



**UNIVERSITY of LIMERICK**  
O L L S C O I L L U I M N I G H

Modelling Footsteps: Procedural Audio in Games

Christopher Giles Paton

10050701

Master of Science in Music Technology

University of Limerick

Supervisor:

Dr. Kerry Hagan

Dr. Conor Ryan

Mr. Andy Farnell (external)

Submitted to the University of Limerick, September, 2011.

## Declaration

Modelling Footsteps: Procedural Audio in Games.

Supervisors: Dr. Kerry Hagan, Dr. Conor Ryan and Mr. Andy Farnell (external).

This Thesis is presented in partial fulfillment of the requirements for the degree of Master of Science in Music Technology. It is entirely my own work and has not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of the work of other people it has been fully acknowledged and fully referenced.

---

Christopher Giles Paton

27th September 2011

## *Acknowledgements*

*To Mr. Andy Farnell, who gave up his precious time and valuable knowledge to guide me throughout this project;*

*To Dr. Kerry Hagan and Dr. Conor Ryan, my two main supervisors, for their essential knowledge of program design, structure and problem solving;*

*To Messers Leonard J. Paul, Dylan Menzies, Federico Fontana, Fabio Morreale and Nicolas Fournel for their expertise and support;*

*To my peers, who helped me, encouraged me and tested my project;*

*And finally, to my parents, for their unwavering support, encouragement and pride.*

*Thank you all.*

## Table of Contents

DECLARATION .....	I
<i>ACKNOWLEDGEMENTS</i> .....	II
ABSTRACT .....	IV
INTRODUCTION.....	1
BACKGROUND INFORMATION AND RESEARCH.....	3
PROCEDURAL AUDIO.....	3
THEORY .....	8
IMPLEMENTATION.....	13
MODELLING FOOTSTEPS .....	17
INTRODUCTION .....	17
LITERATURE .....	18
PYTHON AND LIBPD .....	20
PURE DATA.....	22
PROBLEMS ENCOUNTERED .....	28
EVALUATION.....	31
FUTURE WORK .....	32
CONCLUSION .....	35
REFERENCES.....	37
APPENDIX.....	41
APPENDIX A - INTERVIEW WITH NICOLAS FOURNEL TRANSCRIPT .....	41
APPENDIX B – EMAILS FROM ANDY FARNELL.....	54
<i>Email Excerpt A:</i> .....	54
<i>E-mail Excerpt B:</i> .....	54
APPENDIX C – WORK OF FABIO MORREALE .....	55
APPENDIX D – EMAIL FROM FEDERICO MONTANA .....	56
APPENDIX E - EMAIL INTERVIEW WITH DYLAN MENZIES.....	57
APPENDIX F - CODE EXAMPLES.....	60
APPENDIX G – INTERVIEW WITH LEONARD J. PAUL .....	64
APPENDIX H – USER FEEDBACK FORMS .....	68

## Abstract

Modelling Footsteps: Procedural Audio in Games.

Christopher Giles Paton.

This thesis aims to support the applicability of procedurally generated audio for game audio development. Through existing research and studies of design approaches to synthesised audio modelling, and supporting interviews from academic researchers and industrial professionals alike, the suitability of existing methods and approaches are examined. Using Pure Data and Python code as a prototype, the author demonstrates footstep modelling using a subtractive synthesis approach, which responds dynamically to user input in a game environment. While results are still largely aesthetically subjective and synthetic, there is a clear argument for variable, dynamic and interactive audio in today's demanding, complex and ever-expanding games.

## Introduction

The purpose of this thesis is to implement existing research on footstep synthesis in a single environment, while exploring the applicability of Procedural Audio in games. The project is developed in Python using Pure Data (Pd) as a sound engine. The Libpd code library ([gitorious.org](http://gitorious.org), 2011) allows pre-designed Pd patches to be 'called' in the Python code. The resulting game demonstrates real-time footstep synthesis based on in-game parameters and user input.

First, though, it is important to discuss the existing literature and research regarding the fields of both Procedural Audio (herein referred to simply as 'PA') and footstep modelling. Much research has been done in both fields – PA has taken keen focus from academics and industry professionals alike, while researchers worldwide have made interesting developments in synthesising the sounds of footsteps. I shall also discuss existing methods of implementation within the gaming industry and various software solutions available and in development.

The second stage is exploring the initial phase of the project – building the game environment. As previously mentioned, this uses Python as a core language while Pd processes audio design. As the project developed, problems and issues arose in relation to structure, library functionality and compatibility, as well as cross-platform performance.

Programming Pd was certainly the most challenging element, as this meant attempting to incorporate existing research from different studies and approaches into a suitable, cohesive design. While some research was initially designed using Pd, other work was written using languages like C, and required suitable adaptation (despite Pd being largely written in C code). Discovering which elements of particular research worked best proved challenging too, as it required regular experimentation and testing. Many avenues proved too time consuming for the scope of the project and were abandoned.

While issues arose, it became clear that there is plenty of scope for future research and development. With a larger time frame and suitable foundations laid, opportunities for pursuing abandoned elements are profoundly viable. These areas shall be discussed in the penultimate section of this thesis.

## Background Information and Research

### Procedural Audio

It is important to begin by first defining what PA is and means. As Andy Farnell, a prominent figure in PA research and author of *Designing Sound* (2010a) states, 'it's probably better to describe procedural sound by what it is not. It is not pre-sequence, pre-recorded sound and music' (Farnell, 2007b). Essentially, PA is synthesised sound often based on a pre-existing sound or model, created in real-time, dependent on a rule-based structure and user input. PA models are defined by their user-specified limitations (or characteristics) that shape their behaviour and functionality. Procedural refers to the 'process that computes a particular function' (Fournel, 2010) (or 'procedure'), that ultimately leads to some end goal, in this case, the creation of audio. It is important to differentiate PA from sample-based audio too, as recorded sound 'implies it has already been realised (concretised) as a signal, while "procedural audio" is still code (potential sound). You can't just play procedural audio; you have to *run it* [sic] in a context that makes sense' (Farnell, 2010b, p. 315-316). The definition I offer particularly suits game audio design, in that it allows for the creation of *dynamic* audio that has interactivity (user input) at its core.

There are many methods of creating synthesised audio, which are undoubtedly common knowledge to many people already; these may be frequency or amplitude modulation, subtractive synthesis or granular synthesis or indeed a vast plethora of pre-existing methods widely used today. The key element is to create a sound using these techniques that is useful in the context of a game soundscape. For this purpose, rules are required that govern the behaviour of synthesis parameters. These rules may often include generative and genetic algorithms, fractals, L-systems, expert systems or cellular automata, which may govern parameters such as frequency spectrum, amplitude curve, probability distribution, damping, physical object interactivity (material excitation in relation to game physics) or indeed any other parameter deemed viable to audio synthesis. What ultimately results is a computable algorithm, or model (as code),

that creates audio in real-time, triggered by a variety of possible actions (physical events in the game engine, user interaction).

In game audio today, sound designers are able to create incredibly detailed sounds through modern recording and production techniques. Many consoles today allow high quality 96kHz, 24-bit audio samples to be used in-game offering the user a vivid soundscape – so why use PA?

The main reasons for using PA are inherent flaws of sample-based audio itself; sample-based audio doesn't change, bar tweaking with various effects, which ultimately uses up CPU processing power. To create dynamics, multiple samples must be recorded, edited, stored in physical memory, buffered into temporary memory, cued and then removed from temporary memory when no longer required. These samples take up incredibly large amounts of data. With a sample rate of 44,100 samples per second, at 16-bits (or two bytes), in stereo (2-channels), 60 seconds of audio is just over 10 megabytes in size (TheAudioArchive.com, 2011). This consumes both hard disk memory and RAM, which are both limited, as all game systems require both media to implement physics and game rules, too. In fact, developers place limitations on exactly how much memory teams can use (interview with Nicolas Fournel, see Appendix A). While hard disk memory, portable media and RAM capacities are always increasing, so are game worlds and environments, meaning that this dependence on sample-based audio shall become increasingly difficult to sustain.

PA has been around since computer games began, thanks to chips like the General Instruments AY-3-8910, or the Texas Instruments SN76489 (both with three square wave oscillators and one noise generator). These chips generated the bleeps and honks that we remember from the seventies and eighties (Collins, 2008). These sounds in no way represented real-world sounds, so it is understandable that advanced sampling technology triumphed. Today, though, with the power of multi-core processors and hyper-threading, computing synthesised audio offers far greater potential than it did over three decades ago.

Procedural generation of game content has been widely used already – not just for audio, but for game environments, too. Procedural, Inc. is the company behind CityEngine, a procedural city generation toolkit that can generate blocks of highly detailed cityscapes in real-time. Parameters control the age, height, dimensions and style of buildings, as well as road dimensions and layout (Procedural, Inc., 2011). Streets can be drawn in and populated in a rather simplistic fashion, so why shouldn't this be done for sound?

To a certain extent, it already has been. Electronic Arts have been widely commended for their attempt at procedural music in their game Spore (2008), in which user behaviour governs what type of music is being played at that time. Instrumental density increases and fades depending on user interactivity, creating a far more dynamic score, rather than having themes loop and thus become repetitive and boring (Collins, 2009). Music can be generated procedurally too using previously mentioned algorithms to control musical parameters and performance, while actually synthesising instrumental tones in real-time thanks to advances in modal and physical modelling approaches.

PA for sound design is useful for a multitude of reasons. It is most suitable for repetitive sounds that occur regularly and require minor variations to eliminate the possibility of sounds becoming boring, like the sound of footsteps. Wind and water sounds normally consume a large amount of memory to keep a sense of realism, while a PA model could be used to generate these sounds endlessly. Sounds *en masse* are another good situation where PA would be applicable. Impact sounds or sounds that are dependent on user interactivity are suitable, as parameters between objects would be dependent on the physics parameters of the game engine (stimulated by the player). Motion sensor-based controllers represent direct user activity, and thus audio must sound varied and contrasting to retain a sense of realism and create immersive environments. One last area (though this list is not the entirety of PA-applicable areas) is the realm of sandbox games, open-ended universes that have so many varied sounds based on similar models that to recreate them all would just take up far too much memory. Procedural generation of such a vast array of sounds would also reduce the

requirement to create vast sample libraries, which can be difficult and awkward to manage and navigate (email interview with Dylan Menzies, 2011, see Appendix E).

Currently, game developers use development kits such as UDK (developed by Epic Games) or Unity (developed by Unity Technologies), and middleware including Wwise, XACT, FMOD (FMOD is developed by Firelight Technologies, Wwise by Audio Kinetic, and XACT by Microsoft), to implement their creations, some of which have procedural audio tools built-in, but there is still a heavy dependence on sample-based sound design. Some developers, such as Sony Computer Entertainment and Electronic Arts have made attempts at creating their own, in-house procedural audio tools, which I shall discuss later. These attempts are among the foundations for innovative PA implementation.

One of the potentially negative aspects of procedural audio is that it often requires scientific or technical knowledge of how biological or mechanical systems function (the object one wishes to model) and sufficient experience in programming to recreate these sounds using relative software or script compilers. If PA is to be encouraged, education at third level or even second level in relative fields would be essential to ensure sufficient knowledge to future audio designers and programmers. Also, a vast majority of current employees within the industry would need a certain element of retraining. As one can see, this has obviously led to difficult obstacles, and industrial inertia has caused increasing setbacks (Farnell, 2007a; Fournel, 2010).

It is interesting that most research into modelling and procedural audio seem to remain largely academic, with little collaboration between industry and academia. Both Leonard J. Paul, a sound designer and PA advocate from Vancouver, Canada and Dr. Dylan Menzies, an academic from De Montfort University in the UK, feel that the gap between the two realms is quite vast. Essentially, academics have a much greater timespan to work on hypothetical situations, 'to generate ideas and turn them into papers' (Menzies, see Appendix E), while 'industry is typically a short [term] view on solving practical problems'

(Paul, see Appendix G). Despite this divide, Paul theorises that a solution would be to introduce academia to industry via research departments, but due to the financial responsibility placed on publicly-held companies for quarterly profits, coupled with the fact that research can often 'produce no direct benefits in a given quarter', it is unsurprising that this has not been ventured before (Paul, see Appendix G). Paul's hopes seem to lie with privately held companies with 'a longer view of producing interactive entertainment, such as Valve, [that] might have more of a capability of having a strong research department'.

## Theory

Model design is one of the keystones of PA, coded templates of a sound that designers can alter via specialised parameters and rule systems. There are two different approaches to creating these models: teleological (using the laws of physics, or, bottom-up) and ontological (modelling something based on its biological appearance, or, top-down) modelling (Fournel, 2010).

While essentially analysis is a top-down method, and re-synthesis is a bottom-up method, teleological and ontological modelling reflect *attitudes* toward efficient model design. Even though both approaches may end up with the same result, analysis, re-synthesis and computational efficiency could be completely different.

Teleological modelling is the basis of taking a sound from the real-world, analysing the physical components of that sound, and then recreating them using software or suitable programming tools. Farnell takes this as his core approach to sound design. His approach is based upon a 'pillar' system, where physical, mathematical and psychological theory are the basis for developing technique, which one finally implements in the design stage (Farnell, 2010a). Through this approach it is possible to recreate accurate representations of various sounds using synthesis techniques and clever Digital Signal Processing (DSP). One major advantage of the teleological approach is that it leads to re-usable 'pieces' of code that allow future system construction and adaptation speed to increase dramatically.

