their coordinates swapped shown on the Mapillary map.

imagery captured outdoors within a square kilometer Geo-fenced area. The images were taken at several times ranging form early morning to early evening and facing many directions at different locations. Images included street imagery with roads and grass fields with no roads. All experiments succeeded to upload, process, and display the spoofed imagery.

OpenSFM AR Swap

With the ability to add images to any position in the shared state, it follows that AR objects added using these images may also have their positions manipulated. Figure 5.11 shows the core mechanics of the AR swap attack. The AR objects or augmentations are placed relative to map features gathered from images and the map features are merged in 3D space based on GPS coordinates. With simple spoofed GPS coordinates, we can move the map features, poisoning the shared state, and the augmentations that were placed relative to them also move to potentially disastrous results.

To demonstrate this, we utilise Mapillary's underlying OpenSFM along with some

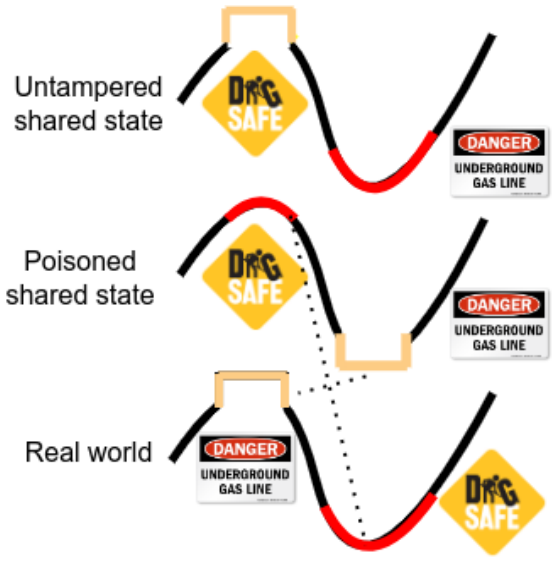


Figure 5.11: AR GPS swap attack mechanics. Swapped visual feature locations (red curve and orange square) in the poisoned shared state cause the victim to view a "dig safe" hologram instead of "danger" in the real world.

of our own additional programs to construct a simple AR application. OpenSFM takes as input a sequence of Images and outputs a map of 3D features, in the form of a JSON file. Figure **??** shows the flow of the AR application from image sequence to augmentation. First the initial maps are generated using OpenSFM, then the maps have augmentations placed in them relative to the first camera pose. The maps from each image sequence contain no global positioning information so this is added in order to facilitate the merging of the maps into a shared state. Finally, new images and camera poses (resolved using OpenSFM) use the shared state map to render nearby augmentations into the new images.

For an example we use construction and caution signs as augmentations. This is to demonstrate a possible AR application to mark areas as safe or dangerous to dig in. Tampering with the signs in this hypothetical could lead to confusion, damaged property, and potential harm.

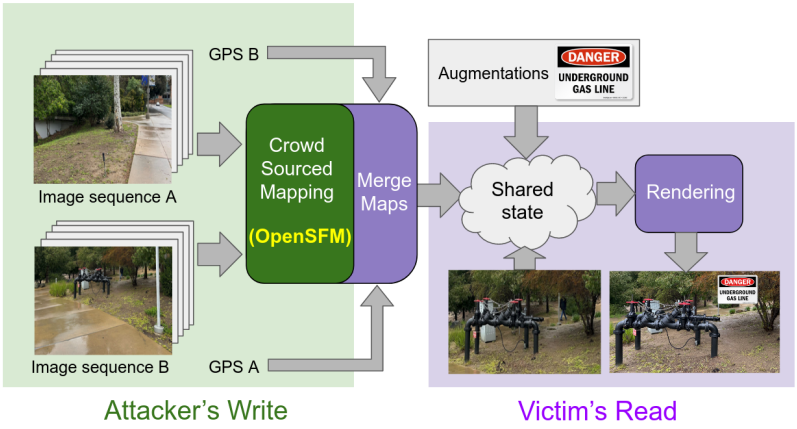


Figure 5.12: Overview of Global AR system (portions in purple are our own programs added to Mapillary's). The swap attack is performed by simply swapping GPS A and B.

The impact of this system with spoofed GPS is shown in the evaluation figure 5.13. To evaluate we used two image sequences with five images each to construct the shared state map with one additional image per sequence reserved. These sequences were captured approximately 100 meters apart and this distance was used to merge the maps generated by OpenSFM. Augmentations were placed 5 meters in front of the initial image in the sequence. Two shared states were created through merging, one un-tampered and one with the GPS swapped causing the merged maps to swap places in the shared state. the reserved images had their camera poses resolved by OpenSFM and the augmentations within view were rendered after being rotated to face the camera, one augmented image per sequence to obtain the final augmentations.

Additional shared state AR components for augmentation, merging, and rendering were created in the form of python 3.9.13 [122] scripts using numpy 1.21.5 [113].

(a) An AR safe to Dig sign in a park

(b) An AR NOT safe to dig sign near a pipe

(c) An AR NOT safe to dig sign in a park

(d) An AR safe to dig sign near a pipe

Figure 5.13: AR swap via GPS spoofing.

Object detection

Mapillary regularly performs object detection on images uploaded to their service [91]. When an image sequence is uploaded these detected objects are then added to the shared state map at the positions they were detected. This presents attackers with the ability to add fake real-world objects to the shared state. Such an attack can be performed by overlaying an image of an object of interest, such as a stop sign or other traffic sign, onto otherwise empty street imagery. Figure 5.14 shows a fake stop-sign taken from street imagery cropped-out and overlaid on a new image captured with a mobile device and uploaded to Mapillary. A sequence of three manipulated images were needed to successfully add the fake stop-sign to

the Mapillary shared state. Similar fake objects could cause issues for navigation systems or AR applications. For example an AR app that tries to find the nearest trash receptacle could be manipulated to point to nothing as a fake receptacle had been added to the shared state.

5.6 Shared State Attack Mitigation

The question at issue for these AR attacks is how to establish the real location (or existence) of a device or object. All of these attacks "lie" about their location in order to read or write data maliciously. With this in mind, we discuss potential mitigation strategies.

Ground-up Application Design Perhaps the most straight-forward solution is to make sure that the core design of these applications use more traditional security measures to prevent tampering. Non-curated shared states may be updated to be curated with a permissions system where only trusted users may perform writes. This may be done to turn non-curated applications into curated applications (similar to [24]) but for those applications where non-curated-ness is desirable a compromise involving a user reputation system based on past good behavior may prove sufficient. Even in non-curated applications, only accepting appropriately watermarked uploaded imagery [86] may be a useful layer to prevent image tampering and to verify the source of the imagery. GPS information may also be encoded into these water marks to make manipulation more difficult than plain EXIF data.

