Procedural 3D Audio for AR Applications

Angeliki Skandalou
John Jeremy Ireland

Supervisor:
Michael Rose

Kongens Lyngby 2017

Abstract

This is the first paragraph

THis is the second

THird paragraph of abstract

Four paragraphs is enough I guess

Acknowledgements

We would like to express our gratitude and appreciation to our supervisor for his support and guidance throughout this thesis work. Several discussion sessions and advice helped us take the most out of this project and make this study possible.

We would like to express special thanks also to the rest of our classmates who did their thesis at the same time under the same supervisor and offered us their advice. Furthermore, we would like to thank **To do** name of the recording guy (1) for providing us with the acoustics room to perform recordings. And last but not least to our family and friends whose support throughout this thesis was invaluable.

List of Figures

List of Tables

Contents

Abbreviations

ADSR Attack, Decay, Sustain, Release (sound envelope).

AI Artificial Intelligence.

AR Augmented Reality.

BPF Band-Pass Filter.

BW Bandwidth.

CPU Central Processing Unit.

DAC Digital-to-Analog Converter.

DLL Dynamic-Link Library.

DSP Digital Signal Processing.

DTU Denmark's Technical University.

FBX Filmbox.

FEM Finite Element Method.

FFT Fast Fourier Transform.

FPS Frames Per Second.

GUI Graphical User Interface.

HRTF Head-Related Transfer Function.

MUSHRA MUltiple Stimuli with Hidden Reference and Anchor.

OSC Open Sound Control.

Pd Pure Data.

Q Q factor.

UI User Interface.

VBAP Vector Base Amplitude Panning.

VR Virtual Reality.

Nomenclature

A Matrix of oscillating amplitudes on every point of an object.

CFM Collision force magnitude.

D Parameter corresponding to type of material.

K Kinetic energy.

M Matrix of modal data.

Q Parameter describing the damping of an oscillator.

ω Angular velocity.

d Decay rate.

f Frequency.

Chapter 1

Introduction

With the increasing popularity of virtual environments such as video games, simulators, virtual reality (VR) and augmented reality (AR) applications, it has become crucial to offer the user a rich and compelling experience. Today's physics engines are capable of providing realistic graphics and animations which are a source of audio events (e.g. a toolbox falling on the ground, a marble rolling on a table, ...). High quality audio which is consistent and in synchrony with the virtual world is necessary to convey a sense of presence [Larsson et al., 2002].

Sounds are strongly related to our everyday life and the ways we understand things. Through our experience, we can visualize an event by only hearing the sound it produces (e.g. a car approaching). The vibration caused by the collision of two objects produces sound that depends on the collision force, the duration of the interaction and its changes over time, but also on the size, shape, material and texture of the two objects. All these attributes form a unique sound and the sound waves produced from the interaction give the information to the listener [Gaver, 1993b].

The aim of a sound designer that produces sounds for a virtual environment is to make it difficult to distinguish them from the real ones. At first, one could suggest the use of prerecorded sounds for this matter. This method is the most used in the video game industry nowadays and offers good quality contact sounds with little computation (amplitude, pitch and filter envelopes). Despite this, it is necessary to record and store the audio clips which generates high memory usage and loading times. Additionally and most importantly, due to the sound variations that an object presents along its surface and the significant amount of interactions that can excite the object, prerecordings do not appear to be an optimum technique to render sounds for real-time applications where interactions are not known in advance[Verron, 2010, Van Den Doel et al., 2001].

Procedural audio, on the other hand, generates highly dynamic sounds that can be modified on the go according to unpredictabe events [Farnell, 2010]. Additionally, the use of sound synthesis solves the problem of having to store various prerecorded sound files. More specifically, modal synthesis uses physical models to model the behaviour of a vibrating object which can be decomposed in a set of resonant modes [Bilbao, 2009]. Through the manipulation of different parameters it is possible to change the sound based on the object's characteristics and interactions. For example, scraping the side of an object sounds differently than rubbing its middle, the same as striking an objet harder makes the resulting sound louder and affects the bandwidth of vibration. Within virtual environments, these interactions are interpreted by the physics engine which drives the synthesizer to provide audiovisual coherence to the user. The flexibility that parametric modeling offers over prerecorded sounds makes it an appealing solution for run-time applications that require realistic audio [Cook, 2002].

This thesis deals with real-time synthetic impact and continuous contact (rolling and scratching) sounds that are produced automatically from physical interactions of everyday objects in a 3D virtual world. Recordings of struck objects in different locations have been performed in the analysis stage to compute their corresponding modal model. Recent papers [Lloyd et al., 2011, Bonneel et al., 2008] make use of modal synthesis to simulate high quality audio events in game engines but fail to make the controls of the synthesizer accessible to game developers. The aim of this work is to bridge the gap between advanced modal sound approaches and commercial game engines by designing a tool within Unity® that allows to control a synthesis model through high level parameters (object's size, roughness, material). The idea is to easily associate an audio texture to an object the same way a colour or material are rendered on a mesh surface. In [Pruvost et al., 2015] the authors present a framework for interactive and real-time synthesis of solid sounds driven by a game engine that can be controlled during game play. A similar approach is developed in this paper with the addition of using two different techniques for impact sounds and comparing their outcome through user tests.

The first part of this paper describes previous work that conforms the state-of-the-art for physics-based synthesized sounds in virtual environments as well as principles of contact sound production and synthesis techniques. The next section gives an overview of the tool and goes through the different instruments used for its development. In the *Implementation* section the procedure used to record the struck objects is outlined as well as the development of the different contact sounds, user interface and the scripting. Then, the user tests used to evaluate the synthetic sounds are discussed. Before the conclusion there will be a section in which the outcome of the work is analyzed and future work is listed.

To do write stuff about 3d sound?? (2)

Chapter 2

Theoretical Background

2.1 State-Of-The-Art

Researchers have been working on automatically generated synthesized sounds based on physical interactions for quite some time now. Different publications that have contributed to the creation of procedural sounds in 3D virtual environments and that have inspired this thesis are discussed in this section.

Sound spatialization is a crucial component for building 3D virtual scenes and the listener's immersion. Different techniques such as Ambisonics, VBAP and Wave Field Synthesis have been developed to simulate 3D sounds on loudspeakers. Binaural techniques use HRTF functions to simulate the position of a sound source with respect to the listener through the use of headphones. In this thesis spatialization and sound synthesis are considered two separate processes. Here all sounding objects are considered point sources that are spatialized with Unity's Audio Spatializer SDK [Unity Scripting Reference,] which produces the desired effect. This is why sound propagation is not studied in depth.

Regarding studies related to sound simulation frameworks, in [Gaver, 1993a, Gaver, 1993b] Gaver proposes the analysis and synthesis of contact sounds based on what he calls "every listening" which consists in focusing on the perceptual features of a sound event. Another pioneering work [Takala and Hahn, 1992] presents a methodology to produce synchronized sounds with animations. [Van den Doel and Pai, 1998] decribes a framework that uses modal analysis to generate sounds based on the object's material, shape and impact location. This analytical model however can not handle objects with arbitrary or more complex shapes. A Finite Element Method (FEM) in [Director-O'Brien, 2001] takes advantage of existing simulation techniques models the surface vibration of virtual objects. Despite being more accurate, this method is expensive and non-interactive. In his book [Cook, 2002], Cook covers extensively physically based sound synthesis concepts and defines a parametric model as one that can be manipulated to change the interaction, object and sound.

Different techniques have been used to extract modal parameters necessary for modal synthesis. [Pai et al., 2001] scans the response of real objects to model the interaction behaviour of 3D objects. Despite offering the possibility to synthesize sounds for arbitrary shaped objects, the hardware needed is a clear limitation. A FEM approach is used in [O'Brien et al., 2002] for modal analysis and [Raghuvanshi and Lin, 2006] uses spring-mass systems to model each object's deformation and vibration.[Van Den Doel et al., 2001] and [Lloyd et al., 2011] make use of modal models derived from recordings of struck objects which is similar to the approach described in this thesis. This model is used to create impact, rolling and sliding sounds.

As far as scratching sounds are concerned, [Van Den Doel et al., 2001] generates a fractal-noise based force profile that is sampled at audio rates to simulate friction force variations. A similar approach is used for rolling sounds by adding a low-pass filter to obtain a rolling quality. [Rath, 2003] develops a model in which the uneven ground surface is significant compared to the size of the rolling object. In [Farnell, 2010] the author presents a rolling model that takes into account the uneven ground texture plus the object's surface irregularities. This thesis is heavily inspired by the fore-mentioned paper while improving the model's interactivity in a game engine.

