Creating a Low-Cost Virtual Camera System

A thesis presented to
The Clemson School of Computing

In Partial Fulfillment
of the Requirements for
Senior Departmental Honors

---

by
Dylan Andrew Bruss
May 2022

Abstract

Computer Animation has become more and more popular, outpacing traditional animation in feature-length films. These films use virtual cameras to represent the viewer's perspective in a virtual world. In addition, many visual effects rely on reproducing a physical camera's position, orientation, and properties in a virtual space. While these virtual cameras can be moved manually through techniques such as key-framed animation, this technique can produce, rigid, unnatural movement without careful operation. Virtual Production is a technique where a real-world camera is tracked with multiple degrees of freedom, producing more realistic, natural camera movement. This project aims to create a simple virtual production system with widely available consumer software and hardware.

Acknowledgments

I would like to thank Dr. Jerry Tessendorf for advising me throughout this project. I would also like to thank Shuvagata Shuvo for his assistance with testing and evaluating this project. Finally, I would like to thank the Clemson Honors College and the Clemson School of Computing for making this project possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The movie-making industry has had several significant breakthroughs over the past few years. Leaps in computer graphics mean that films created with computers can now look more realistic than ever before while giving movie-makers artistic control that would be impossible with practical effects. More recently, improvements in real-time graphics rendering processes, fueled by the gaming industry, have allowed many graphical effects that once took weeks to process to be generated almost instantaneously. Similarly, motion tracking and augmented reality advances are also applied to film-making. As a result, a new type of film-making has emerged: Virtual Production.

1.1 Virtual Production

Virtual Production is a process used in the film and television industries that use motion tracking and augmented reality to combine visual effects with live performances on set, often allowing for viewing a final or near-final shot at the time of filming. Virtual Production is an ongoing field and encompasses many different types of production technologies. However, the three aspects that these technologies typically share include the recording of a live performance, the use of a virtual set, and the use of film-making interfaces (usually a camera) that resemble the interfaces used in traditional film-making.

Virtual Production techniques originally reached mainstream recognition in the film *Avatar*, in which it was used to view the actions of performers in the context of digital characters in the fictional world of Pandora, in which the movie takes place (see figure 1.1). Using virtual Production,

Figure 1.1: Virtual production techniques being used on the set of *Avatar* (Image ©Weta Digital)

director James Cameron was able to use an interface similar to a traditional film camera to record and view performances by live actors replicated on virtual characters. [1]

More recently, effects companies such as Industrial Light and Magic have created Stagecraft, a virtual production system used on the Disney+ series *The Mandalorian* (see figure 1.2). Stagecraft uses a series of bright LED panels to recreate virtual environments around the actors on set. It can recreate backgrounds from the camera's perspective and reproduce the lighting that would be experienced in the replicated environment on the physical stage[2]. Virtual production systems such as these drastically reduce the amount of time used to create visual effects, as many elements can be adjusted and captured on set instead of painstakingly reproducing the nuances of motion and visuals to match the filmed content after the fact.

This paper explores one particular aspect of Virtual Production: the virtual camera.

1.2 Virtual Cameras

As referred to in this paper, a virtual camera system refers to a type of virtual production system that tracks and records the motion of a physical camera in real-world space and applies that motion to a camera in a virtual world. Additionally, the system must render a scene from

Figure 1.2: Virtual production techniques being used on the set of *The Mandalorian* (Image ©Industrial Light and Magic)

the view of the virtual camera and provide the rendered images as feedback to the camera person. A virtual camera system allows users to film a virtual world as if they were recording a real-life scene with a traditional camera. The virtual camera is typically an essential part of any virtual production workflow, as the camera view is the root of any visual effects shot. Even in virtual production systems that use traditional cameras such as Stagecraft, the camera's view must always be considered to place virtual elements in the set correctly and accurately project the perspective of the set's virtual background. Without this, the illusion of the virtual world is immediately broken.

In addition to being an integral part of many virtual production systems, a virtual camera alone can be considered one of the simplest forms of virtual Production. The three aspects of virtual Production are all present. A virtual camera records the live performance of the camera person. It is used to view a virtual set and is meant to resemble the cameras used in traditional film-making. Even so, it only requires the tracking of the position and orientation of a single point on the camera itself and does not require complex and potentially expensive technologies like facial and body motion capture or the large, power-hungry, and costly screens of a system like Stagecraft.

