Music Visualizer and Synesthesia Simulation in Virtual Reality

Celia Choy
Middlebury College
Middlebury, VT
celiac@middlebury.edu

Maya Reich
Middlebury College
Middlebury, VT
mreich@middlebury.edu

ABSTRACT

In this paper, we present our senior seminar project, a Virtual Reality application that simulates synesthesia and is a music visualizer. This project falls into several fields of computer science: VR, music visualization, game development and user interface design. We start with a general definition of visualization and the emerging field of data visualization in computer science. Virtual Reality has provided a new medium to explore data visualization techniques, and thus an increase in the number of music visualizers. Secondly, we describe synesthesia and explain the uses of music visualizers. We provide a brief overview of projects related to ours such as VR music visualization apps and synesthesia simulators and explain potential psychological benefits of using our application. In methods, we elaborate on hardware integration steps, including putting together an Android development environment in order to make our application compatible with Oculus Go which required setting up Unity for Android and integrating Unity and Oculus Go. We explain how we designed a user interface that allows for a customizable VR environment and how we mapped specific features of audio signals detected by the VR headset's microphone to visuals.

Keywords

CS701; Synesthesia; Music; Unity; Oculus Go; VR; Audio Spectrum; Visual Effects

1. INTRODUCTION

Visualization is defined as the act or process of interpreting in visual terms or of putting into visual form. The creation of abstract visualizations of data has been practiced for centuries in order to help people interpret data in different ways, allowing for many uses such as increased memory retention or heightened potential for creative thinking. Data visualization, the study of visualizing relationships in numerical data, has become an increasingly popular field of study with the rise of software. Music visualization is a form of data visualization in which features of sound such as pitch and loudness are converted into discrete values that are used to create a visual representation of the sound.

Virtual Reality is a new technology in which a user wears a headset over their eyes and the headset simulates a 360 degree 3D environment by displaying an image or animation on a screen through the lenses. The headset detects the movement and rotation of the user's head to display a 360 degree virtual world. Since its invention, its uses range far and wide, including being used for training for specialized jobs, gaming and creating a distraction for patients undergoing painful treatments. VR is a useful technology because it tricks the brain into thinking that a simulated environment is reality without having to actually be in that environment. The invention of Virtual Reality has allowed for plethora of opportunities to visualize data in new ways.

Since VR creates a fabricated environment, it can be used to depict scenarios and environments that may not be possible in real life. Synesthesia is a rare neurological condition that affects 1 in 2000 people in which those affected perceive sensory data through multiple senses. Many Synesthetes possess a form in which the pitch of any sound they hear appears as specific bursts of color in their vision. Thus, VR allows for an exciting opportunity to simulate this condition for those that do not have it.

In this paper, we will describe several projects and papers that have explored music visualization and Synesthesia simulation. We will explain how we created a simple user interface that allows for a customizable VR environment, how we mapped the audible range of frequencies of audio signals to a visual with a corresponding color and how volume data was used to create a visualizer that responded to changes in the loudness of sounds. We end the paper with a summary of our results and ideas for improvement and further work.

2. PROBLEM STATEMENT

Our goal in creating this application is multi-fold. Firstly, we aimed to develop a deep understanding of developing in Unity and programming in C# which were essential in solving the problem of creating an application that provides its users entertainment and stress relief.

We imagine our application being used mostly for leisure and to reduce stress. There are many instances in which professionals feel the need to escape their job temporarily or students feel the need to leave campus for a few hours in order to clear their head and remove themselves from a stressful environment. Our VR application provides an outlet for those who wish to shut off their brain for a few moments and listen to the music they enjoy, all while creating an environment in which they can "escape" their work without actually physically leaving their workplace.

When audio and visual stimuli are coordinated, it creates a more immersive music experience in which the user not only hears music and sounds, but "sees" them. Listening to music is shown to increase levels of endorphins in the brain, lower cortisol levels and boost mood. Adding an extra layer of stimulation by transforming the music listening

experience into VR boosts these mind enhancing benefits even more.