Real Space Security Read attacks like in Scenarios A and B, allow a user to resolve augmentations using images captured from some location and re-used somewhere else. For some locations it may be possible to simply enforce non-entry and photography restrictions

(a) Real world ground truth

(b) Tampered image with fake stop sign inserted

Figure 5.14: Tampered images written into AR shared state. A fake stop-sign has been photo-shopped into a sequence of images in order to trick an object recognition system. The fake stop-sign is added to the shared state.

(Gates, ID badges, ect.). Using markers placed in the space that are used to resolve augmentations (marker-based AR) may also provide a guarantee of locality [79] if the attacker has no way of knowing the markers configuration from outside the location.

Local Moderators As these applications may be generally be considered content hosting services, human moderators may be used to great effect as in other successful applications like YouTube and Facebook. While one of the most powerful mitigation strategies, moderator teams are expensive and come with the additional hurdle of needing to be located close to the locations of the uploaded imagery to verify their veracity. This may, additionally, be undesirable for non-curated applications.

Automated Mitigation Some attacks may be made for more difficult through automated means. Checking for duplicate imagery or manipulated images have a long history in computer security [11, 187] and may be deployed to great effect in global scenarios to catch sloppy attacks. AI depth segmentation [104] could potentially be used to check for photographs or screens used to present images as in Scenarios A and B. The planes that contain these false image could be ignored but this is expensive and untested. Depth cameras could provide the needed information to notice when features are coming from a flat plane and not the varying depths of a real location as in figure 5.15. Computer Networks can be used to establish a users rough device location [120, 139] either by cell tower or Building WiFi Signal.

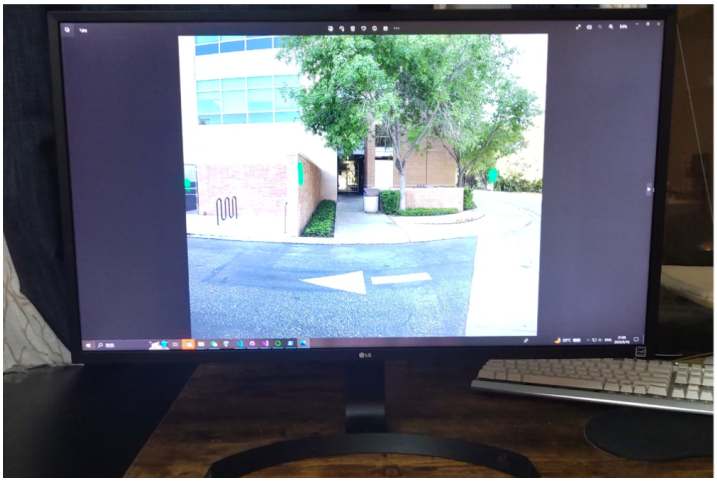
Collaboration The vulnerability of global map data to general inconsistencies, errors, attacks, and the need to constantly update with the world create a massive quality assurance problem. Some efforts like the Overture maps foundation [117] pool resources, expertise, and

map data from many organizations. Of particular interest to these attacks is the ability for one compromised shared state to check itself against another map that covers the same area. Disagreements in map data may be due to time, errors, or malicious actions. While major corporations like Meta and Microsoft can collaborate together, time will tell if the global size of the problem and its potential benefits warrants collaboration on a nation-state level.

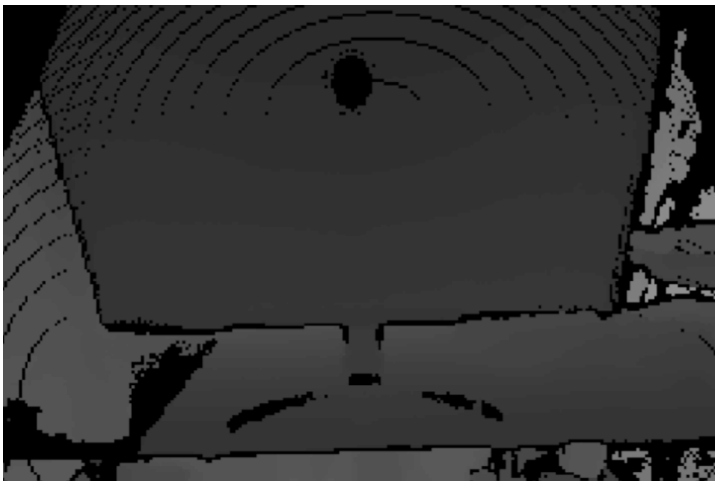
Other Available Sensors With the increasing integration of sensors on AR/VR devices, the vulnerability of share-state attacks can be mitigated by assessing the coherence between the share-state and other accessible sensor data. For instance, the Microsoft Hololens 2 incorporates not only RGB cameras but also a depth camera [164]. As illustrated in Fig.5.15, we can leverage the depth camera to detect the presence of a monitor or a photo. Subsequently, a comparison can be made with the output of the RGB cameras. If any inconsistencies are identified between the two camera types, the read or write operation can be rejected as a precautionary measure.

5.7 Related Work

AR/VR Security and Privacy Overviews VR/AR have taken off in recent years but research into potential security and privacy issues has pre existed popular adoption [133]. Recent overview [132] and literature review [28] broadly cover the existing issues. Literature covering the specific cases of Multi-User AR also exist [79] where our work squarely lies within. The global scenarios also intersect with Geospatial information services security covered in [13].



(a) Hololens 2 RGB camera



(b) Hololens 2 depth camera

Figure 5.15: Mitigation via other sensors on Microsoft Hololens 2. Depth sensors show the screen as flat and lacking detail of an image captured of the real location.

AR/VR Threat Mitigations As these technologies are newly reaching approaching adoption, the threats to them are new and even more so the mitigations. Mitigation of manipulation of the input data from users such as image manipulation [188, 10] have become sophisticated in recent years. GPS spoofing mitigations [64] focus on real-time mitigation but apps like Mapillary seek to offer the ability to upload batched imagery at later times for user convenience and mitigation for this arrangement may need additional attention. The most effective mitigations are likely to come in the form of permissions systems like in [24] but these will require non-curated shared states to become curated. Care must be taken to prevent permissions systems from being *location* based (as shown in this chapter such permissions could be "stolen" by spoofing location data) and to instead use more traditional authentication methods like passwords.