Many other researchers take this approach, too. The Precomputed Acoustic Transfer (PAT) model developed by James et al. (2006) analyses the wave propagation effects of a solid object and attempts to simulate this through the development of a run-time algorithm that incorporates elemental physical components of acoustic propagation.

The work of van den Doel and Pai on *Modal Synthesis for Vibrating Objects* (2003), illustrates the use of modal synthesis to replicate the resonant

characteristics of a solid, vibrating object. They do this through the use of dampened harmonic oscillators, whose frequencies and dampening are decided by the geometric and material properties of the object, while the coupling gains of the oscillators are governed by the force by which the object is struck by an exciter.

Cook et al. (2001) worked on synthesis based on the motion of physical objects. This was done by analysing the physical properties and geometric construct of the objects, relative to their motion in space. They isolate the vibrational components of the object that correspond to audible frequencies. Their work supplemented a visual synthesis project as well, and commented that ‘the additional cost of computing the audio with our technique is negligible’, and that the sounds are generated in real-time without the use of pre-recorded sound (Cook et al., 2001, p. 530).

Each of the above is an example of a teleological modelling approach, which has its advantages when it comes to audio synthesis. Physical properties can be controlled and designed to the model’s requirements, and the end result can be incredibly realistic with efficient computation. Interaction between objects can be simulated accurately if interactions are analysed and understood.

Unfortunately, these can be negative elements, too. In-depth mathematical and physical knowledge (not to mention incorporating acoustic propagation and psychoacoustic effects) are required to create suitable models, which to most sound designers could be quite overwhelming. Knowledge of DSP is crucial, though this may be less problematic, as many sound designers may have knowledge of this area. Models may also be difficult to adapt to different situations, depending on the requirements of the project and the intentions of the sound designer. In this situation, a variety of models would be necessary. Creating models through teleological processes can be time consuming, due to the requirements of data gathering and in-depth analysis of various physical properties and functions. When a sound designer is under pressure due to deadlines and time is of the essence, this may not be realistic or feasible. Sounds

contrast from project to project and from environment to environment, so it is not unreasonable to presume that a model for one particular material or environment may not be suitable to another, and thus tweaking of physical (or many other) elements may be necessary. In a worst case scenario, a new model may be required, increasing time consumption.

Ontological modelling is a slightly different approach. Sounds are analysed for their natural characteristics and behaviours from an original sound. It is far more parametric, meaning that 'salient spectral or time features are extracted for resynthesis in the absence of an underlying causal model,' (see Appendix B) which creates a polar position between teleological and ontological approaches.

A sound designer may record, edit and manipulate their own sound – creating a model for this teleologically would be rather difficult, especially if the sound has no basis in real-world physics. To model this sound then, various analysis processes must be carried out. Fourier Transform analysis, amplitude curve distribution, peak detection, pitch contour, beat detection and windowing function detection are just a few audio analysis processes that are available to sound designers, commonly available in software packages like Spear, AudioSculpt, Sonic Visualiser or MatLab.

Analysis of audio files can give the user a more accurate picture of how the sound is constructed and thus offer the user control over certain parameters when reconstructing their model of the original audio. One of the advantages of this approach is that once model is analysed, various aspects of that sound could be applied to other sounds to create variability, or new sounds; the amplitude curve of a dog bark could be applied to the frequency distribution of a car engine revving, which would perhaps create a much more aggressive acceleration sound, due to the sharp attack segment of a dog bark amplitude curve. While this is a simplistic example, the illustration is clear.

The biggest problem facing this approach from the sound designer's position is that specific tools for constructing models are not widely available, though it is

clear that many developers have begun working on such a process (Fournel, 2011b).

Higher-level tools that reduce the need to deconstruct and reconstruct audio are crucial to the successful implementation of PA, regardless of the approach taken. These tools would reduce time consumption and increase workflow, as well as allowing the designer to create models with suitable control and adaptability. Paul discusses how,

‘The issue in the industry is time, so any tools that can make most of the process faster for the sound designer is likely going to help the overall sound of the game. The problem is that often one tool or middleware is meant to solve all the problems of the sound artist. Hopefully the audio programmer is able to help the designer by providing solutions that help the remaining challenges.’

(see Appendix G)

This is particularly relevant to procedural methods, which at the moment sound far too ‘synthetic’ to create the right impact in a proper framework, such as the first-person shooter genre.

Various analysis and synthesis methods are widely used to create synthesised audio that efficiently model original sounds. Modal synthesis is used to create accurate representations of solid objects with vibrating modes. Boneel et al. (2008) use short-time Fourier Transforms to approximate modes, creating representations of modes of multiple objects in a three-dimensional environment. Frequency-domain modal synthesis does seem to have a slight degradation in overall quality, though is computationally far more efficient in comparison to *Time*-domain modal synthesis. This is a key development, particularly for use in game audio design, as computation limitations restrict the amount of audio processing power.

James and Zheng (2009) developed an algorithm for synthesis of fluid harmonics, that is, bubbles. In their model, ‘bubble oscillators’ induce fluid-air vibration, which creates audible sound pressure in the air. Using Helmholtz

Green's function they are able to simulate various liquid scenarios, that are possible to be generated at run-time, incorporating physical parameters from a three-dimensional environment. Their model incorporates physical elements such as entrainment, vibration, advection and radiation. Acoustic considerations such as the head-related transfer function, as well as using an amplitude filter to remove noise artifacts that may occur during synthesis, are implemented. While not efficient enough for calculations to work in real-time, work by Raghuvanshi et al. (2010) demonstrates a way of creating dynamic real-time diffraction of sound in a three-dimensional environment by measuring impulse responses and separating early and late reflections to simulate acoustic effects such as reverberation, low-pass filtering (created by obstructive objects) and diffraction. While not technically *synthesising* sound, this work reduces the requirement for pre-processing of audio files, and ultimately would allow for greater dynamics in three-dimensional environments. There is also a dependence on synchronisation with the game engine, from which the acoustic radiation transfer system draws key parameter information, which in turn is usually deduced from user input (character movement or action). They implement their design using Valve's Source game engine SDK (Software Development Kit), which illustrated realistic auralisation of characters within the environment, as well as user behaviour and interaction with objects. The research above and that carried out by others has led to greater developments in synthesised audio and PA design interest. It is clear that a diverse field of research exists, and perhaps partly due to increase computational efficiency, developments are occurring.

## Implementation

There are various methods for implementing PA; a multitude of commercial analysis and synthesis software exists, while many in-house programmers are working on their own tools to aide accurate PA design and implementation. Middleware software solutions provide pre-designed PA models that are designed to work with various other game engines (the aforementioned *SoundSeed*, by AudioKinetic). Game engines are the driving force behind most of the action the gamer sees on screen, and ultimately, it is essential that game engines are compatible with PA models and designs.

This, unfortunately, is where problems begin to arise. Most game engines are designed to work with sample-based audio, and thus are designed to cue files stored in memory, not patches or modules with data that *represent* audio. While some studios and developers use their own game engines, other game engines like the Unreal Engine and Unity are widely used throughout third-party developers. Microsoft's Xbox consoles use the Xbox SDK and XACT (Cross-platform Audio Creation Tool) to design games and implement audio.

Pure Data, or *Pd* as it is known by its users, is an object-oriented module-based programming language that bypasses conventional coding through pre-made objects, and is designed (by Miller Puckette) with audio in mind. Complex patches (originally coded in C) can be made with relative ease through connecting various function objects using virtual patch cords to perform a particular operation. As an open-source program related to the MaxMSP software (which was also initially designed by Miller Puckette), many users have created their own extended modules for open use.

Pd has been used as a design tool by Farnell, who has created models for sounds ranging from traffic-crossing signals to alien blasters (Farnell, 2010a). This amalgamation of code examples stems from his ambitions to create an audio demonstration reel using purely PA-generated sounds, which he first completed

in 2005. A near ten-minute sequence of synthesised environments was the result (see Appendix B).

EA also used Pd in the score development for EA's title *Spore*, which has been hailed as a successful implementation of Procedural Music in games (GameSpy, 2008). I shall speak more of Pd when discussing my own work.

MaxMSP has been widely used, also. Synthasaurus is a Masters project developed by Robert Martino (2000) that synthesises animal vocalisations using Max and modified MSP objects. It draws upon previous work by Perry Cook, who created a waveguide model for vocal tracts called SPASM (Cook, 1991).

Böttcher and Serafin (2009) used MaxMSP and a Wii remote control to create virtual sword-based action sounds, again using modal synthesis models.

While these models may be very dynamic and interesting, no attempt (that I am aware of) has been made to integrate Pd or MaxMSP as modelling tools that can efficiently be implemented into a game engine (EA's adaptation of Pd was used for musical scores, rather than sound effects). Phya is a library written in C++ that can be attached to a game engine to facilitate 'physically motivated audio in virtual environments' (Menzies, 2009). Menzies aim was for Phya to be a framework that allowed for the creation of dynamic models that can be controlled by the physics engine. While C++ is an object-oriented programming language, it still takes considerable time to write code; Menzies realised that this may be an issue for sound designers, and thus began working on an authoring system to test and experiment with models, which he calls Vfoley. The system is designed to enhance controller function and will hopefully act as a catalyst for the Phya library.

TAPASTREA (Techniques and Paradigms for Expressive Synthesis, Transformation and Rendering of Environmental Audio) is an analysis and resynthesis software solution designed to create synthesised audio environments using specific algorithms, breaking down audio samples into

deterministic and transient events, and including stochastic background information. The user is given control over a suitable range of parameters, including 'density, periodicity, relative gain, and spatial positioning of the components' (Misra et al., 2006, p. 1). The analysis and resynthesis is quite efficient in terms of quality, though after arrangement there seems to be no ability to embed models into a game engine. Rather, it is simply a tool for generating new, more complex scenes.

SPARK (which stands for Scee Procedural Audio Real-time Kernel) is one of the first analysis and resynthesis software tools designed specifically for PA. Designed in-house at Sony Computer Entertainment Europe's (SCEE) *London Studio* by Lead Audio Programmer Nicolas Fournel, SPARK is a tool that instantly creates dynamic models of audio samples, while still retaining parametric control. In an interview with Fournel (see Appendix A), he explained to me how SPARK is a high-level sound design tool created with the game audio designer in mind. Once modelled, parameters can be mapped onto other models, creating different sounds and resonance characteristics. Once a model is designed (through GUI interaction, rather than coding), it can be implemented into the game engine used at the *London Studio* (Fournel, 2011a; 2011b).

Audiokinetic's *WWise* middleware solution allows audio designers to implement pre-made PA models through their *SoundSeed* module. Various effects can be implemented and controlled efficiently and dynamically, though the variety of models are limited. A key concern of this implementation method is that although the sounds may be dynamic, they may not contain the acoustic properties desired by the sound designer, resulting in a return to sampled audio files (AudioKinetic, 2011).

Game engines then, which are the core to game implementation, need to facilitate new sound design methods, such as PA. Various game engines exist, such as Unity and the Unreal Engine. Unity can communicate with MaxMSP through UDP and OSC mapping, which has already been used by students at Expression College in Emeryville, California, to create new audio environments

for games (Creative Digital Music, 2009). It remains to be seen whether this can be used to create truly dynamic effects that allow MaxMSP to communicate with the Unity engine, though Leonard J. Paul illustrated the OSC routing possibilities of Unity that can send various packet data to Pd, and then onto the sound driver code (Paul, 2010).

Each of the above design systems are possible implementation processes that are potential systems for creating PA models for games. It is evident from research (particularly in relation to the work by Menzies; Böttcher and Serafin; and Nordahl), that these designs are not suitable for game implementation, yet. Also, Fournel expressed concerns over the computational efficiency of Pd and MaxMSP, as the more complex patches become, CPU processing becomes far more expensive. Each object used within a patch requires even greater calculations. Thus on larger scales, Pd and MaxMSP may not be viable solutions for sound designers, though this remains to be seen.

TAPASTREA allows for creative sound design and dynamic modelling, though parametric control is not retained after the resynthesis stage, nor is there any function to communicate with (or implement within) a game engine. It is purely a design tool.

SPARK is perhaps the most concise solution at time of writing. This is undoubtedly due to the fact that it has been designed with game audio as a key concern, with PA as a core focus of the design stage. It is unclear at this stage as to whether SPARK has been implemented and used within a released game, though it is certainly a step forward.

# Modelling Footsteps

## Introduction

It was deemed appropriate for this thesis project that a practical work demonstrating applied theoretical knowledge was the correct direction to take. With sufficient research on PA and various modelling approaches and techniques, a project that attempts to incorporate this information would help to reinforce aforementioned knowledge.

While my own knowledge of programming was limited, over the projected period I attempted to incorporate various elements of the gaming pipeline using PA as the core focus. Using the Python coding language as the game engine with Libpd and Pd as the audio engine, this project would attempt to incorporate existing research into a model of footstep impact sounds over various textures, that would respond to user input in real-time. Footsteps were decided upon as a focal point due to several key factors: the repetitive nature of the sound, the varied surfaces that could be designed, the relatively sufficient literature and research already available on the area.

## Literature

One of the key components of this project was the existing research on footstep modelling. The aim of the project was to incorporate elements of that research into a singular model using one language (in this case, Pd would be responsible for audio synthesis). It was important that no sample-based audio be used, as that would only increase RAM and hard disk consumption. This ruled out elements of granular synthesis and subtractive synthesis that use complex waveforms, as complex waveforms require being buffered from disk memory into temporary memory – a cost to be avoided. Also, this work was *specifically* aimed at an interactive, traditional PC-based or console-based, casual gaming environment, and as such also ruled out work on virtual reality simulations like that of Serafin et al. (2009) and Nordahl (2004). The key literature focused on then is that of Cook (2002), Farnell (2007a, 2010a), Fournel (2010), Bresin et al. (2001) Fontana and Bresin (2003) and Morreale (via e-mail, 2011, see Appendix C).

