

Bucknell University

Bucknell Digital Commons

Honors Theses

Student Theses

Spring 2023

The Implementation of Augmented Reality and Low Latency Protocols in Musical Instrumental Collaborations

Qixiao Zhu
qz003@bucknell.edu

Follow this and additional works at: https://digitalcommons.bucknell.edu/honors_theses



Part of the [Audio Arts and Acoustics Commons](#), [Graphics and Human Computer Interfaces Commons](#), [Other Music Commons](#), and the [Software Engineering Commons](#)

Recommended Citation

Zhu, Qixiao, "The Implementation of Augmented Reality and Low Latency Protocols in Musical Instrumental Collaborations" (2023). *Honors Theses*. 638.
https://digitalcommons.bucknell.edu/honors_theses/638

This Honors Thesis is brought to you for free and open access by the Student Theses at Bucknell Digital Commons. It has been accepted for inclusion in Honors Theses by an authorized administrator of Bucknell Digital Commons. For more information, please contact dcadmin@bucknell.edu.

**The Implementation of Augmented Reality and Low Latency Protocols in Musical
Instrumental Collaborations**

by

Qixiao Zhu

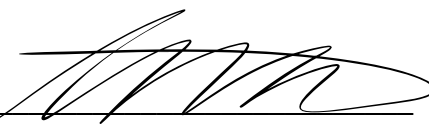
A Senior Thesis Submitted to the Honors Council

For Honors in Department of Computer Science

April 3rd, 2023

Adviser: _____

Xiannong Meng

Co-advisor: _____

Paul J. Botelho

Table of Contents

Introduction	1
Background	3
Design and Implementation	7
Components	9
MIDI Signal	9
Oculus Quest 2	11
Unity Game Engine	13
Open Sound Control	15
Musical Instrument 1: Vocal Chorus	16
Oscillator	17
Musical Instrument 2: Modular Synthesizer	19
Modular Synthesizer User Interface	20
Network	22
Process	23
Results and Discussions	26
Reference	29

Introduction

This thesis investigates the issue of music collaboration using augmented reality and related topics. My software, “AR Jam”, aims to investigate the various possibilities of the use of augmented reality in music studies and collaborations. I attempt to use existing ideas and technologies in Augmented Reality, AR digital musical instruments, computer networks, and game development to put together a piece of new software for musicians to use. The objective of this project is to analyze the feasibility of this type of software.

The original intention of this thesis lies in serving the need for long-distance interactions among musicians and overcoming the challenges brought on by the COVID-19 global pandemic, which impacted a lot of countries. China, for example, believed that quarantine and lockdowns were the only way to stop the spread of the virus. Consequently, traveling between the U.S. and China had become extremely difficult. As an international student from China, making music with friends in China was impossible during that time period. “AR Jam” was a way that I came up with to play music with people from everywhere else. Music performance students might even take virtual online lessons using this platform. While the lockdown of the pandemic is coming to an end, the idea of this project is still valid, since more and more people are looking into socializing and working together in the virtual world. I believe that musicians can have an opportunity to play music together in the virtual world as well, and mixed reality might be the best solution to this task yet.

“AR Jam” uses networking methods to transmit music and body gesture data between the users. Body gestures are a crucial part in music playing in bands and ensembles. Since musicians cannot cue each other by talking, body gestures are what they use to communicate when playing music most of the time. When musicians play ensembles across different physical locations, it would be ideal to have software that is capable of tracing hand and head

movements, sending them over the internet, and playing them back in some device on the receiving end. Such software provides a very good collaborative medium for musicians, since seeing other people's physical hand and head gestures is not possible on any other types of platforms.

Virtual/augmented reality music software are often not designed for practical music playing. Software designers often do not care about the playability of the virtual instruments, but only care about the novel feeling that the users acquire when playing with the program. For example, most of the known projects that involve musical instruments in AR/VR implement the instruments entirely virtually. These instruments are not as easy to play as playing a physical musical instrument, and thus cannot be used by musicians to practically create and play music. "AR Jam" seeks to use the characteristic of AR, which is a mix between virtual and physical signals, to create physical musical instrument interfaces with virtual sound production and transmission.

This thesis is divided into multiple sections to further present the research. The Background section will present the initial idea of this project, previous projects done by other researchers, and all the tools involved in creating this piece of software, "AR Jam." The Background section will also compare the different existing software, analyze their pros and cons, and explain the logic behind the design of "AR Jam." The Design and Implementation section will highlight the general structure of the working project. The Components section will dive deeper into each part of the project and look into the tools and implementations used in detail. The Process section will present the general development process of the project, issues encountered during development, compatibility problems, and describe the pros and cons of each tool that was used. The Results and Discussions section will reflect on the whole project, what works well and what does not, and propose future research ideas and improvements in relation to "AR Jam."

To summarize, this project is not only about making a piece of music software, but also an experiment on the capability of augmented reality in the art of musical instrumental collaboration.

Background

The importance of body language in music is immeasurable, as musicians use their bodies to show musical gestures and give other musicians physical cues when collaborating. Without body language, instrumental collaboration would be harder, as there would be no other way than giving audible cues to other musicians. Musical gestures can also be expressed much more clearly with the use of body language. In all genres, musicians move their bodies a lot when playing music. To the audience and other musicians, body language is an indication of how the musicians understand the music and shape the music. In *Bodily movement and facial actions in expressive musical performance by solo and duo instrumentalists: Two distinctive case studies* (Davidson, Jane, 2012), the authors analyze the performance of Lang Lang playing Franz Liszt's *Nocturne no. 3*, based on a poem by Freiligrath, *O lieb, so lang du lieben Kannst* (*O love, as long as you can love*). The authors specifically talk about how Lang Lang's expression is distributed on his whole body. They argue that there is a strong connection between musical structures and the body gestures of Lang Lang. The change of musical rhythm and harmony is always accompanied by the physical change in body gestures and face expressions. "Whether consciously employed or not, the physical expressions of the body and face offer a means by which the performer can generate expression which is both integrated into the musical effects created – timing and dynamic effects – and which exists independently of them. That is, I can observe these expressions of physical release in the still images, or better still, experience them as I observe

the dynamics of the performer as observed on film, or indeed in the live performance context.” (Davidson, Jane, 2012, p. 618) In response to this finding, “AR Jam” aims to introduce bodily movement to networked music playing.

Previous research works have been inspirations for “AR Jam” in countless ways, such as “PianoVision” (Reid, 2022), which has a big influence on the design of “AR Jam.” The software features a combination of physical musical instruments and virtual user interfaces, along with multiple ways for the users to connect their physical instruments to their VR headsets, which “AR Jam” also uses. “PianoVision” presents a virtual interface similar to the layout of a piano roll, in which the user can see the notes coming from far away. When the notes reach the location of the physical keyboard, the user plays the corresponding notes that just arrived at their designated keys. The software also features virtual pianos, which are primarily for users who do not own a digital piano. “PianoVision” allows users to collaborate with each other online, enabling them to see each other in a virtual environment. The use of handtracking can also be seen in “PianoVision.”, which inspired handtracking implementation in “AR Jam.” When using the handtracking function, players can use their hands to navigate the virtual controls and interface. This makes navigating the virtual environment simpler, since there is no need to pick up a controller each time the user stops playing in order to interact with the virtual environment (see Figure 1).