AR Leakage Vectors As the adoption of augmented reality (AR) and virtual reality (VR) devices becomes increasingly widespread in various facets of individuals' daily lives, a plethora of prior research papers [145, 102, 179, 161, 88, 152, 182, 6, 83, 71, 160, 48] have highlighted the issue of unauthorized acquisition of sensitive information from AR/VR devices, exploiting both `software` and `physical` leakage vectors. In the software-based approach, certain studies [145, 102] demonstrate the feasibility of inferring the user's location by analyzing network traffic information. Other investigations [179, 161] showcase the extraction of sensitive data from VR network traces. More recent works [88, 152, 160] establish the ability to deduce keystrokes based on the user's head motions. Additionally, Zhang et al. [182] explore the utilization of rendering performance counters to execute side-channel attacks on AR/VR systems. In the realm of physical vectors, Arafat et al. [6] employs WiFi CSI side-channel information leakage to infer keystrokes. Furthermore, a

cluster of studies [83, 71, 48] prove that attackers can exploit vision and sensor-based side-channel leakages to exfiltrate sensitive information

Computer Vision Attacks AR uses computer vision techniques as part of its foundation and attacks on Computer vision systems can apply to the AR systems that depend on them as shown in section 5.5.1. Previous work on attacks on computer vision object detectors is wide reaching from attacks on machine learning models[60, 180] to on-board vehicles [191]. While our work uses photographs, screens, manipulated images and gps to trick Computer vision systems, attacks using additional hardware like lasers have been explored [178]. Simultaneous Localization and Mapping (SLAM) attacks exist [62, 147] and also would impact AR systems like CloudAnchor that depend on these techniques to function. Our work takes inspiration from these to show computer vision attacks can cascade into interesting behaviors in AR systems

Sensor Spoofing and Confusion Our work uses GPS spoofing by simply altering data with freely available mobile applications or our own programs. While not necessary for our attacks, more sophisticated GPS spoofing has existed for nearly as long as the technology has reached wide-spread use [159]. Tricking sensors such as the IMU was not done in this chapter but is possible with acoustic waves [65, 140] and can be used to assist in attacking computer vision systems that use this sensor to improve accuracy, or stabilize camera imagery.

5.8 Shared-State Attack Conclusion

As these AR applications become more ubiquitous, there is a growing for additional research into security and privacy risks unique to AR. This paper introduced and explored attacks on multiple shared-state augmented reality applications and frameworks. Specifically we show

that using GPS and Camera imagery are not sufficient to establish the location of a device or the objects that are visible to that device without additional steps to prevent tampering. We formed a threat model that can apply to many scenarios and demonstrate them on current systems. We show that these attacks can be performed in a variety of environments successfully. Simple mitigation like duplication detection and image manipulation detection can and have been implemented but in the future, further work on mitigation strategies like map merging policies and fraud detection is paramount.

Chapter Acknowledgements

We would like to acknowledge the engineers at Mapillary that helped set-up test areas and provide feedback after the experiments were completed on future work. Special thanks to Yicheng Zhang, Erfan Shayegani, and Pedram Zaree for agreeing to help extend this chapter in the future for publication. Portions of this Chapter are under review to be published in USENIX 2024. This work was partially supported by the NSF grants CNS-1942700, CNS-2053383, CCF-2212426, and a Meta faculty research award.

Chapter 6

Conclusion

We conclude this dissertation with a summary of contributions and a discussion of future work.

6.1 Contribution Summary

6.1.1 AR accuracy evaluation

As the AR applications continue to move towards marker-less, simultaneous localization and mapping solutions, augmentations are placed relative to tracked visual features that are detected and tracked on the fly. These methods inherit core computer vision issues like imperfect feature correlation and tracking that may cause localization inaccuracy, which is inherited by the augmentations causing them to drift across the images as viewed from different angles or points in time. Augmented reality adds extra steps after this for updating augmentations and rendering to the display that may cause additional drift or jitter. There is a need for tools to evaluate the drift or spatial inconsistency that are cheap and simple to utilize.

We contribute RealityCheck, a system that uses more accurate, marker-based computer vision techniques to check the accuracy of less accurate but more flexible marker-less systems with no additional hardware needed beyond the application device and a personal computer. Reality check is accurate to human-eye evaluation to within 1.5cm on average. This tool was released to the public for use in evaluating the spatial consistency of Augmented Reality applications in order to improve their Quality of Experience.

6.1.2 Web-based XR application latency

Deploying XR experiences to many devices is a problem where one current solution is to make these experiences for web-browsers. Due to their ubiquity, Web browser based XR has the potential to reach the widest array of users. We observed that these experiences can take minutes to render the first frame of video to the device display as all assets must be downloaded over a (typically wireless) network, significantly hampering Quality of Experience. We identify multiple causes of undue delay in the time to first frame and address them in VIA: Visibility Aware WebXR. We divide the XR scene into smaller objects and prioritize them using a combination of heuristics proven to minimize latency. VIA shows latency improvements depending on scene and initial viewpoint of up to 50 percent and maintains improvements over the control even if the user rotates their head during download by up to 90 degrees in any direction.

6.1.3 Head Tracking Typing Inference.

When typing text using a head mounted XR display, we tend to look at the keyboard as we type. These HMDs track the device pose and allow any application to record this data. To show the potential security and privacy risks, we developed a typing test VR application to

perform a user study, prompting the users with common english words and recorded the head pose data. We show that using "off the shelf" machine learning methods can achieve an over 50% accuracy in inferring common English words from over 20 participants using this pose data.

6.2 AR application shared-state attacks

Multi user AR applications use a shared map of the real world constructed from Images and position data. Augmentations placed into this map are dependent on the underlying imagery to be able to be rendered at the correct locations. We show that it is not only possible for an attacker to read augmentations in from areas they were not written to, but also that an attacker can write map data, augmentations, and fake real-world objects to false locations using only the inputs accessible to a normal user. We demonstrate these attacks on three different application frameworks, namely Google ARCore and Geopspatial anchors, as well as Meta's Mapillary service.

6.3 Future work

Quality of Experience is a wide category of improvements that may include latency and accuracy improvements as in this paper but also network usage, device power usage, and memory usage. Mobile devices provide restrictions on available resources that often need special solutions different from higher powered devices, Many Computer vision algorithms are iterative or can scale down in the event of a time constraint which means mobile device solutions will always be less accurate and lower quality but, accepting this, techniques to improve these results are still being developed with surprising results.

Future improvements to shared-state consistency across devices is on the horizon as when multiple users attempt to add or manipulate augmentations, conflicts inevitably arise. Strategies for updating the shared state using lockstep "wait until all the network information has arrived every frame" solutions are vulnerable to pauses, jitter, and "lag" as they wait to update the display. Investigating methods to allow augmentations to render freely and resolve conflicts as they arise may take inspiration from networked games and the "optimistic" solutions that have developed in the last decade.

Security and privacy research is a continuous "arms race" of threats and mitigation in XR as in many other fields. However, the unique sensor combinations and human-centered displays provide unique angles of attack which prove to set the research on its own path.

The ever increasing amount and variety of sensors on mobile XR devices provide rich data for inferring private information. With Engineers becoming ever-more aware of dangerous side-channels left open to malicious actors, some remain under-investigated. Computer Network packets sent from and to devices may be encrypted but follow patterns in size over time that may betray information about the underlying application and activity [182]. Images with lossy compression, even when encrypted, maintain their compressed size and may warrant investigation into what information can be gleamed from the time series of image sizes.

