Sound Design for Digital Media

Dimitrios Emmanouil Karavias- s141278

Abstract — I will describe my experience creating sound effects and procedural audio using Pure Data and Unity3D.

I. INTRODUCTION

The idea for this project is to implement sounds that can be used by a game. The original idea was to create sound effects and procedural background audio for a very specific 3D Space shooter game that I was working with Unity3D. But after having a lot of trouble making Unity3D work with all Pure Data patches, in the end I decided to change the idea and create a demonstration of how Pure Data patches can be used from Unity3D to achieve some specific sound effects and generate procedural audio. So instead of having a solid game, and implementing the necessary sounds for it, I created a Unity3D scene that demonstrates the different ways these sounds can be used inside a game.

This idea can be interesting for those who want to see how they can combine a game engine like Unity3D with a tool like Pure Data for several reasons. One reason is that many indie developers, like myself before I start this project, know enough about a game engines and graphics, but not much about sound design. I always caught myself having a game and later searching for free sounds that can cover my game's needs, something that isn't always easy. After this project I have a better understanding about several audio synthesis techniques for games and I will most likely use Pure Data for my future games.

II. DESCRIPTION & FEATURES

The concept of the project is for the player to discover the sounds that are in the virtual world I created. The player controls a character, and can hear him walking or running on different terrains. The terrains are dirt, grass, snow and wood. There are two areas with wood where the player can also see an illustration of other sound design synthesis techniques. In the first area that can be located inside the snow terrain, the player will experience a 3D sound effect, which technically is an additive synthesizer that is combined with a spatial patch to convert the sound into a 3D sound relative to the player. In the second area that is located in the grass terrain, the player can experience an example of a procedural audio that could be used in a game when the level of difficulty change.

From the first idea with the 3D space shooter game, one big challenge was to create sound effects like explosions and laser guns. So I thought that a Pure Data patch for spatializing an audio would be necessary feature for anyone who designs 3D games and wants to use Pure Data.

Procedural music is the second feature that I wanted to create. It is necessary for 2D and 3D games and can give a different dimension to the game. This sounds will be 2D sounds and volume of the sound is not depending on the position of the character.

In the same concept with procedural music, I wanted to create procedural sound effects, like footsteps for the character. The idea for the game is that the footsteps will change when the player walks or runs on different terrains.

When I started working on this project there were a number of challenges that I expected to face. One of them was how I would do a physical analysis of what a sound should sound like and how I could be able to recreate a sound like an explosion on pure data. Unfortunately the biggest challenge turned up to be the compatibility of the available Unity's Pure Data libraries with all the features of Pure Data. A challenge that in the end forced me to change my original idea.

III. IMPLEMENTATION

In this section I will talk about the implementation of the patches and the connection of the patches with Unity3D.

A. 3D sound

This is a feature that came up to me because I encountered the problem it solves first. So first I will talk about the problem, and then about the implementation.

Unity3D is a game engine that helps you very easily create 3D games, but the current available libraries for pure data do not include a functionality for having 3D sounds. Luckily while I was developing the patches for the original idea, a new library appeared for Unity that is called "libpd4unity tools" that had this feature. Unfortunately this new library right now contains a lot of bugs and cannot be used for production, but only inside the Unity's editor. One other problem with this library is that it is not working at all on android devices, and android was my target platform for the original idea.

After trying hard to debug the library myself I finally faced some bugs that were impossible to fix from the library's Unity scripts. So I decided to solve the 3D sound problem myself, following the idea behind this library which is quite simple, but challenging to make it work correctly while still learning the tools.

The first thing before the implementation was to understand what a 3D sound is. A 3D sound has a source position in the 3D

space. In our 3D world we have an audio listener which is the character, so the sound that the player hears is related with the position of the audio source and the position of the character, as well as the angle between them. The implemented patch (figure 1) is being used as sub patch in patches that have 3D feature as requirement.

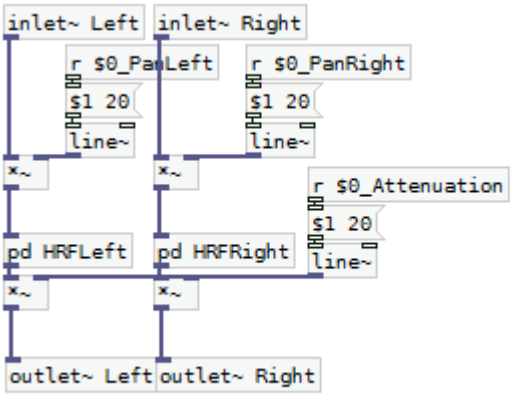

Figure 1. Patch code for 3D sound effect.