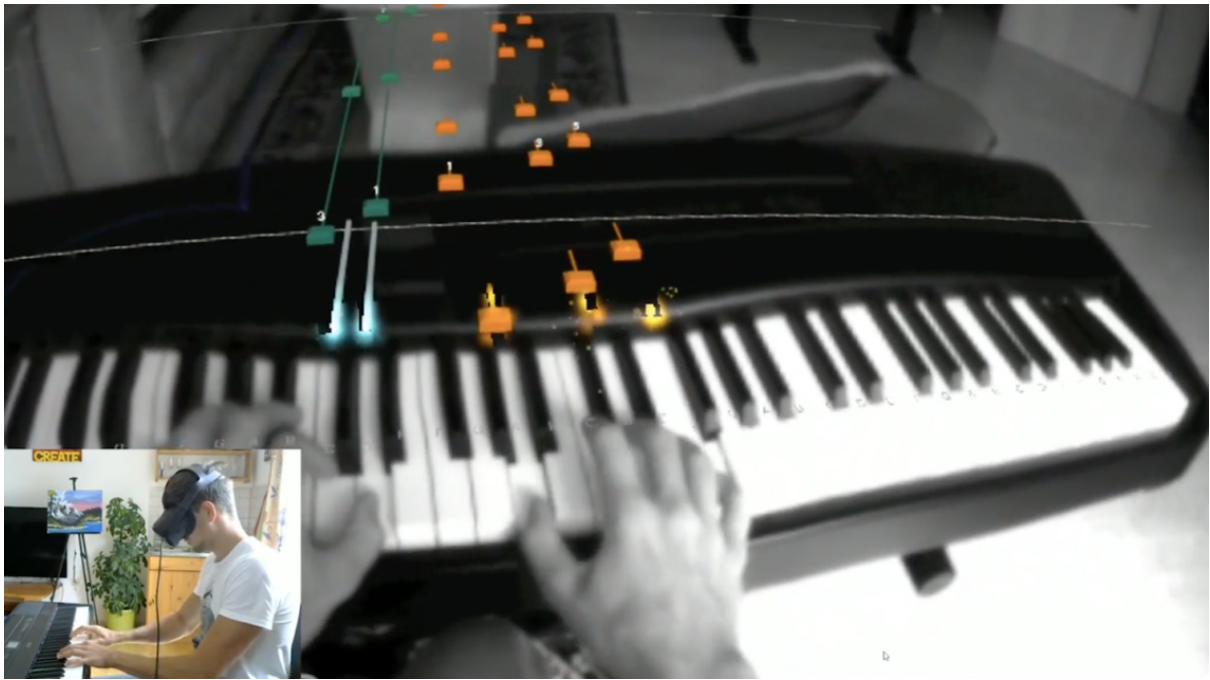


Figure 1: PianoVision, Third Person & First Person (Player) View (Melnick, 2021)

The difference between “PianoVision” and “AR Jam” is that PianoVision’s virtual user interfaces are mostly focused on learning the piano in a game-like way, while “AR Jam” does not teach users to play the piano, but creates an environment to craft new sounds and design new ways to play music.

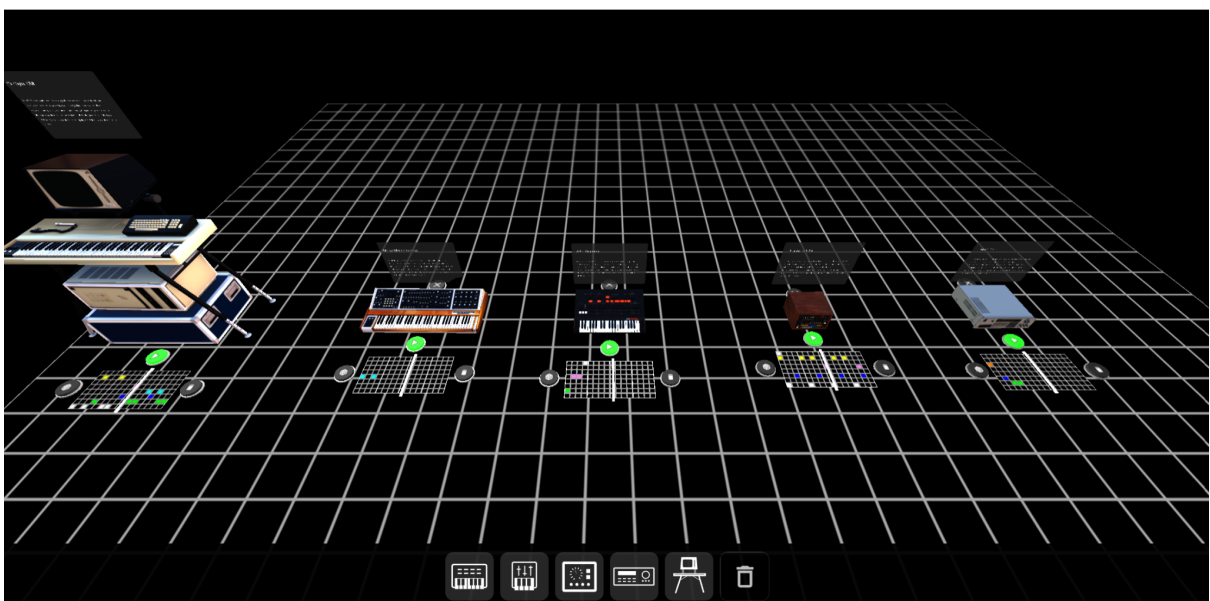


Figure 2: Screenshot of AR Synth (Google Arts & Culture)

“AR Synth” (Google Arts and Culture) influences the synthesizer implementation in “AR Jam”. The web app features five virtual instruments that are all classic synthesis and sampling machines. The instruments are “Moog Memorymoog Synthesizer”, “APR Odyssey Synthesizer”, Roland CR-78 Drum Machine”, “AKAI S900 Floppy Disk Sampler”, and “Fairlight CMI Music Station”. While the samplers play pre-recorded samples when the user asks it to play notes, the synthesizers in the environment all compute multiple oscillations to produce sound. The classic synthesizer from the 1980s, “Moog Memorymoog”, is featured in the experimental app. It runs multiple oscillators at the same time, and the user can adjust the frequency spread between the oscillators. While being an augmented reality software, “AR Synth” cannot be played by pressing either physical keys or virtual keys. Each of the instruments in “AR Synth” uses a tool/method called sequencer to map the notes according to the user’s will. Sequencers often appear as 2D grids with their horizontal axis representing time and their vertical axis representing notes, which is exactly how “AR Synth” implements it. The user clicks on one of the blocks in the grid to place a note on a specific pitch and time. With each tick of the universal metronome, the sequencer’s line moves to the right horizontally, indicating that another tick has passed and the next notes assigned to the tick now is played (see Figure 3)

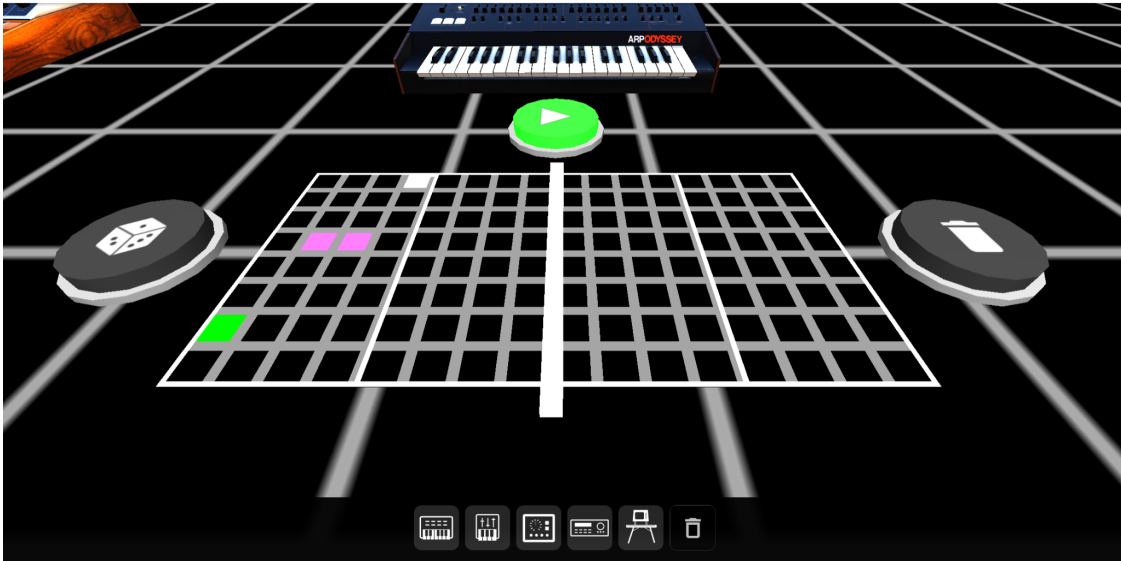


Figure 3: Sequencer Implementation of “AR Synth” (Google Arts & Culture)

The goal of “AR Jam” is to combine the ideas of the two programs and add in networked musical collaboration systems. “AR Jam” is aimed toward a low network latency collaborative experience while providing users an opportunity to see others’ body language and play music together.