With this in mind, we created an application that stimulates the senses by synchronizing components of audio signals with visuals that dance across the environment in an aesthetic and captivating way.

3. RELATED WORK

Music visualization has become increasingly popular with the rise of VR, which has allowed for a more immersive and interactive music experience. Many music visualizers have been made including Intone, which is one that uses one's voice to create clusters of different colored blocks that move or are different colors based on the sounds the user makes.

A video named "What's It Like to Hear Colors?- A VR 360 Synesthesia Experience" on Youtube aims to illustrate what having the condition Synesthesia is like. In this video, a woman depicts what having Synesthesia is like. The video consists of a person playing violin in which specific notes correspond to visuals that flash on the screen with distinct shapes and colors. We used this video as a basis for understanding how the frequency of sounds is linked to visuals that Synesthetes see.

4. METHODS

4.1 Creating environment in Unity

We used Unity as a platform to build the 3D environment. We decided to visualize the audio in two different ways: a ring of cylinder bars indicating the audio frequency spectrum and visual effects indicating the pitch values. Skybox and terrain components in Unity were also used to set up the customizable backgrounds. Finally, we created a scene for main menu, where users can customize the backgrounds, with the Canvas.

4.1.1 A Circle of 32 Cylinders

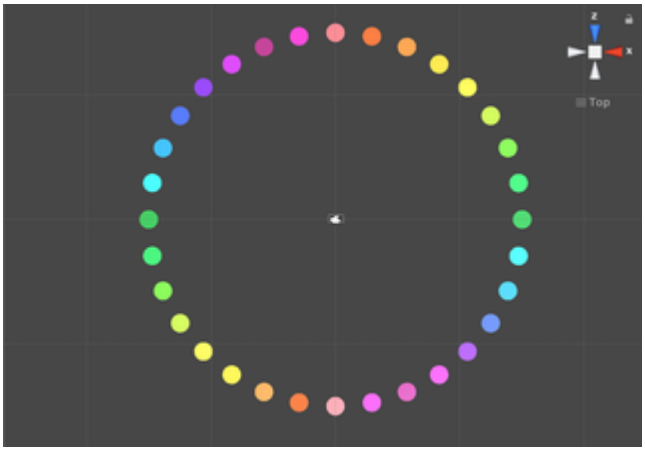
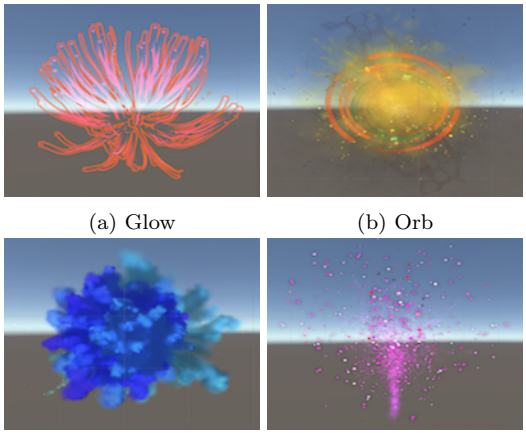


Figure 1: A ring of 32 bars

The Scene opens up to a ring of 32 bars that respond to changes in frequency and volume as shown in Figure 1. They surround the user who is placed in the center of the ring. 16 distinct colors are assigned to these cylinders: red, orange red, orange, orange yellow, yellow, yellow green, light green, green, dark green, light blue, blue, navy, purple, pink-purple, wine, and pink. Each cylinder has the same cylinder mirrored on the opposite side because we wanted to provide a user a full view of the cylinders from red to pink when facing one side. The cylinders from red to pink indicate the audio frequency spectrum from 0 to 20,000 Hz. Thus, the red bars refer to low frequency and the pink, high frequency. It will be further explained how the frequency spectrum was distributed to the cylinders in the third section.

