BandAR: A live mixing and real-time effects augmented reality app.

Candidate Number:	148508
Module Name:	Sound for Interactive Media
Module Code:	W3085
Academic Year:	Spring 2018

Abstract

This paper explores the creation and development of BandAR, an augmented reality based app that incorporates distance based live mixing and real-time effects processing. This was my first project fully in Unity3D/C#, and so this paper is an exploration of my thought processes throughout the project, including the creation of C# scripts and creative coding to solve problems with computational thinking.

Keywords:

iOS, Unity3D, XCode, C#, augmented reality, real-time effects, live-mixing

Keywords:

AR*: Augmented Reality*

VR*: Virtual Reality*

Acknowledgements:

Chris Kiefer*: for inspiring much of my work over the last three years during this undergraduate degree and for offering me many opportunities to explore my own creative musical ideas.*

Thor Magnusson*: for doing much the same and also inspiring me with his teaching of computational thinking and humorous live coding.*

The Team at BeFries, Brighton*: for putting up with my use of their printer for the last six weeks printing numerous QR codes for this project and being a great support throughout my degree.*

Introduction

This report aims to investigate BandAR, the app I have created for this creative music project. My motivation for this project is partly trying to get the most out of my smartphone and new technology such as augmented reality and image recognition, but also the apparent lack of engaging and intuitive augmented reality apps on the App Store. BandAR at its core is an exploration into the collaboration between music and augmented reality. The user is presented with a camera view, that they can point at up to 6 printed QR codes. The image recognition service paired with my Unity project replaces these QR codes with 3D models of 6 different instruments on separate stages. The user can move their device closer to the instruments resulting in a louder volume, and click on the instrument to change some effects parameters for a reverb, echo, chorus, and a high-pass filter.

Research / Context

Augmented reality and virtuality reality have become known for their explosive entry into the industry of mobile applications, and more recently video gaming with the advent of technologies such as the Oculus Rift, HTC Vive and cheaper systems such as Google Cardboard and PlayStation VR. However, outside of gaming, there hasn't been too much activity after the novelty of it wore off for users. On mobile devices, the supply of augmented reality applications is limited by the very low demand for them. The only truly useful application I have come across is the IKEA AR app that uses the new ARKit depth camera code to 3D map your room for furniture measuring. The general consensus seems to be that VR/AR is "cool" but there isn't much about it that is truly 'game-changing' at this current moment in time outside of gaming (Busel, 2017). This got me thinking about possible musical uses of AR, perhaps without the snazzy new depth perception cameras in the iPhone 8+/X, but still using digital camera overlays and aural stimulation. My idea for this project is mainly to explore the crossroads of 3D art/music and augmented reality. I have done this by creating a mobile application, and by adding to the increasing amount of research and development in the field of art-based augmented reality applications and installations such as Circa69's "Whilst the Rest Were Sleeping". Circa69's installation combines 17 VR experiences and an AR smartphone app and trail as well as live electronic music, audio-visual performances, video installations (Circa69, 2017)

Development

The development of this project spans three months, from February to May and can be split into into four stages:

The Image Recognition API **The Interface**

The Audio **The Scripts**

Unintentionally, this ended up being a very efficient workflow method for this project. The four sections/stages of the app interacted with each other in a simple enough way that allowed problem solving to be relatively straightforward and modular even when the project was inactive for weeks at a time due to time constraints on another project I had on simultaneously.

The Image Recognition API

The first stage of development was researching into image recognition and augmented reality APIs. I found out very quickly that Vuforia is the most used image recognition API with Unity3D implementation. First, I created 6 QR codes, and registered them as image targets on my Vuforia account (Figure 1). Then I imported the Vuforia API into my Unity3D project, and set up an ARCamera, which is what the device's camera sees through (Figure 2).