Previous researches on material identification focus on the decay time of vibrating objects. [Wildes and Richards, 1988] proposed material type recognition depending on a parameter named the coefficient of internal friction which depends on the damping of the material. The authors in [Giordano and McAdams, 2006] point out that damping remains a robust acoustic descriptor to identify material macro-categories independently of the size of objects. [Aramaki et al., 2011] proposes a control strategy for the perceived material in an impact sound synthesizer but does not include these controls within a game engine.

Offering the possibility to control the synthesizer that produces the contact sounds of 3D objects is one of the objectives dealt in this work. Little has been made to enable game developers and sound designers to control physic-based sounds through high level parameters within the game engine. In [Lloyd et al., 2011] a set of plugins for Wwise [Audiokinetic Inc.,] have been implemented that enable the audio designer to choose how varied the sounds will be. In the very recent paper [Pruvost et al., 2015] from the PHYSIS project [PHYSIS,], the authors introduce a controllable framework for interactive and real-time synthesized sounds of solids driven by a game engine. This approach shares similarities with the presented tool in this paper.

2.2 Sampled sounds vs Procedural audio

Sampled sound is still the most typical technique for digital audio in the game industry since its sophistication in the late 1990s when game music moved to recorded tracks on CD. Present game audio systems can compare to professional samplers on account of the improvement of hardware decompression and compressed audio formats [Farnell, 2007]. But this method has limitations due to its static and repetitive nature which does not allow to model the sound of an object based on its physical interactions in a game scene. This leads to inconsistencies between the visuals and their associated soundtrack [Picard-Limpens, 2009]. Despite the use of different processes such as filtering, pitch-shifting and time-scaling to name but three, sampled sounds can not compare to the versatility that procedural audio offers. With the increasing interest that the game industry has put into Virtual Reality (VR) and Augmented Reality (AR), it is becoming essential to provide realistic and compelling sounds in these virtual environments to provide immersion and presence to the user. An obvious advantage of procedural audio over recordings is the large memory storage space that is necessary due to the huge amount of audio assets present in modern games. The ability of procedural audio to generate sounds automatically based on the interactions of the objects in a scene eases the asset management task for the audio team. Instead, the sound designer becomes more of a programmer by taking into account the object's behaviour and physical properties to create sounds. This does not mean that the sound designer is replaced by procedural content as certain tasks need special attention to deliver high quality sounds. Regardless of its advantages, procedural audio still lacks of development tool chains and presents conflicts with current methods due to its still scarce use in the industry [Farnell, 2010].

2.3 Sound Production

For computer sounds to be produced, force models are used as input to the sound synthesis algorithm. In other words physics is used as the excitation parameter to produce sound. [Van Den Doel and Pai, 2003] refers to four different interaction models that produce the excitation force.

1. Impact force, produced during collisions,
2. Continuous contact force, produced during rolling or scratching of an object,
3. Combustion engine forces and
4. Live data streams

This thesis examines sounds produced by the first two models. Impact forces are applied when two objects collide. They last for a short period of time and depend on the physical attributes of the objects. Constant contact forces are produced while an object is rolling on a surface of scratching/sliding on it. They can be modeled as successive impact forces, where one adds on top of the other. The distinction between roll or scratch depends on the shape of the object and on the hardness of both surfaces.

In this thesis we focus on sound produced by rigid-body objects. When two of these objects collide, the energy from the impact deforms them. This deformation is propagated through the whole object, making its outer surfaces vibrate and emit sound waves to the environment. Sound waves impact on other objects in the environment and get reflected and absorbed from them before reaching the ear [Van den Doel and Pai, 1998]. All the interactions that happen before reaching the ear, contribute to the characterization of the sound, making the listener able to visualize the impact just by sound. Since in this thesis we examine only object-related interactions and not sound propagation, below we explain in depth how each of these physical attributes affect sound.

2.3.1 How physical attributes affect sound

Interaction

An object that receives an impact is deformed during a very short period of time. The vibration lasts until the initial energy is fully damped. Hence, the amplitude of the oscillation depends only on the object's damping. On the other hand, scraping sounds are characterized by a continuous supply of energy that varies throughout the object interaction. The force of the interaction plays the most significant role for the amplitude of the oscillation. The stronger the force, the bigger it imposes the amplitude value to be - and the louder the sound. Additionally, the force affects the spectrum of the resulting sound. The greater the force, the more high frequency components [Gaver, 1993b].

Material

The material of interacting objects affects their vibration amplitude over time. Several studies [Wildes and Richards, 1988, Giordano and McAdams, 2006]have examined that damping is a material-specific attribute that is independent from the shape of the object. The more the system is damped, the faster it loses energy and thus the oscillation lasts less. For example, wood has way bigger damping ratio than metal and this is why they produce a "thud" and a "ringy" sound respectively. Moreover, in [Klatzky et al., 2000] they have proven that material and spectral characteristics are correlated, for example glass is found to include in general higher frequencies than rubber.

Configuration

Moreover, the configuration of the object also affects its vibration. The size of it determines how high or low pitched the sound produced will be. More specifically, the smaller the object, the higher pitched is the sound. Elasticity, on the other hand, influences the impact force since the more elastic an object is, the bigger the excitation area when colliding.

Table 2.1 shows the effects on the sound wave generated by each physical attribute.

Attribute	***Changes on Sound Wave***
Interaction	
Type	Amplitude, spectrum
Force	Amplitude, bandwidth
Material	
Density	Frequency
Damping	Amplitude, frequency
Homogeneity	Amplitude, frequency
Configuration	
Shape	Frequency, spectral pattern
Size	Frequency, bandwidth
Elasticity	Amplitude, frequency

Table 2.1: Acoustic effects of source attributes [Gaver, 1993b].

2.4 Modal parameters extraction

Before the sound synthesis can be done we need to extract certain modal parameters that are characteristic of the vibrating solid object. The data needed for a physically motivated synthesis, namely modal synthesis, is shown in the table 2.2.

Symbol	**Description**	**Derivation**
A_n	Initial amplitude	Modal data extraction
d_n	Decay rate	Material properties
f_n	Modal frequency	Modal data extraction

Table 2.2: Derivation of data used in modal synthesis.

Since every different point struck on the object produces different deformations and thus sound, matrices are used to represent the data needed for each point. More specifically, we need a vector $\mathbf{f} \sim (K \times 1)$ corresponding to the modal frequencies of every point, a vector $\mathbf{d} \sim (K \times 1)$ corresponding to the decay ratios and a matrix $\mathbf{A} \sim (K \times N)$, which corresponds to the amplitudes of each mode in every point of the object, where K is the number of modal frequencies calculated in one point and N the number of points. All the above gives the modal model which can be represented as $\mathbf{M} = \{\mathbf{f}, \mathbf{d}, \mathbf{A}\}$ [Van Den Doel et al., 2001]. To obtain this model different techniques such as the FEM, spring-mass systems and the example-guided approach have been used for data extraction and are reviewed here below.

2.4.1 FEM

FEM is a simulation technique that is commonly used for modal analysis, which studies the response of a structure when it vibrates freely. [O'Brien et al., 2002] uses this approach on meshes to precompute the shape and frequencies of an object's deformation modes. It is an accurate method to create *sound maps* of complex-shaped objects without the need of recording equipment or the actual physical object. Most importantly it computes the resonant modes at different locations on the object. A downside of this method is that it complicates the audio pipeline in a game production as the sound designer would need to deal with complex computations and material properties that are not intuitive for him to obtain the desired modal parameters.

2.4.2 Spring-mass systems

A mass-spring system consists in constructing a model that approximates the object's surface based on its geometry and material properties. In [Raghuvanshi and Lin, 2006], the authors convert each mesh vertices into masses and the edges into damped springs. They solve an ordinary equation based on the input mesh to calculate the vibration modes and damping parameters. This modal analysis technique is faster than FEM but less accurate [Ren et al., 2010].