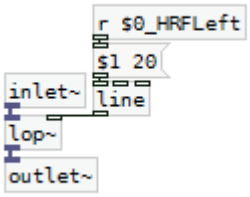

Figure 2. HRF patch, applies a low pass filter to the input signal.

The patch receives the signal of the original audio as input in the inlets. Then it expects 5 inputs from Unity. These inputs are the Attenuation, left Pan, right Pan, left HRF and right HRF. All these variables are calculated every frame from a Unity script and they are send in the Pure Data client. In Unity we can choose the attenuation to be linear or logarithmic from the location of the character. When Unity sends Attenuation to the patch, this value is used as a multiplier to the final signal and controls the volume. PanLeft value controls the amplitude of the signal for the left speaker, and PanRight for the right speaker. HRFLeft and HRFRight apply a low pass filter in left and right output signal respectively to generate the effect of the character moving around the sound.

In figure 3 we can see how we can use the patch to create any sound signal into a 3D sound.

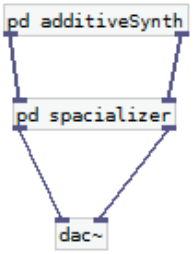

Figure 3. Show case of spacializer patch
Which spatializes a simple additive synthesizer.

In the original idea of the 3D space shooter game, the spatializer patch worked good as expected. One big issue that occurred with it however is that in the game there were a lot of explosions and laser guns that could generate a 3D sound at the same time, and the idea of having one patch instance for every game object was not efficient for the CPU. I managed to solve this problem with unity by pooling 10 patches and using them for the nearest to the player objects. In the final idea however we only have one object that generates 3D sound as a showcase and the problems with performance does not exist.

Lastly one problem occurred with Android devices and libpd4unity. The library cannot handle the situation where a Pure Data file is included in a different Pure Data file. The original idea was to make spatializer patch a different file, which we can be included in any patch we want to make it 3D. However because of the issue with Android devices I decided to copy the spatializer code inside every patch that needs this feature.

B. Additive synthesizer

For the new idea of just the showcase, I replaced the laser beams and explosions with an additive synthesizer. The synthesizer created only for showing the results of the 3D sound so I wasn't very ambitious with it. However while I was learning additive synthesis the resulting sound became interesting enough for me.

The new idea of this patch was to create something like the background music of a platformer game. In figure 4 is the Pure Data patch I implemented for this, and in figure 5 is a sub patch that creates the basic tones for the synthesizer.

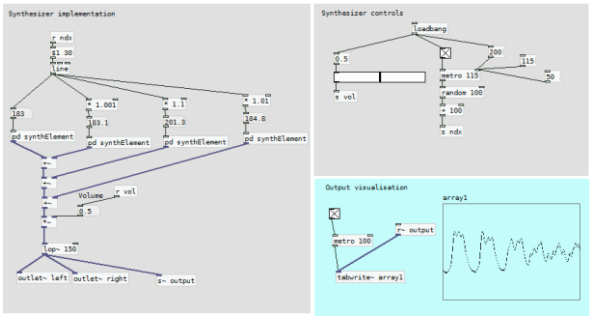

Figure 4. Simple additive synthesizer. Contains
Simple control panel (uses random)
And a visualization panel to see the output signal.

As we can see in Figure 4 the main patch contains 3 sections, one for the part of the patch that manipulates the signal and generates the output, one for the section that controls the basic inputs of the synthesizer and affects the sound creating a melody. Finally there is a section for the visualization of the output signal.

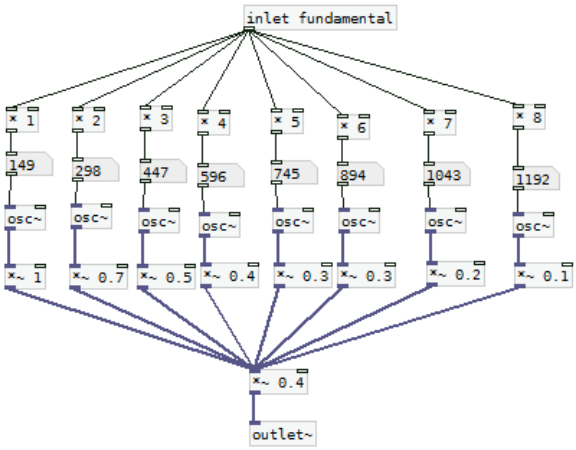

Figure 5. Additive synthesizer that
creates basic tones.