Design and Implementation

On the non-networking level (running on one VR headset, single user), the structure of “AR Jam” can be seen in Figure 4. A USB cable connects the piano keyboard to a computer, where a small Python program is run. The program receives the MIDI signal when a key is pressed/released and, using network protocols to transmit the MIDI Signal to the headset. One of the instruments running in “AR Jam” receives the signal, and from the signal extract the key pressed/released, the strength the player uses to press the key, and the status of the key (pressing or releasing). The AR instrument currently running will then play/unplay the pitch specified by the MIDI signal.

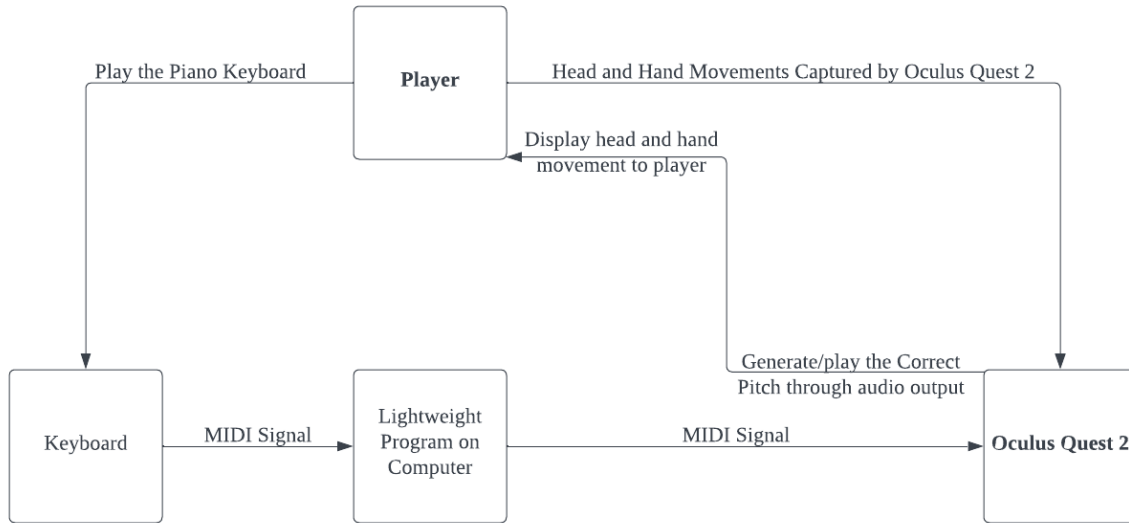


Figure 4: “AR Jam” Structure

The VR headset, while doing all the above tasks, is also keeping track of the player’s head and hand movements. This makes it possible for the player to manipulate in-game objects and instruments (turning them on/off, adjusting the timbre).

Being a networked application, “AR Jam” also sends all of the information to other headsets so multiple players can play the game at the same time.

In Figure 5, we added an extra step in which the headset synchronizes information (head and hand movements, in-game objects’ locations, sound) to another headset. The two headsets run the same environment concurrently so that the players can see each other in the same environment and experience the same sound, and objects in the environment. The process diagram of the other headset is omitted, since all the headsets are involved in the exact same structure.

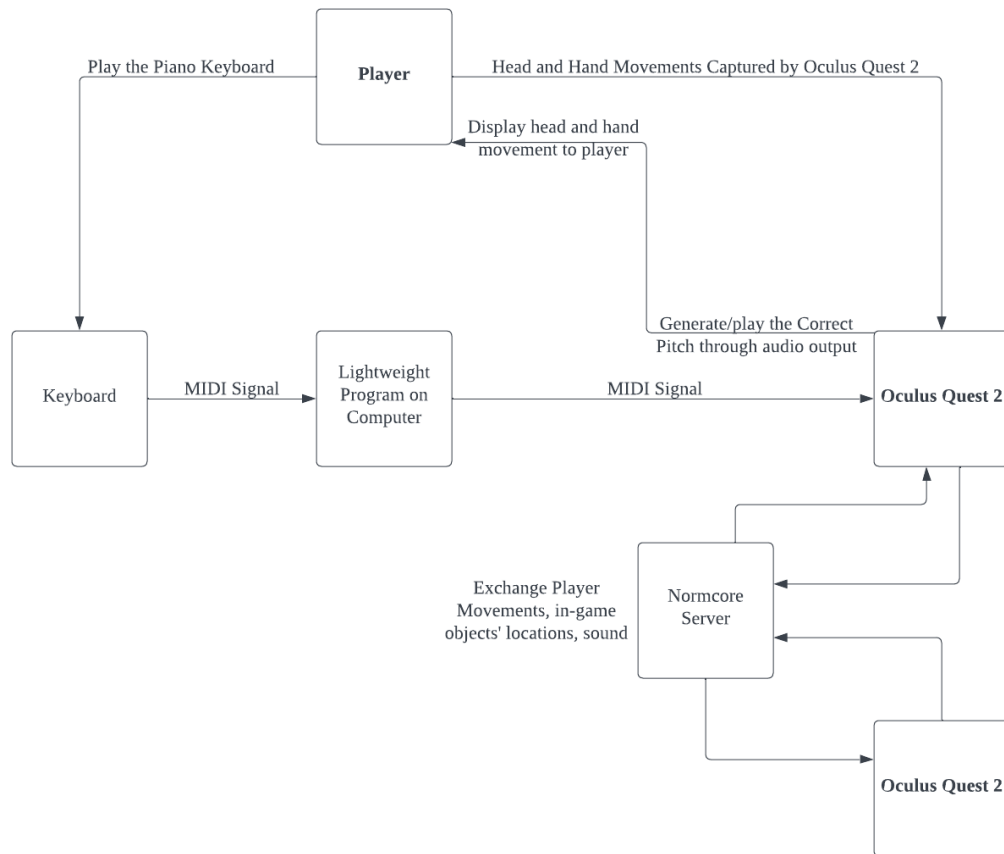


Figure 5: System Diagram with Added Networking Process

Components

This section will be separated into several different parts in which we will discuss the different tools and implementations used to build “AR Jam.”

MIDI Signal

As mentioned in the Design and Implementation section, there are two main hardware components used in this project. The first one is the MIDI keyboard. A lot of people tend to mix up the two types of hardware: electric piano and MIDI keyboard. While an electric piano

can be played on its own (without connecting to a computer), a MIDI keyboard cannot. The MIDI keyboard can be thought of as a controller (input device), like a computer keyboard, but looks like a piano. Each key, when pressed or released, tells the computer a specific value, called a MIDI signal. The MIDI signal contains several key components:

1. Status: signifies the status of the note (pressing/releasing)
2. Note number: each note has a corresponding number
3. Velocity: the strength that the user uses to press the key

Please see Figure 6 and Figure 7 for formats and examples of MIDI signals.