Merging XR map data securely provides an interesting Byzantine Generals problem [74]. How do we find consensus between many maps, some of which may be incorrect or malicious? Establishing that two map segments "agree" with one another is not straight forward computer vision-wise and assuring that attackers or "bad" maps do not outnumber good ones is open to investigation. An additional wrinkle is that long term maps must

update as the real world adds more buildings, new roads, and removes landmarks. How do we treat these updates when they disagree with old shared-states?

XR research has reached a new wave with the advent of affordable, high-quality XR devices. New problems and solutions are being discovered every year towards improving human communication and collaboration in XR.

Bibliography

[1] D. F. Abawi, J. Bienwald, and R. Dorner. "Accuracy in optical tracking with fiducial markers: an accuracy function for ARToolKit". In: *IEEE ISMAR*. 2004.

[2] Dragomir Anguelov et al. "Google street view: Capturing the world at street level". In: *Computer* 43.6 (2010), pp. 32–38.

[3] Kittipat Apicharttrisorn et al. "Characterization of multi-user augmented reality over cellular networks". In: *Proc. IEEE SECON*. 2020.

[4] Apple. *ARKit - Apple Developer*. https://developer.apple.com/arkit/.

[5] Abdullah Al Arafat, Zhishan Guo, and Amro Awad. "VR-Spy: A Side-Channel Attack on Virtual Key-Logging in VR Headsets". In: *IEEE Virtual Reality and 3D User Interfaces (VR)*. 2021.

[6] Abdullah Al Arafat, Zhishan Guo, and Amro Awad. "VR-Spy: A Side-Channel Attack on Virtual Key-Logging in VR Headsets". In: *IEEE Virtual Reality and 3D User Interfaces (VR)*. 2021.

[7] Dmitri Asonov and Rakesh Agrawal. "Keyboard acoustic emanations". In: *IEEE Symposium on Security and Privacy*. 2004.

[8] Ulf Assarsson and Tomas Moller. "Optimized View Frustum Culling Algorithms for Bounding Boxes". In: *Journal of Graphics Tools* 5 (July 2000). DOI: 10.1080/10867651.2000.10487517.

[9] Andrea Barisani and Daniele Bianco. "Sniffing keystrokes with lasers/voltmeters". In: *Proceedings of Black Hat USA* (2009).

[10] Sevinc Bayram et al. "Image manipulation detection". In: *Journal of Electronic Imaging* 15.4 (2006), p. 041102. DOI: 10.1117/1.2401138. URL: https://doi.org/10.1117/1.2401138.

[11] Sevinç Bayram et al. "Image manipulation detection". In: *Journal of Electronic Imaging* 15.4 (2006), pp. 041102–041102.

[12] Donald J Berndt and James Clifford. "Using dynamic time warping to find patterns in time series". In: *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 1994.

[13] Elisa Bertino et al. "Security and Privacy for Geospatial Data: Concepts and Research Directions". In: SPRINGL '08. Irvine, California: Association for Computing Machinery, 2008, pp. 6–19. ISBN: 9781605583242. DOI: 10.1145/1503402.1503406. URL: https://doi.org/10.1145/1503402.1503406.

[14] Blender. *Blender Overview*. https://www.blender.org/.

[15] Kevin Boos, David Chu, and Eduardo Cuervo. "Flashback: Immersive virtual reality on mobile devices via rendering memoization". In: *ACM MobiSys* (2016).

[16] F. Bork et al. "Empirical Study of Non-Reversing Magic Mirrors for Augmented Reality Anatomy Learning". In: *IEEE ISMAR*. 2017.

[17] Caridad F Brito. "Demonstrating experimenter and participant bias." In: *Activities for teaching statistics and research methods: A guide for psychology instructors* (2017).

[18] Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* 35.10 (2016), pp. 1157–1163.

[19] Michael Butkiewicz et al. "Klotski: Reprioritizing web content to improve user experience on mobile devices". In: *USENIX NSDI*. 2015, pp. 439–453.

[20] T.P. Caudell and D.W. Mizell. "Augmented reality: an application of heads-up display technology to manual manufacturing processes". In: *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*. Vol. ii. 1992, 659–669 vol.2. DOI: 10.1109/HICSS.1992.183317.

[21] Cesium. *Cesium 3D Tileset*. 2017. URL: https://github.com/CesiumGS/3d-tiles.

[22] Kaiming Cheng et al. "Exploring User Reactions and Mental Models Towards Perceptual Manipulation Attacks in Mixed Reality". In: *USENIX Security*. 2023.

[23] *City Grid Block*. https://sketchfab.com/3d-models/city-grid-block-3488e40ceca846bb9023f894a749c398. Accessed: 2021-06-27. 2021.

[24] Luis Claramunt et al. "SpaceMediator: Leveraging Authorization Policies to Prevent Spatial and Privacy Attacks in Mobile Augmented Reality". In: (2023).

[25] CNBC. *Coronavirus could be catalyst to reinvigorate virtual reality headsets*. https://www.cnbc.com/2020/05/02/coronavirus-could-be-catalyst-to-reinvigorate-virtual-reality-headsets.html. 2020.

[26] Xavier Corbillon et al. "Viewport-adaptive navigable 360-degree video delivery". In: *IEEE ICC*. IEEE. 2017, pp. 1–7.

[27] Mark Davies. "The Corpus of Contemporary American English as the first reliable monitor corpus of English". In: *Literary and linguistic computing* 25.4 (2010), pp. 447–464.

[28] Jaybie A. De Guzman, Kanchana Thilakarathna, and Aruna Seneviratne. "Security and Privacy Approaches in Mixed Reality: A Literature Survey". In: *ACM Comput. Surv.* 52.6 (Oct. 2019). ISSN: 0360-0300. DOI: 10.1145/3359626. URL: https://doi.org/10.1145/3359626.

[29] Angus Dempster, François Petitjean, and Geoffrey Webb. "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels". In: *Data Mining and Knowledge Discovery* 34 (Sept. 2020). DOI: 10.1007/s10618-020-00701-z.

[30] JEITA (Japan Electronics and Information Technology Industries Association). *EXIF data format*. https://www.loc.gov/preservation/digital/formats/fdd/fdd000146.shtml.

[31] F. Faion et al. “Camera- and IMU-based pose tracking for augmented reality”. In: *IEEE International Conference on Multisensor Fusion and Integration*. 2016.

[32] *Forest Loner*. https://sketchfab.com/3d-models/forest-loner. Accessed: 2021-06-27. 2021.

[33] Thomas Forgione et al. “DASH for 3D networked virtual environment”. In: *ACM Multimedia*. 2018, pp. 1910–1918.