In figure 5 we can see the sub patch that implements an additive synthesizer that creates the basic colours for the synthesizer. Here we use eight "osc~" objects to generate sine tones with different frequencies. All the frequencies are a multiplication of a fundamental frequency that is the input of the patch. Like this we create a chord based on the overtone series. Before combining the sine tones together we change their amplitudes so bigger frequencies have smaller amplitudes so the different tones blend better, and create a sound colour.

The patch in Figure 4 creates every 250 milliseconds a different frequency between 100Hz and 200Hz. This frequency then is used in the sub patch in figure 5 to create a sound color. We use four instances of the sub patch where each one is provided with a slightly different input frequency as a fundamental frequency. This way we can create a better melody with more "colors".

Finally I use the output signals of each sub patch and add them together to create the final melody. At this step I experienced two issues, one was that the sound itself was very loud, so I created a number input that I called "vol" for volume and I used this number to reduce the amplitude of the final signal. Even if that created a better experience, there were cases where the final signal had peaks with very high amplitude than the average amplitude of the signal, making the sound often disturbing. To fix this I used a one-pole low pass filter to cut the high frequencies. The resulting sound after these changes was satisfying.

C. Generative music and procedural audio

This has been a very interesting part of the project. Generative music (or procedural audio) is used in games in order to provide an audio feedback for the player that is synchronized with the state of the game in order to create a better game experience. In this part I will describe the implementation of a patch that can work as a procedural audio generator.

In practice there are a number of ways to create generative music using synthesis techniques to achieve the basic sounds and parametrizing aspects of the audio during the. I gave a try on different methods but in the end I used the most convenient to me. The method is similar to the method that has been used to create the generative sound for the game Sim Cell.

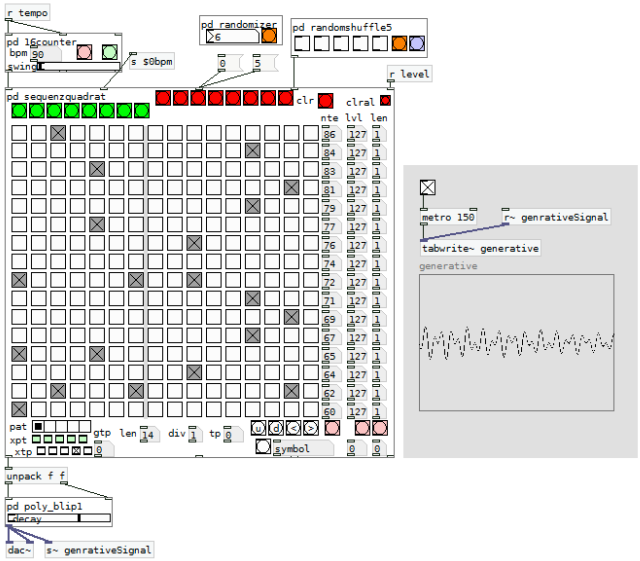

Figure 6. Pure Data patch for generative music.

The patch contains four noticeable features as we can see in figure 6. The first one is the counter which counts up with a number of beats per second, which controls the speed of the counter. The counter's output is a number that then is inserted into the sequencer.

The sequencer receives the output of the counter, which then uses to choose the beat number to play. The sequencer also supports eight different windows where we can synthesize different audio and choose which one we want to play. I use this option to change the audio as it would have changed on different states of a game, for example the level of difficulty increases.

The output of the sequencer is the current selected notes that correspond to the given sequence. These notes are then imported into an additive synthesizer that uses a "blip" sound to create the output signal.

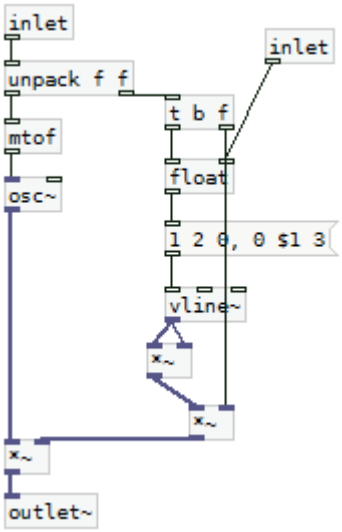

Figure 7. Blip patch.

The last noticeable section of the patch is the one that does the visualization of the output signal.

The additive synthesizer uses the blip patch to create instrument like sounds. By changing the input of this patch we can create different instrument sounds and blend the sounds together to create quality audio, like they have done in Sim Cell game. Another improvement would be to use granular synthesis to smoothly change between different sequences.