4.1.2 Visual effects



(a) Glow (b) Orb

(c) Smoke (d) Sparkle

Figure 2: Visual Effects

The users can also observe visual effects spawning all around them. We created 4 different visual effects: we named them as glow, orb, smoke, and sparkle as shown in Figure2. These visual effects were created with Unity's particle system, which simulate fluid entities by animating a number of 2D sprites. We imported interesting particle sprites from Unity Asset Store and assigned them as particle textures. In case of orb, smoke and sparkle, we added multiple particle systems to create a visual with multiple effects. When the audio is detected and the pitch value is calculated, one of the 4 visual effects will be randomly chosen and instantiated with the color that corresponds to the pitch value. It will be further described how the color is assigned and how they are spawned in section 3 and 4, respectively.

4.1.3 Backgrounds: Skybox and Terrain

In order to give users more fascinating and realistic experience, we added more variations in backgrounds along with the bars and visual effects. A skybox, which wraps the entire scene with a panoramic texture, was used for adding realism to the scene. As the skybox's panoramic view is split into six textures representing six directions visible along the main axes (up, down, left, right, forward and backward) and rotates to match the current orientation of the camera, skybox can give the user impression that they are actually staying in such environment. We created 5 different skybox textures: pink, green, purple solar systems, bright sky, and early dusk sky.

Using terrain and shader components in Unity, we also created 5 different terrains: mountain, moon, ice, stone, and water. We sculpted and painted the mountain and moon terrains in Unity, and for ice, stone and water, we applied

the ice, stone, and water shaders we imported from Unity Asset Store.

4.1.4 Unity Canvas

After we finished designing the backgrounds and the terrains, we created Main Menu Canvas in Main Menu scene where the users can customize their own environment by choosing from 5 different backgrounds and 5 different terrains and Menu Button Canvas in AudioVisualizer scene (a main scene where a circle of 32 cylinder bars and visual effects exist) to enable users to go back to Main Menu from AudioVisualizer scene. The Main Menu Canvas in the Main Menu scene contains Background options panel and Floor options panel, where the user is given various options to set up their own environment. These panels are attached with SkyboxChanger script and FloorChanger script, respectively. In these scripts, the skybox material and floor (terrain) game object in the scripts are set as static variables so that they can be accessed from other scripts. For example, we declared skybox and floor static variables in SkyboxChanger and FloorChanger scripts as following:

```
public static Material skybox;
public static GameObject floor;
```

When the user chooses the background skybox and the floor (terrain) in the Main Menu Canvas, the static variables of skybox material and floor game object will be set to the chosen ones. When the AudioVisualizer scene starts after the user choose all the options, the Game Manager script attached to a game object called GameManager in AudioVisualizer scene will set the skybox material and instantiate the floor (terrain) object as following:

```
RenderSetting.skybox = SkyboxChanger.skybox;
GamObject floor = FloorChanger.floor;
Instantiate(floor);
```

As a result, the user will the environment chosen from the Main Menu scene.

The Menu button in the AudioVisualizer scene is rendered in the bottom of the view. The Menu button Canvas is attached with Buttons script, and we set the Button on Click() to call GoMenu() from Buttons script when the Menu button is clicked, which will load the Main Menu scene.

4.2 Building Application and Installing it to Oculus Go

Oculus is the main platform where we built the application. There are several steps that we went through to integrate Unity and Oculus, build the application that is compatible to Oculus Go, and install it into Oculus Go.

4.2.1 Setting Android Development environment

We used Oculus as our main platform to deploy our VR application. Since Oculus Go only supports Android application, we decided to use Windows machine and set up Android Development environment. We first installed Android Studio, and in Android Studio, we configured the SDK manager by installing LLDB, Android SDK Platform-Tools, Android SDK Tools 25.2.5 and NDK. We then set our environmental variables in system properties for JDK, Android SDK, and Android NDK location. The information for the file paths were copied from project structure in Android Studio.