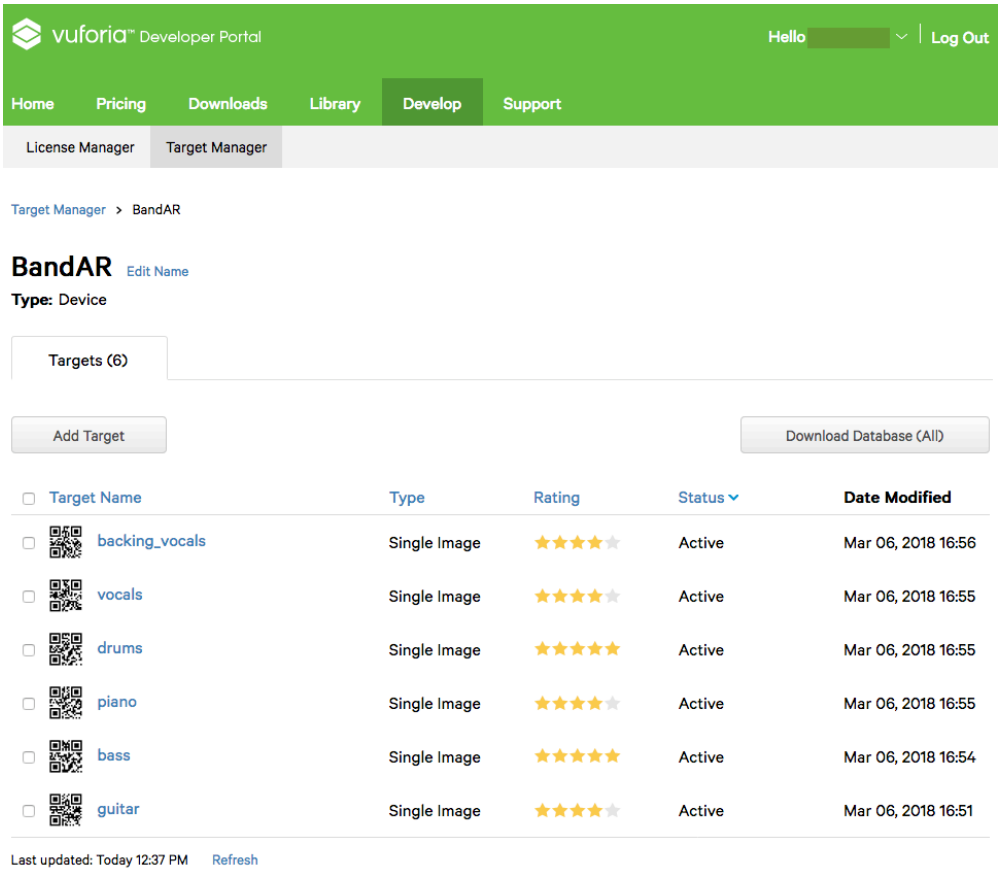

Figure 1: Vuforia Developers Portal

Then I essentially "logged in" to my Vuforia account in Unity by assigning a unique private API key to the ARCamera GameObject. The next step was creating the image targets in the editor window and assigning them the images that I had uploaded onto the dev portal (Figures 3, 4).