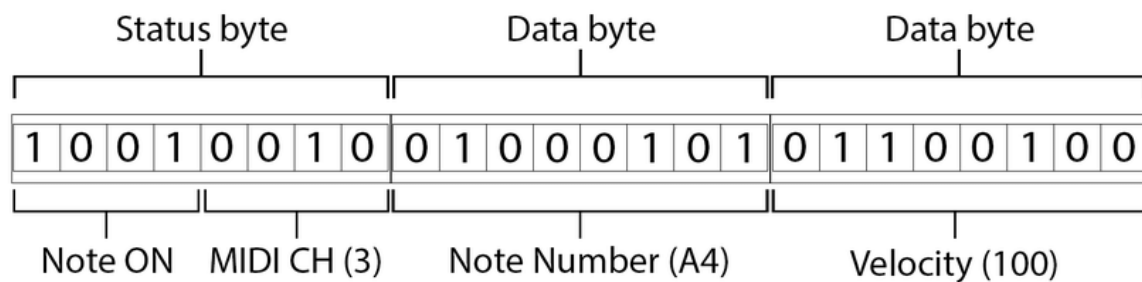


Figure 6: The MIDI Signal (Breve, Cirillo, Desiato, Cuofano, 2020)

```
[144, 60, 67, 0]
[128, 60, 0, 0]
[144, 60, 66, 0]
[128, 60, 0, 0]
[144, 52, 45, 0]
[144, 59, 43, 0]
[128, 59, 0, 0]
[128, 52, 0, 0]
```

Figure 7: Example of MIDI Signals
Format: [note on/off, Key Number, Velocity, Channel]

Oculus Quest 2

The other important hardware used in “AR Jam” is an Oculus Quest 2 (Meta, 2020) headset. It features displays for the two eyes, stereo sound, and a sensor for tracking the motion of the user’s head.



Figure 8: Oculus Quest 2 (Meta, 2020)



Figure 9: Displays and Speakers

The Oculus Quest 2 also has front facing black and white cameras so that the user can see through the headset when playing AR games. These cameras allow the Quest 2 to use machine learning algorithms to track the player's bare hand movements, specific to the movements of each finger.



Figure 10: Four Front-Facing Cameras

Unity Game Engine

Various software components were used in “AR Jam.” Unity (Unity Software, 2005) is a game engine that is developed by Unity Technologies. Simply put, Unity is a collection of tools to put objects into an environment to make up 2D or 3D game scenes. There are a lot of pre-built objects and effects that game developers can use to create the game scenes to what they desire. At the same time, Unity has tools that support AR and VR development for Oculus Quest 2, so I chose Unity to be the primary tool when building “AR Jam”.

In the game engine, it features a scene view in which the users can put objects anywhere in the scene, and specify the initial location of the players when launching the game.

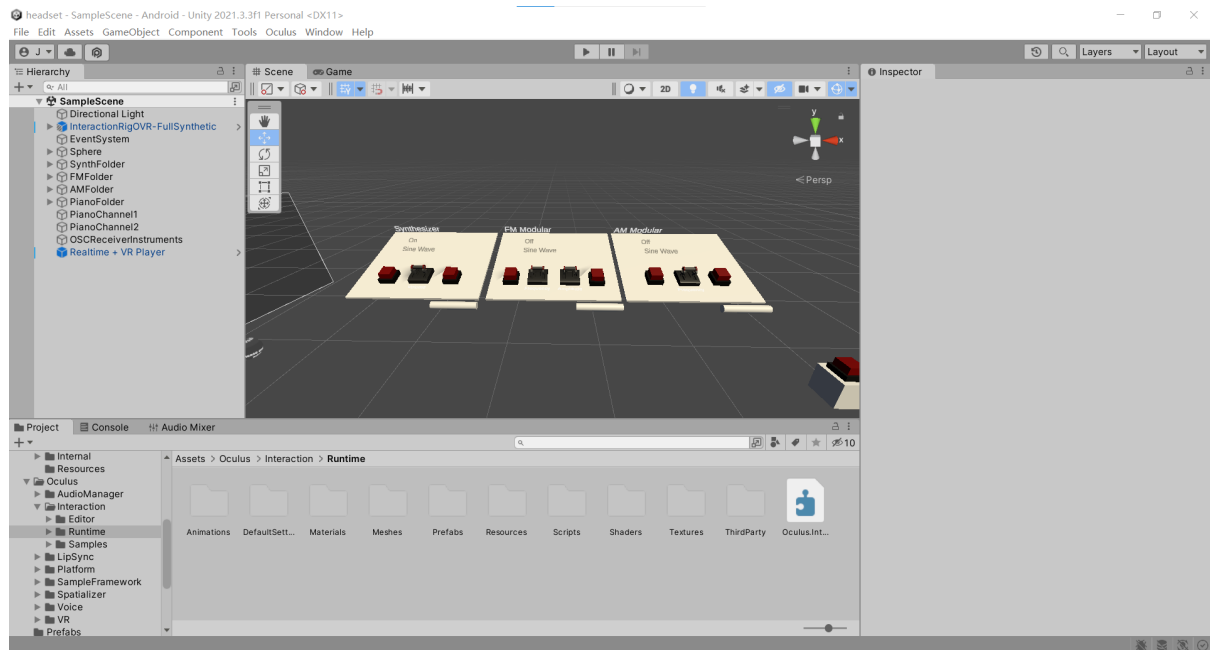


Figure 11: Unity Graphical User Interface

The objects are managed in a hierarchical fashion, in which there are child objects attached to parent objects. When we change the status of the parent object (location, active/inactive, etc.), the child object changes with the parent object relatively. The child object has a location on the X, Y, Z axis relative to the parent object. If an object does not have a parent, its location is the actual universal location relative to the central point of the environment.