2.4.3 Example-guided

The method used in this thesis for modal parameters extraction is the "Example-guided", where data get extracted using example recordings of the objects being struck as seen in [Lloyd et al., 2011] and [Ren et al., 2013]. Using suitable DSP algorithms it is possible to extract features from the recordings such as the fundamental frequency, its harmonics and the frequency peaks of the signal (amplitudes). An advantage of this technique is that it is easy to use for the sound designer as he just needs to supply an audio clip to obtain the corresponding modal model. The sound designer is used to selecting sounds from audio libraries although in a virtual scene with a high number of audio sources this process could be tedious.

2.5 Modal Synthesis

When an object is excited, the forces applied cause deformations to it, emitting sound waves through the vibration of its outer surfaces [Van Den Doel et al., 2001]. The object vibrates at specific frequencies which are called resonant modes and that decay over time with high frequency modes decaying faster than low ones [Lloyd et al., 2011].

Modal synthesis is a method that simulates the complex behaviour of a vibrating object by decomposing it into the sum of damped harmonic oscillators each corresponding to a modal frequency through additive synthesis [Bilbao, 2009]. This is procedure used in this thesis to generate physics-based contact sounds.

Mathematically, at a struck point k when vibrating in mode n, the impulse response of the model is:

$$y_k(t) = \sum_{n=1}^{N} A_{nk}\ e^{-d_n t}\ \cos(2\pi f_n t) \tag{2.1}$$

if $t \leqslant 0$ and $y_k = 0$ if $t < 0$ [Van Den Doel et al., 2001]. The decay rate d_n of each mode is object-dependent, it determines the energy loss due to the vibration and is related to the material of the object. The amplitudes A_n and the frequencies f_n are the resonant data

obtained during the data extraction phase. Modal frequencies are a set of frequencies that characterize the object and remain the same, while amplitudes are vertex-specific and change depending on the excitation point.

Two approaches for modal synthesis are presented, the "Sinusoidal Additive Synthesis" model and the "Filter-based Modal Synthesis" model. Later, different approaches regarding sound variation along the vibrating object's surface are suggested.

2.5.1 Sinusoidal Additive Synthesis

This method is based on Fourier theory which states that any sound can be expressed mathematically as a sum of sinusoids. The term "additive" refers to sound that is generated by adding together the output of multiple sine wave generators which are modulated by amplitude and frequency envelopes [Smith III, 2011].

The frequencies used for the sound synthesis are the resonant modes at which an object vibrates when struck. On excitation, the sine waves representing the mode vibrations peak to the designated amplitude and then start decaying over time. For physics-based synthesized sounds the decay rate depends on the material of the vibrating object.

[Cook, 2002] points out that the vibrational modes of a metal plate can be predicted by the shape and the location of the impact on the object, which makes it possible to recognize the power of a sound synthesis model based on the summation of several sinusoidal modes. Figure 2.1 shows a model that enables to control the amplitudes and frequencies of a bank of sinusoidal oscillators.

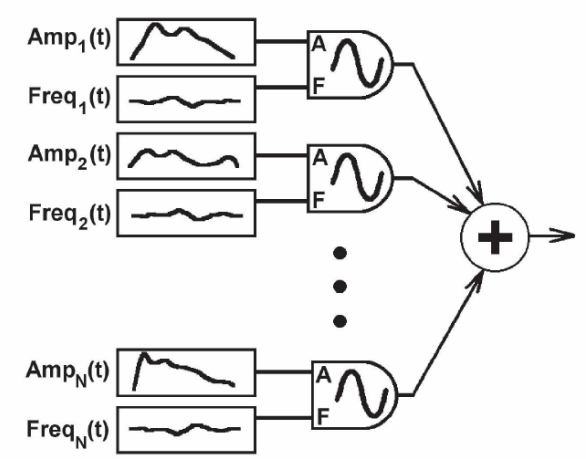


Figure 2.1: Sinusoidal Additive Synthesis Algorithm [Cook, 2002].

2.5.2 Filter-based Modal Synthesis

This method is also additive, since we are adding the outputs of a number of band-pass filters. To synthesize a sound using this method, we use as many filters as the modal frequencies.

Band-pass Filters

At this point we will give some basic description of the band-pass filter since it is widely used in this thesis. Band-pass filters (Band-Pass Filter (BPF)s) take a signal as input and let only a range of frequencies to pass while attenuating the rest. This range depends on the central frequency f_c and the bandwidth. A filter of this kind is a result of a cascading of a low-pass and a high-pass filter circuit.

The Bandwidth (BW) is the passing range or "band" of frequencies. Defining as 0db the resonant peak, we can find the two cut-off frequencies ($f_{c,low}$ and $f_{c,high}$) at -3dB. The difference between those two is the bandwidth

$$BW = f_{c,high} - f_{c,low}. \tag{2.2}$$

In figure 2.2 we can see the frequency response of a BPF [AspenCore, Inc.,].

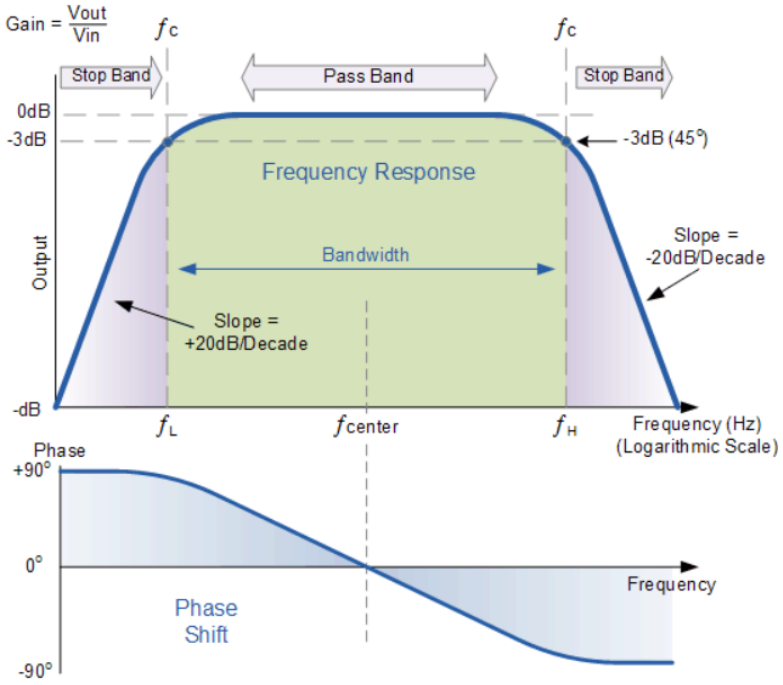


Figure 2.2: Frequency Response of a Band-pass Filter [AspenCore, Inc.,].

Synthesis

In this method, a number of filters are constructed using modal frequencies of the object as center frequencies and the damping of the material is used to control the Q factor (Q) of the filters. The Q is a dimensionless parameters that is inversely proportional to the bandwidth ($Q = f_c/BW$), so the lower the Q, the wider the bandwidth and vice-versa. Using amplitudes from matrix **A** (section 2.4), the level of the passing frequencies is controlled and resonators can be created through the sum of these filters.

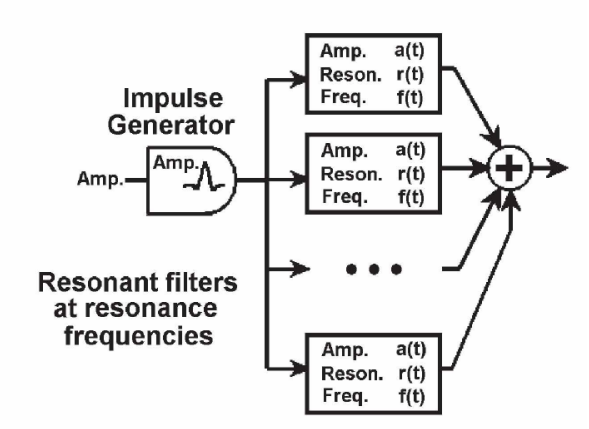


Figure 2.3: Filter-based Modal Synthesis Algorithm [Cook, 2002].

2.5.3 Sound Variations

Each point of an object produces a different sound when struck. This happens because of the different amount of excitation each resonant mode experiences. As mentioned in section 2.4, during data extraction we get a matrix of amplitudes of size $K \times N$ corresponding to K different resonant modes of each of the N points of the object. . Note that the frequencies and dampings are determined by the geometry and material properties of the object while the gains of the modes depend on the contact location on the object [Van Den Doel et al., 2001]. This means that even though resonant frequencies are the same for all surface points, each mode gain is different depending on the location.