[34] Rafael Medina-Carnicer Francisco J.Romero-Ramirez Rafael Muñoz-Salinas. “Speeded up detection of squared fiducial markers”. In: *Image and Vision Computing*. Vol. 76. 2018, pp. 38–47.

[35] *Future House*. https://xeogl.org/examples/#importing_gltf_BranchHouse. Accessed: 2021-06-27. 2016.

[36] Google. *ARCore Cloud Anchor API*. https://developers.google.com/ar/develop/cloud-anchors/management-api.

[37] Google. *ARCore Overview*. https://developers.google.com/ar/discover/.

[38] Google. *Cloud Anchor Dev Guide*. https://developers.google.com/ar/develop/unity-arf/cloud-anchors/developer-guide-android.

[39] Google. *Google ARCore Geospatial API*. https://developers.google.com/ar/develop/geospatial.

[40] Google. *Google's Visual Positioning System (VPS)*. https://ai.googleblog.com/2019/02/using-global-localization-to-improve.html.

[41] Google. *How Street View works and where we will collect images next*. https://www.google.com/streetview/how-it-works/.

[42] Google. *Logcat, Android Developers*. http://android-doc.github.io/tools/help/logcat.html. 2022.

[43] Google. *SnapChat Lenses*. https://www.snapchat.com/.

[44] Google. *Tensorflow releases*. https://github.com/tensorflow/tensorflow/releases. 2022.

[45] Google Creative Labs. *Just a Line - Draw Anywhere, with AR*. https://justaline.withgoogle.com/.

[46] Google Developers. *Get Ready for Priority Hints*. https://developers.google.com/web/updates/2019/02/priority-hints. 2019.

[47] *Google Lighthouse*. https://developers.google.com/web/tools/lighthouse. 2021.

[48] Sindhu Reddy Kalathur Gopal et al. “Hidden Reality: Caution, Your Hand Gesture Inputs in the Immersive Virtual World are Visible to All!” In: ().

[49] Gautam Goswami. *Council Post: Augmented Reality's Applications And Future In Business*. https://www.forbes.com/sites/forbescommunicationscouncil/2020/10/15/augmented-realitys-applications-and-future-in-business, year=2020.

[50] Daniel Gruss et al. “Flush+ Flush: a fast and stealthy cache attack”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2016.

[51] Yu Guan et al. “Pano: Optimizing 360 video streaming with a better understanding of quality perception”. In: *ACM SIGCOMM*. 2019, pp. 394–407.

[52] Tzipora Halevi and Nitesh Saxena. “Keyboard acoustic side channel attacks: exploring realistic and security-sensitive scenarios”. In: *International Journal of Information Security* 14.5 (2015), pp. 443–456.

[53] Bo Han, Yu Liu, and Feng Qian. “ViVo: Visibility-aware mobile volumetric video streaming”. In: *ACM MobiCom*. 2020, pp. 1–13.

[54] John Paulin Hansen et al. “Gaze typing compared with input by head and hand”. In: *ACM Symposium on Eye tracking research & applications*. 2004.

[55] Richard L Holloway. “Registration error analysis for augmented reality”. In: *Presence: Teleoperators & Virtual Environments* 6.4 (1997), pp. 413–432.

[56] Hristina Hristova et al. “3CPS: a novel supercompression for the delivery of 3D object textures”. In: *ACM MMSys*. 2020, pp. 66–76.

[57] S-Y Hu et al. “Flod: A framework for peer-to-peer 3d streaming”. In: *IEEE INFOCOM*. IEEE. 2008, pp. 1373–1381.

[58] Xing Hu et al. “Deepsniffer: A dnn model extraction framework based on learning architectural hints”. In: *ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2020.

[59] Yonghao Hu et al. “WebTorrent Based Fine-Grained P2P Transmission of Large-Scale WebVR Indoor Scenes”. In: *ACM Web3D*. 2017.

[60] Lifeng Huang et al. “Universal Physical Camouflage Attacks on Object Detectors”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[61] IKEA. *IKEA Place*. https://apps.apple.com/us/app/ikea-place/id1279244498.

[62] Muhammad Haris Ikram et al. “Perceptual Aliasing++: Adversarial Attack for Visual SLAM Front-End and Back-End”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4670–4677. DOI: 10.1109/LRA.2022.3150031.

[63] InShot Inc. *XRecorder*. https://play.google.com/store/apps/details?id=videoeditor.videorecorder.screenrecorder&hl=en_US&gl=US.

[64] Ali Jafarnia-Jahromi et al. “GPS vulnerability to spoofing threats and a review of antispoofing techniques”. In: *International Journal of Navigation and Observation* 2012 (2012).

[65] Xiaoyu Ji et al. “Poltergeist: Acoustic Adversarial Machine Learning against Cameras and Computer Vision”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 160–175. DOI: 10.1109/SP40001.2021.00091.

[66] Li Jinyu et al. “Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality”. In: *Virtual Reality & Intelligent Hardware* 1.4 (2019), pp. 386–410.

[67] Li Jinyu et al. "Survey and evaluation of monocular visual-inertial SLAM algorithms for augmented reality". In: *Virtual Reality & Intelligent Hardware* 1.4 (2019), pp. 386–410. ISSN: 2096-5796. DOI: https://doi.org/10.1016/j.vrih.2019.07.002. URL: https://www.sciencedirect.com/science/article/pii/S209657961930052X.

[68] Kasirat Turfi Kasfi, Andrew Hellicar, and Ashfaqur Rahman. "Convolutional Neural Network for Time Series Cattle Behaviour Classification". In: *ACM Workshop on Time Series Analytics and Applications*. 2016.

[69] Eamonn Keogh et al. "Segmenting time series: A survey and novel approach". In: *Data mining in time series databases*. World Scientific, 2004, pp. 1–21.

[70] Kickstarter. *Oculus Kickstarter*. https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game. 2015.

[71] Tadayoshi Kohno et al. *Display leakage and transparent wearable displays: Investigation of risk, root causes, and defenses (technical report)*. Tech. rep. Microsoft Research, 2015.

[72] Manu Kumar et al. "Reducing Shoulder-Surfing by Using Gaze-Based Password Entry". In: *USENIX Symposium on Usable Privacy and Security (SOUPS)*. 2007.

[73] J. Laaksonen and E. Oja. "Classification with learning k-nearest neighbors". In: *International Conference on Neural Networks (ICNN)*. 1996.

[74] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925. DOI: 10.1145/357172.357176. URL: https://doi.org/10.1145/357172.357176.

[75] Steven LaValle. "Virtual reality". In: *Cambridge University Press* (2016).

[76] Steven M. LaValle et al. "Head tracking for the Oculus Rift". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[77] Guillaume Lavoué, Laurent Chevalier, and Florent Dupont. "Streaming Compressed 3D Data on the Web Using JavaScript and WebGL". In: *ACM Web3D*. 2013.