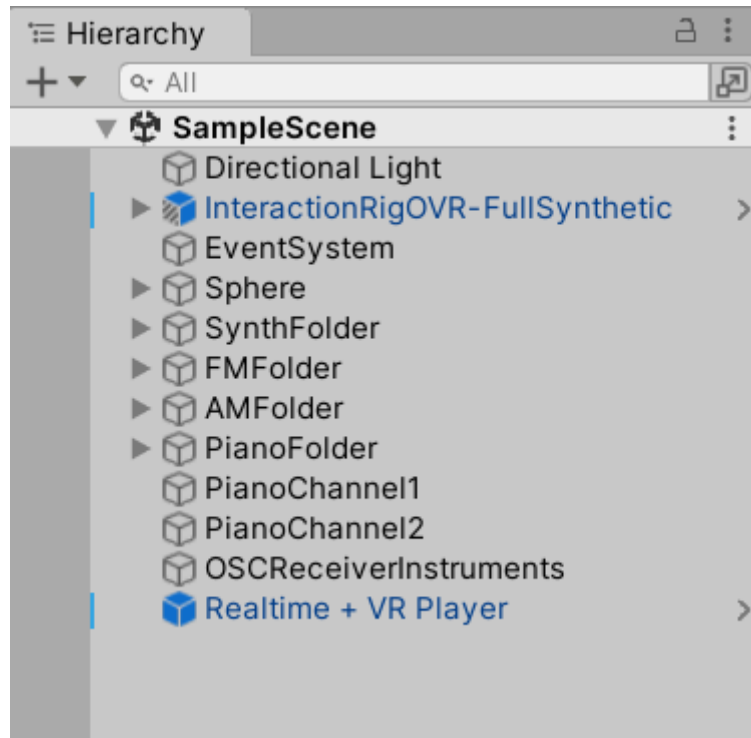


Figure 12: Scene Hierarchy

Open Sound Control

The wireless transmission of the MIDI signal from the computer to the headset is realized using OSC (Open Sound Control) (Wright, Freed, Momeni, 2003). OSC is developed using the UDP protocol (Postel, 1980), which is a fast protocol that does not guarantee the reception of the signals. The computer program's OSC is from the Python library "pythonosc" (PyPI, 2023). On the headset end the project uses "extOSC" (Sigalkin, Finnegan, 2018), which is a Unity package developed by Vladimir Sigalkin. While the computer software mainly sends the MIDI signal received via USB to the headset using OSC, it also makes the user choose which instrument the signal gets sent to.

```

17     # create client
18     client = SimpleUDPClient(server_ip, port)
19
20     # ask user which instrument to use
21     while True:
22         instrument = input("Choose your instrument: (piano / synthesizer)")
23         if instrument != "piano" and instrument != "synthesizer":
24             print("Not a valid answer. ")
25         else:
26             print("You have chosen " + instrument)
27             break
28
29     # infinite loop to detect midi input, if detected, print
30     try:
31         while True:
32             if midi_input.poll():
33                 message = midi_input.read(num_events=16)
34                 print(message[0][0])
35                 pass_message = message[0][0]
36                 client.send_message("/") + instrument, pass_message)
37     except KeyboardInterrupt as err:
38         print("Stopping...")
39

```

Figure 13: Portion of MIDI Sender's Code

Musical Instrument 1: Vocal Chorus

There are currently two instruments in “AR Jam”. While the two are both played by a MIDI keyboard, the timbres and the sound-producing methods are largely different. One of the instruments in “AR Jam” is the vocal chorus. It is a polyphonic instrument that can play different pitches of sampled vocals at the same time. In the Unity environment, the vocal chorus is just one game object with a C# script attached to it. Once the environment launches, the object generates 88 child objects, each of which is an audio output and plays a unique sample of a vocal. When the vocal chorus instrument receives the MIDI signal from the computer, it checks which notes are being sent and tells the corresponding child objects to play the samples.

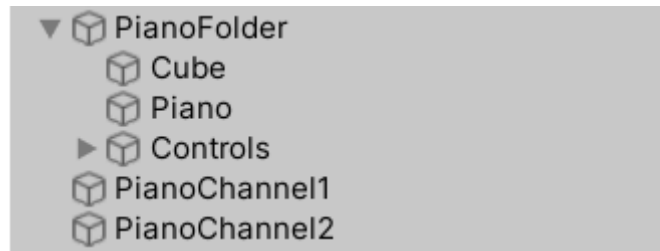


Figure 14: Vocal Chorus Hierarchy

Once initialized, notes are generated under the Piano object. Cube is the base of the virtual button for users to turn on the vocal chorus, and Controls contains the virtual button.

Oscillator

A digital oscillator is a computer-generated wave form, in which the computer uses different formulas to generate different types of waveforms. There are generally four types of waveforms in my oscillators: sine wave, square wave, triangle wave, and sawtooth wave.

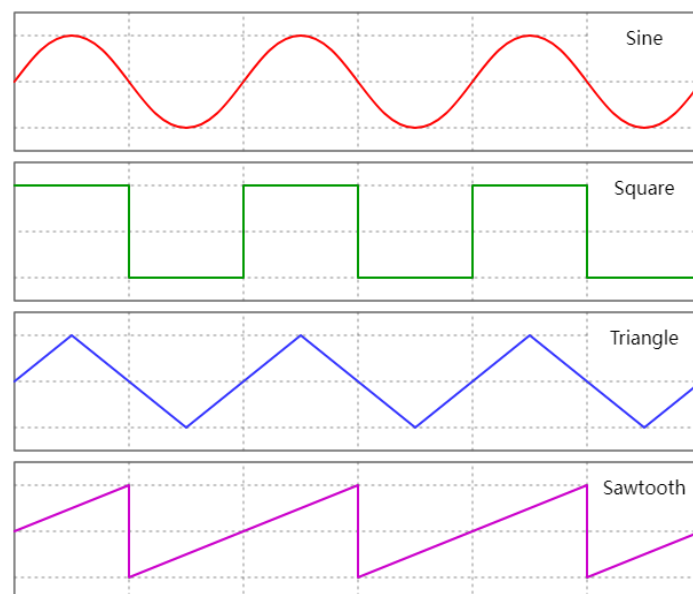


Figure 15: Four Different Waveforms

When producing digital oscillations, we need to consider the sampling frequency, which is the number of samples created every second. In Unity, the default sampling frequency is 48000 samples per second as opposed to the conventional 44100 samples per second. This means that in Unity, the program produces 48000 samples per second, and combining each of

the samples we simulate the oscillation.

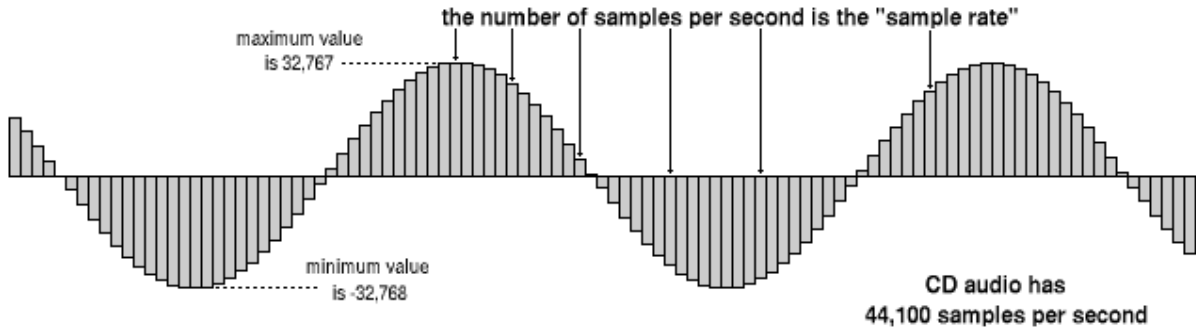


Figure 16: Sampling (Hydrogenaudio Knowledgebase)

Each of the four types of waveforms are produced using a unique Python function. Take a sine wave for example, we need to first calculate the increment of phase of each sample using Equation (1), which is added to each sample's phase value. For each sample, we then add the same increment value to the phase value, and we use Equation (2) to calculate the actual sample using the unique phase value. Putting together all of the samples ultimately simulates the sine wave. Similarly, we can add a sign function to Equation (2) to make the equation a square wave, as seen in Equation (3). The sign function is shown in Equation (4).

$$\text{increment} = \text{frequency} \cdot 2\pi / \text{samplingfrequency} \quad (1)$$

$$\text{sample} = \text{amplitude} \cdot \text{sine}(\text{phase} + \text{increment}) \quad (2)$$

$$\text{sample} = \text{sgn}(\text{amplitude} \cdot \text{sine}(\text{phase} + \text{increment})) \quad (3)$$

$$\text{sgn}(x) = \begin{cases} -1 & \text{for } x < 0 \\ 0 & \text{for } x = 0 \\ 1 & \text{for } x > 0. \end{cases} \quad (4)$$

```

92 // Square wave
93 private void square(float[] data, int channels)
94 {
95     increment = frequency * 2 * Math.PI / sampling_frequency;
96     for (int i = 0; i < data.Length; i += channels)
97     {
98         phase += increment;
99         data[i] = (float)(amplitude * Math.Sign(Math.Sin(phase)));
100
101         if (channels == 2)
102         {
103             data[i + 1] = data[i];
104         }
105
106         if (phase > (2 * Math.PI))
107         {
108             phase = 0.0;
109         }
110     }
111 }

```

Figure 17: Function for Square Wave

Musical Instrument 2: Modular Synthesizer

The instrument currently implemented that uses digital oscillators is a modular synthesizer. This instrument consists of three oscillators, each serving a different function. Each oscillator's frequency and amplitude can be adjusted on-the-go. The first oscillator, the one that produces sound, is the main synthesizer. When the synthesizer receives a MIDI signal, it converts the MIDI key number into the correct pitch using a correlation shown in Equation (5), and adjusts its own frequency to that desired pitch.

$$f = 440 \cdot 2^{(n-69)/12} \quad (5)$$

where n is MIDI note number and f is frequency (this equation assumes an equal tuning system with $A4 = 440$ Hz). The second oscillator, frequency modulator, is mapped to the frequency of the synthesizer to use its wave to modulate the frequency, thus modulating the pitch of the synthesizer. The third oscillator, amplitude modulator, is mapped to the amplitude of the synthesizer and thus uses its wave to modulate the amplitude. The use of the two different modulators can result in novel and interesting timbre. By changing the waveforms

of the modulators, the frequency and amplitude of the modulators, novel and interesting timbres can be produced. The user can even combine the two modulators by turning both on to result in a synthesizer with both its frequency and amplitude modulated simultaneously by similar or different waveforms.