Cook's work was arguably the most influential, as he illustrated clear model design, processes implemented, as well as testing data. Following an ontological approach, the entire process is broken down into core components of analysis (envelope extraction, Linear Prediction Coding (LPC) and wavelet decomposition) and resynthesis (parameterisation, filter design, particle density), which gave a much clearer understanding of his approach. As influenced by his previous work 'PhISEM' (Physically Inspired Stochastic Event Modelling) (Cook, 1997), I have presumed that his work is similarly written in C++ code. Despite mentioning in the literature that, 'all source code is made publicly available for free', I was unable to locate it on his own website, and he failed to respond to further communication attempts. As a result, analysing his work in detail became difficult.

The work of Fontana and Bresin (2003) was implemented in a Pd environment, which was a key reason why Pd was chosen as a suitable environment to develop this project. Their approach was far more teleological than Cook's, using complex

mathematics to resynthesise the core physical components of various crushing and crumpling sounds, as well as introducing music-based control models such as *legato* and *staccato*, initially proposed by Bresin et al. (2001). As a secondary objective, it would have been interesting to implement some of these aesthetics in my own system. Unfortunately, Mr. Fontana explained via e-mail (see Appendix D) that much of this work was unavailable to retrieve. This too, created difficulties.

Farnell's work was undoubtedly the most accessible, with source code for many of his models freely available and illustrated in *Designing Sound* (2010a). He also explained a great deal of the signal processing with a specific focus on Pd, as well as introducing interesting filtering techniques using pre-designed Pd objects.

The work of Fabio Morreale at the University of Verona (2011) on footstep modelling was a great insight into current work. His approach uses an Arduino to measure readings from sensors to give an amplitude curve, which influenced my own technique of creating amplitude curves from existing sample data.

## Python and Libpd

Initially, it was intended that Blender (an open-source graphical development software) be used as the visual element of the project, but due to the complex nature of the software and its secondary function in the overall project, it was decided that the PyGame modules available for Python coding be used to generate the game environment. PyGame is designed as a highly portable set of modules for game development using the Python development language. A game-style environment was chosen to tie-in with the notion of interactivity, and that PA is generated in part by user input – this approach allows the user to directly interact with the system.

Python was designed with a similar purpose to that of PyGame – being portable, and quite high-level, with an interpreter between the code and the processor. This allows for code to be written quickly and easily, and this was a primary reason for using it as the game engine. Python was also one of the languages available that had libpd written for it. Libpd is a ‘wrapper’ for various coding languages that allows one to use Pd patches as a form of audio engine.

Due to the relative simplicity of Python, developing an elementary game environment was completed within a few weeks. This included learning how Python operated, reviewing libraries and GUI testing. It was initially intended that the main display should have interactive features, such as being able to change surface textures manually. But, the wxPython GUI library that was initially used proved difficult and rather incompatible when working with PyGame (embedding PyGame in wxPython frames), resulting in unwanted errors. After much testing and review, I decided that it would be best that the project continue without a GUI. It was indeed unnecessary for the final setup.

Embedding libpd was undoubtedly challenging and at first confusing. There were no well commented or instructed ‘hello world’-style demonstrations available either from the libpd source files or that could be found online for the true beginner, as many assumed a certain level of programming ability. Installing the package was difficult too, but putting ‘Python setup.py install’ into the Terminal

(on Mac OSX) or command line when in the appropriate directory successfully built and installed the required files.

Through several attempts I managed to adapt the 'pygame\_fun\_test.py' program included in a sub-directory of the Libpd source files, which included all necessary code to allow me to use my patch within the PyGame environment. Apart from the initial setup, all that was needed was a while-loop that buffers data coming from Pd (see Fig. 1 in Appendix). It is perhaps important to mention, too, that Pd does not actually run along side Python. Rather, the patches are read inside the Python program when it is compiled or run, reducing the computational cost of having a separate program running.

A simple controller system was setup using a for-loop looking for PyGame 'events'. Events in PyGame are user-triggered moments that can be easily detected by the PyGame module. A computer keypad is used, with 'Shift' and 'Ctrl' as speed modifiers. When PyGame detects an event, it sends a specified message to the pre-designed Pd patch previously loaded by Python – in the case of this project, it informs the Pd patch 'Footsteps2.pd' of the speed of the character on screen. All values for speed are justified by ear.

## Pure Data

The main stage of the project was the audio engine designed in Pd, and was undoubtedly the most challenging element of the project. Once the Python engine had been built and successful communication occurred between Python and the main patch, with real-time audio synthesis, it became important to develop the model that would synthesise footsteps.

Before design began, it was important to understand the *process* of footstep synthesis, according to pre-existing research. It was evident that the process was split up into two key components: the audio synthesis and the exciter, or the surface and the foot, respectively. It is also important to understand that none of the research done before has been designed to directly function in a gaming environment, so all research had to be adapted for this purpose. All Pd objects herein are marked with closed square brackets.

In the work of Fontana and Bresin (2003) two abstractions exist; the first one is called [control\_crump], which feeds control data into the second called [sound\_synth~], which becomes responsible for synthesising the appropriate audio. Essentially, the control data creates an amplitude curve for the audio signal generated by the audio synthesis component. This is also seen in the work of Farnell (2007b), Cook (2002) and Morreale (2011), where Farnell uses a three-point polynomial coefficient, Cook uses control data gathered from sample analysis, and Morreale's data is gathered from pressure points triggered by sensors and received by an arduino.

The GRF, or Ground Reaction Force is a common element, as well, which simulates the pressure with which the foot interacts with the surface and is the force returned by the surface. Morreale extracts his directly from the pressure sensors, while Farnell uses an approximated scaling factor for his three-point polynomial curve.

Due to the stochastic nature of footstep/surface interaction and white noise, subtractive synthesis using various filter methods seems to be synthesis method of choice. Fontana and Bresin (2003) created a variable filter depending on the physical parameters of their crushing or crumpling object. Farnell (2007b) used [vcf~], a Pd object that works as a voltage-controlled band-pass filter that takes a signal as a control variable, creating peaks and spikes at stochastic locations. Morreale (2011) used user specified band-pass filters incorporating frequency and Q value, while Cook (2002) used LPC to determine filter values.

This project uses amplitude values gathered from pre-analysed sample data of recorded footstep sounds, as control data for various synthesis models representing varied surface textures. Float control values in a range from 0.0-1.0 represent the amplitude of the current texture. Amplitude, ultimately, is the control value that represents the current position of the footstep strike. The audio is generated in a separate component, through a subtractive approach using various filters and white noise generators. Each surface texture, as well as player speed, can be changed through the player's interaction in the Python game, and alters the control value destination (which texture sub-patch the values are sent to) or value (the speed at which the values are read), respectively.

This modelling approach decouples control and signal values as carriers of behavioural and spectral features, respectively. What this means is that the values created by the amplitude curve generator are responsible for behavioural characteristics of the resulting sound. Parameters such as weight and speed can be factored in at this point to affect variability. If one were designing a model that was affected by in-game physical parameters, such as gravity for instance, information from the physics engine would be incorporated at this point.

The spectral process is the filtering process, or whatever form of audio generation one wishes to use. Here we can incorporate other variables that would affect the sound only, regardless of character dynamics. This would factor in various environmental variables drawn from the physics engine, such as

texture variability (how much gravel there is) or surface density (hard ground or soft mud). Keeping these behavioural and spectral elements separate ultimately leads to greater control and model design, a key factor of any object-oriented programming environment.

I began by using Farnell's work as a template, as it had a pre-existing functional structure. This made updating components much simpler, as sub-patches or abstractions could be disconnected and reconnected for testing and comparison. It was important to work on the project in segments, leaving no element unattached or idle, as it would affect the overall functionality of the patch. The function of each component must be exact to allow efficient construction and debugging. This became clear from the offset, as time was spent debugging one component when it was eventually realised that it was a separate component that was causing issues, such as distortion caused by filter rather than amplitude curve.

As Farnell's polynomial coefficient curve was an approximation, it was decided that this would have to be replaced by a method that more closely modelled gait. I was most interested in Cook's work, in which he creates a variable amplitude curve generated from multiple data samples and created new curves by using averaging and standard deviation. This is no easy task in Pd, so I felt it worth attempting to create an object for Pd using 'C' code. After several days though, the task was identified as being out of the scope of this project (which shall be discussed further in a future section), and a scaled down version was drawn up.

The final amplitude curve generator was also partly inspired by the work of Morreale, who created arrays in Pd using .txt data files containing numerical values for Y-values in a two-dimensional array, which were loaded into separate buffers using a Java external (written by Morreale). In my own model, I used the 'Amplitude Follower' plug-in from Sonic Visualiser to retrieve amplitude values of an pre-recorded sample and exported the annotation layer as a .txt data file. This file contained two columns: one with the time value and the second with the amplitude value. I wrote a small Python program to separate these two values

(only the amplitude value was required) and to create an end-of-line marker (';') that Pd would read (see Fig. 2 in Appendix).

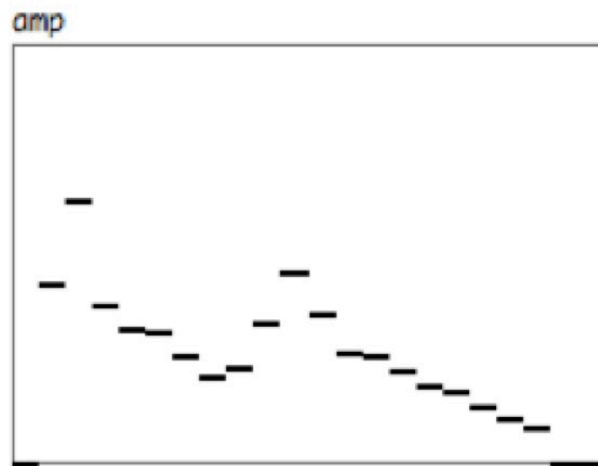


Figure 1 - 'amp' array filled with data from [textfile]. The vertical axis represents a linear value from 0-1 while the horizontal axis represents time, which varies due to control rate.

In the subpatch [pd amplitude\_generator], using the [textfile] object in Pd I designed a patch that would read in each amplitude value from the .txt file as Y-points on a specified array (called 'amp', see Figure 1) for each X-point available (see Fig. 3 in Appendix). The size of the array depended on the amount of data gathered from the original sample. This break-point method creates a scalable method when outputting the values to a signal, both in terms of speed (creeping, walking or running) and amplitude value (weight of character or footwear). This offers parametric control over particular aspects of the model design, creating a more adaptive model for use in game environments.

The next stage involves reading the array as a control variable (numerical rather than signal values). [Tabread4~] uses a four-point polynomial to read through a look-up table (in this case, 'amp'). To read through the array efficiently a subpatch, [pd amp\_counter], was created which cycles through each array value at a customisable speed variable.

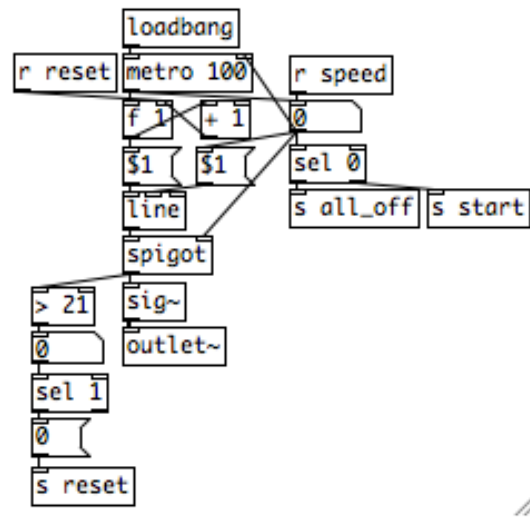


Figure 2 - The [pd amp\_counter] object.

This variable can then be controlled from the Python environment using the libpd messaging system. The [metro] object connected to the [snapshot~] object allows the patch to extract numerical values at a specified interval (by [metro]), which becomes the amplitude value at that point. Using a [random] object it was possible to create a scaling value that would approximate a standard deviation value *à la* Cook (2002) of an estimated  $\pm 10\%$  deviation. This means that the amplitude curve is never exactly the same, despite being from a single data source. This helps to simulate the variability of real-life and reduce the monotony found in sample-based sound design approaches, a key feature of PA.

For the spectral element of the patch, most studies used subtractive filter combinations from noise sources to approximate textural properties of a specified surface (Farnell, 2007b; Cook, 2002; Morreale, 2011; Fontana & Bresin, 2003). Keeping the synthesis process separate to the control data, a separate subpatch, [pd audio\_generator] was created. This contains separate subpatches that synthesis different textures.

For more efficient design, a filter subpatch, [pd Filter1] was created that allowed easy adjustment of filter values. In this way, multiple filters could be combined to create a signal from white noise, which would receive its amplitude value from [pd amplitude\_generator].

The most important element of this patch is that no value is ever permanent – they are always changing. This reinforces the stochastic characteristics of footstep resonance (see Appendix F for further code examples).