4.2.2 Unity Setup for Android and VR

After all the required packages for Android development environment were installed, we set up Unity to build for Android. We switched our platform to Android from PC Standalone platform and set the SDK, JDK and NDK file paths to Android section in External tools tab in Unity preferences. We then went to Project Settings for the player and changed the XR Settings by checking the box next to Virtual Reality Supported. Since we are using Oculus, we added Oculus to the list of Virtual Reality SDKs.

4.2.3 Integrating Unity and Oculus

Unity provides built-in VR support for Oculus Go. To use the Oculus controllers and implement Oculus camera behavior in the application, Oculus Unity Integration package that contains prefabs, scripts for camera behavior and API for controllers to supplement Unity's built-in support was imported.

When using Oculus Go application, we don't use a keyboard or mouse as a controller but a Go controller. Thus, it is necessary to enable the interaction between the Oculus Go controller and the Canvas, which is used for Menu User Interface, in Unity by using the "UI Helpers", a prefab from Oculus Unity Integration package. The laser pointer, which is attached to UI Helpers, is added as a pointer to a OVR Raycaster script attached to the Canvas. The Graphic Raycaster component is removed from Canvas since the Canvas is not interacting with the computer mouse anymore. The Laser Pointer behavior is also set to be displayed when the laser hits the target, which is the Canvas it is interacting with.

We also used the OVRPlayerController, which includes a physics capsule and a movement system. The OVR Player Controller will act as a user, and it will collide with the objects that have colliders attached. This prefab also contains OVR Camera Rig as a child, which is a custom VR camera that replaces the regular Unity Camera in the scene. The OVR Manager script is attached to OVR Camera Rig and provides the main interface to the VR hardware. With the two Game Objects for the left and right eyes and one Game Object for tracking space, the rig also helps the users to fine-tune the relationship between the head tracking reference frame and the surrounding environment.

4.2.4 Building Application and Installing to Oculus Go

To develop the application locally for Oculus Go, we enabled Developer Mode in Oculus app. Once the Oculus Go app installed in our mobile devices is connected to the Oculus Go headset, we toggled the Developer Mode on. Then we downloaded Oculus Go ADB driver and connected the Oculus Go headset to the computer via USB. In Unity Build Settings, we added scenes that we would like to build and clicked 'build'. Once the build was completed, it generated the .APK file, and we copied the file and moved it to the same directory where ADB is installed. In the ADB window, we ran adb devices to confirm that the Go headset is detected. Once it was detected, we installed the generated .APK file to our Oculus Go headset. The application was installed in Library > Unknown Sources.

4.3 Analyzing Audio

In this section, it will be explained how the audio spec-

Table 1: MIDI and Frequency

MIDI note	Frequency (Hz)
0	8.17578125
12	16.3515625
24	32.703125
36	65.40625
48	130.8125
60	261.625
72	523.25
84	1,046.5
96	2,093
108	4,186
120	8,372
132	16,744

trum data was retrieved and used for scaling the ring of 32 bars, how the pitch value was calculated, and how the colors were assigned to the visual effects.

4.3.1 Microphone input to audio spectrum data

We first converted the microphone input into Unity's audio source. Once the audio is converted to audio source, we can use AudioSource.GetSpectrumData(float[] samples, int channel, FFTWindow window) in Unity that provides a block of current audio source's spectrum data. The audio spectrum represents a sound in terms of the amount of vibration at each individual frequency and is presented as a graph of power (decibel) of a frequency. The first parameter of GetSpectrumData is the array to populate with the requested data, the second parameter is the channel to sample from, and the third, the FFTWindow type to use when sampling. For the ring of 32 bars, we used the audio spectrum data of 512 samples and for calculating the pitch values that are used for assigning colors to the visual effects, 1024 samples were used. BlackmanHarris FFT window was used for both.