Modular Synthesizer User Interface

Because of the many possibilities that the modular synthesizer provides, a simple and effective user interface needs to be developed. The user interface contains three modules, one for each of the oscillators. The synthesizer module contains two buttons, one is an on/off button, the other is a button to switch between different waveforms. There is also a throttle that controls the volume (amplitude) of the wave. The frequency modulator module has an on/off button, a throttle that controls the amplitude of the wave, a throttle that controls the frequency of the wave, and another button that controls the waveform. The amplitude modulator module also has an on/off button, a frequency throttle, and a waveform button. Ultimately, the user puts their physical MIDI keyboard in front of these three modules so that all the virtual controls on the modules are easily reachable when the user is physically playing the keyboard.

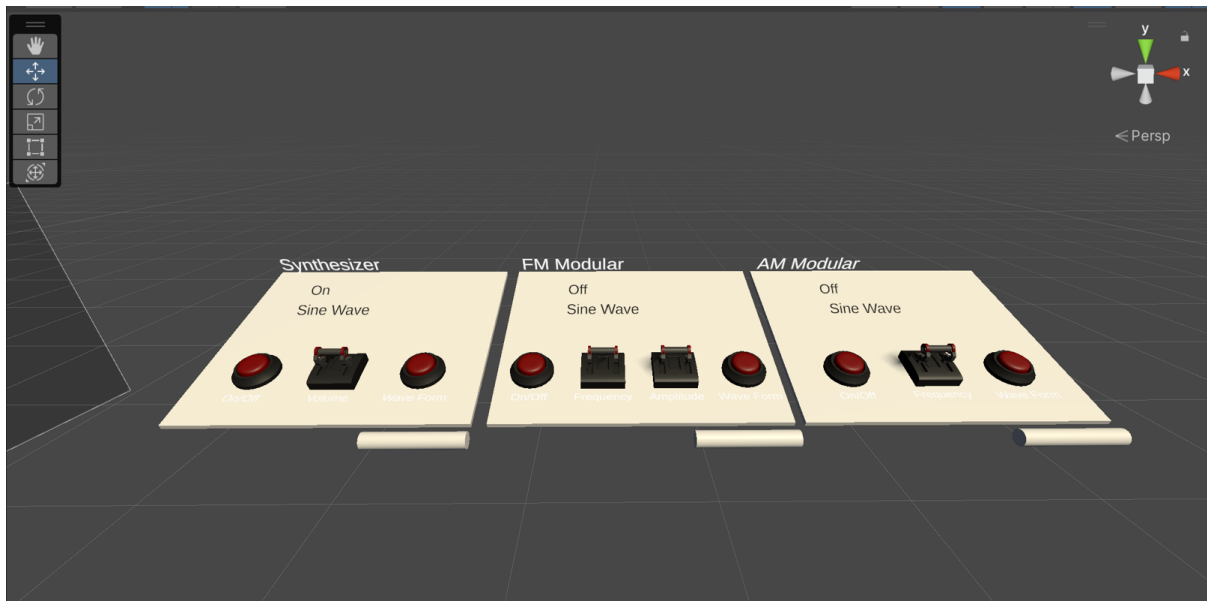


Figure 18: Modular Synthesizer Virtual Interface

To make interacting with the virtual controls easy, handtracking is implemented. The idea is to avoid having to reach for a controller whenever the user wants to adjust the virtual controls. Oculus Interaction SDK (Meta Platforms, Inc.) is used for hand and head movement tracking.

The InteractionRigOVR from Oculus Interaction SDK (Meta Software, Inc.) is a rig that contains everything from the player's camera, hands, and controllers. Since in "AR Jam" only handtracking is implemented, Controller Hands (object that tracks controller movements) is disabled. The child objects underneath OVRCameraRig are mainly prefabs, which are the models that represent how the objects may look like when rendered (see Figure 19).

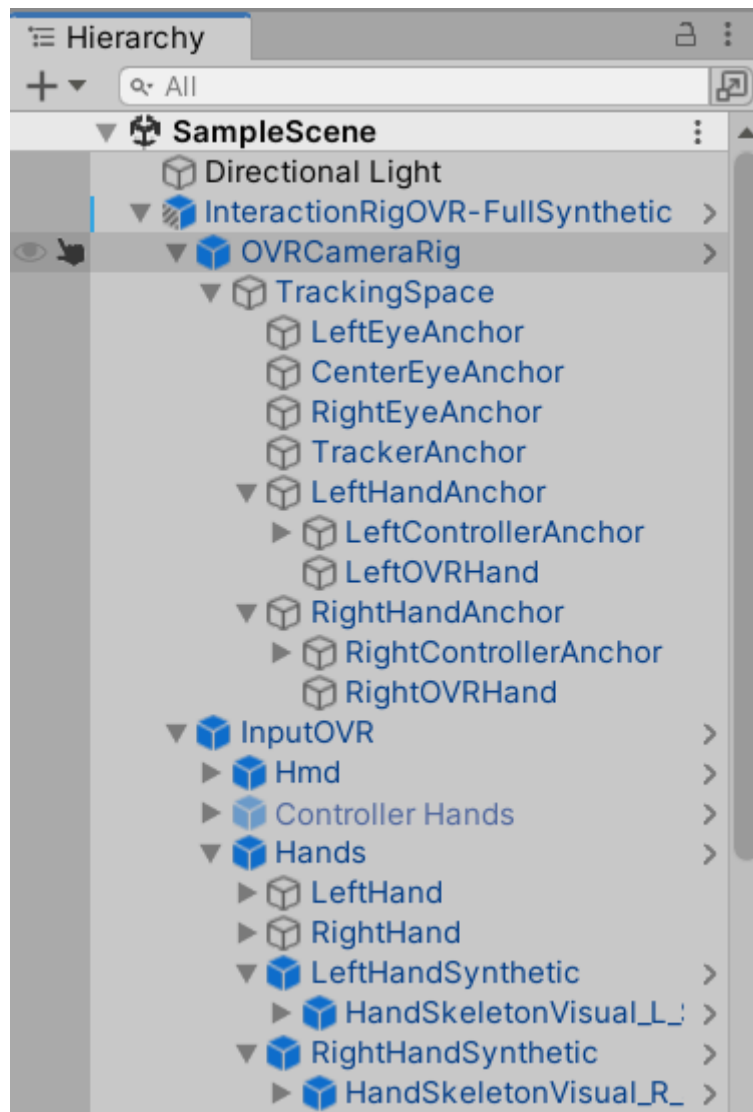


Figure 19: InteractionRigOVR Hierarchy

Network

The network part of the program connects headsets together so that multiple users can be in the same virtual environment at the same time. This project uses the Unity package, Normcore (Normal), to send the interaction data between users. Normcore hosts servers to run clients' games, which means that when using their plugin, "AR Jam" is uploaded to their server and hosted there. In result, all of the users in this environment are the clients who send and receive information from their server.