Modern mobile devices contain enough sensors and processing power to host a simple virtual camera on their own. However, this system would be severely limited by such a device's rendering and power restrictions. Conversely, modern consumer desktop systems are powerful enough to render complex scenes in real-time but lack the sensors and portability needed for a virtual camera system. As a result, this project will attempt to combine the benefits of both platforms, using the mobility and sensors of mobile devices in combination with the power of desktop systems to produce an

effective consumer virtual camera system.

1.3 The purpose of this project

While virtual production systems are widely used throughout the movie-making industry, these technologies are expensive, hard to maintain, and often created specially for the companies or productions that use them. Independent filmmakers and students outside of these companies often do not have access to these technologies. This project attempts to assess the current state of technologies available to such filmmakers by constructing a simple yet effective virtual camera system using consumer technologies.

Chapter 2

Related Work

Virtual production is an active field of research explored by many groups worldwide. This section features a few projects related to virtual camera systems and describes how they relate to this project.

Genesis is a pipeline designed by production company MPD for Virtual Production workflows. [6] This pipeline includes many aspects of the virtual production process, including asset creation, set design, and recording. While this system integrates several tracked devices for use as virtual cameras, it does not describe how such a system is designed. [6]

VPET or Virtual Production Editing Tools [5], is a system developed by German film production company Filmakademie Baden-Wuerttemberg to assist in virtual production environments. The system features the ability to use multiple mobile devices to view and edit virtual production scenes in real-time. However, unlike this project, these devices are only used for on-set scene changes, not as the primary scene camera. In addition, the project uses on-device rendering for the display on these devices and not the full-quality scene as rendered by the central server. As a result, more expensive and extensive virtual production equipment is needed to provide tracking for the main camera view. [5]

Finally, the project that is likely closest to this project in functionality is FXHome's CamTrakAR application for iOS-based mobile devices. This application integrates camera tracking, recording, keying, and many other aspects of virtual production into an iOS application. Unlike this project, however, this application renders pre-visualizations on the device and is thus subject to the rendering limitations of handheld devices. [3]

Chapter 3

Research Design and Methods

The virtual camera system created during this project consists of two major components: a mobile application that runs on an android phone and a Unity Engine-based program that runs on a Windows computer. The two applications are connected with a simple TCP-based protocol in a client-server relationship, with the android application as the "client" and the desktop program as the "server." The RTSP streaming protocol is used to transfer video information from the server. Finally, the motion data is saved to the hard disk, and a specially written extension for the 3D software Blender is used to import the data to allow it to be exported to other formats. Each of these components is described in its own section below.

3.1 Client System

The client system is written in Java using the Android SDK. This application handles several essential tasks: It must use the Android device's sensors to track the environment around it and generate position coordinates. It must be able to open a video stream from the server and display it onscreen. It must communicate position data and start and stop events (from an onscreen record button) to the computer via TCP.

The application flow is as follows. First, the user opens the application on their android device. It prompts the user for the IP address of the server. After the user enters the IP address, the app connects to the server and displays a camera view of the environment. The user is told to move the camera around to start tracking the environment. Once the tracking begins, the user can

place a marker in the environment. The marker serves as a "scene root" and gives the application a reference point to derive the camera's location. After the marker is placed, a video stream showing the contents of the server's display appears on the phone, and it is ready to start the recording process. A record button is provided on the device for convenience.

To track the environment, the Google ARCore SDK was used. The ARCore SDK is typically used to create Augmented Reality applications for Android devices. It is a comprehensive system that combines input from several sensors, including the camera, gyroscope, and accelerometer. Under the hood, ARCore uses Simultaneous Localization and Mapping[4] to generate a comprehensive understanding of the environment around your device and, more importantly, the location of your device relative to the environment.

The result is robust 6-degree of freedom tracking that continues to work, even when the camera observes previously unseen parts of the environment, a crucial feature for a virtual camera system.

The stream is displayed using the ExoPlayer Framework. Once the IP address is picked, the video stream is opened using ExoPlayer and rendered over the camera view for the remainder of the session.

An asynchronous TCP client system manages other TCP communications. The client itself runs in a separate thread from the rest of the application and uses the Android Socket API to manage the connection. Messages to be sent are added to a FIFO (First-In, First-Out) Queue structure in the main application thread and then asynchronously sent by the messaging thread soon after. Received messages undergo the reverse process, being placed into a queue by the messaging thread to be retrieved from the main application thread.