4.3.2 Populating the spectrum data to bars

We used the retrieved spectrum data to Human hearable sound ranges from 20 to 22050 Hz. We decided to represent the frequency from 0 to 20200 Hz and grouped the intensity of the frequencies found between 0 and 20200 into 512 elements. Thus, each element contains a range of about 40 Hz (20200/512).

In Table 1, we can see that as the MIDI note grows linearly, the frequency value increases 2 times from the previous one. That is, for each octave, the frequency doubles. We applied the same logic when populating the frequency elements to the ring of bars. The frequency value increases two times and the number of elements to be populated increases exponentially for every two bars. However, since we have 16 bars to populate the frequency values, we decided the adjacent bar to have the same number of elements. The result of the population is described in Table 2.

Once the population is done, the values of the elements populated to each bar will be used for calculating the total frequency intensity. The sum of the intensities will then be applied to scale the bar.

4.3.3 Analyzing the sound

For calculating the pitch value, the array of 1024 samples was used. Thus, each element refers to frequency resolution of approximately 20 Hz. We then found the maximum amplitude element and multiply its index by the frequency resolution to extract the dominant frequency. To get a better, precise result, we interpolated the index between the neighbors since the frequency value can fall between two elements. This computed pitch value is then used for assigning the color to the visual effect, a method which will be explained in the next section.

Table 2: Frequency resolution population

Bar	Frequency	Frequency Range	Elements
1 (red)	40	0 - 40	1
2 (orange red)	40	41 - 80	1
3 (orange)	80	81 - 160	2
4 (orange yellow)	80	161 - 240	2
5 (yellow)	160	241 - 400	4
6 (yellow green)	160	401 - 560	4
7 (light green)	320	561 - 880	8
8 (green)	320	881 - 1200	8
9 (dark green)	640	1201 - 1840	16
10 (light blue)	640	1841 - 2280	16
11 (blue)	1280	2281 - 3560	32
12 (navy)	1280	3561 - 4840	32
13 (purple)	2560	4841 - 7400	64
14 (pink-purple)	2560	7401 - 9960	64
15 (wine)	5120	9961 - 15080	128
16 (pink)	5120	15081 - 20200	128

Table 3: Color Bins

Color bins	MIDI range
Red	< 24
Orange	<34
Orange-Yellow	<44
Yellow	<54
Yellow-Green	<64
Green	<74
Light Blue	<84
Dark Blue	<94
Violet	<104
Pink	104 <

4.3.4 Mapping Color

Once the pitch value is calculated, it is then converted into MIDI notes. The color is then assigned based on the range the MIDI value falls into. The table below describes 10 distinct color bins which are made based on HSL spectrum and the MIDI notes range for each color bin. Once the color is decided, the visual effect of the assigned color will then be instantiated.

4.4 Spawning Visuals

In order to prevent the visuals from instantiating right in front of the users and overlapping with each other, we created 18 spawn areas and placed them in distinct locations.

The red dots in the figure indicate the spawn areas. These locations are stored in a dictionary with index as a key and SpawnArea as a value:

```
public Dictionary<int, SpawnArea> dictionary =
new Dictionary<int, SpawnArea>();
public class SpawnArea
{
 public SpawnArea
 (bool occupied, GameObject spawnPosition)
  {
   this.occupied = occupied;
   this.spawnPosition = spawnPosition;
  }
}
```

SpawnArea is a class that holds a boolean (whether the spawn area is occupied) and the spawn position, which indicates one of the red dots in the figure above. When the pitch is detected from the audio, GenerateRandomList() is called and returns the list of 2 random, unique indexes that are not occupied from dictionary. Then the returned list will be iterated, and for each index, the spawn position will be retrieved from the dictionary:

```
Vector3 spawnPosition =
dictionary[index].spawnPosition.transform.positoin;
```

The spawn position of the given index will be set as occupied:

```
dictionary[index] =
new SpawnArea(true, dictionary[index].spawnPosition);
```

And the visual effect will be instantiated at the given position:

```
GameObject visual =
Instantiate(visual effect, spawnPosition);
```

The spawned visual will be the child of the current game object:

```
visual.transform.parent = gameObject.transform;
```

5. RESULTS

In the end, we were able to build the application and install it into Oculus Go headset and demonstrated it at the poster session. We saw many of the users enjoying the customizable environment.