There are several methods to achieve spatial variation on sound produced by the same object.

- Modal analysis using FEM and spring-mass systems (as seen in section 2.4.1 and 2.4.2)provide the frequencies of resonant filters and the amplitudes for multiple locations on the object (potentially for each vertex). This is why it makes it an accurate method that provides varying sounds along the object's surface. However, as discussed previously, it is not a suitable method for an audio pipeline due to its computational complexity.

- Another method is to separate the object into areas and assume that each point belonging to the same area and struck with the same force will sound exactly the same. Depending on the resolution of the object's division cells the overall sound of the object varies. The higher the resolution the more varied the sound is and thus the more persuasive results. Additionally, the higher the resolution the more data has to be stored for the model.

- A third method is to store only one amplitude matrix and randomize the values for each impact sound, but this can lead to inconsistent sounds with the collision location. For example, impacting the same location with the same force would produce different sounds when theoretically the same sound shoule be heard. This method is used in [Lloyd et al., 2011].

- Finally, a better approach to the previous method is to retain the same amplitude values for all points of the object, but apply a texture map on the object which indicates pitch changes of the sound all over the object. For instance, the near-edge points of an object produce a higher-pitched sound than the ones in the center of each faces.

2.6 3D Audio

In VR/AR applications, the location of the incoming sound plays as significant role as the sound itself.

To do explain HRTF (3)

Chapter 3

Overview of Method and Tools Used

The target of this thesis is to provide sound designers with an easy-to-use tool which enriches video games with procedural and physics-based audio. The challenge we want to outrun is to offer realistic real-time and event affected audio, without consuming more CPU cycles than the limited offered for audio in games.

To do replace figure with a better one (4)

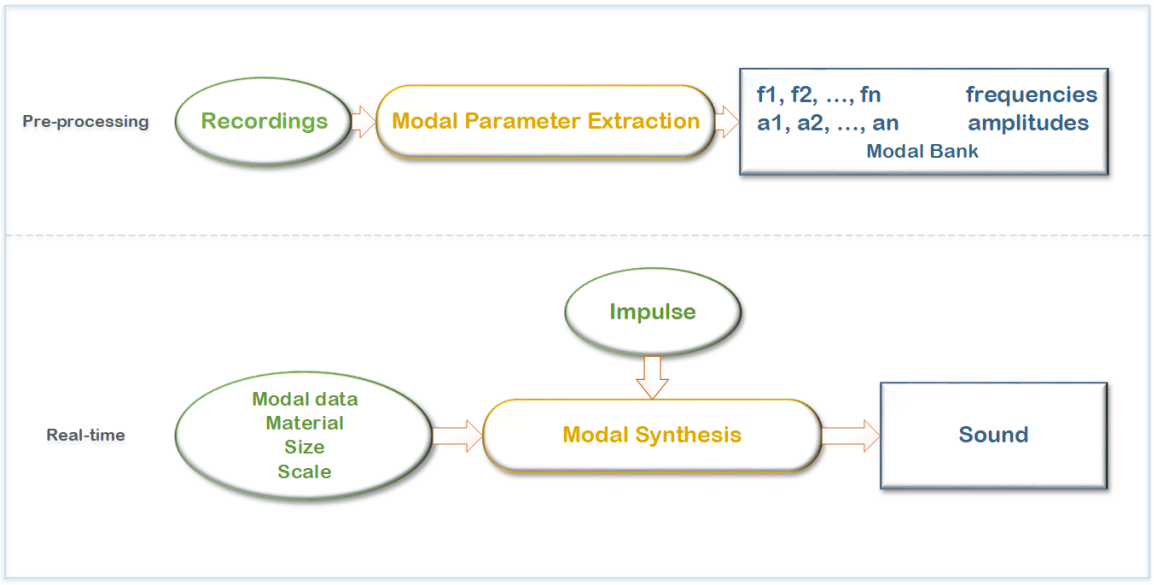


Figure 3.1: Tool overview.

To create this tool, we followed the procedure described in figure 3.1. More specifically, the first step we carried out was to find several everyday objects, separate them into different areas that sound similar when struck and record impact sounds for each area. Afterwards, we used those recordings to extract the data needed for sound synthesis. At the same time, we made a 3D model of every object we used. Next, we generated two different Pure Data patches, introducing two different synthesis methods. Then, we combined the 3D models with their corresponding data and the synthesis patches inside Unity® software [Unity Technologies,]. At this point, the frequencies and amplitudes corresponding to the point of collision are assigned to the patch. The sound is then sent to the audio DSP chain and is played back.

3.1 Recordings

The first step of this tool creation is to obtain the necessary data for audio synthesis. Thus, we performed recordings of the sound produced of the object when being hit in several areas. The signal of those recordings was used to extract the modal frequencies and their peaks.

Since we only used simple shape everyday objects, we could easily assume that nearby points produce almost the same sound and thus separate the object into "sound areas" instead of calculating different modal matrices for each vertex. Unofficial tests proved that this accuracy-computational complexity trade-off was acceptable. A sample object and its division in areas is shown in figure 3.2.

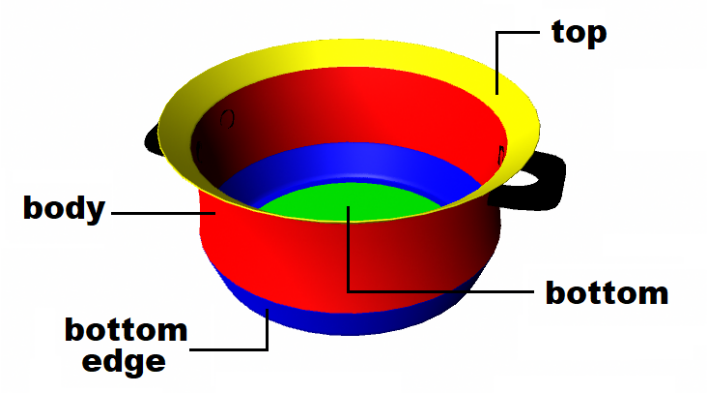


Figure 3.2: Division of an object into areas with similar sound.

The procedure of the recordings will be explained in detail in section 4.1.

The reason why we chose to extract the modal data from recordings instead of using the FEM method as in [Director-O'Brien, 2001] or [O'Brien et al., 2002] or using spring-mass systems as in [Raghuvanshi and Lin, 2006] lays in simplicity. More specifically, recordings of impact sounds is much easier for our target users to perform, if they want to extend our tool, than having to use complex calculations or software.

3.2 3D models

To achieve realistic sounds, we have to correspond the impact recordings of the real objects with same size 3D models of them. Hence, we measured the dimensions and weight of the eleven objects and used those data to model the objects. In table 3.1 the maximum dimensions and the weight of the objects are displayed. To create the 3D models of the objects we used **Maya Autodesk** software [Autodesk, Inc., b] and exported them as FBX® files [Autodesk, Inc., a] which is a format recognizable by Unity®.

Name	Dimensions(cm)	Weight(g)	Picture
Cooking pot	$21.5L \times 21.5W \times 10.8H$	680	
Cup	$7.5L \times 7.5W \times 6H$	125	
Cutting board	$41.5L \times 26.5W \times 4.5H$	2200	
Jug	$14L \times 10W \times 21.5H$	150	
Mortar	$11L \times 11W \times 19H$	850	
Bowl	$20L \times 20W \times 10.5H$	100	
Plate	$25.5L \times 25.5W \times 2.5H$	700	
Rolling pin	$43L \times 7W \times 7H$	700	
Wine bottle	$7L \times 7W \times 29H$	400	
Wine glass	$8L \times 8W \times 18H$	120	
Wok	$35L \times 35W \times 9.5H$	1225	

Table 3.1: Maximun dimensions and weight of the eleven objects.

3.3 Modal data extraction

Chuck language is a music programming language, made for "real-time sound synthesis and music creation" as mentioned in their website [Wang,]. It's biggest advantage is the way it manipulates time. More specifically, the user specifies how long a sound will last, independent of other sounds that may play at the same time.

We used the ChucK language at the starting point of our thesis to identify and extract the peaks of the recorded audio files. The algorithm used in this part of the thesis is made by Perry Cook for the course **"Physics-Based Sound Synthesis for Games and Interactive Systems"** held by *Perry Cook* and *Julius O. Smith* at **Kadenze Academy** [P. Cook,].