3.2 Server System

The server system has several functions for which it is responsible. First, it must accept connections from a client application. Second, it must retrieve camera data from the client and render frames with the given camera position and location in 3D space. Next, it must stream these frames over the network using RTSP. Finally, on command, it must write received camera data to a file on disk to be imported by another application later. In addition, it must allow a user to load a custom scene from a file on disk, including any material information that the scene may require.

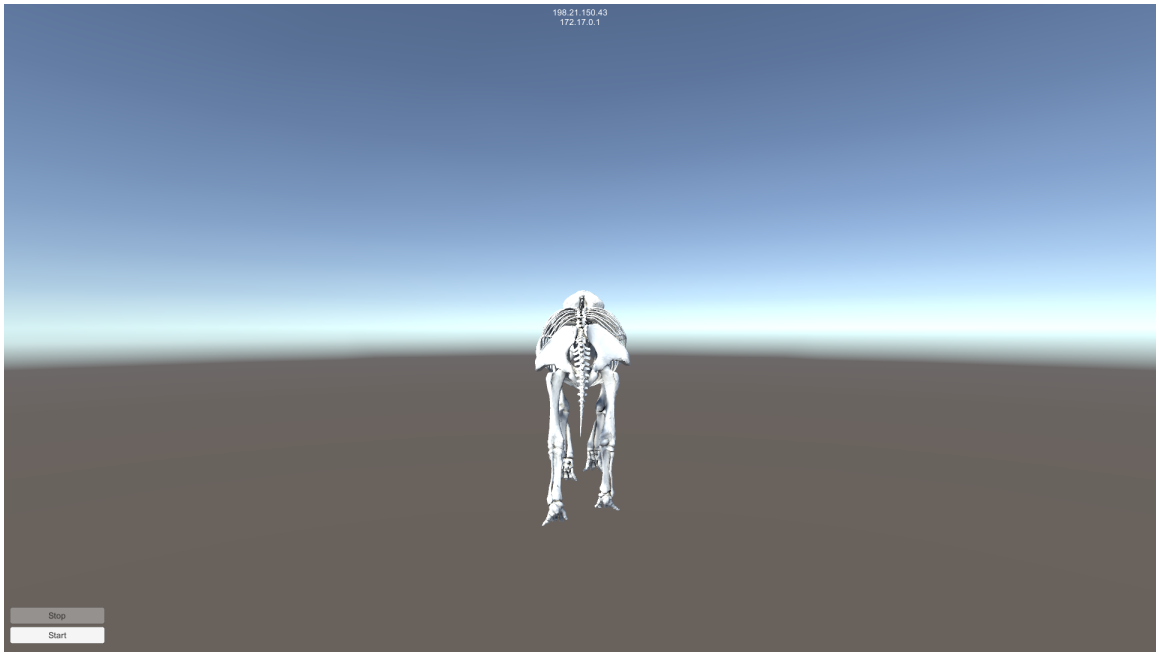

Figure 3.1: A screenshot of the server software, with an example scene loaded.

The Server System is implemented within the Unity Game Engine. Unity provides a high-quality real-time rendering system and an editing environment that allows for scene customization. Scenes are imported from GLTF files on program startup, using a system dialog. GLTF was chosen because it enables the application to import scenes at run-time, it is open-source and portable, and it can store models, textures, physically-based materials, and more within the same file.

Once this file is imported, the TCP server is started alongside the RTSP stream. FFmpeg is used to capture the screen and stream the video data in MPEG-4 format to the RtspSimpleServer application, which manages the incoming connections from the client. If available, FFmpeg is told to use the NVidia NVENC hardware video encoder to reduce streaming latency. The TCP communication server is structured very similarly to the TCP client in the Client application, with the added functionality of accepting new TCP connections.

Once the server portion of the application has started, the user is presented with a 3D view of the loaded scene (see figure 3.1). At the top of the scene, the server's local IP address is listed to facilitate the connection. In addition, Start Recording and Stop Recording buttons are provided in the lower-left corner, though it is expected that the user will use the record button on the client.

3.3 Communication System

A simplified TCP protocol is used to communicate camera data and events to the server. RTSP is used to stream video data back to the mobile device. The details of these systems are as follows.