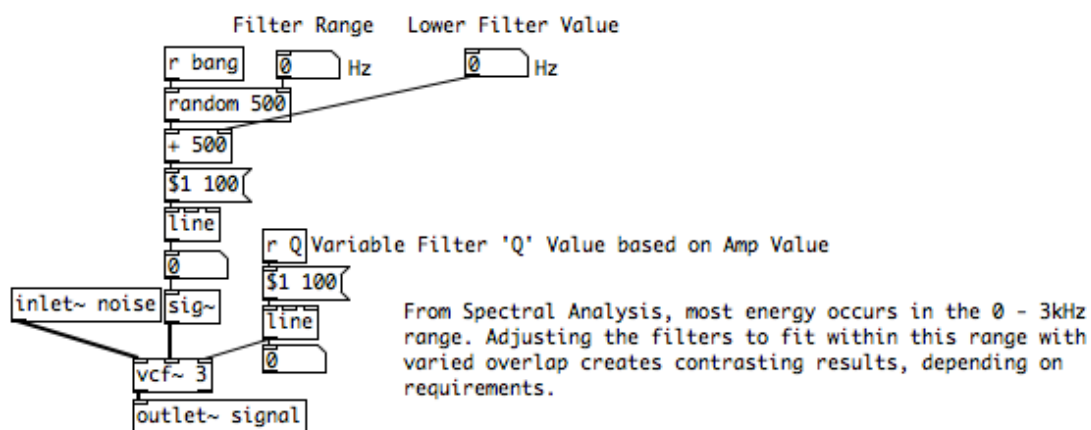


Figure 3 - The filter subpatch created. Notice how adjustable values control 'ranges' rather than specific values.

## Problems Encountered

Finding the correct approach to audio modelling was certainly a difficult obstacle. Does one analyse each physical component in detail to understand its acoustic and sonic make-up, or does one approximate these behaviours by perceptual testing and adjustment based on spectral analysis? Should one's approach be ontological or teleological? Ultimately, and to quote Husserl and Merleau-Ponty (cited in Farnell, 2010b), 'If it works, use it'. This philosophy is informed by phenomenology, believing that as long as the aesthetic result is correct, the path is irrelevant. While most of my work is ontological in construct, a clearer understanding of the physical processes involved would have undoubtedly yielded better results. Footsteps are undoubtedly stochastic in nature, but why does gravel create the sound it does? What is it about compacting snow that has such a distinct resonance?

Scalability was difficult to judge. This refers to comprehending the processes involved just by reading the literature available. This was in part due to the lack of accessibility to original source material of existing research. It is one thing to understand the processes when written in English, but translating that into C++ code or Pd objects is quite another thing entirely. While I have experience with writing in Java and C++, I had never worked with synthesised audio before, and this created very time consuming hurdles. Perhaps the most convoluted element was attempting to design Perry Cook's amplitude curve model in a Pd object. While this was certainly not impossible, it would have taken more time than was available and was ultimately abandoned for the final scaled-down approach.

Another issue, which is most relevant to the gaming element of the project, was the development time of the project. Taking roughly two months of development to synthesise footsteps is completely unrealistic in terms of gaming project deadlines. Obviously, the code does not need to be rewritten every time it is needed, but developing an entire library of sounds (anything from World War Two to farm animals) would take a colossal amount of time to begin with. While I am not as experienced a coder as one would find in a typical game development

studio, time for research and study would certainly need to be undertaken to understand components necessary for model construction. This approach contrasts greatly with Fournel's SPARK software.

From my own experience and reviewing the above literature, I feel that subtractive synthesis of white noise is perhaps not the ideal way to synthesise footsteps. While some results are convincing in ways, when one understands the source material the façade fails, and ultimately all surfaces have a 'noisy' finish. This may of course have been partly due to my own filter design. Basing my filter frequency choices on data gathered from my own spectral analysis (in Sonic Visualiser), there is a certain margin of error due to FFT bin size, ambient interference and frequency variability. Also, the Pd filter objects available give limited control. Cook's work uses frequency decay values, which is not possible in Pd.

As mentioned by Dylan Menzies (see Appendix E), instances of code in Pd cannot easily be destroyed and recreated when needed, as one could in C++. This means that CPU power could be unnecessarily wasted. Pd can be unreliable and unpredictable, making it a poor choice for commercially released products. Menzies also mentions the possibility of a software solution like Pd, that offers the programmer module-based interaction, though also offers the option of interacting directly with the module's code, creating on-the-go alterations to core C/C++ code. This could perhaps create more control and reliability, provided the user has sufficient programming experience. Obviously, standardisation of such software could prove difficult.

A final issue was in the libpd functionality. Libpd only supports the 'vanilla' version of Pd, rather than the 'extended' version, which contains various extended libraries and features not available in the vanilla version. This resulted in certain code having to be designed in a more complex manner, using far more modules than intended. It also limits the range of functionality that patches can have (see Figures 3 & 4).

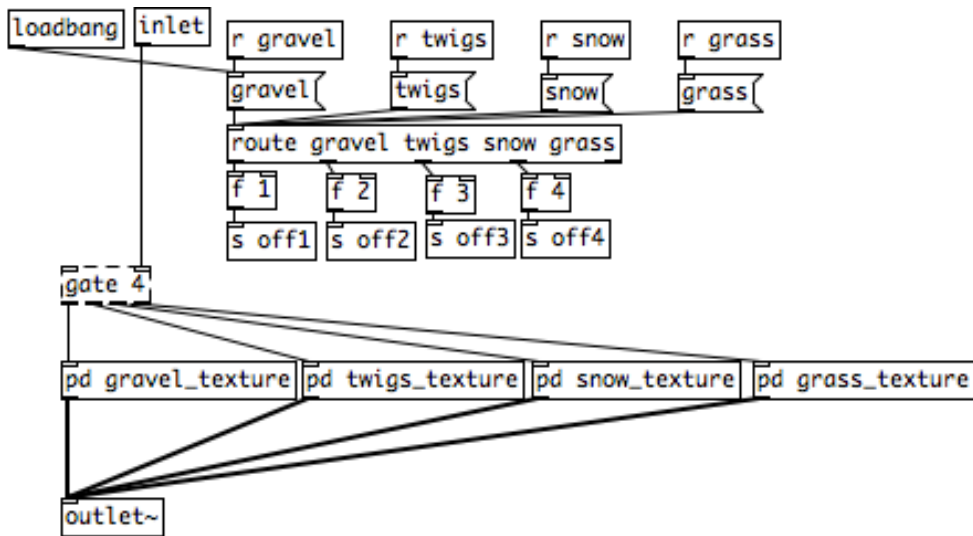


Figure 4 - The [gate] routing system. As one can see, it is far less complex than that of Figure 4.

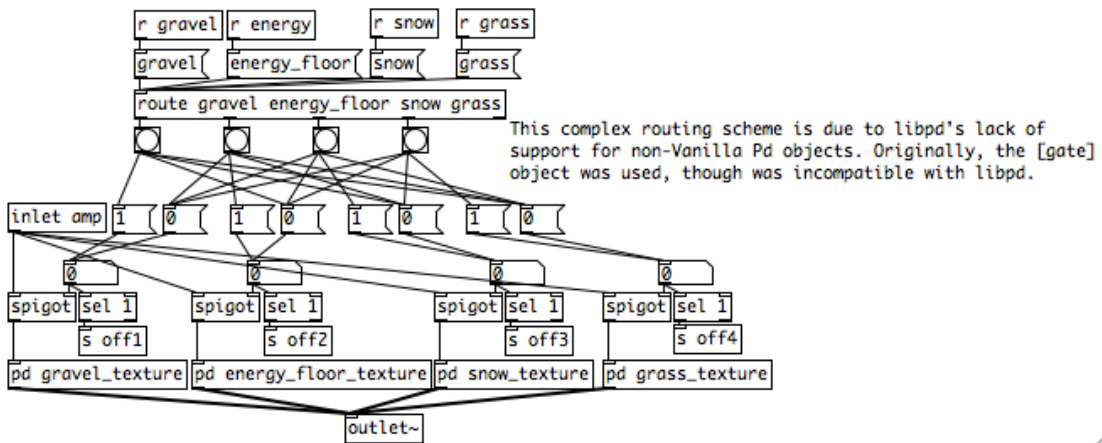


Figure 5 - The final routing system. Much more complex than the previous [gate] design.

## Evaluation

The project contains four possible surfaces for the user to walk, run or creep on: a conceptual 'energy bridge', snow, gravel and grass. The character is controlled using the basic arrow keys on a standard keyboard, while the 'Shift' (creep) and 'Ctrl' (run) modifiers are used to adjust speed variability.

Several participants took part in a feedback session, in which they were able to play the game, then answer a few short questions based on usability and experience. All found the game very intuitive and easy to use, though it was advised that the modifiers be swapped. It is presumed that the reason for this is to stick with 'traditional' gaming control layouts, where 'Shift' allows the player to run, and 'Ctrl' is used to crouch, or creep. This has been adjusted for the final version.

In terms of perception, the feedback is quite mixed. The Snow and Energy Bridge textures were both regarded as the most and least convincing sounds, from different participants. While one participant felt the Energy Bridge was 'exactly as I would have expected even though I have never walked on one', another had difficulty identifying the surface. With Snow, one participant felt it was too loud, while another felt there was a lack of low end sound from the compression of snow into ice underfoot. A third participant though preferred Snow over all sounds basing judgment purely on 'intuition'. It is interesting that the two users who felt Snow was the least successful were using objective points-of-view, while the one who preferred it was being entirely subjective, based on their own personal preference (see Appendix H). For the final version, Snow had its amplitude reduced and more emphasis placed on lower frequencies by reducing the frequency range of the [vcf~] input signal from 0-300Hz to 0-200Hz.

## Future Work

There is an incredible scope for future work and development on this project and footstep modelling in general. Pd is undoubtedly a great development environment for such a task as it is quite modular and a lot of key code is pre-written, cutting down on development time (rather than coding entirely in C++ or Java). Together with Libpd, it also offers an interesting potential for Pd as a game audio engine, too. Abstraction and environment arrangement is quite simple, with sends and busing available to transmit audio and data between independent sub-patches.

In terms of the initial footstep phase, I would certainly like to introduce GRF data to be used as a control variable to influence the intensity of a given amplitude curve. With sufficient research, this could also help to identify gait characteristics such as the sex and weight of a character. Unfortunately, it was not within the scope of this project.

It would certainly be feasible to attempt to develop the work of Perry Cook as a Pd object. The object would potentially create an amplitude curve by carrying out the following tasks: reading .txt files containing information as previously mentioned. Then, create arrays for each file for both time and amplitude, as well as creating average and standard deviation values for each point (assuming all data files had the same number of points). Finally, with each new trigger (of a footstep), new arrays are created for both time and amplitude based on the standard deviation values, and the amplitude values are output at the corresponding time values. This would hypothetically create a much more dynamic amplitude curve based on statistical data of actual footsteps. This model could then be adapted for other sound models, perhaps gunshots or animal vocalisations. The design philosophy of this amplitude curve generator is also influenced by Fournel's SPARK software, which allows sound designers to use their own original data in model construction.

An element that was in my research but that I was not able to implement (due to time restrictions) was in the work of Bresin et al. (2001): using musical parameters such as legato, staccato and ritardando to simulate walking, running and slowing-of-pace respectively. The most interesting aspect of this would be controlling the 'phase' difference of each step, which increases with speed, which is also discussed greatly by Farnell (2007b).

Processor efficiency was also disregarded for this project, due to the added research into DSP and computer science subjects such as modularity, coupling and cohesion, reusability, design patterns, algorithm classes and morphologies. These would undoubtedly be great fields of future research in an attempt to find more efficient ways of creating powerful and dynamic models with minimal cost. Though, today it is arguable that processors are so well designed that PA is not nearly as taxing as it is commonly believed.

Greater character dynamic variables to increase parameterisation would be another interesting field of research, much akin to that done by Cook (2002), but perhaps more related to a gaming environment. In a shooter genre one could incorporate the weight of the characters equipment: the heavier the pack gets, the slower and more staggered their gait. RPGs (Role Playing Games) could introduce terrain variables like sudden earthquakes that cause the player to stumble, or steep hills that cause their feet to slide backwards. Or, maybe the character is a zombie that drags its right leg behind it. Some of these parameters would draw information from the gamer's own actions, but others would require information from the game engine itself. Developing game physics for this project was beyond its scope, though is certainly a future possibility.

I feel that one key issue with surface texture modelling is that there is no 'one solution' to each surface. We walk on different surfaces in different ways, and this is not reflected well in this project, mainly due to the scope of the work involved. Analysis of a persons gait on each surface would have to be carried out. GRF and the overall amplitude curve would be different, so various arrays would be needed for each surface – this could result in high overheads in terms of CPU

and RAM, which would increase in cost with every array required, though this would have to be quantified.

Speed, too, needs to be made relative, not just in terms of the speed at which the array is read, but how the values are affected. When someone runs, their footfall is different to when they walk, as it is when they creep. Weight distribution varies, and thus different emphasis is placed on the foot strike. This is not reflected in this project. With further analysis and measurement of GRF values perhaps, it could be possible to design a model that places greater emphasis on different array values due depending on the speed modifier used.

## Conclusion

From the preparatory research it is clear that with advancing technology synthesising audio for use in game audio design is essential, or at the very least will be. Along with the current necessary knowledge requirements of programming and physics, it will take some time before Procedural Audio becomes the 'go-to' element of the sound designers tool kit. Most research remains largely academic and, at the time of writing, yet to be used at any great length in commercial gaming projects.

Despite this, PA techniques and aesthetics have already taken the eye of game developers such as Electronic Arts and SCEE. While SCEE have yet to release any titles (as of February, 2011), the work done by Nicolas Fournel is undoubtedly impressive. With next generation consoles ramping up in capability and new gaming markets emerging (mobile gaming via iPhone and iPad, online social gaming *à la* Facebook), there may be new cause to introduce more efficient ways of creating dynamic audio.