From a modal analysis one can find out that each object vibrates in a very high number of modes. Although, most of them are inaudible and do not contribute to the sound model. It is, therefore, desirable to preserve CPU cycles by reducing the number of calculated modes. Based on the recommendations of the author Perry Cook, we chose ten as the sufficient amount of modes for the analysis/synthesis. Afterwards, the algorithm having taken a recording as input, computes its histogram and identifies its peaks. The frequencies where peaks occur are the modal frequencies candidates. Depending on the numbers of peaks we chose, the algorithm outputs the strongest peaks. Finally, the algorithm finds the maximum value of the signal on each peak, calculating the corresponding amplitude of each mode.

ChucK language was chosen, because of its build-in functions to manipulate sound like *Fast Fourier Transform (FFT)* of input audio samples and windows functions [ChucK Documentation,]. Another option to extract the peaks of the sound waves could be to use python programming language on audio files, but this would request to program a number of functions or include a number of libraries that perform actions like file input/output, FFT and more.

To do should we explain why we used FFT? (5)

3.4 Modal synthesis patches

Pure Data (Pd) is another music programming language. It is open source and the main difference from ChucK language is that Pd is a visual or "patcher" programming language, using objects instead of code, linked together to form a sequence [Puckette,]. We chose to use this software as our synthesis engine mainly because of the ability to compile the patches into C# code, as explained below in section 3.5. Another important reason for choosing it is the possibility of real-time parameter manipulation and easy testing during implementation period.

All synthesized sounds (impact, rolling and scratching) are being synthesized under one main pd patch. However, two different patches have been developed, one for each of the two examined methods. Since the audio synthesis patch takes the modal data as input, every object can use the same patch. The synthesis patches will be described in detail in section 4.3.

To do develop more? (6)

3.5 Heavy Compiler

Heavy is a compiler that generates audio plugins from Pd patches in interactive sound and music applications [Enzien Audio, Ltd.,]. In this thesis we used it to compile Pd patches into Unity® audio plugins. Heavy's interface is their website where users upload patches and then are able to download the corresponding plugins and put them into their applications. The plugins we used consist of DLL files and a C# (Unity® code) script that allows communication of the plugins with the rest of the scripts and also enables the sound card to play audio.

Through the generated C# script, we are able to send float values to the audio plugins - which are the compiled Pd patches - as inputs to generate the appropriate sound. Those floats are the frequencies and their corresponding amplitudes, the quality factor (Q-factor) of the band-pass filters, the impact force of the collision, the roughness of the object, the multiplier of the size of the object, the velocity of the object and the rolling and scratching duration times. A difficulty we encountered by using this compiler was the inability of sending a whole array or list of floats. Thus, we had to send every frequency and every amplitude individually, creating a float parameter for each.

3.5.1 Why not OSC?

The most popular way to communicate between Unity® and Pure Data is the Open Sound Control (OSC) protocol. The reason why we did not use it is because it requires establishing a connection between two software programs and send data between them. On the other hand, Heavy makes everything work inside Unity® and it is as simple as passing floats between scripts.

3.6 Unity®

Unity® is a game engine software. This is where all previous work is combined together and outputs the final product. For the purpose of this thesis, several demonstration scenes are made inside Unity® where objects are struck in several points and produce different sounds.

The first part of the Unity® implementation is the assignment of modal data to every different area of the object. This is done in linear time ($O(n)$ for n modes). The whole procedure of assigning the appropriate data includes the identification of the area of the

object that collided, filling the arrays with the corresponding data and set the parameters of the plugins. Afterwards, the type of collision is identified and a number of other parameters are calculated and sent to the plugins, like the impact force and the duration of the collision.

Audio in Unity® is enabled using the *OnAudioFilterRead()* function. This function is running in the audio thread, which is a different one from the main thread. Its job is to send the audio buffer to the sound card and is called every $\sim$ 20ms, so it does not require a function call from the programmer [Unity Scripting Reference,].

3.7 Microsoft Hololens Emulator

To do keep it or not? (7)

Chapter 4

Implementation

4.1 Recordings

The measurements were conducted in an anechoic chamber at DTU's Department of Electrical Engineering using a microphone placed one meter away from the struck object. To record the sounds we used Bruel & Kjær's half inch pressure field microphone type 4192 and microphone preamplifier power supply type 5935L, as well as the 744T digital audio recorder from Sound Devices at 44100 Hz sampling frequency. The setup can be seen in figure 4.1.

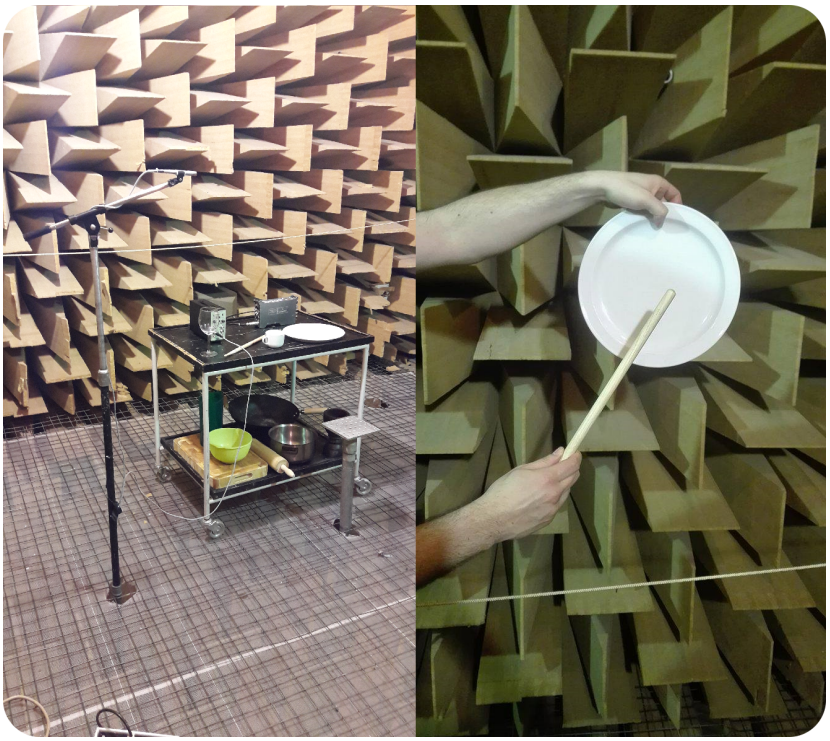


Figure 4.1: Picture of the setup for the measurements (left) and of a struck object (right).

To control the impact we hit the objects by hand with a wooden drumstick (figure 4.1) while trying to use the same impulsive force. Prior to the recordings, every object was divided into different surface areas depending on its shape and the sound produced by these areas (see figure 3.2). Therefore, several impact locations were chosen and recorded for every object.

Eleven objects of everyday life made of five different materials (plastic, wood, ceramic, glass and metal) were used for the experiment. The idea of choosing these objects came firstly from the need of owning them (to perform the recording) and our ability to model them for the demo (as they should be simple enough). Secondly, since we wished to test the immersion of the synthesized sounds on users, we wanted to be sure that the sounds used are familiar to them. In figure 4.2 both real and modeled objects are shown.

(a) Real objects.

(b) 3D models of the objects.

Figure 4.2: The eleven objects used in the thesis.

4.2 Modal data

Using the peak locator algorithm described in section 3.3, we extracted the modal data - frequencies and their corresponding amplitudes - from the recordings. Figure 4.3 shows the data extracted for the wine bottle object. Each graph corresponds to one of the "sound areas" we have divide it into.

Although, theoretically, each object can be modeled using one vector of frequencies and multiple vectors of amplitude data, one for each different vertex, we can see in figure 4.3 that this does not happen in practice. There are multiple reasons why, with the most notable

one being the possible discrepancies of the data extracting algorithm. In addition, to include all the resonant frequencies of one object, instead of the ones with the strongest peaks per area, we would need a very big frequency vector. This would take up a lot of memory space, without improving significantly the result. However, other causes could be the environment noise, the difference in the force when hitting the object or the way it is hold when being hit, during the recordings.