A single TCP connection is used to transfer data about the camera's pose and events such as the record button being pressed. A message in this stream is formatted in ASCII as follows:

```
[Message Type] Message Data
```

There are three major message types: "locrot," "start," and "stop." "Locrot" is used to transfer camera pose information from the Android device to the server. Information is transferred in 7 values, x,y, and z location, and w,x,y, and z quaternion data. This data is represented as comma-separated ASCII values.

"Start" and "stop" signals have no message data. They are both interpreted the same way: by toggling the recording state of the server. While "start" is intended to start the recording, and "stop" is meant to stop it, the client and server can become out-of-sync. In this case, it appears more responsive to the end-user if any "start" message received while the system is recording is interpreted as a "stop" message and vice-versa.

An additional message is included in the server but goes unused. This message is the "matrix" message. The "matrix" message is used to define the camera's fundamental distortion matrix. It could be helpful if you intend to duplicate the camera's field of view and other properties in the virtual environment. However, while this value is applied to the camera in the server itself, it is not recorded and thus cannot be reproduced in the final output.

3.4 Pairing process

Each time the system is put to use, a simple pairing process is required to link the client and server. First, the user must start the server software and select a scene to be viewed. Next, the client software is started. The user must then type the IP address of the server (listed at the top of the main screen of the server software) into the client app. Once this occurs, the client establishes a TCP connection with the server application and prompts the user to select a location to be used as the root of the viewed scene.

```
timestamp,position x,position y,position z,rotation x,rotation y,rotation z,rotation w
1076861824,-1.47248,1.368756,0.4356973,-0.1901845,-0.5747964,-0.1387087,0.7837083
1076861824,-1.472545,1.369437,0.436482,-0.1887854,-0.5741138,-0.1379721,0.7846764
1076861952,-1.472645,1.369538,0.4368883,-0.1883429,-0.5745955,-0.1380723,0.7844125
1076861952,-1.472849,1.369774,0.4372776,-0.18776,-0.5742177,-0.1385641,0.7847421
1076861952,-1.47432,1.370075,0.4361213,-0.1866281,-0.5747944,-0.1378219,0.7847205
1076861952,-1.474235,1.370085,0.4362324,-0.1861181,-0.5758243,-0.1378284,0.7840853
1076862080,-1.474065,1.370106,0.4363132,-0.1857297,-0.5769861,-0.1382707,0.783245
1076862080,-1.472828,1.370463,0.4357818,-0.1868331,-0.578471,-0.1383537,0.7818714
1076862080,-1.472676,1.370551,0.4357148,-0.1867578,-0.5784878,-0.1397281,0.7816325
1076862080,-1.472745,1.370883,0.4357337,-0.186018,-0.5778832,-0.1411155,0.7820067
1076862208,-1.472294,1.371703,0.4355758,-0.1857441,-0.5779849,-0.1413499,0.7819545
1076862208,-1.472257,1.371614,0.4354643,-0.1861996,-0.578477,-0.1409616,0.7815522
1076862208,-1.472131,1.37141,0.4352925,-0.1864162,-0.5795237,-0.1401461,0.7808715
1076862208,-1.471859,1.371773,0.4349603,-0.1871707,-0.5803966,-0.1391761,0.780216
1076862336,-1.471906,1.371428,0.4347564,-0.1880438,-0.5803721,-0.1387652,0.7800975
1076862336,-1.472079,1.371192,0.4346206,-0.1882578,-0.5799174,-0.1384815,0.7804343
1076862336,-1.471416,1.37134,0.4340048,-0.1886879,-0.5797877,-0.1380505,0.7805031
1076862464,-1.471429,1.371082,0.433865,-0.1889901,-0.5798097,-0.1376093,0.7804916
1076862464,-1.471489,1.37088,0.4337248,-0.1894528,-0.5796489,-0.1373295,0.7805481
1076862464,-1.471512,1.370969,0.4331503,-0.1895553,-0.5796363,-0.1367906,0.7806272
1076862464,-1.471766,1.37097,0.4329952,-0.1893366,-0.5799068,-0.1361618,0.7805895
1076862592,-1.472041,1.371003,0.4327272,-0.1888154,-0.580636,-0.1354119,0.7803038
1076862592,-1.472332,1.371134,0.4318693,-0.1881586,-0.5813661,-0.1344827,0.7800797
1076862592,-1.472692,1.3712,0.4316274,-0.1874339,-0.5816342,-0.1333335,0.7802514
1076862592,-1.472945,1.371202,0.4314808,-0.1873011,-0.5816132,-0.1323987,0.7804582
1076862720,-1.473225,1.371562,0.4305421,-0.1868276,-0.5818315,-0.1315288,0.780556
1076862720,-1.473353,1.371502,0.4303478,-0.1866836,-0.582342,-0.1309967,0.7802992
1076862720,-1.473347,1.371309,0.4301464,-0.187048,-0.5832321,-0.1303324,0.779658
1076862720,-1.473123,1.371431,0.429661,-0.1870254,-0.5840947,-0.1295623,0.779146
1076862848,-1.473223,1.371501,0.4297048,-0.1866193,-0.5843619,-0.1295189,0.7790501
1076862848,-1.473364,1.371638,0.4298269,-0.1860696,-0.5843172,-0.129713,0.7791829
1076862848,-1.473345,1.371888,0.4297097,-0.1857217,-0.5841196,-0.1300254,0.779362
1076862848,-1.473504,1.3721,0.4299639,-0.1850493,-0.5839122,-0.1302131,0.779646
1076862976,-1.473723,1.372343,0.4303021,-0.1838311,-0.5839009,-0.1299155,0.7799921
1076862976,-1.473324,1.372621,0.4302202,-0.1831861,-0.5841208,-0.129753,0.7800064
1076862976,-1.473258,1.37276,0.4305475,-0.182174,-0.5845869,-0.1295881,0.7799217
1076862976,-1.473085,1.372842,0.4308469,-0.1813151,-0.5850474,-0.1295395,0.7797844
1076863104,-1.47308,1.372791,0.43046,-0.1815141,-0.5854855,-0.1297352,0.7793767
1076863104,-1.472902,1.372679,0.4306163,-0.1815739,-0.5858415,-0.1299179,0.7790648
1076863104,-1.472899,1.372617,0.4307421,-0.1814863,-0.5859099,-0.130217,0.7789839
```