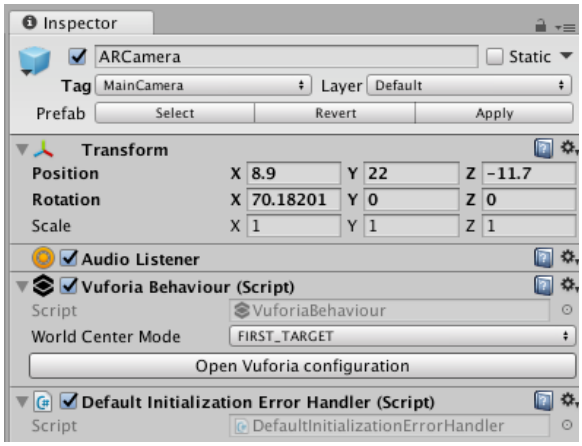

Figure 2: ARCamera

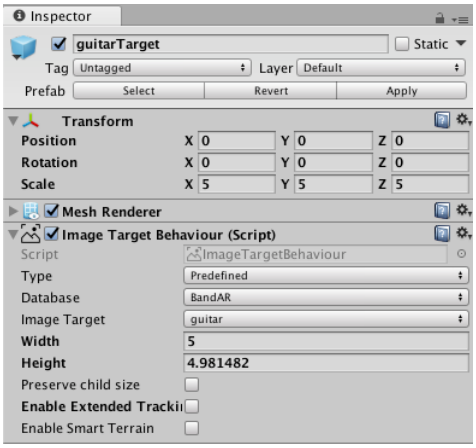

Figure 3: ImageTarget Properties

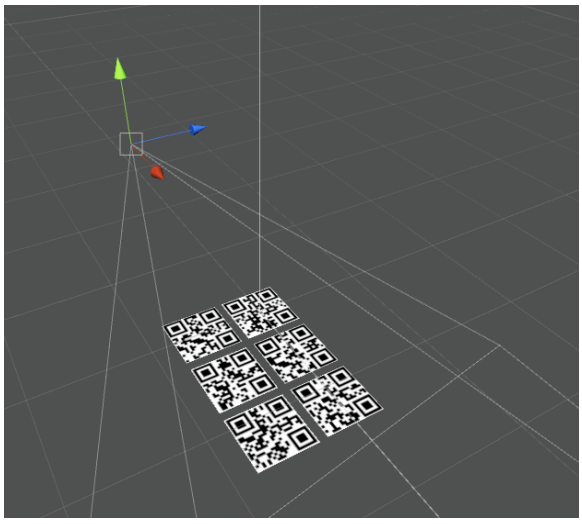

Figure 4: ARCamera and ImageTargets

The next step was creating the 'stages' for the instruments. I used imported 3D assets from the asset store and from some online asset libraries to create six fun looking stages for the six different instruments, Guitar, Bass, Piano, Drums, Vocals, and Backing Vocals. These are each children of their own respective QR targets in the hierarchy window and trigger whenever the QR code is recognised by the 'Default Trackable Event Handler' script. I finished this stage of the development by early March.