Modelling footsteps was a chosen project because it demonstrates many key areas of PA: parameterisation, interactivity, audio generation at run-time in real-time and development in a gaming environment. While this gaming environment may not be up to the same standard as the Unreal Engine, it can certainly get the job done.

Not only does this project demonstrate PA as a modelling approach, but it also argues the notion of Pure Data as an audio engine. With simple modularity and quick programming style, Pd certainly has great potential. The open-source community that keeps Pd running is a minefield of knowledge and commitment that ensures constant development and technological refinement and adaptation. This is perhaps one of the many reasons why EA adapted it for their own work.

While much work and research is necessary before models become sufficiently convincing, the groundwork is well and truly covered. From Pd to C++ and using

a multitude of data analysis techniques and modelling aesthetics, it should not be too long before we are hearing someone's convincing 'first steps' in a game somewhere.

## References

TheAudioArchive.com (2011) *Audio Bit Rate and File Size Calculators* [online], available: [http://www.theaudioarchive.com/TAA\\_Resources\\_File\\_Size.htm](http://www.theaudioarchive.com/TAA_Resources_File_Size.htm) [accessed 4 Aug 2011].

AudioKinetic (2011) *Introduction to Wwise* [online], available: <http://www.audiokinetic.com/en/products/wwise/introduction> [accessed 21 Apr 2011].

Bonneel, N., Drettakis, G., Tsingos, N., Viaud-Delmon, I. and James, D. (2008) Fast modal sounds with scalable frequency-domain synthesis. *ACM Transactions on Graphics*, 27(3).

Böttcher, N. & Serafin, S. (2009) Design and Evaluation of Physically Inspired Models of Sound Effects in Computer Games, *Proc. Of the Audio Engineering Society 35<sup>th</sup> International Conference*, London.

Bresin, R., Friberg, A. & Dahl, S. (2001) Toward a New Model For Sound Control, *Proc. COST-G6 Conf. Digital Audio Effects (DAFX-01)*. Limerick, Ireland.

Cook, P. R. (1991) *Identification of Control Parameters in an Articulatory Vocal Tract Model, with Applications to the Synthesis of Singing*, unpublished thesis (Doctorate), Stanford University.

Cook, P.R. (1997) Physically informed sonic modeling (PhISM): Synthesis of percussive sounds, *Computer Music Journal*, 21(3), p.38–49.

Cook, P. (2002) Modeling Bill's Gait: Analysis and Parametric Synthesis of Walking Sounds, *Proceedings of the AES 22nd International Conference on Virtual, Synthetic, and Entertainment Audio*.

Cook, P. R., O'Brien, J. F., and Essl, G. (2001) Synthesizing sounds from physically based motion, *SIGGRAPH 2001: Proceedings of the 28<sup>th</sup> annual conference on*

*Computer graphics and interactive techniques*, New York, New York, USA: ACM Press, p. 529-536.

Creative Digital Music (2009) *Teaching Adaptive Music with Games: Unity + Max/MSP, Meet Space Invaders!*, available:  
<http://createdigitalmusic.com/2009/04/teaching-adaptive-music-with-games-unity-maxmsp-meet-space-invaders/> [accessed 21 Apr 2011].

Farnell, A. (2007a) An introduction to procedural audio and its application in computer games, *Audio Mostly Conference*.

Farnell, A. (2007b) Marching Onwards: Procedural synthetic footsteps for video games and animation, *Pd Conference 2007*, Montreal, Canada.

Farnell, A. (2010a) *Designing Sound*, London: MIT Press.

Farnell, A. (2010b) Behaviour, Structure and Causality in Procedural Audio, M. Grimshaw, ed., *Game Sound Technology and Player Interaction: Concepts and Developments*, Information Science Reference, p. 313-339.

Fontana, F. & Bresin, R. (2003) Physics-based Sound Synthesis and Control: Crushing, Walking and Running by Crumpling Sounds, *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*. Firenze, Italy.

Fournel, N. (2010) Procedural Audio for Video Games: Are we there yet? *Game Developers Conference*, San Francisco, USA.

Fournel, N. (2011a) Procedural Audio: Challenges & Opportunities, *Game Developers Conference: Audio Boot Camp*, San Francisco, USA.

Fournel, N. (2011b) Writing High-Level Game Audio Tools: Game Audio Evolution, *AES 41<sup>st</sup> Conference on Audio for Games*. London, UK.

GameSpy (2008) *The Beat Goes On: Dynamic Music in Spore* [online], available: <http://pc.gamespy.com/pc/spore/853810p1.html> [accessed: 20 Apr 2011].

Gitorious.org (2011) *Libpd* [online], available: <http://gitorious.org/pdlib> [accessed: 4 Aug, 2011].

James, D.L., Barbič, J. and Pai, D. K. (2006) Precomputed Acoustic Transfer: Output-sensitive, accurate sound generation for geometrically complex vibration sources, *ACM Transactions on Graphics*, p.987-995.

Paul, L. J. (2010) Procedural Sound Design, *Game Sound Conference*.

Martino, R. (2000) *Synthasaurus: An Animal Vocalization Synthesizer*, unpublished thesis (M.A.), Northwestern University, Chicago.

Menzies, D. (2009) Phya and VFoley, Physically Motivated Audio for Virtual Environments, *AES 35<sup>th</sup> International Conference*. London, UK.

Misra, A., Cook, P. R. and Wang, G. (2006) A New Paradigm for Sound Design, *Proceedings of the International Conference on Digital Audio Effects*.

Nordahl, R. (2004) Self-induced footsteps sounds in virtual reality: Latency, recognition, quality and presence, *Proceedings of the Eighth Annual International Workshop Presence*, London, UK.

Raghuvanshi, N., Snyder, J., Mehra, R., Lin, M. and Govindaraju, N. (2010) Precomputed Wave Simulation for Real-Time Sound Propagation of Dynamic Sources in Complex Scenes, *SIGGRAPH 2010: The 37th International Conference and Exhibition on Computer Graphics and Interactive Techniques*, Los Angeles.

Serafin, S., Turchet, L. and Nordahl, R. (2009) Extraction of ground reaction forces for real-time synthesis of walking sounds, *Audio Mostly Conference*.

Van den Doel, K. (2003) Modal synthesis for vibrating objects, *Audio Anecdotes*, pp. 1-8.

Zheng, C. and James, D.L. (2009) Harmonic fluids, *ACM Transactions on Graphics*, 28(3).

## Appendix

### Appendix A - Interview with Nicolas Fournel Transcript

Nicolas Fournel (as of February, 2011) is a Lead Audio Programmer at SCEE (Sony Computer Entertainment Europe) as well as a keen promoter of PA.

Nicolas Fournel: So, what were your questions?

Chris Paton: I suppose the best thing to do is to tell you where I'm coming from. First off, for our thesis we have to come up with some idea to develop from now until September. I had a few ideas at first, but then stumbled upon procedural audio and started looking at that. I've always been into games, and it's always been an interest of mine. So I thought this could be quite interesting as a project – something to get into. So, I started reading up on it – one of my biggest sources was Andy Farnell's *Designing Sound* book, which has been an absolute gem to me. I read a few of his overview papers and then read some of the papers off your website...

NF: I should probably update them! (Laughs) I didn't know anybody was actually looking at it!

CP: That's where I found your information. I was looking for anything really. There were a ton of papers with general information such as modeling footsteps and was very interesting, until I started reading this [*Designing Sound*], when I realized that it was 'relatively' simple to implement in Pd and Max/MSP – I wasn't really sure how it was done [before that]. I started discovering that apart from EA's use of it, it's never really been used in the game environment. What I was hoping to talk to you about was once people have passed this stage, what happens?

NF: Procedural Audio, in itself, is not used that much in games. You have special cases where it is very well-adapted and where it is used. There are some hybrid methods like for car engines, where you have granular synthesis and you are using more or less samples but with a granular engine. There are a few very

special cases there. There are very few if even no [sic] systems at all which is dealing with a lot of different types of sounds in the game right now. Some companies have ported Pd, like EA did but that was just for the music as far as I know. Also, MaxMSP and Pd are very good to do the research. At the prototyping stage it is very interesting, too, because it's very fast to develop something with them, but they are not adapted to game development. So you have to keep in mind that when you are doing game development your audio engine teams will only allow you to use 5% of the CPU. It's not like running on a full-blown PC; you have to have something that's running quick and very fast. The main problems with Pd and Max is [sic] that they are very granular. What I mean by that is: your modules can be very, very small. You can have a module to multiply or add, and when you are at run time you will have to send the data between all those modules. If you keep sending buffers between all those modules then you are spending a lot of memory and we cannot afford that in the game engine. That is the main problem. The other issue is that, as long as you are doing research work or you have time on your hands, it is exciting because you are playing with modules. You are testing models, 'What if I add that? Oh yeah, that sounds more realistic,' and 'What if I read a book on animal vocalisations to see how it works?' and that's the work Andy [Farnell] does in his book. Starting with bottom-up design. How does it work in real life? The fire is done with this, and this and the physics behind it is that, so I will do something to emulate that... But in real life, in game development: Can you give me that fire sound in five minutes? You do not have time to do that kind of thing. Even though you may have slower periods, where you can do some research, it's pretty rare. That's one of the main issues with procedural audio right now – there is no central place or group dealing with that in the gaming industry. So where do you get your models from? As long as there is no collaboration within the game industry to share models we are pretty stuck. It is a Catch 22. People will tell you: you don't have a model for that, so why should we be using that? But, if we don't research it we will never get the models. So it's a bit of a vicious circle.

CP: Do you think there is a potential for this work?

NF: I definitely think there is a potential for this kind of stuff (laughs), or I would not be talking about it all the time! I went to some conferences as well, and some people were saying, 'Wow, that's the future of game audio!' No that's not the case, it's just one of things we will be using. There are obviously very important points in favour of Procedural Audio: it's more dynamic; it uses less memory. If it's well done and you optimize it, it doesn't use as many CPU cycles. One of the things people will tell you is: 'Oh yeah it's synthesis, so we are not playing a sound that is already recorded, so we have to calculate everything so it takes more time, more CPU cycles.' But, in some cases that's not true actually. Even when you are playing samples you have features for the pitch, for resampling... So, you still have a lot of work to do.

Procedural Audio is still very interesting. To come back to the advantages, we have games, which are becoming huge. Obviously not the social gaming part of things but the AAA titles, first person shooters and all that stuff... It's becoming really huge with a lot more things you can interact with, like physics: things can be destroyed in one way or another depending on the impact of the force, so you need something which is a bit more flexible than just playing with samples. We will get there, but for now we are really at the beginning.

CP: Would it be a case where, like EA rehashed Pd to be used in Spore, is that the kind of way people would do it? Or would there be a particular audio engine designed for it?

NF: Yeah, I think that's a possibility, that's one option. You would have to design some thing that takes care of the granularity of your parameters and that kind of stuff, that's a possibility. Obviously you would win all the knowledge of the Pd community so that's a very important thing. I have a main concern about that way: basically, most of the procedural work done right now is done... how can I put this nicely (laughs)? You will build a model in Pd and you will give that to the sound designer, but that's not his sound, and that's a major issue when working with a game team with sound designers. You can't tell them, ' Well, this is how the wind sounds in your game.' So we need to go the other way around. We don't want to build models and tell the sound designer, ' You will use that model.' We

need the sound designer to bring his sounds. It must sound like that in the game, and take their sounds and turn them into something dynamic. For that, we need new types of tools: tools that take samples and make models from them. And, that's what we're working on here.

CP: What kind of languages do you use to code in information? Is it C++?

NF: Yeah. Mostly in game development it's done in C++. That's interesting, because obviously things like audio synthesis must be fast. Again, we were talking about that 5% or maybe 3% (laughs) that they will let us use! So the interaction with the other subsystems of the game will do that in C++. For the synthesis part, it's better if it's done at the lower level, especially on the PS3 where we have the chance to have SPUs (Synergistic Processing Units), which are really fast for all that DSP stuff. So, the best thing is to write the code to use the SPUs. That can be done in assembly if you want something really fast.

CP: If someone comes up with a design in Pd, and they want to implement it into the game environment, that translation from Pd to C++ code, how would that be feasible?

NF: Pd is already C++ no? Or is it C? That's not as much a language program as much an architecture problem. But also at the same time, it's true the audio generators must be optimized as much as possible. If I was told to port Pd to PS3 the generation would be done in C++, but the audio generation would probably be written in assembly to the SPU.

CP: Why is it that audio is so second place to everything else in the game environment?

NF: Yeah, we could talk about that for hours! I'm not sure what to say!

CP: 5% seems so little...

NF: You can do a lot with 5%, even though physics may have the lion's share.

CP: So do you do sound design? Or are you purely a programmer?

NF: I have done some sound design, a long time ago, but now I pretty much work on tools and systems.

CP: Making tools?

NF: Yeah, general ideas about the tools and writing tools and run time code and implementing that in the game.

CP: Nowadays, what kind of tools would sound designers use to create the sounds they want? Is it all recording, editing and implementing, or would there be an element of procedural methods?

NF: Well you have tools that you would probably know, like Wwise or Fmod. At Sony we are using something internal, which is also used by 3<sup>rd</sup> parties called Scream. It's mainly a sound scripting tool, so you have different actions that you can do, but it's still sample based but you have ways to repeat samples or change their pitch. Things like that. That's a tool that probably comes from the 90s. We are still using that tool – many generations after of course. But to tell you how slowly game audio has developed, we are still using the same methods, especially when you see how graphics have moved.