Figure 3.2: An example of a csv file generated by the virtual camera server

The user must now use the cell phone to establish tracking and select the root of the scene. Once this occurs, the client application will begin streaming rendered video frames from the server.

3.5 Using produced camera data

Once data is recorded, it is written to disk as a CSV (comma-separated values) file. An example file can be seen in figure 3.2. The file contains camera position and rotation data as well as relative timestamps. Timestamps are sent from the client application and represent the time in milliseconds since the start of the application rather than seconds since the epoch. As a result, they are intended to only be used in reference to one another rather than in reference to a certain zero point.

As a part of this project, an add-on was written for the Blender 3D software package that can be used to import CSV data exported from the server application. This motion data is applied to an empty object in the scene, using the relative timestamps to determine keyframe placement. From Blender, a variety of formats can then be exported, including popular formats such as BVH, Collada, or FBX, to use in other professional software packages such as Autodesk Maya and MotionBuilder.

3.6 Latency Evaluation system

In addition to the systems that make up the project itself, a specialized system was used for the evaluation of video streaming latency from the server to the client. This system uses visual QR (quick-response codes) to measure complete system screen-to-screen latency between devices.

3.6.1 Evaluation system setup

The evaluation system consists of a few different components. First, a Python script is used to generate a series of sequential QR codes, each containing a number in ASCII format increasing at a rate of 1 integer value for every $\frac{1}{5}$ of a second. The QR codes are generated at a resolution of 290 by 290 pixels. These frames are then compiled into an h.264 encoded MPEG-4 video file.

For testing, the system is first set up as it would be for normal use (see section 3.4 above). Instead of streaming rendered video frames from the server, however, the previous encoded video file is both displayed on the server screen as well as streamed to the client device.

Finally, a third video capture device records the displays of both the client and the server devices (see figure **??**). This is recorded for the duration of the previously encoded video file. The recorded data is then processed using the OpenCV library. Both devices are cropped from the entire video and are scanned with a QR code reader. The numeric values are then decoded and recorded to a CSV (comma-separated values) file. The format of this file is such that each row corresponds to a frame for the video capture device, and they contain two values: the recognized value for the client device for that frame and the recognized value for the server device. Some frames are dropped through this process due to glare on the client and server device screens, as well as other interference. However, enough frames are captured to generate a good picture of the screen-to-screen latency of the device.