[78] Kiron Lebeck et al. "Securing Augmented Reality Output". In: *IEEE Symposium on Security and Privacy (SP)*. 2017.

[79] Kiron Lebeck et al. "Towards Security and Privacy for Multi-user Augmented Reality: Foundations with End Users". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 392–408. DOI: 10.1109/SP.2018.00051.

[80] S. M. Lehman, H. Ling, and C. C. Tan. "ARCHIE: A User-Focused Framework for Testing Augmented Reality Applications in the Wild". In: *IEEE Virtual Reality*. 2020.

[81] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. "BRISK: Binary Robust invariant scalable keypoints". In: *2011 International Conference on Computer Vision*. 2011, pp. 2548–2555. DOI: 10.1109/ICCV.2011.6126542.

[82] Max Limper et al. "Fast delivery of 3D Web content: A case study". In: June 2013, pp. 11–17.

[83] Zhen Ling et al. "I know what you enter on gear vr". In: *IEEE Conference on Communications and Network Security (CNS)*. 2019.

[84] Chien-Liang Liu, Wen-Hoar Hsaio, and Yao-Chung Tu. "Time Series Classification With Multivariate Convolutional Neural Network". In: *IEEE Transactions on Industrial Electronics* 66.6 (2019), pp. 4788–4797. DOI: 10.1109/TIE.2018.2864702.

[85] Xiangyu Liu et al. "When good becomes evil: Keystroke inference with smartwatch". In: *ACM Conference on Computer and Communications Security (CCS)*. 2015.

[86] Jan Lukas, Jessica Fridrich, and Miroslav Goljan. "Digital camera identification from sensor pattern noise". In: *IEEE Transactions on Information Forensics and Security* 1.2 (2006), pp. 205–214.

[87] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. "HoloLogger: Keystroke Inference on Mixed Reality Head Mounted Displays". In: *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2022.

[88] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. "HoloLogger: Keystroke Inference on Mixed Reality Head Mounted Displays". In: *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2022.

[89] B. MacIntyre, E. M. Coelho, and S. J. Julier. "Estimating and adapting to registration errors in augmented reality systems". In: *IEEE Virtual Reality*. 2002.

[90] Anindya Maiti et al. "Smartwatch-based keystroke inference attacks and context-aware protection mechanisms". In: *ACM Asia Conference on Computer and Communications Security (ASIACCS)*. 2016.

[91] Mapillary. *Mapillary Object Detection*. https://help.mapillary.com/hc/en-us/articles/115000967191-Object-detections.

[92] Mapillary. *Mapillary: make better maps*. https://www.mapillary.com/.

[93] Mapillary. *OpenSFM: open source structure-from-motion*. https://github.com/mapillary/OpenSfM.

[94] Mapillary. *OpenSFM: open source structure-from-motion*. https://www.mapillary.com/desktop-uploader.

[95] MeetinVR. *MeetinVR*. https://www.meetinvr.com/.

[96] Mehrube Mehrubeoglu and Vuong Nguyen. "Real-time eye tracking for password authentication". In: *IEEE International Conference on Consumer Electronics (ICCE)*. 2018.

[97] Meta. *Accessibility Improvements, and Air Link for Quest 1 in the Latest Oculus Software Update*. https://www.oculus.com/blog/multitasking-accessibility-improvements-and-air-link-for-quest-1-in-the-latest-oculus-software-update/. 2021.

[98] Meta. *Education — Oculus*. https://www.oculus.com/experiences/go/section/161391067688077/.

[99] Meta. *Latest Oculus Quest Update Fosters Developer Creativity With App Lab and Connects People With Messenger*. https://www.oculus.com/blog/latest-oculus-quest-update-fosters-developer-creativity-with-app-lab-and-connects-people-with-messenger/. 2021.

[100] Meta. *Workrooms — VR for Business Meetings*. https://www.oculus.com/workrooms/.

[101] Ülkü Meteriz-Yıldıran et al. "A Keylogging Inference Attack on Air-Tapping Keyboards in Virtual Environments". In: *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2022.

[102] Gabriel Meyer-Lee, Jiacheng Shang, and Jie Wu. "Location-leaking through network traffic in mobile augmented reality applications". In: *IEEE International Performance Computing and Communications Conference (IPCCC)*. 2018.

[103] Microsoft. *Microsoft Hololens 2*. https://www.microsoft.com/en-us/hololens/buy.

[104] Yue Ming et al. "Deep learning for monocular depth estimation: A review". In: *Neurocomputing* 438 (2021), pp. 14–33. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.12.089. URL: https://www.sciencedirect.com/science/article/pii/S0925231220320014.

[105] Hoda Naghibijouybari et al. "Rendered insecure: Gpu side channel attacks are practical". In: *ACM conference on computer and communications security (CCS)*. 2018.

[106] Duckkyoun Nam et al. "A Comparative Study on 3D Data Performance in Mobile Web Browsers in 4G and 5G Environments". In: *International Journal of Internet, Broadcasting and Communication* 11.3 (2019), pp. 8–19.

[107] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. "Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning". In: *ACM conference on Security and privacy in wireless & mobile networks (WiSec)*. 2014.

[108] Ravi Netravali et al. "Polaris: Faster Page Loads Using Fine-grained Dependency Tracking". In: *USENIX NSDI* (2016).

[109] Gerhard Neuhold et al. "The mapillary vistas dataset for semantic understanding of street scenes". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 4990–4999.

[110] Pauline C Ng and Steven Henikoff. "SIFT: Predicting amino acid changes that affect protein function". In: *Nucleic acids research* 31.13 (2003), pp. 3812–3814.

[111] Niantic. *Buddy Adventure coming soon*. https://pokemongolive.com/post/buddyadventurelaunch/?hl=en. 2019.

[112] Naheem Noah, Sommer Shearer, and Sanchari Das. "Security and privacy evaluation of popular augmented and virtual reality technologies". In: *IEEE International Conference on Metrology for eXtended Reality, Artificial Intelligence, and Neural Engineering*. 2022.

[113] NumPy. *numpy 1.21.5*. https://pypi.org/project/numpy/1.21.5/.

[114] Ilesanmi Olade et al. "Exploring the Vulnerabilities and Advantages of SWIPE or Pattern Authentication in Virtual Reality (VR)". In: *ACM International Conference on Virtual and Augmented Reality Simulations*. 2020.

[115] *OpenCV ArUco Marker Board*. https://docs.opencv.org/master/db/da9/tutorial_aruco_board_detection.html.

[116] Openwall. *John the Ripper*. https://github.com/openwall/john. 2013.

[117] Overture. *Overture map foundation*. https://overturemaps.org/.

[118] Bruno Patrão, Samuel Pedro, and Paulo Menezes. "How to Deal with Motion Sickness in Virtual Reality". In: *22o Encontro Português de Computação Gráfica e Interação 2015*. Ed. by Paulo Dias and Paulo Menezes. 2020.