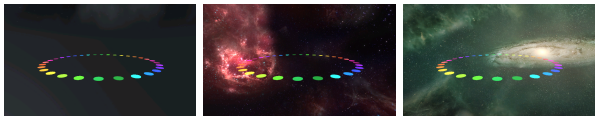
5.1 Customization

One of the key features of the application is the customizable environment. In the main menu, the users can choose from 5 different backgrounds and 5 different terrains. The users can experience the visualizer in different environments and make them feel as if they are actually there. Figure 3 and Figure 4 show all the options the user can choose.

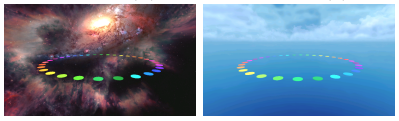
Once the background and the floor are chosen in the menu, it will lead the user to the Audio Visualizer scene, a main scene that renders a skybox and instantiates the floor chosen by the user. Figure 5 shows the Audio Visualizer scene.

5.2 Workflow

The figures in Figure 7 are the screenshots of the application's VR scenes when the user wears the Oculus Go headset like Figure 6. When the start button is clicked, the application is triggered to show the Background option panel. When "Next" is clicked, it will show the Floor option panel. Once they are all selected and the start button is clicked, it will jump into the Audio Visualizer scene with the selected environment that contains the ring of bars and visual effects that are scaled and instantiated in response to the audio spectrum data and pitch values, respectively. Just below where the user is standing, there is a menu button that will load the Main menu scene once clicked.

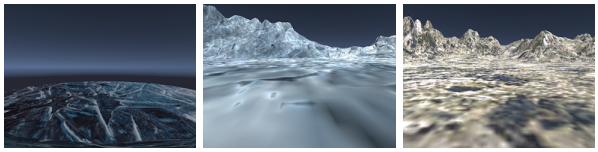


(a) Early Dusk Sky (b) Pink Solar (c) Green Solar

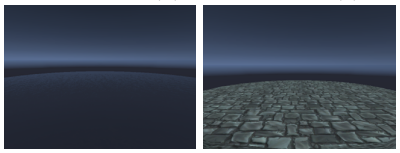


(d) Purple Solar (e) Bright Sky

Figure 3: Backgrounds



(a) Ice (b) Moon (c) Mountain



(d) Ocean (e) Stone

Figure 4: Floors

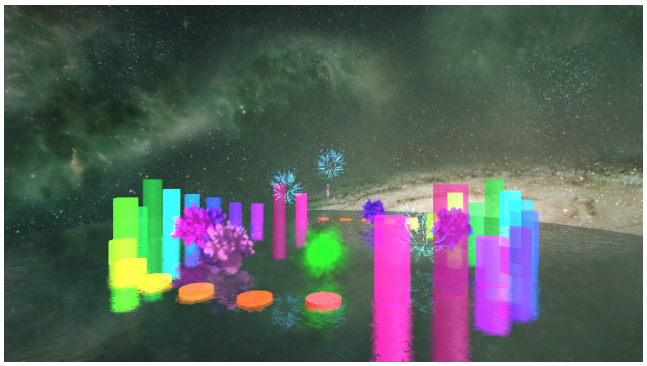
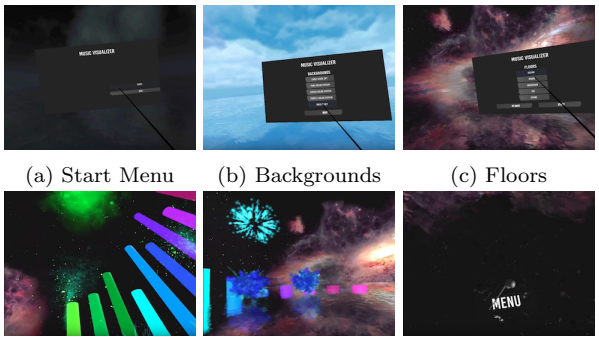