Process

Putting together all the components in “AR Jam” was not an easy task due to my unfamiliarity with the Unity game engine, the packages, and the C# language which is used to build the software. Because there have not been similar AR modular synthesizers produced before, I had nowhere to search for a solution to build a working AR synthesizer. A lot of the attempts in creating the virtual environment resulted in compatibility issues between different Unity packages and the hardware.

In order for “AR Jam” to be an augmented reality software, passthrough technology is required. Passthrough is when a headset uses its front-mounted cameras to allow the user to look through the headset at their surroundings. When setting up the scene, the Oculus Passthrough API was used. Because of undisclosed privacy issues and other concerns, Oculus has decided that when a software is running on Unity (instead of on the headset) with passthrough enabled, the background may not be seen and is all dark. This means that the developer cannot test the software by running the software on the computer and displaying on the headset every time something new is added, but they need to first connect the Quest 2 to the computer via a USB-C to USB cable, build the project into an APK (Android Packaging Kit) file, and use the SideQuest (Oculus) computer platform by Oculus to sideload the APK file onto the headset. Most independent developers use this method because it is the most official and safest way to sideload their own project. This process of testing the software made the development process a lot slower than it could have been. We can conclude that testing my AR software on Oculus Quest 2 was not the most efficient process.

When developing the buttons for the synthesizer interface, round buttons were initially used (see Figure 18). These buttons were developed (I/O Labs). The buttons used

Unity's built in physics simulation for the press and rebound motion. This use of physics seems natural, but when moving the virtual modules to different locations, the buttons also move horizontally and get stuck with their bases, and thus stop functioning. After some research, it was clear that using physics in these buttons was not the best choice, so I implemented new buttons that use the locations of hands to determine how far down the buttons are pressed. If the buttons reach a certain distance, they will output a message. Physics collider was also added with this implementation so that the models of the user's hands could not go through the buttons.

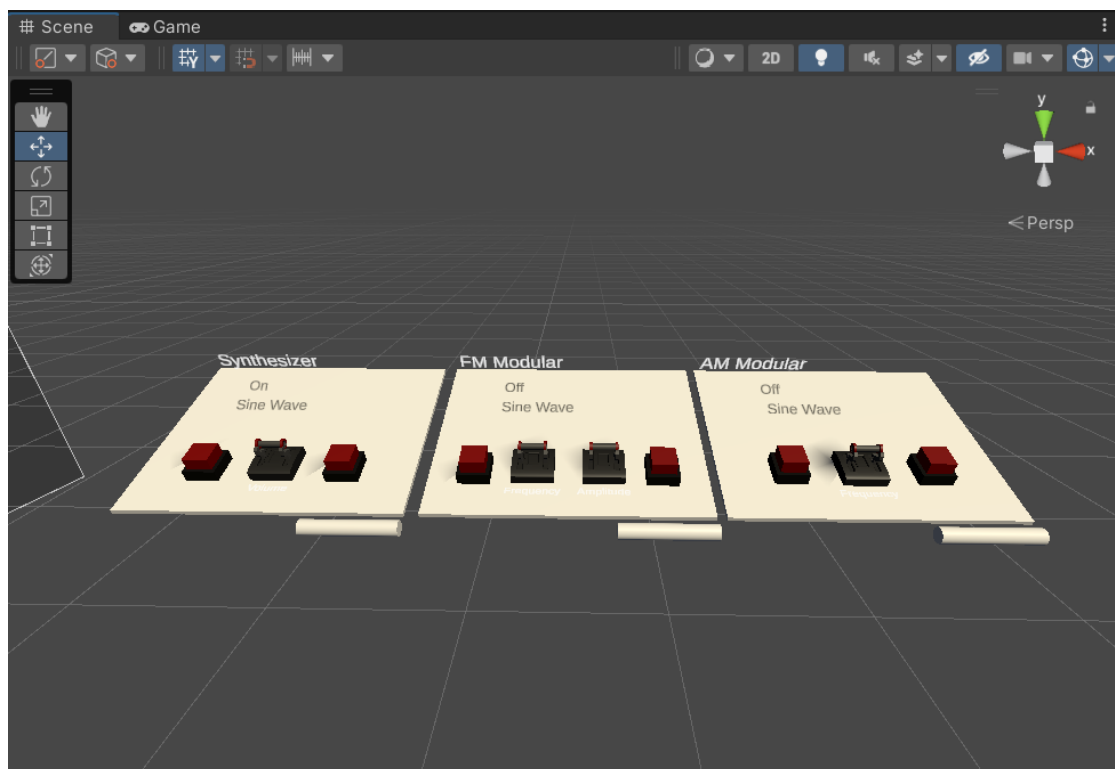


Figure 20: Square Buttons

The implementation of networked handtracking using Normcore was half completed. The Normcore plug-in appears as a game object in the Unity scene. The object accepts two parameters, one is the prefab (model) of the head and the other one is the prefab of the hands. After trying multiple ways to provide Normcore with a hand prefab, I discovered that Oculus Interaction SDK's hand tracking is not compatible with Normcore. The fingers' data are not

transmitted through Normcore through their servers, but only the general location of the hands are transmitted. Furthermore, in order for the location of the hands to get transmitted, two models of hands need to be stucked on the user's two arms so that the two models provide the general locations of the two hands. Fortunately, there are ways to disable the rendering of the two models.

One of the most unexpected issues that I suffered with was bugs in the tools that were used in "AR Jam." Because of some unknown conflicts in the development of SideQuest, the software was having some minor issues for developers. The issue that I encountered when developing "AR Jam" was that sideloading was not working properly. When I successfully built my project into an APK file and tried to sideload the file, SideQuest said that the process was successful but the APK file did not appear on the headset. I consulted with technology support at SideQuest (Figure 21) and ended up having to reset my headset to factory settings. This method ultimately worked but resulted in deleting every software in the headset. The same problem did not rise again during my later development, but there is no assurance that it will not occur again.