Chapter 4

Results

The Virtual Camera system produced by this project has the ability to record camera motions at about 30 frames per second while rendering a scene and streaming the scene to the mobile device. There are, however, a few shortcomings of the system in the current iteration. While the latency of the camera's position and rotation is relatively low, video streaming can have up to a few seconds of latency (see Figure 4.1).

4.1 Recording Performance

Recording performance is measured using the number of frames captured for each second of time. Four trials were recorded. Each trial consisted of a camera motion one minute in length. Trials were then divided into non-overlapping second-long segments, and the number of frames in each segment was totaled. The average and standard deviation for each trial were calculated from the set of frame counts among all segments in a trial. The overall average and standard deviation were then calculated for the set of all segments for all trials. Results can be seen in table 4.1.

Trial	Average Frames per Second	Standard deviation
Trial 1	40.000	9.392
Trial 2	31.029	6.101
Trial 3	23.737	6.354
Trial 4	24.581	7.116
Overall	29.788	9.658

Table 4.1: Recording performance, measured in average and standard deviation of the number frames recorded in a second, for each of four trials, as well as for all datasets combined.

4.2 Streaming Latency

Streaming latency in this context is measured in a manner consistent with that described in section 3.6. A total of 4 trials were run, completely stopping and restarting the system between each trial. All trials were completed in succession using the same hardware and network and thus were performed in similar network conditions.

The results of the four trials can be seen in Figure 4.1. All trials were conducted over a wireless network. From these results, a dataset was constructed from the differences between the first frames in the footage in which a specific QR code is recognized on both devices. Taking a mean of this dataset results in an average latency of 123.75 frames in the capture device, with a standard deviation of 85.63 frames among all four tests. Assuming that the camera is consistently running at its set frame rate of 30 frames per second, this calculates to an average latency of 4.12 seconds with a standard deviation of 2.85 seconds.

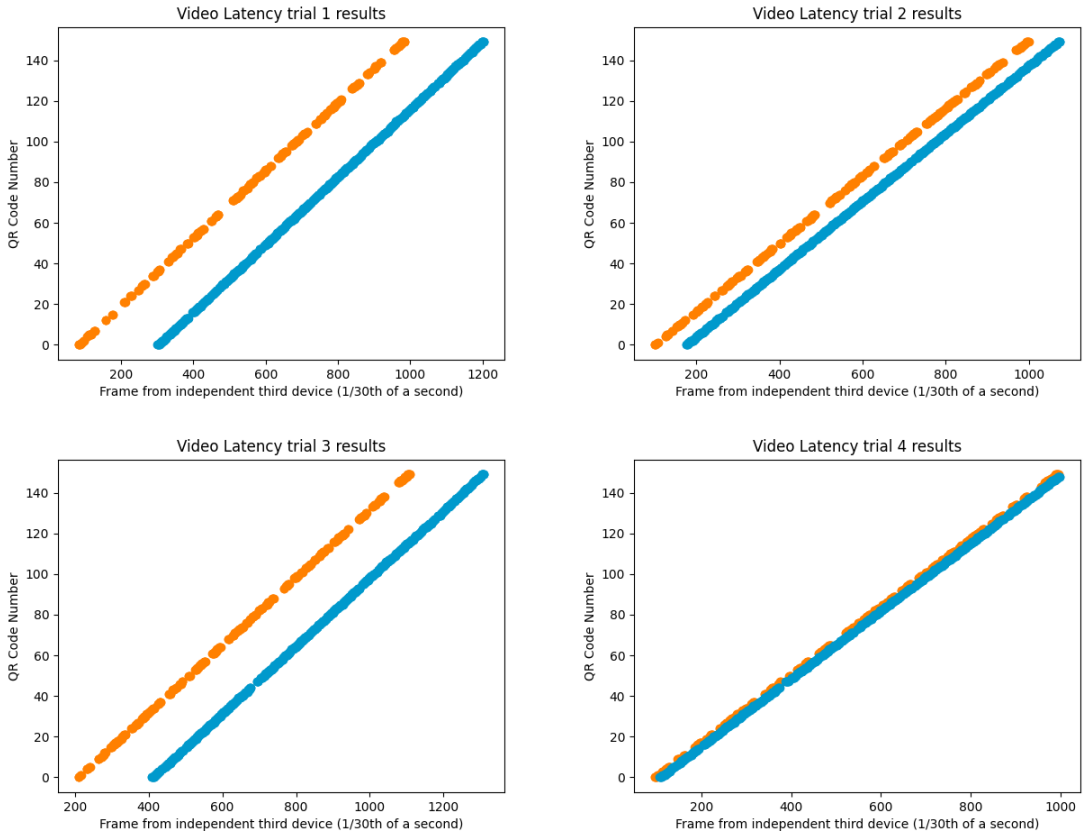

Figure 4.1: The above graphs represent the results from four tests using the evaluation system described in section 3.6. Orange points represent frames recognized from the server machine, and blue points represent frames recognized from the client device