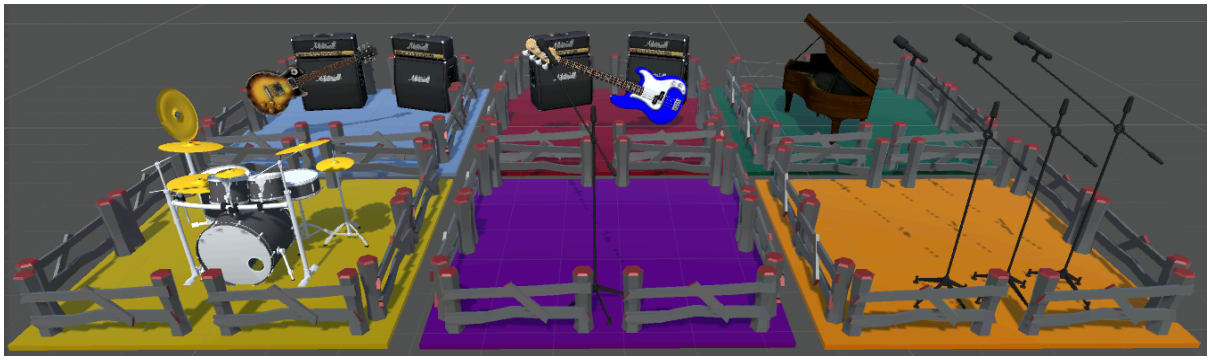

Figure 5: Six instrument stages

The Audio Pt.1

Figure 6: Working Image Recognition

After the image recognition was working in the patch without fault (Figure 6), the next task was to implement audio playback. There was a very simple method to achieve this, and that was adding an AudioSource to each instrument's GameObject in the inspector window and modifying the previously mentioned DefaultTrackableEventHandler.cs script. This script by Vuforia deals with the recognition of targets and has a method that triggers whenever a target is recognised and then ceases being recognised. A simple injection of a few lines of code meant that I could trigger each instruments AudioSource when it was recognised by the app, this is shown in Figure 7. What was initially an experiment became a core part of the app when I enabled 3D sound settings on the AudioSources. After setting a minimum and maximum distance from the imageTargets, what happened was a unique live mixing experiments in one of my app tests. I could move the phone closer to certain instruments and their volume would slowly ramp up in volume. I find this 'feature' extremely intuitive and allows the user to be truly creative with the placement of the QR codes in order to achieve the live mixed sound that they want.

```
if (newStatus == TrackableBehaviour.Status.DETECTED ||
    newStatus == TrackableBehaviour.Status.TRACKED ||
    newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
{
    OnTrackingFound();

    // Play AudioSource
    AudioSource audio = GetComponent <AudioSource>();
    audio.Play();


}
else
{
    OnTrackingLost();

    // Stop AudioSource
    AudioSource audio = GetComponent <AudioSource>();
    audio.Stop();


}
```

Figure 7: AudioSource code

There is more to the audio side of the development of this project, but it came later in the chronology, so I will revisit it later in this report. This short audio section was complete in mid March.

The Interface and Scripts

After a short break on BandAR of approximately 4 weeks due to another project, I came back to the project with fresh eyes and managed to achieve the interface and scripts section of the project development in about a week. I wanted the app to have a sleek appearance, not too dissimilar to Logic Pro X's dark grey colour palette, and with a simple UI toggle button that allowed full camera view. This section on interface is very closely intertwined with the scripts section of the development, so in the interest of flow, I have joined them together.

I found the GUI building experience in Unity3D fairly unintuitive. Being used to software such as Photoshop and Illustrator for graphic design, the child hierarchy approach to creating GUI panels seemed clunky in Unity, and the addition of proportionality anchors that could not be snapped to the RectTransform functions of GameObjects without a third party script I had to install, meant that I spent nearly 2 hours just getting the layout I wanted. Essentially the GUI is made of three sections, the Hide UI Button, the Selected Instrument panel, and the FX panel, of which there are 6 (one for each instrument).