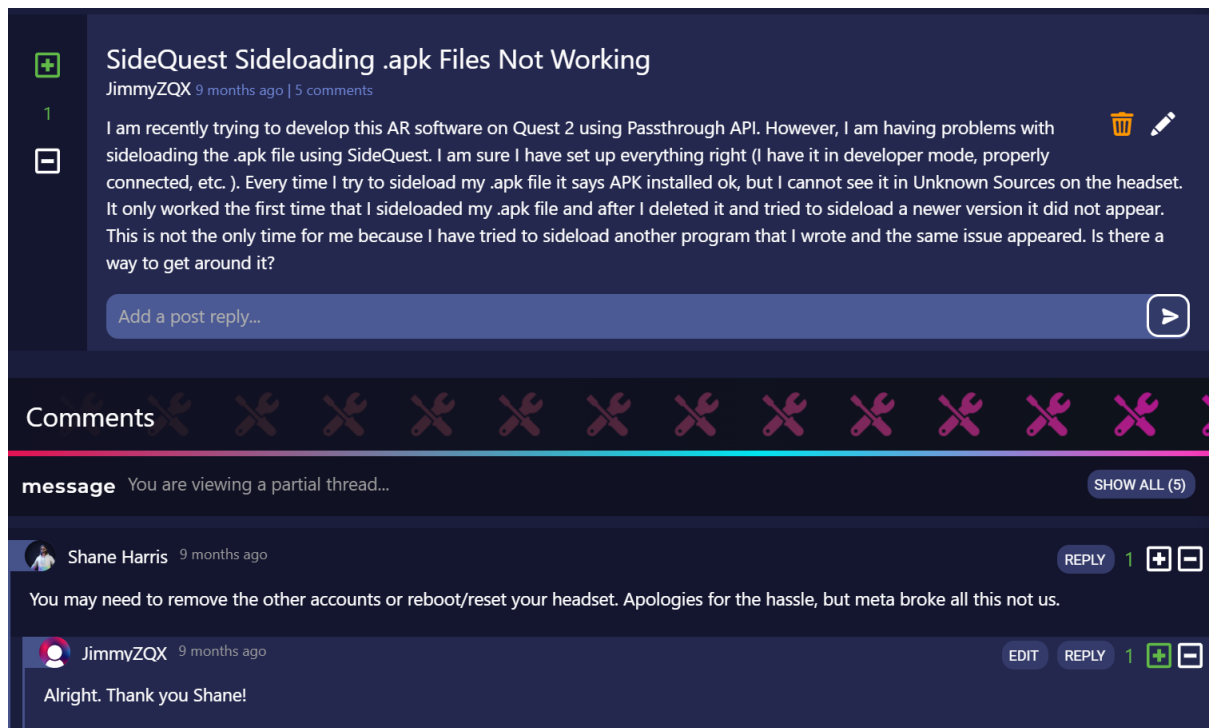


Figure 21: Asking SideQuest Technology Support

Results and Discussions

The resulting “AR Jam” software provides multiple new materials and perspectives to the AR musical instrument making area. The general project explores the idea of musical instrument making in an augmented reality environment, how effective this new way of music-making may be, and how people can interact with such software.

At the same time, after a thorough search on the internet, it is almost certain that the modular synthesizer in “AR Jam” is the only synthesizer in an augmented reality environment that has the ability to use modulators and change their frequencies and amplitudes. This means that it is the closest to a real synthesizer in terms of playability and timbre controllability. It is also the only synthesizer that breaks the boundary between the real world and the virtual world, in which the user plays the notes using a MIDI controller in the

real world but produces the sound in the virtual world. In summary, “AR Jam” is a software that approaches the act of making music a mixed-reality approach. In this software, the musical instruments are hybrid between virtual and physical. The introduction of the virtual environment in music making thus opens new possibilities of producing sound and working with other musicians. A video of this project can be watched here: https://www.youtube.com/watch?v=_QUZKdOx3rg&t=43s

While this is a creative project without any large number of quantitative results, there are a few points that future developers working on similar software can take notes of.

1. There is notable (0.1s ~ 0.2s) latency between the MIDI signal being sent out and when there is audio output. This latency is present with both the vocal chorus and the modular synthesizer, with synthesizer having a larger latency. This latency is even present when I am testing the MIDI transmission locally, with 127.0.0.1 as the destination IP address. This likely suggests that the OSC which uses the UDP protocol and is implemented in “AR Jam” is not as effective as expected. Further research is needed to find out exactly what is causing this surprising problem.
2. As mentioned in the first point, the two virtual instruments, the vocal chorus that plays samples and the modular synthesizer that generates digital oscillations, have a difference in latency, with the modular synthesizer being slower when producing sound. Furthermore, the latency gets worse when the other two modulators are turned on. This likely signifies that the Oculus Quest 2 headset has limited computational power and cannot compute the digital oscillations in a low latency fashion. One optimization method is called wavetable synthesis. In this method, the waveforms are pre-made, meaning that the synthesizer can extract a sample from the wavetable every 1/48000 seconds and play the sample. The synthesizer no longer needs to calculate 48000 samples every second. A benefit to changing the synthesis method to

wavetable is that users can draw the wavetable themselves. The waveforms can be whatever the users want them to look like and sound like, adding a new dimension of possibilities in shaping sounds in AR environments.

3. As mentioned in the Process section, the hand tracking model is not yet developed. Further research is needed on finding what is compatible with Normcore. Oculus Interaction SDK does not work with Normcore. There are previous projects (for example: <https://www.youtube.com/watch?v=Akw7fVgmcLc&t=2s>) by other developers that have realized networked handtracking, so it should be possible.
4. Game engines like Unity update frequently. Searching for methods in developing certain features may be difficult because of the updates. A newer version of Unity sometimes means the removal of certain packages. This may be complemented by an addition in functionality to one of the existing packages or it may not. For information about certain packages when something is changed, consulting with Unity documentation is the most effective way to understand the update. For Unity documentation see here: <https://docs.unity3d.com/Manual/index.html>

In general, the development of a mixed-reality music software needs to always adapt to the technological advancement of the headsets and the related hardware. Developers need to consider the computational capabilities of VR headsets early in the development process. In my case, using a wavetable might be much more efficient than using the traditional synthesis method. On the other hand, getting to know the game engines, the operating systems of the headsets, their versions and dependencies are the most important things before developing the software. This gives the developer a sense of what is achievable in what ways and what is not achievable. Sufficient research in software and hardware dependencies is necessary.

Reference

AR Synth - Google Arts & Culture. (n.d.). Google Arts & Culture.

<https://artsandculture.google.com/story/ar-synth/7AUBadCIL5Tnow?hl=en>

Breve, B., Cirillo, S., Desiato, D., & Cuofano, M. (2020). Perceiving space through sound: mapping human movements into MIDI. *Distributed Multimedia Systems*.

<https://doi.org/10.18293/dmsviva20-011>

David's MIDI Spec. (n.d.).

<https://www.cs.cmu.edu/~music/cmsip/readings/davids-midi-spec.htm>

Davidson, J. W. (2012). *Bodily movement and facial actions in expressive musical performance by solo and duo instrumentalists: Two distinctive case studies*.

Psychology of Music, 40(5), 595–633. <https://doi.org/10.1177/0305735612449896>

Documentation | Normcore. (n.d.). <https://normcore.io/documentation/>

File:Digital wave.png - Hydrogenaudio Knowledgebase. (n.d.).

https://wiki.hydrogenaud.io/index.php?title=File:Digital_wave.png

I/O Labs. *VR Ready Controls [Button, Leaver, Knob] | Modeling | Unity Asset Store*. (2016, October 1). Unity Asset Store.

<https://assetstore.unity.com/packages/tools/modeling/vr-ready-controls-button-leaver-knob-65520>

Interaction SDK Overview | Oculus Developers. (2022). Oculus.com.

<https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/>

Melnick, K. (2021, August 31). *Oculus Quest App “Magic Keys” Teaches You Piano Using AR - VRScout*. VRScout.

<https://vrscout.com/news/oculus-quest-magic-keys-teaches-you-piano-ar/>

Meta Platforms, Inc. (n.d.). *Interaction SDK Overview*. Meta Quest Developer Center.

<https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/>

Oculus Quest 2. *Meta Platforms, Inc.* VR Headset, 2020.

Postel, J. (1980). User Datagram Protocol.. *RFC*, 768, 1-3.

python-osc. (2023, January 15). PyPI. <https://pypi.org/project/python-osc/>

Reid, Z. (2022, August 4). *PianoVision - An Augmented Reality Piano Platform*. (n.d.). <https://www.pianovision.app/>

SideQuest: Oculus Quest Games & Apps including AppLab Games (Oculus App Lab). (n.d.). <https://sidequestvr.com/>

Sigalkin, V., Finnegan, T. (2018). I. (n.d.). *GitHub - Iam1337/extOSC: extOSC is a tool dedicated to simplify creation of applications in Unity with OSC protocol usage*. GitHub. <https://github.com/Iam1337/extOSC>

Unity. *Unity Software Inc.* Game Engine, 2005

Wright, M., Freed, A., & Momeni, A. (2003). OpenSound Control: State of the Art 2003. *Zenodo (CERN European Organization for Nuclear Research)*. <https://doi.org/10.5281/zenodo.1176575>