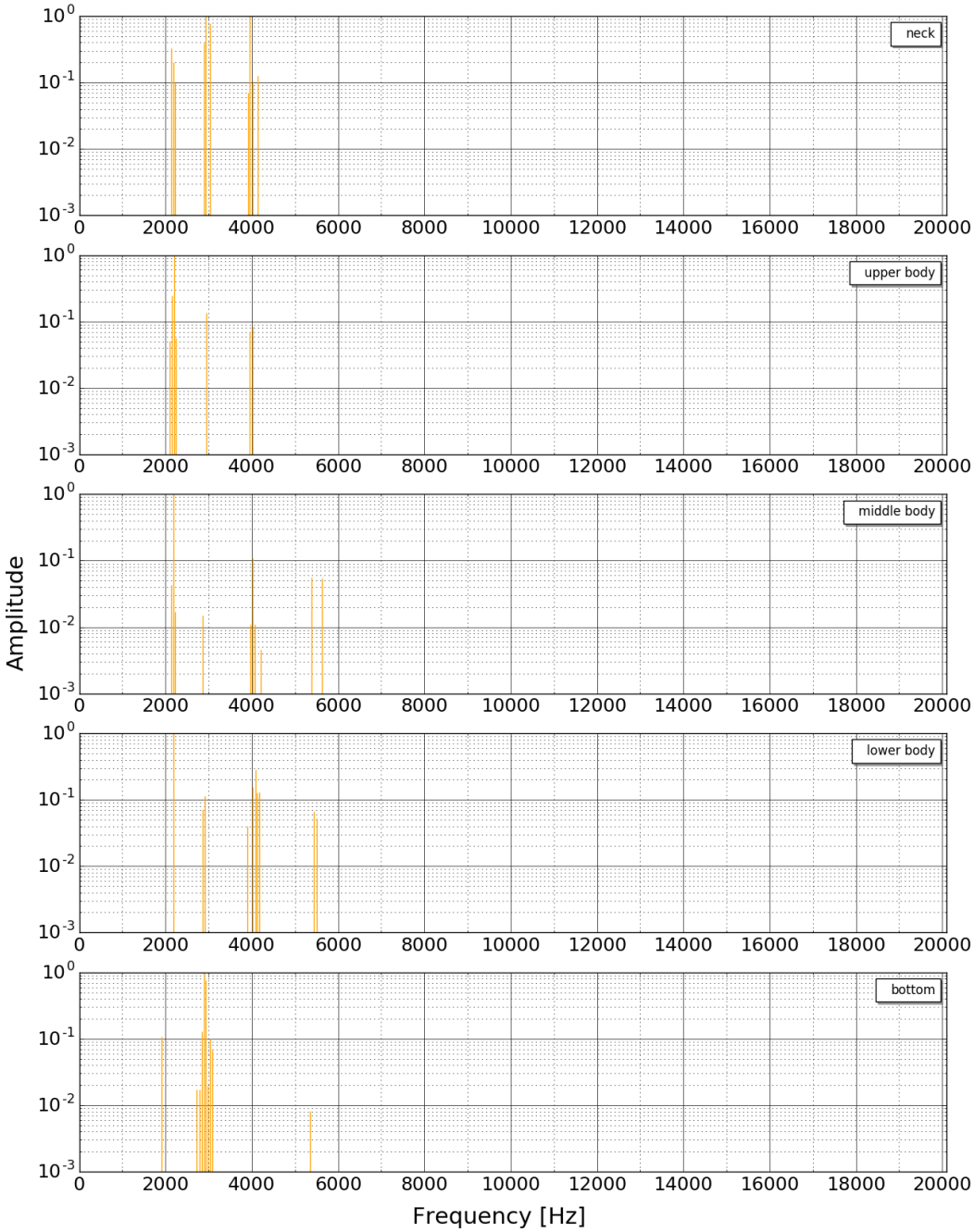


Figure 4.3: The frequencies and their corresponding peaks for each part of the wine bottle object.

Nevertheless, it is obvious that the fundamental frequencies are similar for each graph and lay just above 2000Hz for the wine bottle.

4.3 Sound Synthesis

All synthetic sounds have been created in Pd patches and are interpreted by Heavy which generates audio plugins and a C# interface for Unity®. This C# script is attached to the GameObject in the scene so that the sound is processed within the game world.

We have created our own C# script that assigns the modal parameters, that we extracted in the analysis part with the ChucK code (see section 3.3), to every one of the objects. This is done independently of the synthesis methods used here below.

We consider two different types of force models as input to the synthesis patches. Impact forces that are used for sounds produced by a collision and constant contact forces, used for rolling and scratching sounds.

4.3.1 Impact Sounds

4.3.1.1 Sinusoidal Additive Synthesis

In this section we describe in depth how the Pd patch corresponding to the sinusoidal additive synthesis of impact sounds works. The patch attempts to translate equation 2.1 into the programming language of Pd. Some of its terms are referenced in the following explanation.

First of all the frequencies and amplitudes matching the ten modes of the object are initialized. We can therefore feed these frequencies, which we identified as f_n in the equation 2.1, into the different oscillators. In Pd, oscillators output a cosine wave which is equivalent to $cos(2\pi f_n t)$ from the equation which suits our purpose perfectly.

We also translate into Pd the expression $e^{-d_n t}$ which corresponds to the damping of every mode n. Gaver, in [Gaver, 1993a], states that for each partial the decay rates d_n are controllable through a parameter D which corresponds to a material and that a useful heuristic, that we use in our patch, is to have $d_n = 2\pi f_n D$. By experimenting we established that values of D range from approximately 0.0002 for metal to about 0.05 for plastic sounds, with glass, ceramic and wood sounds in between. The higher the damping the higher the values. Then we multiply the damping by the partial's initial amplitude A_n to obtain an amplitude envelope that varies over time and which we multiply by the oscillator's signal. The output is what we call a partial which is illustrated in figure 4.4.

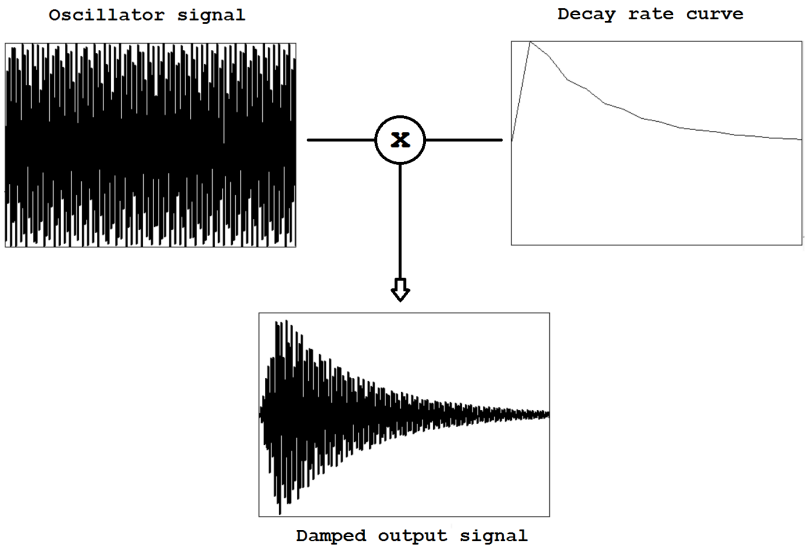


Figure 4.4: Diagram showing how the output partial is created from a 5000 Hz cosine wave and a decay rate curve with D = 0.005.

The final sound is produced by adding together the ten partials. The resulting signal is multiplied by the magnitude of the impact. For this, we calculate the kinetic energy with Unity®'s physics components (see section 4.5.2). As described in [Farnell, 2010], before sending the signal to the DAC we pass it through a clipper. This gives richer harmonics and produces brighter sounds the harder the impact is [Aramaki et al., 2009].

The patch produces an impact sound whenever the OnCollisionEnter method from Unity® is called. This is done when the collider, that has the script attached to it, has begun to touch another collider. When this happens we set the magnitude of the collision and then send an event to excite the patch. This is done by setting the value of t in 2.1 to zero which increases over time making the sound to decay.

4.3.1.2 Filter-based Modal Synthesis

This synthesis method is based on the utilization of a bank of ten bandpass filters. Pd's bandpass filters have three control inputs as seen in figure 4.5. The left inlet is the incoming audio signal, the middle one sets the center frequency and the right input sets the Q factor value. The characteristics of these filters define the virtual object.