CP: With the way things are at the moment, would it be worth making complete models or would I be better off at looking at ways of assimilating samples with procedural techniques and ways that that could be implemented?

NF: What do you mean by 'samples with procedural techniques'?

CP: Well I suppose using the best of both worlds, like with granular synthesis methods but with using samples or wavetables.

NF: Oh yeah I see what you mean. Well it depends on what your goals are at the end of it. What do you want to achieve with it?

CP: Well, I'm still in the research stage - I want to gather as much information as I can. When I started looking at procedural audio I thought, 'Wow, this looks really cool!' What interests me the most is that a lot of it is dependent on user input and that the sounds are always fresh every time you use it. For me that is the biggest part. I see the weakness in sample-based technology. Obviously there are some games where the samples are really good, but there are some games, like Fallout New Vegas, where, for example, the footsteps sound similar on every surface and you walk in the desert and it sounds like you're walking in circles. For that kind of thing, procedural audio sounds like the way to go. But, the problems would be making the models. But, there seems to be a lot of people who have done a lot of work on it: water modeling, impact sound, environment modeling for reverb. I saw a really good one using Valve's engine, and it sounded really impressive. So there's that kind of stuff, but it seems the actual cross over from the design stage from the Pd patch and all that, into the game environment is the block.

NF: Yeah, and again the main reasons for that is exactly what we were talking about: firstly, why you need an engine to play that stuff, and if you are using something similar to Pd it will take a lot of time and CPU resources. Secondly, you are starting from your modules, not from the sound, so it's hard to tell the sound design guys to do that. I think what will really put procedural audio on the map is when we start creating models from the existing samples. Until you do that, I really doubt it will start.

CP: What exactly do you mean by that?

NF: What I mean is, sound designers know exactly what sound they want for their game. They won't be able to get...

CP: So they design the sample first, and the model is built around that.

NF: Exactly. They won't be able to build that model from scratch.

CP: So they have to find the sound that they want first, then build the model. That sounds quite good.

NF: And that's not the way people are presenting procedural audio. They are presenting it the other way: 'Oh, because it's like that in physics we will build it like that.' Yeah OK, but first my game is a game, so I don't care about physics and it should not sound like that because my creative vision is more like this.

CP: I started playing with a few samples in the book, but I was trying to find, apart from the modeling, where the creative element was occurring. It just felt like: 'Here is the physics of it, there you go, that's it.'

NF: Yeah exactly, and that's my problem. That book is really good, he's done a ton of research on it and it's great. As a total nerd, and geek, I really like it. As someone who would have to work with procedural audio in games... I cannot apply that. Also, I could use some of that research to get different ideas. The work we are doing here, now, is exactly what I was talking about taking samples, analyzing them and creating the models exactly from that. Even in that book he analyses samples and see, 'Oh the frequencies are like that so the model will be like that,' but we need to do that on a much higher level and to allow the sound designers to do it.

CP: There must be a huge time element to it as well. I mean, to design a model for each sample...

NF: Exactly, that was my third point actually that we talked about earlier! (Laughs). Things are going very fast in the game industry, you don't have the time to spend one week fine tuning a model, especially when you think about the number of sound effects you can have in a game. There is no way you can do that.

Of course, to come back to what we were saying, if you had some kind of industry wide interest group or whatever, sharing the models, then you could do that.

CP: I suppose the key thing to it is financial support then as well.

NF: Yeah. I think the idea is really: if you want to get it in the game, start from the sample, to the model, not the other way around. And I'm not sure why nobody is doing that but I'm pretty sure that's the problem. Because, from talking with sound designers and being in games for a while, that's the problem. The other thing is: you will always hear that procedural audio doesn't sound good, well if it comes from their sound and you can recreate it, they won't have any problems with that. But if you start from sound modules... you have enough theory against it... 'Yeah yeah, it's just a bunch of mathematical formulas, and yeah it sounds like that, it's not great,' it could be very good in the game context, but they already have this theory that we are just messing with sine waves and that kind of stuff.

.....

NF: Ok. Let me show you something (turns to his laptop). You didn't see one of the talks right?

CP: I don't think so. Maybe!

NF: This is the stuff I'm working on. This is Procedural Audio. This is a bit of what we were talking about so we can go through that again, and I can give you that version of that talk so you can get those points. Basically what I was talking about in that presentation was what are the advantages for procedural audio. So good candidates are repetitive sounds, some which have a very large memory footprint like wind (if you want to do wind in a game you must have several large streams of large duration and overlap them together and it takes up a lot of memory and space on the disc), sounds that need more control, like car engines where you have a lot of settings where you want to change the sounds dynamically, sounds that are triggered and generated from movement like the

new Move controller, game physics, or you have too many of them and you want something automatically generate them, like in RPGs where they are so vast, so many things can happen you can't plan everything. So you can use scripting, so you still just randomize assets and change the pitch and the volume but it's not truly dynamic and still uses a lot of memory. Or you can do that with patching, the sound designer has to be aware of physics of everything and it takes a lot of time and CPU resources... .. So doing things bottom-up, like the fire chapter in that book, is something which in practice is not doable. If we had an interest group and some people were doing it and the industry could pick these models yes, but by yourself and a team, no, you don't have time to do that, you don't have the resources. So this one for example is a test program that I wrote which is doing procedural audio so you can generate liquid sounds, wind sounds, footsteps and impact sounds. But even with that I'm going to the sound designer and saying, 'hey, that how your sound must sound!' So that doesn't work. So the idea is to have a tool that will analyse sounds and create models from that. You have to start thinking in asset models terms not in assets – samples. So for example that tool, we analyse existing samples and we create the models from them. You can create the whole model from them or you can just grab elements of it that are interesting. For example, lets say you want to create debris sounds in a game. It won't sound the same if it falls that way, or that way, or if it's... If you record one debris sound it will sound the same but you want to create the same kind of sound but different each time, so you want different distribution, something more dynamic. So what I do is, with that tool, we take the debris sound and we have transient detection, so that's the analysis. Then you get the sound object inside which is this list of transients. From that we have a modeling phase and that's the distribution finer, because now we have offsets of all the events in the sound, and we calculate the distribution – is it Gaussian etc. At the end of that we have a distribution model so now we have something that can generate events which will be different each time if we want to, but which will have the same feeling of the original sound. So that's the start of our debris sound. Already we can generate many debris sounds and they sound quite cool and they are based on a real sample.

Then you want to generate the audio itself, so for example in that case we have a model resonator and we analyse... we do a spectral analysis... so in that case we usually take one part of that debris sound that we had at the beginning, we isolate one part, or another sound (another surface or something) and we analyse it, and from the spectral analysis we track the modes and we create a resonant model of that. Then at this point you can pretty much generate metal sounds, glass sounds, things like that, and you can change the parameters – the damping of the modes, the variation in pitch. But it is still based on the sample, so it is totally dynamic as well. So, at that point you can already trigger whichever distribution you want but which sounds like what you want, with the exact same sound or something different.

So, when you bring this to the sound designer, it's ok because that is their sound, and it is totally dynamic. Of course, you can analyse the overall amplitude, you can use an envelope follower and analyse the overall amplitude of your debris sound and apply that to your sound here. And that is an example of how you can do things. And the idea behind that is because you are using audio analysis to create complex models you don't need as many modules as you would with Pd. If you had to recreate the distribution of something you have a lot of operations and all that data has to flow from one module to one module at runtime. But generating events based on distribution is very fast and you have it in one module here. See, you have your event generator here, and you're done. And you have your model resonator, this is basically what I just showed you. With two modules you are creating any types of debris and those are sounds we have analysed. I just drag a sample here, this resonating sample, then I drag it here it creates the model resonator...

CP: So are these predesigned models?

NF: These are predesigned models that can use any sample as a basis.

CP: So these contain all the physics?

NF: Exactly. Or other things. This is a very old version, and probably buggy. Now we have stuff for some animal vocalisations, impact models, footsteps, wind...

CP: Is SPARK a tool I'd be able to get?

NF: No. It's totally internal.

NF: There are default settings for the analysis. So you could select one part of it... When you drag and drop this sound here, it already does a model for it. So I have my original sound, drag it here, and now I have the model, and now I can play with it. So now I can change the damping, for example, I will have something that is more resonant, if I change the modes I will have different frequencies. So now I can create different sounds all with that model and it's still based on what the sound designer wanted in his game so now they want to use that.

CP: So how does SPARK get used in the game engine, or is it part of the game engine? How would you put this into the game?

NF: So at this stage, this is the patch, and this is very small because you just have a bunch of modes, their frequencies, their amplitudes their damping value etc. And I export that to the game, so there is a patch file and then the audio engine I wrote a super patch coming from SPARK and so I just put it in there and I can trigger that. And that's the same for other things. Let's say I want to create some animal model but I want to get the pitch, because I like that cat singing... Let's say I want to get the pitch envelope of that cat – there you go! So what happened here is I have a curve generator which can recreate any kind of curves based on that model of the pitch of the cat. If you click on that you will see the interesting part. As you can see the blue curve is the real thing, it is an amplitude envelope I calculated. But instead of using that directly, which is on my PowerPoint here. I have a curve-fitting algorithm and I get the model from that. So now I can recreate any kind of similar curve based on the one here.

CP: That would take you so long to make in Pd!

NF: Exactly! So now I can have a variation for that curve, a time scale, so now it will use this fourth-degree polynomial to generate the curve, and every time it can be a bit different. And obviously what I'm storing isn't a full sample. I'm storing a few coefficients for my polynomial, a few bytes! And something for the event distribution stuff... So you analyse that and it creates the corresponding distribution. Of course, you don't have to analyse everything, you can create your own distributions and trigger...

CP: This is a tool you developed yourself?

NF: This is a tool I developed last year. This is a purely internal tool. We are just using it in London.

CP: Not in any other studios?

NF: Well not yet. It's not at a point I feel comfortable I could maintain it for a lot of teams.

So now for example excitations... If you get a scratching things, we do some LPC analysis and then we get the excitations you know, and then you can apply that to your model resonator etc. So even with 2 or 3 models you can a lot of things already.

CP: Do you know of many people in the industry that are doing what you are doing?

NF: Well I've been doing quite a few talks this year to try to encourage people to work this way, so hopefully that worked!

CP: But are any of your colleagues as interested as you are?

NF: No. I'm pretty much the only one annoying everyone with that! But after that they were quite interested so hopefully that's a start.

## Appendix B – Emails from Andy Farnell

Email Excerpt A:

'What I did in 2005/6 was actually something of a first, before getting stuck into the book, to give me confidence that it was worth writing in the face of so much negativity from the game industry... and though few people know of it outside the industry will share it with you so you can say something more interesting.

<http://obiwannabe.co.uk/html/showreel/synthetic-reel.html>

Having already realised any sound could be synthesised in a game my main concern was efficiency. The project began almost in jest, and partly as consequence of poverty (no sob story, its just the way it was) with an attempt to make all the sounds for a game in CPU

"a proof of concept of all the sounds for a viable FPS video game running on a 500MHz CPU" \*\*, because I was really into what they called the "demo scene" back then.

The idea was, if I could do it on a 500MHz machine then that would be about half the available on a 1GHz which were widespread standards at the time.

Ended up presenting a paper a couple of years later at Audio Mostly, using this as a demo.'

E-mail Excerpt B:

'In the literature of Zheng and James, Stefan Bilbao, Dylan Menzies and others into modal, finite element or waveguide based physical methods you will see the term "parametric". This means that salient spectral or time features are extracted for resynthesis in the absence of an underlying causal model.'

# Appendix C – Work of Fabio Morreale

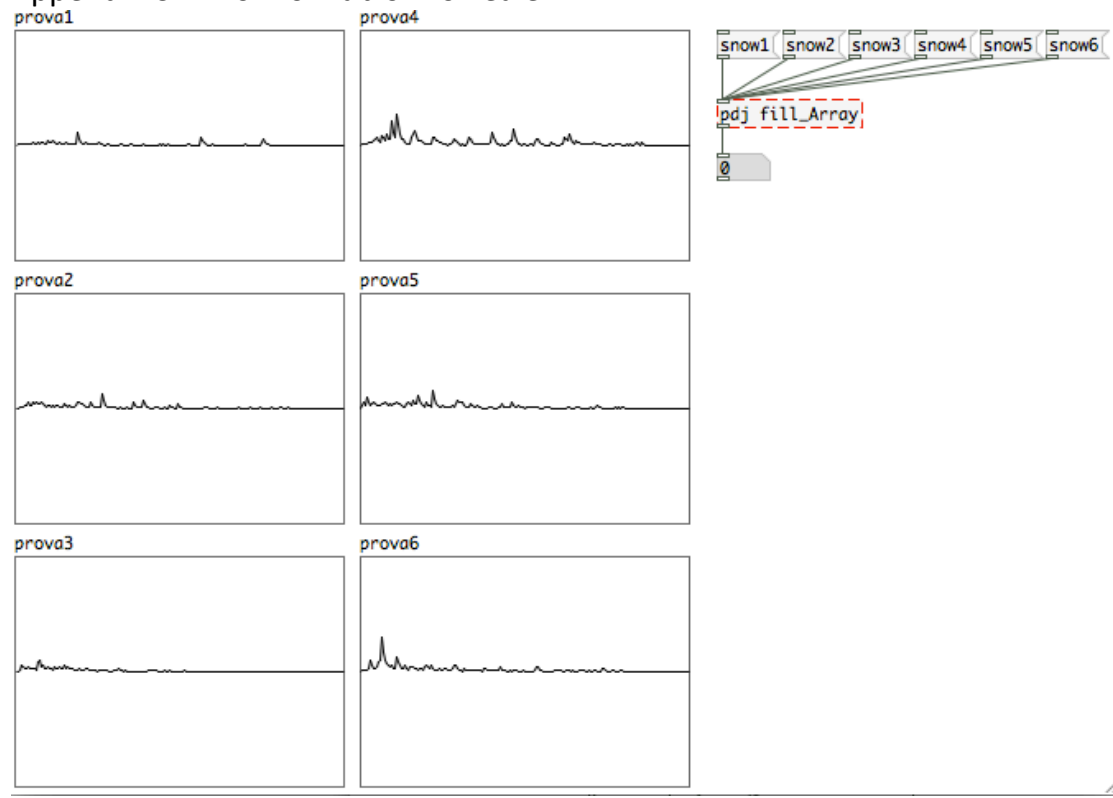


Figure 6 - Arrays loaded from .txt data in Morreale's model.