Chapter 5

Conclusions and Discussion

This project shows that a low-cost system such as this can emulate the function of more expensive virtual cameras. The tracking, transmission, and recording of the camera's position and orientation in three-dimensional space perform adequately to produce a smooth camera motion at close to 30 frames per second with little to no need for correction by the artist (see table 4.1). Additionally, the system allows for a cameraperson to operate it wirelessly, using a lightweight phone instead of a heavy camera rig. It has been shown that rendered data can be wirelessly transmitted over the network, and rendered scenes can be transmitted back from a more powerful device. It has also been shown that modern handheld augmented reality tracking technologies are sufficient to provide the tracking data for a virtual camera system of this variety.

The greatest barrier to the project in its current state is the video streaming latency in sending rendered frames back to the handheld device. As shown in section 4.2, the system was determined to have an average screen-to-screen latency of about 4.12 seconds, which is much too high to provide real-time feedback to a camera operator. The latency can vary wildly, with an experimental standard deviation of 2.85 seconds in latency. However, once the system is initialized, latency remains relatively constant (as seen in Figure 4.1). This suggests that something in the connection process could be changed to improve or reduce latency. However, further research is required to determine what exactly could be done to improve latency.

Chapter 6

Future Work

While this project demonstrates the basic functionality of a virtual camera system, there are many ways that the system can be improved upon in future research.

6.1 Video Latency

As discussed in Chapter 5, the greatest barrier to the usefulness of this project is the high latency between an image being rendered and the display of that image on the client device. Low-latency is a difficult problem, but many solutions have been created to make low-latency streaming easier on consumer hardware.

For example, as the system is currently implemented, it is locked to streaming at 720p and 30 frames per second. However, the system could instead make use of adaptive streaming, where video resolution, frame rate, and other quality factors are changed dynamically by the client due to factors like network traffic, bandwidth, and decoding speed.

Additionally, most stream playback libraries keep a buffer between the latest received frame and the frame currently being displayed, so it can ensure smooth playback and prevent audio distortion. Since no audio is being transmitted, audio distortion is not a problem, and lower latency is much more desirable for this application than smooth playback. As a result, it may help latency to reduce or completely remove the buffer entirely.

Finally, the stream uses the H.264 format for video compression, which consists of "I-Frames," which are static images, and "P-Frames," which are frames constructed by referencing

other frames. The initial source of latency may be due to the client waiting for the server to send an initial I-Frame before beginning playback, resulting in playback beginning later than it should. As a result, increasing the frequency of I-Frames, while it could increase the bandwidth requirement, could assist in lowering latency.

6.2 Additional Functionality

In addition to improving the functionality of the core project, there are many ways the project could be expanded in future works to provide additional virtual production capabilities.

One idea would be to allow the client to stream the video feed captured by the camera back to the server to be combined with the CGI elements. Other data can also be streamed from the device to the server, such as camera projection matrices and real-world depth information, both of which can be queried using the ARCore API.

Finally, the system could integrate elements of motion capture technology to allow live performances to be mirrored by animated characters in the camera view. This would mimic virtual production systems used in the movie Avatar, among others, and allow a user to film a motion-capture performance much in the same way that a live performance would be captured, yet still retain the benefits of a computer-animated scene.

Bibliography

[1] *WetaFX*, 2022.

[2] *Industrial Light & Magic*, May 2022.

[3] *FXhome*, 2022.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[5] Simon Spielmann, Volker Helzle, Andreas Schuster, Jonas Trottnow, Kai Götz, and Patricia Rohr. Vpet: Virtual production editing tools. In *ACM SIGGRAPH 2018 Emerging Technologies*, SIGGRAPH '18, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Robert Tovell and Nina Williams. Genesis: A pipeline for virtual production. In *Proceedings of the 8th Annual Digital Production Symposium*, DigiPro '18, New York, NY, USA, 2018. Association for Computing Machinery.