[119] E. Peillard et al. "Studying Exocentric Distance Perception in Optical See-Through Augmented Reality". In: *IEEE ISMAR*. 2019.

[120] Kushani Perera et al. "Trajectory Inference for Mobile Devices Using Connected Cell Towers". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '15. Seattle, Washington: Association for Computing Machinery, 2015. ISBN: 9781450339674. DOI: 10.1145/2820783.2820804. URL: https://doi.org/10.1145/2820783.2820804.

[121] Stefano Petrangeli et al. "Dynamic adaptive streaming for augmented reality applications". In: *IEEE ISM*. 2019, pp. 56–567.

[122] Python. *Python 3.9.13 release*. https://www.python.org/downloads/release/python-3913/.

[123] *Python release 3.7.0*. https://www.python.org/downloads/release/python-370/. 2018.

[124] Bin Qian et al. "Dynamic Multi-Scale Convolutional Neural Network for Time Series Classification". In: *IEEE Access* 8 (2020), pp. 109732–109746. DOI: 10.1109/ACCESS.2020.3002095.

[125] Feng Qian et al. "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices". In: *ACM MobiCom*. 2018, pp. 99–114.

[126] Shwetha Rajaram, Franziska Roesner, and Michael Nebeling. "Designing Privacy-Informed Sharing Techniques for Multi-User AR Experiences". In: *International Workshop on Security for XR and XR for Security (Vr4Sec)* (2021). URL: https://par.nsf.gov/biblio/10312789.

[127] Xukan Ran et al. "Multi-User Augmented Reality with Communication Efficient and Spatially Consistent Virtual Objects". In: *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '20. Barcelona, Spain: Association for Computing Machinery, 2020, pp. 386–398. ISBN: 9781450379489. DOI: 10.1145/3386367.3431312. URL: https://doi.org/10.1145/3386367.3431312.

[128] Xukan Ran et al. "Multi-user augmented reality with communication efficient and spatially consistent virtual objects". In: *Proc. ACM CoNEXT*. 2020.

[129] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[130] *RealityCheck Project Website*. https://sites.google.com/view/arrealitycheck/home.

[131] Thomas Ristenpart et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds". In: *ACM conference on Computer and communications security (CCS)*. 2009.

[132] Franziska Roesner and Tadayoshi Kohno. "Security and Privacy for Augmented Reality: Our 10-Year Retrospective". In: *VR4Sec: 1st International Workshop on Security for XR and XR for Security* (2021).

[133] Franziska Roesner, Tadayoshi Kohno, and David Molnar. "Security and privacy for augmented reality systems". In: *Communications of the ACM* 57.4 (2014), pp. 88–96.

[134] C. S. Rosales et al. "Distance Judgments to On- and Off-Ground Objects in Augmented Reality". In: *2019 IEEE VR*.

[135] RosTeam. *GPS Emulator*. https://play.google.com/store/apps/details?id=com.rosteam.gpsemulator&hl=en_US&pli=1.

[136] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: (2011).

[137] Alejandro Pasos Ruiz et al. "The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery (* 35.2 (2021), pp. 401–449.

[138] Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. "Secure Multi-User Content Sharing for Augmented Reality Applications". In: *USENIX Security Symposium*. 2019.

[139] Piotr Sapiezynski et al. "Tracking Human Mobility Using WiFi Signals". In: *PLOS ONE* 10.7 (July 2015), pp. 1–11. DOI: 10.1371/journal.pone.0130824. URL: https://doi.org/10.1371/journal.pone.0130824.

[140] A. Sayles et al. "Invisible Perturbations: Physical Adversarial Examples Exploiting the Rolling Shutter Effect". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2021, pp. 14661–14670. DOI: 10.1109/CVPR46437.2021.01443. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.01443.

[141] Tim Scargill, Jiasi Chen, and Maria Gorlatova. "Here To Stay: Measuring Hologram Stability in Markerless Smartphone Augmented Reality". In: *arXiv preprint arXiv:2109.14757* (2021).

[142] Arne Schilling, Jannes Bolling, and Claus Nagel. "Using glTF for streaming CityGML 3D city models". In: July 2016, pp. 109–116.

[143] *scikit-learn: Machine Learning in Python*. https://scikit-learn.org/stable/. 2007.

[144] Jiacheng Shang et al. "ARSpy: Breaking Location-Based Multi-Player Augmented Reality Application for User Location Tracking". In: *IEEE Transactions on Mobile Computing* 21.2 (2022), pp. 433–447.

[145] Jiacheng Shang et al. "ARSpy: Breaking Location-based Multi-player Augmented Reality Application for User Location Tracking". In: *IEEE Transactions on Mobile Computing* (2020).

[146] Cong Shi et al. "Face-Mic: Inferring Live Speech and Speaker Identity via Subtle Facial Dynamics Captured by AR/VR Motion Sensors". In: *ACM International Conference on Mobile Computing and Networking (MobiCom)*. 2021.

[147] Ashutosh Singandhupe and Hung Manh La. "A Review of SLAM Techniques and Security in Autonomous Driving". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. 2019, pp. 602–607. DOI: 10.1109/IRC.2019.00122.

[148] *sktime*. https://www.sktime.org/en/stable/. 2019.

[149] Carter Slocum and Jingwen Huang. *VIA GitHub Repository*. https://github.com/mavens-lab/webxr_latency. 2021.

[150] Carter Slocum, Jingwen Huang, and Jiasi Chen. "VIA: Visibility-Aware Web-Based Virtual Reality". In: *The 26th International Conference on 3D Web Technology*. Web3D '21. Pisa, Italy: Association for Computing Machinery, 2021. DOI: 10.1145/3485444.3487641. URL: https://doi.org/10.1145/3485444.3487641.

[151] Carter Slocum, Xukan Ran, and Jiasi Chen. "RealityCheck: A Tool to Evaluate Spatial Inconsistency in Augmented Reality". In: *2021 IEEE International Symposium on Multimedia (ISM)*. 2021, pp. 58–65. DOI: 10.1109/ISM52913.2021.00018.

[152] Carter Slocum et al. "Going through the motions: AR/VR typing inference using head motion tracking". In: *USENIX Security Symposium*. 2023.

[153] *Solar System*. https://sketchfab.com/3d-models/solar-system. Accessed: 2021-08-01. 2018.

[154] *Sponza*. https://immersive-web.github.io/webxr-samples/tests/sponza.html. 2019.

[155] S. Stephenson et al. "SoK: Authentication in Augmented and Virtual Reality". In: *IEEE Symposium on Security and Privacy (SP)*. 2022.

[156] J. Sturm et al. "A benchmark for the evaluation of RGB-D SLAM systems". In: *IEEE/RSJ IROS*. 2012.

[157] Ivan E Sutherland. "A head-mounted three dimensional display". In: *Proceedings of the fall joint computer conference*. Dec. 1968, pp. 757–764.