## Appendix D – Email from Federico Montana

'Not so easy to retrieve that stuff now.

We have more fresh simulations: you may contact Fabio Morreale, here cc'ed, who has developed an interesting hybrid synthesis model: if he agrees, and likes it, you may for example work in adding interactivity to his current model running under puredata.'

## Appendix E - Email Interview with Dylan Menzies

Chris Paton: You've worked on interactive and parametric audio design for some time now, with a clear orientation for gaming (Phya & VFoley). Do you feel that there is enough development and existing research to encourage developers to look at more alternative synthesis methods when approaching their game audio pipeline? In fact, do you feel that there is much in the way of academic/industrial collaboration today?

Dylan Menzies: There is some, however there is a lot of work needed to create good quality tools that would work successfully in the fast paced game industry. Industry and academia are coming [sic] from very different directions. Academics mainly want to generate ideas and turn them into papers. Industry mainly wants to make money, it doesn't have too much time for development in unproved areas.

CP: A large role of the audio programmer is to design tools to simplify the process between audio design and implementation. In my research I have discovered tools designed take a certain responsibility of model creation away from the sound designer. Do you think over simplistic tools limit creativity? Does procedural audio have the potential to increase creative scope somehow?

DM: Tools can be a limit, but on the other hand programmers are not always going to be good sound designers. In any case the idea of good tools should be to avoid the mental overhead of programming and get straight to the sound design business. If really needed a team can drop back into programming for a special case, and this is what happens in practice in game teams. The main thing procedural audio offers in increasing creative scope is more interesting audio behaviour, which can include more natural interactive sound. This involves more specialist sound design, but it also potentially drastically reduces the amount of work spent organizing large libraries of samples.

CP: Do you think that companies budget enough for retraining as technology

advances?

DM: I think once good enough tools are there, available at the right price, the transition could be quite fast.

CP: In your experience, is there much emphasis on creative audio synthesis with a gaming orientation in many Universities around the UK? If there is, is it generally rather concentrated (perhaps only in London) or is it a more widespread development?

DM: Traditional game audio techniques are taught widely. Research in new techniques is also fairly widespread though obviously not as much.

CP: There is often an argument that Procedural Audio is far too costly in terms of CPU usage to be viable. While incorporating the limitations imposed upon audio designers (roughly 5% of a PS3 cell processor, for example), do you feel that this argument has any weight or relevance to the current development of processor speed?

DM: This is a real red herring and has been for some time. Clearly you can use as much cpu as you like on audio, but very real advantages of procedural audio can be seen with minimal amounts of cpu. There is an attitude problem towards audio in game teams which I think is gradually shifting. It used to be what ever is left we will give to the audio guys. Now there is more planning and its about cost/benefit.

CP: What do you think are the possibilities of module-based programming solutions like Pure Data and Max/MSP becoming viable audio engine solutions? Or at least, model design solutions for prototyping, in the context of game development?

DM: These are great for experimenting, but they have some drawbacks for fully fledged procedural audio implementation. The first being they are not dynamic

enough- you can't create and destroy instances easily and efficiently. Another is creating the framework for overall object and signal management really needs a full spec general programming language, and since we want efficiency this means C/C++. Finally there are reliability and efficiency issues in pd/max.

CP: What other tools/software solutions/languages would you use for model design, and how efficient/creative have you found these approaches? Is there a balance between efficiency and creativity?

DM: It would be nice to have something lightweight and efficient like pd/max developed with a graphical design element, but also immediate access to low level coding, and an interface to incorporate objects in game code. This way prototype code can evolve directly into final code. Failing this its really down to plain old C/C++. My experience has been a lightweight and efficient C++ library structure makes it much easier to develop useful code. But like anything you have to lay down boundaries when you create a framework. You try and make it so it can be extended later in a natural way. Wrestling with overly cumbersome and pedantic api is really death in a complex area like procedural audio.

In the experimenting phase there are other things that can be useful like matlab/octave and general audio tools.

## Appendix F - Code Examples

```
# we go into an infinite loop selecting alternate buffers and queuing them up
# to be played each time we run short of a buffer
selector = 0
leave = False
while not leave:
    # we have run out of things to play, so queue up another buffer of data from Pd
    if not ch.get_queue():
        # make sure we fill the whole buffer
        for x in range(BUFFER_SIZE):
            # let's grab a new block from Pd each time we're out of BLOCKSIZE data
            if x % BLOCKSIZE == 0:
                outbuf = m.process(inbuf)
            # de-interlace the data coming from libpd
            samples[selector][x][0] = outbuf[(x % BLOCKSIZE) * 2]
            samples[selector][x][1] = outbuf[(x % BLOCKSIZE) * 2 + 1]
        # queue up the buffer we just filled to be played by pygame
        ch.queue(sounds[selector])
        # next time we'll do the other buffer
        selector = int(not selector)
```

Figure 7 - The while-loop buffering Pd audio from 'pygame\_fun\_test.py'.

```
amp_f = open("gravel(single2).txt")
fileObj = open("gravel(single2)split.txt", "w")

for line in amp_f:
    (time, amp) = line.split()
    fileObj.write(amp + ";" + '\n')

fileObj.close()
|
```

Figure 8 - Text File Editing Program.

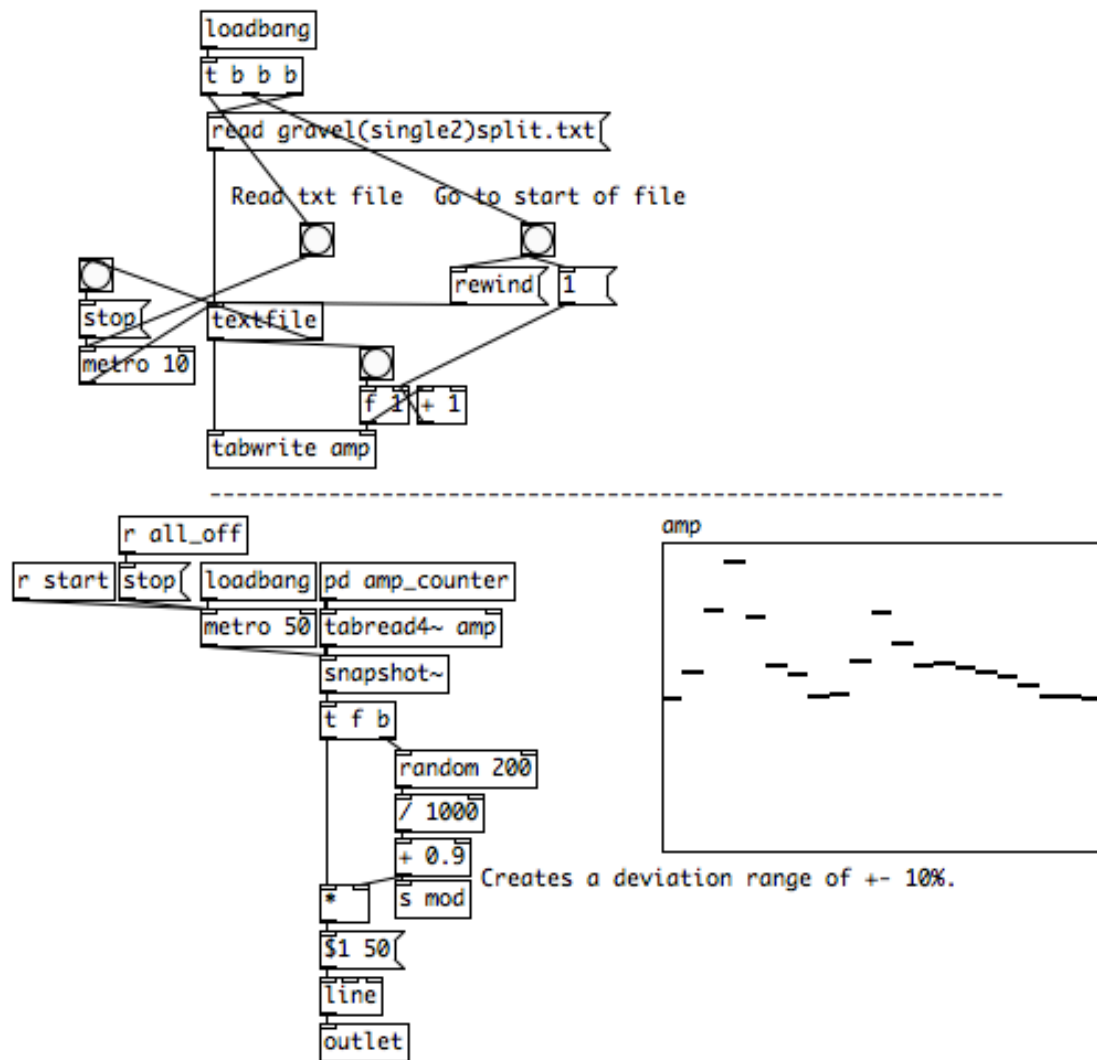


Figure 9 - Amplitude Curve Generator.

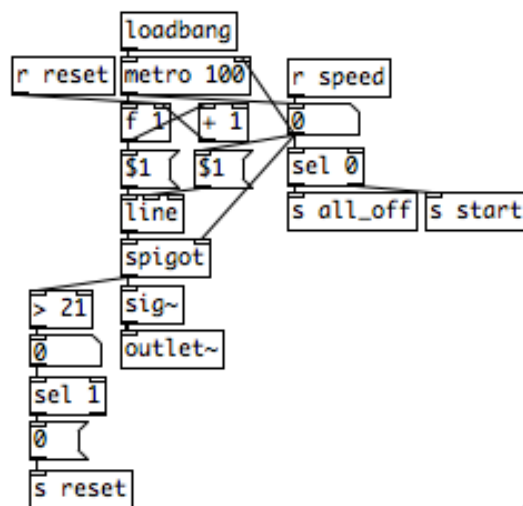


Figure 10 - The [pd amp\_counter] object.

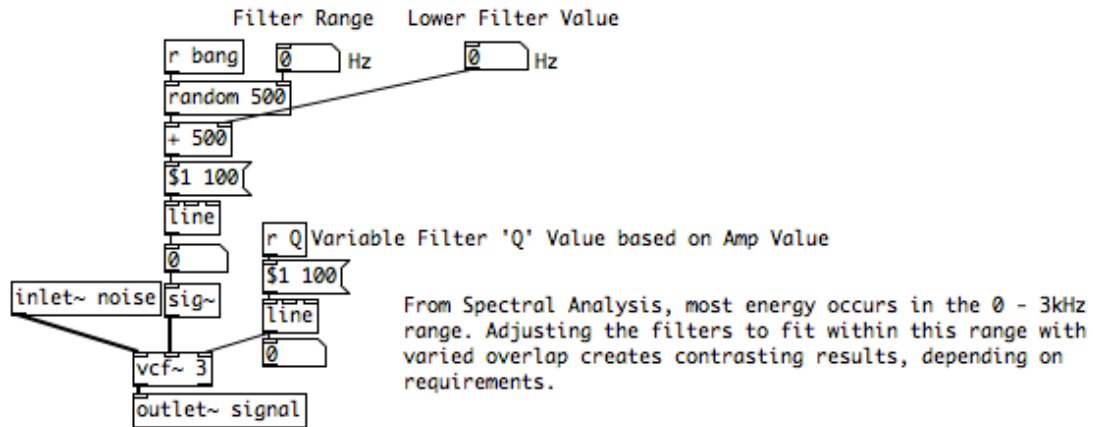


Figure 11 - Filter Design.

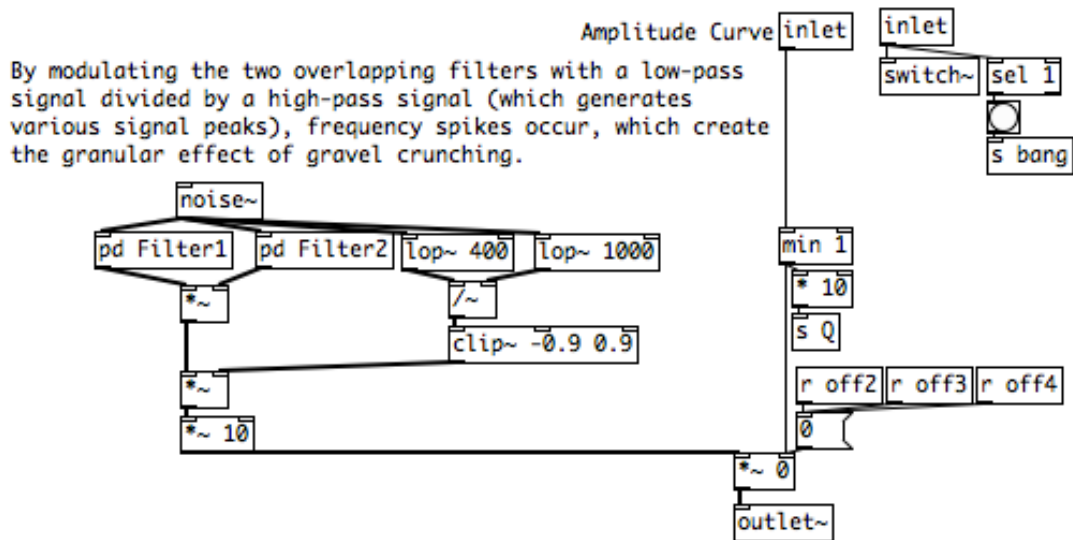


Figure 12 - Signal design for gravel sound. See Figure 7 for [pd Fliter1] design.