Figure 5: Audio Visualizer scene

6. DISCUSSION

The main goals of this project were to learn how to develop a VR application in Unity and use the elements of sound to create a music visualizer that can be used to help users enjoy music immersively. We started with definitions of synesthesia, data and music visualization, VR and the benefits of using VR music visualizers. We then explained the process of integrating software and hardware components and the intricacies behind how the visuals in the application are cre-

Figure 6: Demonstration



(a) Start Menu (b) Backgrounds (c) Floors

(d) Main-1 (e) Main-2 (f) Menu Button

Figure 7: Workflow

ated.

Despite achieving many of the goals we set out for, our project had some limitations. Attempting to create an application that was both a music visualizer and a synesthesia simulator proved to be more difficult than expected. The end result was an application that incorporated elements of each but focused more on music visualization. A lesson we learned from this was to enforce a clearer vision of a final product and more defined steps in order to arrive at the end goal.

Another limitation of our project was time. Our resulting application was satisfactory but with more time, more intricate details could have been added to our project to make the VR environment more engaging and dynamic. Syncing changes in volume to the size of particle systems created such that louder noises would instantiate larger particle systems or syncing the beat to the movement of objects were two ideas we had for improving the project.

This project could be improved by implementing more sound features like beat. Currently, the project only uses amplitude and frequency as the main features of the sound that influence the visualizer. We initially planned to implement beat to add movements in visual effects; however, we did not have enough time to incorporate beat detection algorithm. The project can also be improved by changing the way we assign colors to the visual effect. Right now, the color is determined by color bins. Each color bin refers to the certain range of the MIDI notes, and when the pitch value falls into one of the range, the color that corresponds to the range will be assigned to the visual effect. However, instead of having color bins which limit the number of colors (we only have 10 color bins in this project), we can convert the extracted pitch value into the hue value so that there can be more variations in color.

7. ACKNOWLEDGMENTS

We would like to thank Professor Jason Grant and the Undergraduate Research Office for providing the Senior Research Project Supplement which allowed us to purchase the materials necessary for creating a VR application.

8. REFERENCES

[1] U. Documentation. Unity audiosource.getspectrumdata, 2019.

[2] T. L. Hubbard. Synesthesia-like mappings of lightness, pitch, and melodic interval. *The American Journal of Psychology*, pages 219–238, 1996.

[3] Z. Kaidi. Data visualization. Technical Survey, 2000.

[4] H. S. Kosuke Itoh, I. L. Kwee, and T. Nakada. Musical pitch classes have rainbow hues in pitch class-color synesthesia. *Scientific Reports*, December 2017.

[5] mbryonic. 10 best vr music experience.

[6] Merriam-Webster. Definition of visualization.

[7] L. P, Z. A, and S. G. Absolute pitch and synesthesia: two sides of the same coin? shared and distinct neural substrates of music listening. *ICMPC Proc Ed Catherine Stevens Al Int Conf Music Percept Cogn*, pages 618–23, 2012.

[8] K. Rougeau. Chromesthesia music visualizer, July 2018.

[9] Scriptable. Oculus go unity setup.

[10] Seeker. What's it like to hear colors? - a vr 360 ° synesthesia experience, November 2015.

[11] R. Studios. How to build an oculus go app with unity, August 2018.

[12] J. Tarrant, J. Viczko, and H. Cope. Virtual reality for anxiety reduction demonstrated by quantitative eeg: A pilot study. *NeuroMeditation Institute*, July 2018.

[13] C. I. University. Pitch to midi note number to equal-temperament frequencies chart, 2017.

[14] V. with Andrew. Pointer for gearvr and oculus go, December 2018.