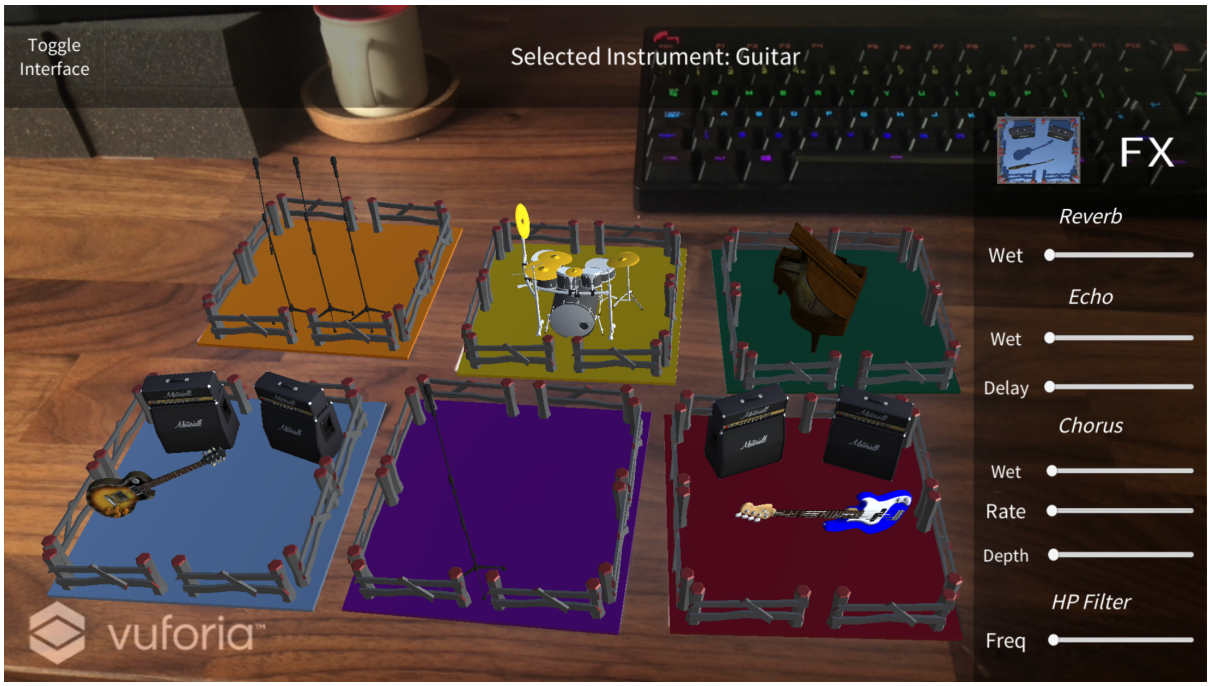

Figure 7: Full GUI in view

The Hide UI button in the top left hand corner has a script attached that deactivates UI panels 'onclick' using the hideGUI.cs script I have written. On initiation of the app, the FX panels will not be visible as they are deactivated by default, but when the user taps on an instrument, the correlating FX panel becomes visible thanks to another script called onHover.cs. This script makes use of the Physics.raycast method that detects user input on the screen and contains conditional if statements that not only change the FX panel but change the text in the Selected Instrument panel to the correlating instrument. This script is attached to the ARCamera GameObject.