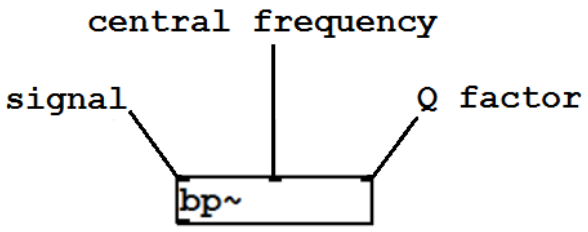


Figure 4.5: Pd's bandpass filter with its three inlets

The same way we did in the previous method, we initialize all ten frequencies and amplitudes of the object. Every frequency f_n is sent into a bandpass filter as the center frequency.

Every filter is characterized by its Q factor which is directly related to the damping. The higher the value of Q, the narrower the bandwidth and the less the resonator becomes damped. Thus, Q determines the material of the impacted object [Gaver, 1993a]. By manipulating Q we can obtain different material sounds. Through experimentation we have found values of Q that range from about 20 for plastic to 5000 for metal.

To cause the object to sound we use an impulse signal that excites the filter. The amplitude of the signal is 1 at $t = 0$ and 0 everywhere else, as represented in figure 4.6. This impulse is multiplied by the value of the kinetic energy when the object impacts with a surface.

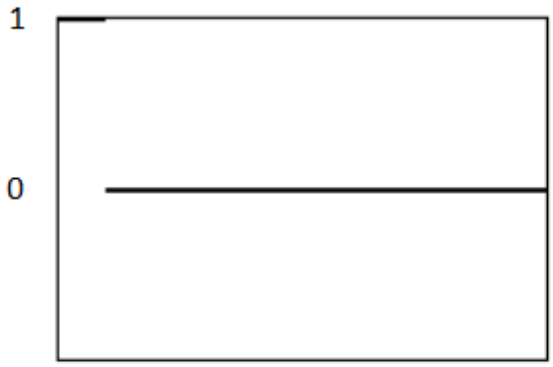


Figure 4.6: Impulse signal used to excite the bandpass filter.

Pd does not include a build-in impulse signal, thus we had to construct one. We created the array shown in figure 4.6 and read the values from it, as the input. Since the lowest amount of time calculated by Pd proved to be 2 milliseconds, it was not possible to read a small fragment of the input signal. Therefore, we made the impulse array to be of size 441, which is the number of samples per 10 milliseconds and we read the input array in a duration of the sample rate.

The output signal of each of the ten filters is multiplied by the corresponding amplitude A_n of the mode. All ten resulting signals are added together. The signal is sent through a clipper as we did in the previous synthesis method.

4.3.2 Scratching Sounds

The sound produced by an object that is scraped across a rough surface can be assimilated to a succession of multiple impacts in a short time according to [Gaver, 1993a]. Additionally, the aforementioned paper shows that the resonant modes present in the spectrum of a struck object, are the same as when the object is scrapped. We can then use the same modal parameters as in the impact methods described here above.

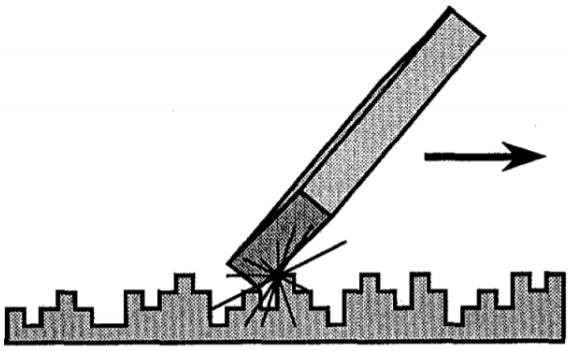


Figure 4.7: Scrapping involves a multitude of micro-collisions against a contact area. Picture from [Gaver, 1993a]

To produce scratching sounds we follow a similar technique to [Gaver, 1993a] and [Van Den Doel et al., 20 who propose the use of filters. We therefore choose the filter-based modal synthesis method to model the resonator as seen in the previous section. The difference lies in the signal that excites the model. We implement this by generating a noise impulse waveform, as seen in figure 4.8, that passes through the bandpass filters. We create this waveform by having a simple impulse signal that is scaled relative to the velocity and material of the object and then multiplied by a white noise signal. From a heuristic approach we deduced that the higher the velocity and the Q factor, the higher the gain of the scratching sound. The length of the excitation signal depends on the time it took Unity® to complete the last frame. The scraping sound is triggered in Unity®'s OnCollisionStay method when the object's collider is touching another one and with the condition that the angular velocity of the object is beneath a threshold. The angular velocity specifies the rotational motion of a rigid body [Sears and Zemansky, 1964]. We therefore add this condition to differentiate between sliding and rolling.

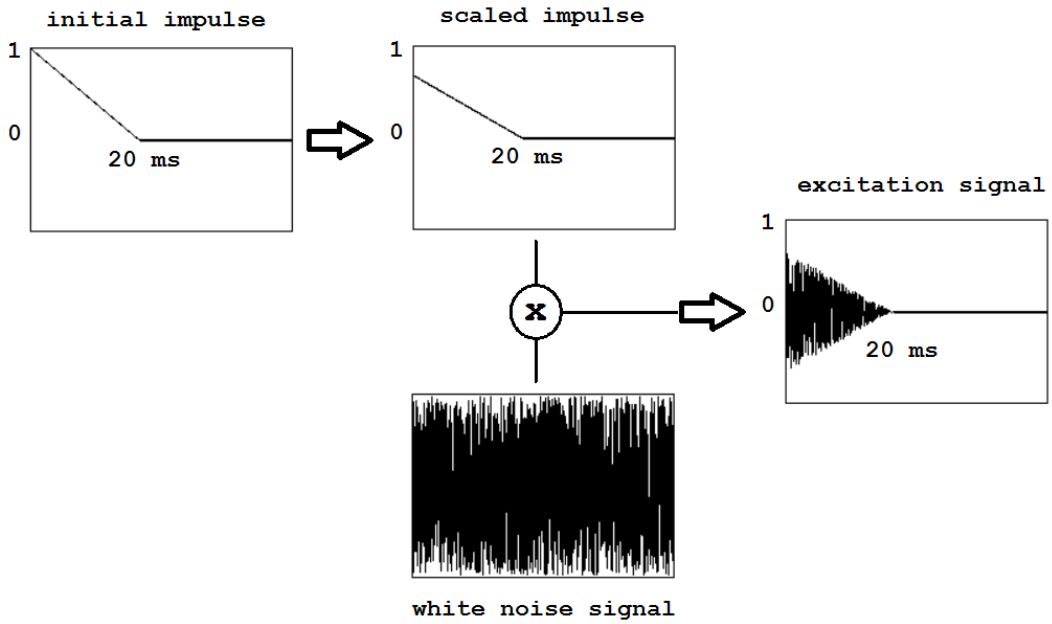


Figure 4.8: Diagram showing the process to get the excitation signal of the resonator to produce scratching sounds.

The authors who's approach we are following state that the center frequencies of the bandpass filters are scaled with respect to the contact velocity. The higher the velocity, the more the proportion of high-frequency energy increases. To recreate this effect we scale the incoming filter frequencies depending on the velocity of the sliding object.

4.3.3 Rolling Sounds

As well as scratching sounds, rolling sounds are produced by the irregularities of the surfaces in contact [Van Den Doel et al., 2001], namely the rolling object's surface and the ground. We therefore focus our study on two models inspired by [Farnell, 2010]: a series of repetitive impulses that correspond to the surface profile of the rolling object and the irregular bumping sounds due to the uneven ground.

First we aim our attention to the sounds produced by the object's surface irregularities. For simplification we take a regular octagon that has received an impulsive force and therefore rolls along a plane. Every time one of the vertices impacts the ground, energy is lost to heat and sound. Thus, as the octagon rolls, a pattern of eight impulses is created for every rotation as seen on figure 4.9. To create these impact sounds we use the filter-based synthesis method. The filters are excited by a succession of eight impulses of different amplitudes as no real object has a perfect geometry. The outcome results in a pattern of quasi-periodic audio cues distributed differently over time depending on the speed of the rolling object. See figure 4.9. [Houben et al., 1999] and [Rath, 2003] suggest that this periodicity which originates from the asymmetry of the object, enables listeners to distinguish between sliding and rolling sounds.