D. Procedural audio, footsteps

In the same area of procedural audio I spent some time trying to figure out how I can create example of a sound effect that is changing from specific attributes in the game. Most of the ideas required physical analysis of the involving natural elements that generate the sounds, like rain and guns. My experience in sound design and the amount of time I could spend on this wasn't enough for me to work on a good example that I could demonstrate on this project. However I manage to make a foot step patch made by Andy James Farnell to work with Unity3D and the character in the game.

The patch gives a good approximation of human footsteps on different terrains and speeds. If I had use audio files to achieve the same result I would have need an enormous amount of files to cover all speeds and surfaces combinations, increasing the size of the game in tens of megabytes. Now however with this script we can achieve the same result with just a few kilobytes of code, and most importantly, we can add functionality for a new surface and different characters, with just adding a few lines of extra code.

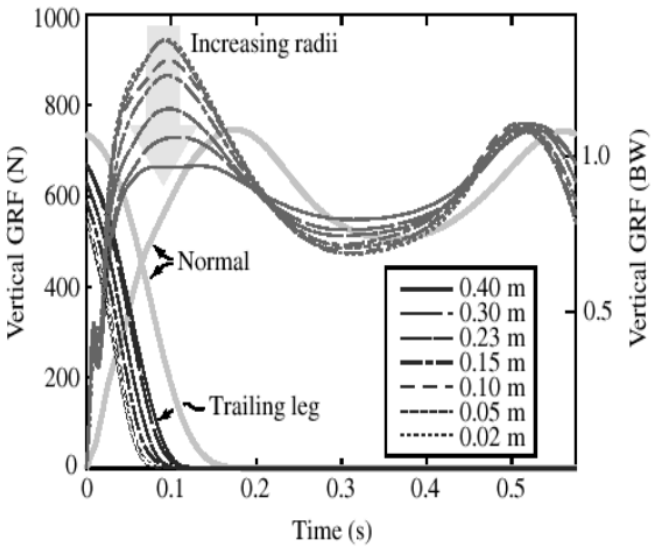

Figure 8. GRF Human step.

The patch tries to approximate the human step signal in different situations as shown in figure 8. Next the signal is passed through a three stage envelope where the physics for the different stages of the footstep (heel, roll and ball) are calculated. Lastly the signal passes through a texture synthesizer to get the corresponding sound for the terrain the character walks on.

From Unity3D I used this patch and I change the speed of the footsteps to achieve the correct walking and running rhythms, as well as I update the current terrain the player is walking or running on.

E. Unity3D – how to Pure Data

In this section I will write about what you will see in the Scene in Unity3D and how Unity3D is interacting with Pure Data.

First of all, after trying different Pure Data libraries for Unity3D I finally used the libpd4unity. This library works with free version of Unity in both Windows and Android platforms. However I also tried a new library called "libpd4unity tools". This new library uses libpd4unity and implements spatialized

sounds and more features that make working with 3D environments and unity easy. The library however is very new and has a lot of bugs, causing more troubles than those it solves.

When you play the scene you will see a character in an island. You can interact with the player by using arrows to move, hold shift button to run, and space button to jump. The terrain of the island is mainly dirt. One circular part of the terrain is grass, and one similar is snow. Inside both circular terrains there is a wooden platform.

The first thing you will notice in the scene is the footsteps of the character that change when the character changes terrain, and when walks in the wooden platforms. The footsteps will also change speed to match the character's speed, which can be changed by either walking or running. The terrain can be seen in the figure 9. On every frame in Unity I send a message to Pure Data to update the speed of the foot step patch. When the character enters a different terrain, a message is sent to update the terrain texture that is used to filter the foot step signal.

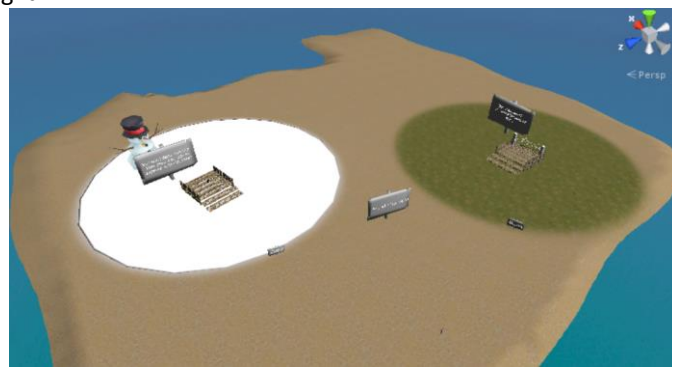

Figure 9. Unity3D scene setup.