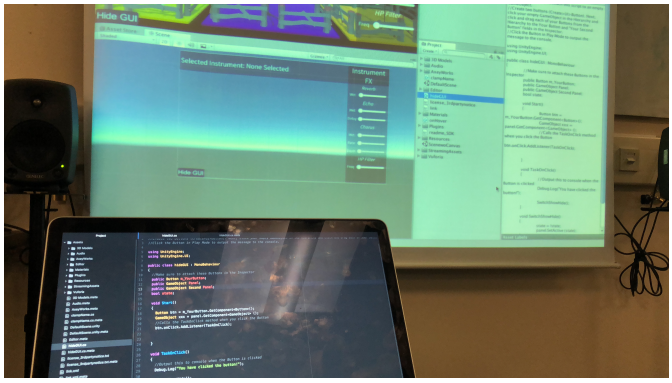

Figure 8: Script editing

The Audio Pt.2

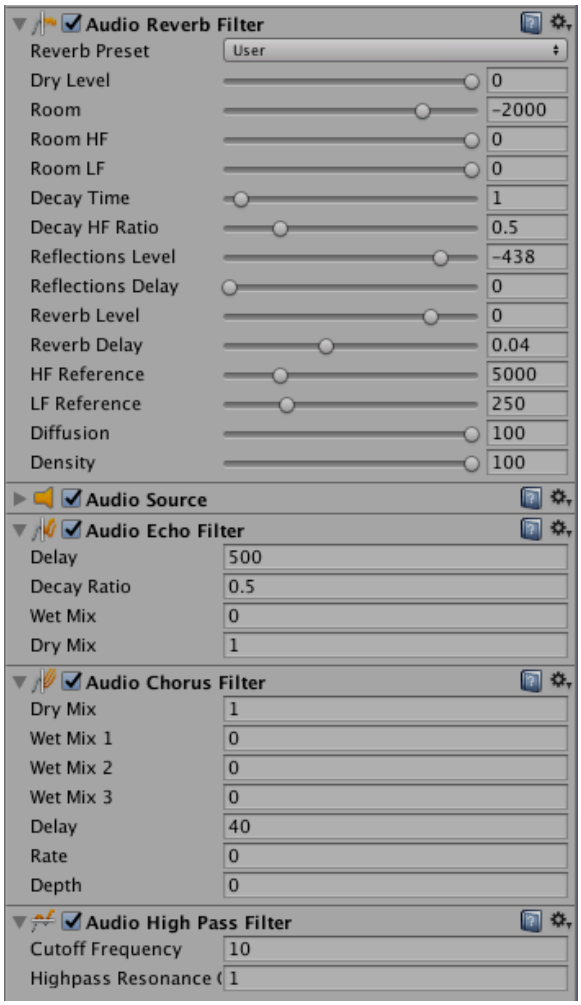

Figure 9: Audio FX Filters

After the GUI was completed and the sliders had been created for the FX panels, it was time to map them to the audio effect filters I had planned to use: reverb, echo, chorus and high-pass filtering. I wasn't sure if there was a more gracious way to achieve the mappings than manually dragging GameObjects to each slider, so thats what I did. With all 42 sliders. Figure 9 shows the four filters and their default settings (Wet/Dry slider always at 100% dry default). Figure 10 shows the mapping set up for the Wet/Dry slider on the Chorus FX of the guitarTarget. The mapping was surprisingly simple to set up, which made a nice change from the GUI-creation.

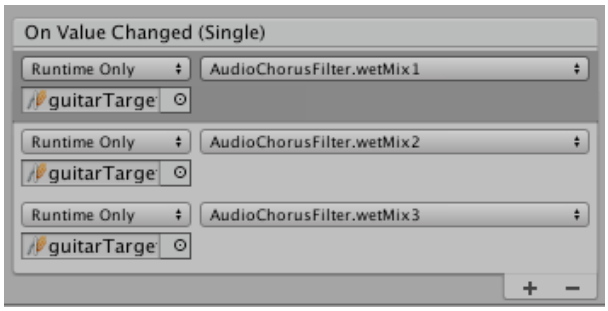

Figure 10: Example slider mapping

Conclusion

I started this project wanting to explore the crossroads of 3D art/music and augmented reality by creating a mobile application that uses QR and image recognition to allow the user to live-mix music and change FX parameters. While I believe that I have achieved this, and work colleagues and classmates have congratulated me on my creative and interesting use of augmented reality, there are still features that I would love to add to BandAR in the future if it was commercially released. Primarily, I would like there to be a way for the user to add their own music to the app. Currently, it is just a 4 bar loop of 6 instruments that I quickly made in a DAW. The reason I have not been able to do this so far is due to my working in Unity3D. If it was a solely Swift/Xcode based project I am sure I would have learnt enough by now to add that functionality. However, due to the fact that I am building the app in Unity, and using the Build/Run feature to make Unity create a custom Xcode project, means that I would have to be very proficient in Swift coding to implement a file system structure alongside the imported Unity project. Another feature I would like to see implemented is 3D animations on each stage, perhaps if the instruments moved when selected, or if the stage changed colour based on a certain musical parameter. User testing the app proved to be invaluable, after testing the app with three people, the consensus was that two features needed changing, and that I hadn't noticed how unintuitive they were. The first issue was the overall volume level. I switched the AudioSources to linear volume settings and decreased the overall sphere of each 3D sound source so that the volume ramp was much more apparent. The second change was increasing the size of the sliders, which made the app a lot easier to use and increased the enjoyability.

I thoroughly enjoyed the idea of computational thinking and creative coding in this project, and thinking about C# as a traditional language with syntax and keywords really helped problem solving in the script-writing of this project.

Supplementary Information

Bibliography

Busel, M. (2017). *The 6 biggest challenges facing augmented reality – Haptical*. [online] Haptical. Available at: https://haptic.al/augmented-realitys-biggest-threats-3f4726a3608 [Accessed 9 Apr. 2018].

Circa69. (2017). *Circa69*. [online] Available at: http://circa69.co.uk/ [Accessed 10 May 2018].