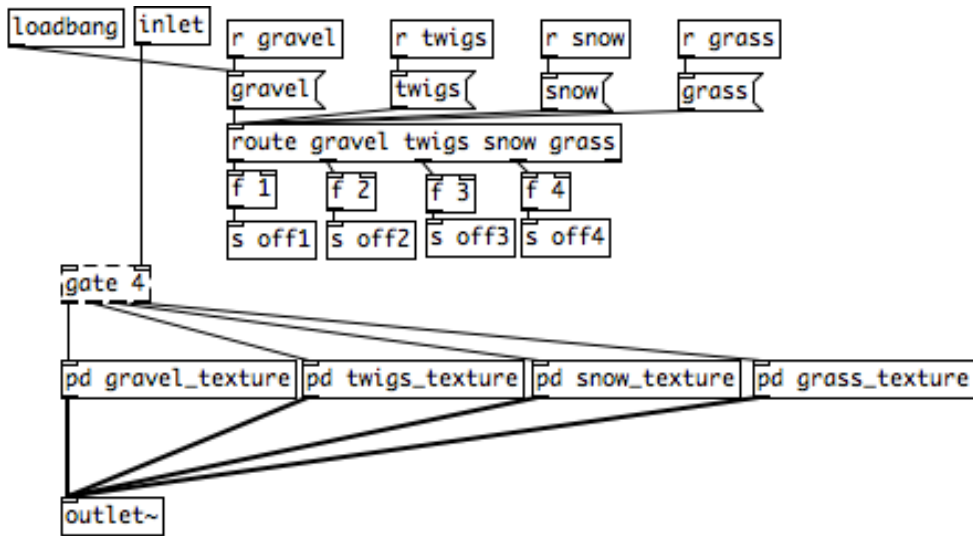


Figure 13 - The first routing scheme used for [pd signal\_generator]. Notice how the gate object has a dotted border in the vanilla version of Pd. This represents a non-functioning object.

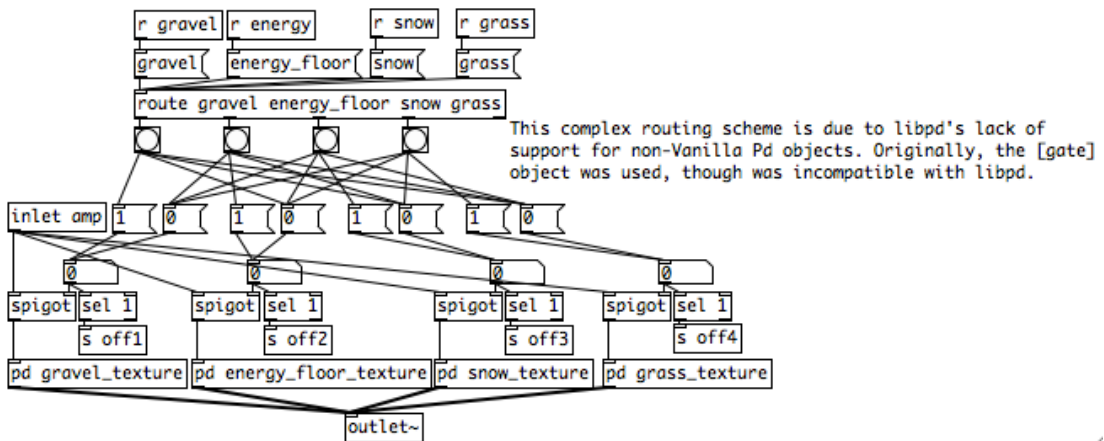


Figure 14 - The final routing scheme for [pd signal\_generator].

## Appendix G – Interview with Leonard J. Paul

Chris Paton: You have recently been working on *Retro City Rampage*, which resorts to 'retro' forms of audio generation (square waves, sine tones from the arcade era). Do you feel games like this promote interest (amongst developers) in game audio, or is this largely a stylistic focus?

Leonard J. Paul: I believe revisiting the 'retro' style of game audio does bring a focus to game audio as it is a style of audio that is unique to games. However, using a retro style in sound production is often quite different than the original sounds of games in the classic gaming era as the means of producing the same tones has become much easier. When the time for the artist to iterate and actualize their artistic intent becomes easier, I would argue that the results are often more engaging than their predecessors as the artist isn't fighting their materials as much and this is apparent to the listener. So, it does promote game audio as well as being a stylistic focus simply in the means of its production. If the stylistic result does not fit the aesthetics of the game then the overall artistic integrity of the game is lessened.

CP: A large role of the audio programmer is to design tools to simplify the process between audio design and implementation. In my research I have discovered tools designed take a certain responsibility of model creation away from the sound designer. Do you think over simplistic tools limit creativity? Does procedural audio have the potential to increase creative scope somehow?

LJP: Usually the issue in the industry is time, so any tools that can make most of the process faster for the sound designer is likely going to help the overall sound of the game. The problem is that often one tool or middleware is meant to solve all the problems of the sound artist. Hopefully the audio programmer is able to help the designer by providing solutions that help the remaining challenges. Procedural audio is expensive to develop for the sound designer as it requires more time to tune the system as well as the audio programmer who needs to create and tune the system. The issue with the audio designer is that

they are often not technical enough to realize the level of control over the system to come up with convincing results with most of the toolsets currently available. As systems such as Audiokinetic's SoundSeed become more popular hopefully the use of procedural audio techniques with convincing synthesis of sound will become as commonplace as synthetic audio effects such as reverb.

CP: As an educator, do you feel that there is enough collaboration between game developers and academics? Why?

LJP: I believe that there will always be a disconnect between academia and industry as academia has more of an ability to take a long view of problem solving whereas industry is typically a short view on solving practical problems. Academia has an advantage of being able to work on problems that are hypothetical and have little direct application in industry. For games, a solution is to actively bring in academia by having research departments but often game companies are publicly held and their emphasis on quarter to quarter financial gains is somewhat antithetical to a research department that requires money and can often produce no direct benefits in a given quarter. Hopefully with companies that are privately held with a longer view of producing interactive entertainment, such as Valve, might have more of a capability of having a strong research department.

CP: In your experience, is there much emphasis on creative audio synthesis with a gaming orientation in many Universities around Canada and the US? If there is, is it generally rather concentrated (perhaps only in Vancouver) or is it a more widespread development?

LJP: Universities and schools that are teaching game audio are changing rapidly so it is difficult for me to say if training people synthesis techniques directly related to game audio is on the rise or not. From what I have heard, there is an issue currently with schools having a difficulty of finding teachers that can teach the topic of game audio effectively as they often are from either academia or industry and need to be good teachers to convey the knowledge to a level that is

useful for industry.

CP: Is there much impetus for the gaming industry to develop audio approaches toward synthesised audio?

LJP: I believe that most synthesized audio still sounds very synthetic and doesn't correspond well to typical middleware solutions that supply a first person shooter framework that often requires more realistic sounds. In my own practice and research I believe that sometimes using a combinations of samples with a process such as granulation can produce good results. I think that as the sphere of games continues to mature that the use of synthesis will also expand. Synthesis is just another tool for realizing the overall artistic result of the game audio experience.

CP: Do you think that companies budget enough for retraining as technology advances?

LJP: In my few years of working as a salaried employee in game audio I was never offered any retraining opportunities. However, in recent years I have primarily been a contractor and have heard that companies do allow for retraining options for their employees. Technology has always been advancing, so it is basically an issue if the company is forward looking or not.

CP: What do you think are the possibilities of module-based programming solutions like Pure Data and Max/MSP becoming viable audio engine solutions? Or at least, model design solutions for prototyping, in the context of game development?

LJP: These systems have been used in game audio commercially since basically the time that Max was started. I believe that the main problem with game audio production is often finding the correct tool to solve the problem. Visual coding environments allow for a granularity of control but require more time to produce those systems. High level middleware allows an ease of control over

common tasks but cannot solve many specific tasks for the audio designer which must now be solved by the game coder. I have found that prototyping in these environments is helpful as I am able to refine my audio designs in a logical manner but not spend the time at the fine granularity of coding the structures in C++. If one is able to solve game audio requirements via C++, visual coding environments (or scripting) and with a middleware game audio tool such as FMOD then one can approach each task at a certain level of granularity appropriate to the task and hopefully maximize one's overall output.

CP: What other tools/software solutions/languages would you use for model design, and how efficient/creative have you found these approaches? Is there a balance between efficiency and creativity?

LJP: This question is partially answered above. I have used Lua, FMOD, Wwise, UDK, Unity and several in-house audio tools when doing audio design for games. I have found that using a combination of C++ (audio coder), Lua (or a system like Pure Data) and FMOD (or Wwise) to be a good balance. At the C++ level anything is possible but is often a large time investment. With each task various portions can be broken down to each implementation level. This model can be seen as similar to instrument design: the design of commonly changed parameters at the FMOD level, commonly changed logic structures at the scripting level and the hard-coded logic at the C++ level.

Overall, I believe that game audio is more of an art than a science and I am happy to have chosen the challenge of an art form where my tools are changing as the art of interactive entertainment continues to mature.

## Appendix H – User Feedback Forms

Participant Name: David Power

Date: 10 – August - 2011

By naming this document you are consenting to the use of your given information (below) in a feedback section for the thesis project 'Modelling Footsteps: Procedural Audio in Games' only.

Please answer all questions honestly and truthfully. If you have any doubts, queries or questions, please e-mail for consultation. Please save your version of this document with your own name as the title.

1. Did you find the program easy to operate? If no, why?

Yes the system was easy to use.

2. Which sound did you find most believable (energy bridge, grass, gravel or snow)? Why?

I thought the grass was the most believable. It sounded the most like how I expect grass to sound like when its walked on.

3. Which did you find the least successful? Why?

I didn't feel the Snow sounded like snow, I would have expected their to be more low end in the sound. The sound of the snow being compacted. The kinetic light walk platform sounded exactly as I would have expected even though I have never walked on one.

4. Do you play computer games of any kind on a regular basis (at least once a week)?

Yes, street fighter.

5. If yes, do you feel, or have you ever noticed repetition or a lack of realism or dynamics in the audio design of the games you play?

In street fighter when a character falls and hits the ground it always results in the same hollow thud. It would be more realistic / enjoyable if the character environment was reflected in sound.

6. What genres would you mostly play?

Fighting games, RPGs.

7. If you have any other points or observations, please note them here:

Is there a difference between what we expect something to sound like and what it actually sounds like in a video game context? In film they don't use realistic sound effects e.g. Punches don't sound as 'huge' in real life.

Participant Name: Stephen O'Connell

Date: 10<sup>th</sup> August, 2011

By naming this document you are consenting to the use of your given information (below) in a feedback section for the thesis project 'Modelling Footsteps: Procedural Audio in Games' only.

Please answer all questions honestly and truthfully. If you have any doubts, queries or questions, please e-mail for consultation. Please save your version of this document with your own name as the title.

1. Did you find the program easy to operate? If no, why? Yes. Very simple although you should use shift for making it faster
2. Which sound did you find most believable (energy bridge, grass, gravel or snow)? Why? Grass definitely. Something cool about the sound
3. Which did you find the least successful? Why? Snow. It's too loud
4. Do you play computer games of any kind on a regular basis (at least once a week)? yes
5. If yes, do you feel, or have you ever noticed repetition or a lack of realism or dynamics in the audio design of the games you play? Never thought about it
6. What genres would you mostly play? Shoot em ups, 1<sup>st</sup> person
7. If you have any other points or observations, please note them here:

Thank you for your participation.

Participant Name: Kevin Quigley

Date: 11/08/2011

By naming this document you are consenting to the use of your given information (below) in a feedback section for the thesis project 'Modelling Footsteps: Procedural Audio in Games' only.

Please answer all questions honestly and truthfully. If you have any doubts, queries or questions, please e-mail for consultation. Please save your version of this document with your own name as the title.

1. Did you find the program easy to operate? If no, why?

Yes, very simple.

2. Which sound did you find most believable (energy bridge, grass, gravel or snow)? Why?

Snow. I'm not sure I could explain, it was an intuitive answer.

3. Which did you find the least successful? Why?

The energy bridge because I couldn't identify what the surface was to have an expectation of what sound it should make.

4. Do you play computer games of any kind on a regular basis (at least once a week)? Yes.

5. If yes, do you feel, or have you ever noticed repetition or a lack of realism or dynamics in the audio design of the games you play?

I have not played games based in 3D environments in a long time so could not comment on current trends. I don't notice in turn based, point and click or sports sims but have often noticed in first person shooters and games like Grand Theft Auto.

6. What genres would you mostly play?

Turn based/point and click, sports sim.

7. If you have any other points or observations, please note them here:

Thank you for your participation.