[158] The Chronos Group. *The glTF standard*. https://www.khronos.org/gltf/. 2021.

[159] Nils Ole Tippenhauer et al. "On the Requirements for Successful GPS Spoofing Attacks". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Chicago, Illinois, USA: Association for Computing Machinery, 2011, pp. 75–86. ISBN: 9781450309486. DOI: 10.1145/2046707.2046719. URL: https://doi.org/10.1145/2046707.2046719.

[160] Pier Paolo Tricomi et al. "You can't hide behind your headset: User profiling in augmented and virtual reality". In: *IEEE Access* 11 (2023), pp. 9859–9875.

[161] Rahmadi Trimananda et al. “OVRSEEN: Auditing Network Traffic and Privacy Policies in Oculus VR”. In: *USENIX Security* (2022).

[162] Rahmadi Trimananda et al. “OVRseen: Auditing Network Traffic and Privacy Policies in Oculus VR”. In: *USENIX Security Symposium*. 2022.

[163] UNESCO. *Rock Shelters of Bhimbetka*. https://whc.unesco.org/uploads/nominations. 2003.

[164] Dorin Ungureanu et al. “Hololens 2 research mode as a tool for computer vision research”. In: *arXiv preprint arXiv:2008.11239* (2020).

[165] Unity. *Scripting API: CommonUsages - Unity - Manual*. `https://docs.unity3d.com/ScriptReference/XR.CommonUsages.html`.

[166] Unity. *Unity version 2020.3.26f1 download archive*. `https://unity3d.com/get-unity/download/archive`. 2020.

[167] Martin Vuagnoux and Sylvain Pasini. “Compromising electromagnetic emanations of wired and wireless keyboards.” In: *USENIX Security Symposium*. 2009.

[168] W3C. *WebXR Device API*. `https://www.w3.org/TR/webxr/`. 2021.

[169] Daniel Wagner et al. “Towards Massively Multi-user Augmented Reality on Handheld Devices”. In: *Pervasive Computing*. Ed. by Hans -W. Gellersen, Roy Want, and Albrecht Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 208–219. ISBN: 978-3-540-32034-0.

[170] Chen Wang et al. “Friend or foe? Your wearable devices reveal your personal pin”. In: *ACM on Asia conference on computer and communications security (ASIACCS)*. 2016.

[171] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. “MoLe: Motion Leaks through Smartwatch Sensors”. In: *ACM International Conference on Mobile Computing and Networking (MobiCom)*. 2015.

[172] Waqas Wazir et al. “Doodle-Based Authentication Technique Using Augmented Reality”. In: *IEEE Access* 8 (2020), pp. 4022–4034. DOI: `10.1109/ACCESS.2019.2963543`.

[173] Widya Andyardja Weliamto et al. “Enhancement of Aligning Accuracy on Zooming Camera for Augmented Reality”. In: *International Conference on Advances in Computer Entertainment Technology*. 2005.

[174] Jane Wilhelms and Allen Van Gelder. “Octrees for Faster Isosurface Generation”. In: *ACM Trans. Graph.* 11.3 (July 1992), pp. 201–227. ISSN: 0730-0301. DOI: `10.1145/130881.130882`. URL: `https://doi.org/10.1145/130881.130882`.

[175] Nick Wingfield and New York times Mike Isaac. *Pokémon Go Brings Augmented Reality to a Mass Audience*. `https://www.nytimes.com/2016/07/12/technology/pokemon-go-brings-augmented-reality-to-a-mass-audience.html`. 2016.

[176] Zhi Xu, Kun Bai, and Sencun Zhu. “TapLogger: Inferring User Inputs on Smartphone Touchscreens Using on-Board Motion Sensors”. In: *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*. 2012.

[177] R. Yagfarov, M. Ivanou, and I. Afanasyev. "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth". In: *International Conference on Control, Automation, Robotics and Vision*. 2018.

[178] Chen Yan et al. "Rolling colors: Adversarial laser exploits against traffic light recognition". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 1957–1974.

[179] Ananya Yarramreddy, Peter Gromkowski, and Ibrahim Baggili. "Forensic analysis of immersive virtual reality social applications: a primary account". In: *IEEE Security and Privacy Workshops*. 2018.

[180] Hantao Zhang, Wengang Zhou, and Houqiang Li. "Contextual Adversarial Attacks For Object Detection". In: *2020 IEEE International Conference on Multimedia and Expo (ICME)*. 2020, pp. 1–6. DOI: 10.1109/ICME46284.2020.9102805.

[181] Ruide Zhang et al. "AugAuth: Shoulder-surfing resistant authentication for augmented reality". In: *IEEE International Conference on Communications (ICC)*. 2017.

[182] Yicheng Zhang et al. "It's all in your head(set): Side-channel attacks on AR/VR systems". In: *USENIX Security*. 2023.

[183] Bendong Zhao et al. "Convolutional neural networks for time series classification". In: *Journal of Systems Engineering and Electronics* 28.1 (2017), pp. 162–169. DOI: 10.21629/JSEE.2017.01.18.

[184] F. Zheng, R. Schubert, and G. Welch. "A general approach for closed-loop registration in AR". In: *IEEE Virtual Reality*. 2013.

[185] Feng Zheng, Dieter Schmalstieg, and Greg Welch. "Pixel-wise closed-loop registration in video-based augmented reality". In: *Proc. IEEE ISMAR*. 2014.

[186] Chao Zhou, Mengbai Xiao, and Yao Liu. "Clustile: Toward minimizing bandwidth in 360-degree video streaming". In: *IEEE INFOCOM*. IEEE. 2018, pp. 962–970.

[187] Peng Zhou et al. "Learning Rich Features for Image Manipulation Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[188] Peng Zhou et al. "Learning Rich Features for Image Manipulation Detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[189] Wu Zhou et al. "Detecting Repackaged Smartphone Applications in Third-Party Android Marketplaces". In: *ACM Conference on Data and Application Security and Privacy (CODASPY)*. 2012.

[190] Yajin Zhou et al. "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets". In: *Network and Distributed System Security Symposium (NDSS)* (2012).

[191] Wenjun Zhu et al. "TPatch: A Triggered Physical Adversarial Patch". In: (2023).

[192] Li Zhuang, Feng Zhou, and J Doug Tygar. "Keyboard acoustic emanations revisited". In: *ACM Transactions on Information and System Security* 13.1 (2009), pp. 1–26.

[193] David J Zielinski et al. “Exploring the effects of image persistence in low frame rate virtual environments”. In: *IEEE Virtual Reality (VR)*. 2015.

[194] Danping Zou and Ping Tan. “CoSLAM: Collaborative Visual SLAM in Dynamic Environments”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.2 (2013), pp. 354–366. DOI: 10.1109/TPAMI.2012.104.