When the player goes to one of the two wooden platforms, he/she will experience a different sound. For the wooden platform in the snow terrain, the player will experience the 3D sound patch with the additive synthesizer. To demonstrate this visually I have placed a 3D model of a speaker in the platform which the player will see, then when he is moving around the speaker and changes distance from it, he will experience the change of frequency and volume, as well as the stereo sound from the speakers. This is demonstrated in figure 10. On every frame in Unity I send values to the spatializer patch that alters the amplitude and the frequency of the output signal in both the left and the right speaker according to the player's distance with the audio source and the angle between them.

To make this work in Unity3D I made a C# script (PDComponent.cs) that the developer has to use in game objects that are 3D sound sources. The script receives the name of the patch, the Volume Rolloff which can be either Logarithmic or Linear. The developer can also choose the minimum and maximum distance for the sound, as well the pan level.

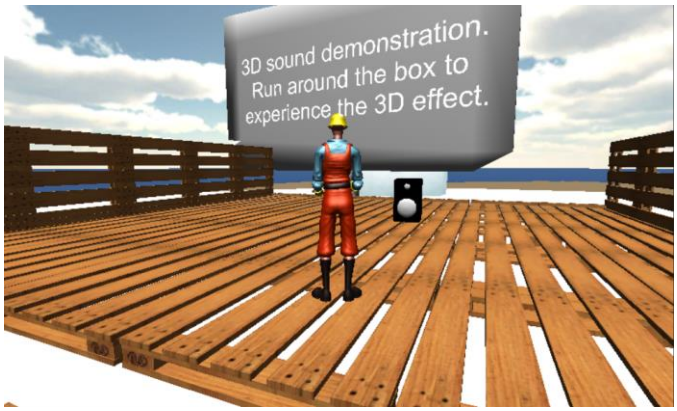

Figure 10. Wooden platform in the snow.
A speaker visualizes the sound source.

In the wooden platform inside the grass terrain the player will see an illustration of the procedural audio patch. He will see a box moving from one side to the other, passing checkpoints. For each check point the box passes, a message is send to Pure Data patch to change sequence of the synthesizer. This is illustrated in the figure 11. In Unity3D I enable this audio when the player enters the grass terrain, and I disable it when he exits the grass terrain.

Figure 11. Wooden platform in the grass.
Illustrates the a showcase for the procedural audio

All pure data files are located inside the Assets\StreamingAssets\PdAssets folder of Unity3D. The 2D sounds are initialized in the Scene from the component LibPDMain, which is attached to the Main Camera. The 3D sounds are initialized with the script PDComponent and is attached to the game objects that require 3D sound, in our scene this is the Speaker. All the scripts that I implemented for initializing and communicating with Pure Data are in the folder Assets\Scripts.

IV. ANALYSIS & CONCLUSION

When I started working on this project I was very ambitious of what I could achieve even though I knew that I didn't have any sound design background. Being a programmer who loves coding, I manage quickly to connect Pure Data and Unity3D, and I was ready to start implementing the patches.

In the original idea I wanted to create an amazing sound experience for a game that I am working for mobile devices, but when I started working on the sounds I had in mind, I soon faced the first problems. The available Pure Data libraries for Unity3D are not very mature to use the full potential of Unity and Pure Data together and Unity3D crashes when some specific Pure Data objects are used in the patches. I spend most of the time on this project debugging different libraries to make my patches work on both PC and Android.

All these difficulties forced me to change the project idea to something simpler that can just demonstrate what I have learned and I only used one patch from the original idea. Fortunately this decision helped improve the learning process since I stopped concentrating on both Unity3D and Pure Data and I focused more on Pure Data.

If I knew in the beginning what I know now, I would spend more time trying all the different synthesis techniques and less time debugging Unity3D. I would also spend more time on practicing with physical analysis.

For the next steps in this project I would like to improve the generative music patch, I would create more instrument sounds and use granular synthesis to smoothly change between different sequences. Something that I will most likely do in the future games I will make. The possibilities are endless. Lastly there are improvements that can be made on the other patches as well, the footsteps even if they keep the correct speed, sometimes after running and then walking, the sound is not synchronized with the foot and they have a small offset with the animation.

REFERENCES

[1] Pure Data additive synthesis, http://www.pd-tutorial.com/english/ch03s02.html

[2] Pure Data Granular synthesis http://www.pd-tutorial.com/english/ch03s07.html

[3] Andy James Farnell, Procedural synthetic footsteps for video games and animation, http://obiwannabe.co.uk/html/papers/pdcon2007/PDCON-2007-FARNELL.pdf

[4] Andy James Farnell Tutorials, http://www.obiwannabe.co.uk

[5] Leonard J. Paul, Pure Data patches/examples, http://videogameaudio.com