Explain script somewhere

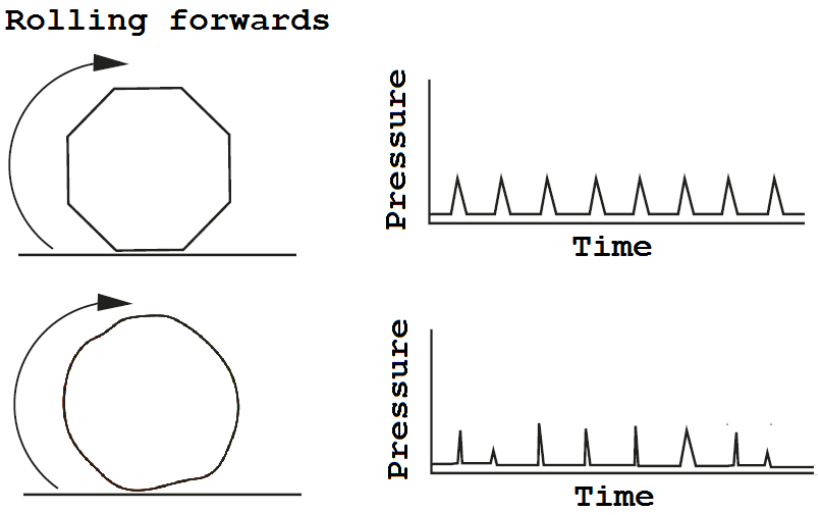


Figure 4.9: An octagon and a real spherical object pressure levels over time as they roll.

Let us dive into the actual implementation of the latterly described model. We mentioned that the amplitude of the impulse signals used to excite the resonator are different. With a heuristic approach, we choose a sequence of eight multipliers that correspond to the prominence of the bumps along the object's surface contour. Additionally, we have incorporated a parameter that determines the object's surface roughness. This parameter, which can be adjusted by the user, scales the values of the eight multipliers. The lower the value of the parameter the smoother the object's surface. In other terms, the smoother the object's surface, the smaller and more homogeneous the amplitude of the impulses are. We can compare the amplitudes of the impulses for different roughness degrees in figure 4.10.

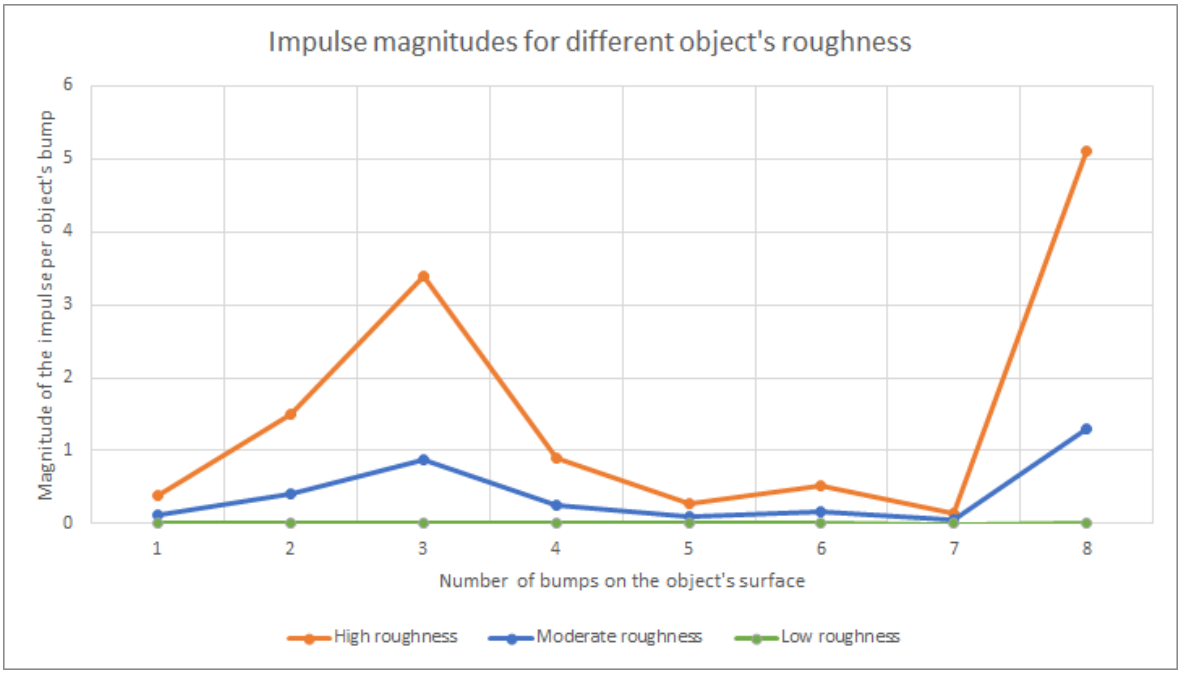


Figure 4.10: Graph showing the amplitude of the impulses for every bump on the object's surface for three values of the roughness parameter.

We now look into the sounds produced due to the irregularity of the ground. As pointed out in [Van Den Doel et al., 2001], even a smooth ball rolling on a rough surface produces sound. This paper and [Rath, 2003] indicate that this is due to the small constant collisions of the ball with the surface's asperities. The bigger the ball, the less the surface details are "felt". In our implementation we consider the surface's irregularities to be very small compared to the radius of the rolling object. See figure 4.11. This suggests that our model for uneven ground is similar to our previously explained scratching model as [Van Den Doel et al., 2001] proposes. The difference is that the gain of the output signal is lower for the rolling as we consider that rolling friction forces are smaller in comparison with scratching friction [Mehta and Mehta,].

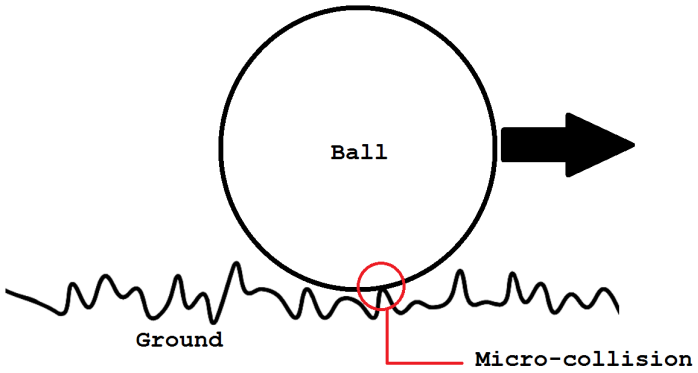


Figure 4.11: A smooth ball rolling over an small ground irregularities.

4.3.4 Sound Variations

The best approach to get the amplitude matrix is through FEM. However, the computational power needed to access all these data exceeds the limitations which audio in games undergoes. Hence, other methods should be developed to achieve spatial variation of sounds.

One method is to have only one amplitude matrix (corresponding to only one point of the object) and randomize the values every time a collision occurs. During the development of this thesis we tried this method, but we abandoned it rather quickly due to undesirable sounding results. In other words, two impact sounds, being produced by two very close locations on the object, were exceptionally different.

Although the last method described in section 2.5.3 of chapter 2 appears to be the best solution, we did not implement it in this thesis due to the inability of Unity® to give us data about the exact point of the object that participated in the collision.

Hence, we decided to carry on with another method where we separated each object into similar "sound areas" and simplified the calculations by assuming that each area produces the exact same sound when struck. Therefore, during measurements, we conducted one recording for each one of these areas, ending up with 4-5 different amplitude matrices for each object instead of thousands.

4.4 User Interface

This section describes the *user interface (UI)* of the tool, where designers are able to tweak parameters and choose the sound they prefer for every object. The UI is made inside Unity®, by programming a custom inspector. *Inspector* in Unity® is a window that shows up after selecting an object, a file etc inside the platform and it displays all information relevant to it. We also used a custom *GUISkin*, which is a set of settings about the Graphical User Interface (GUI).

When the designer wants to insert a new object with the audio implemented, he can either use one of our pre-made prefabs or assign the procedural audio component on a Unity® game object of his choice. In the second case, he only has to select this game object and then from the Unity® menu bar he can select one of the two available methods described above, as seen in figure 4.12.

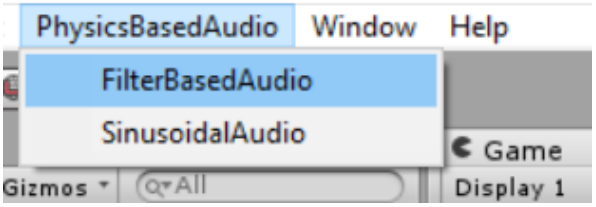
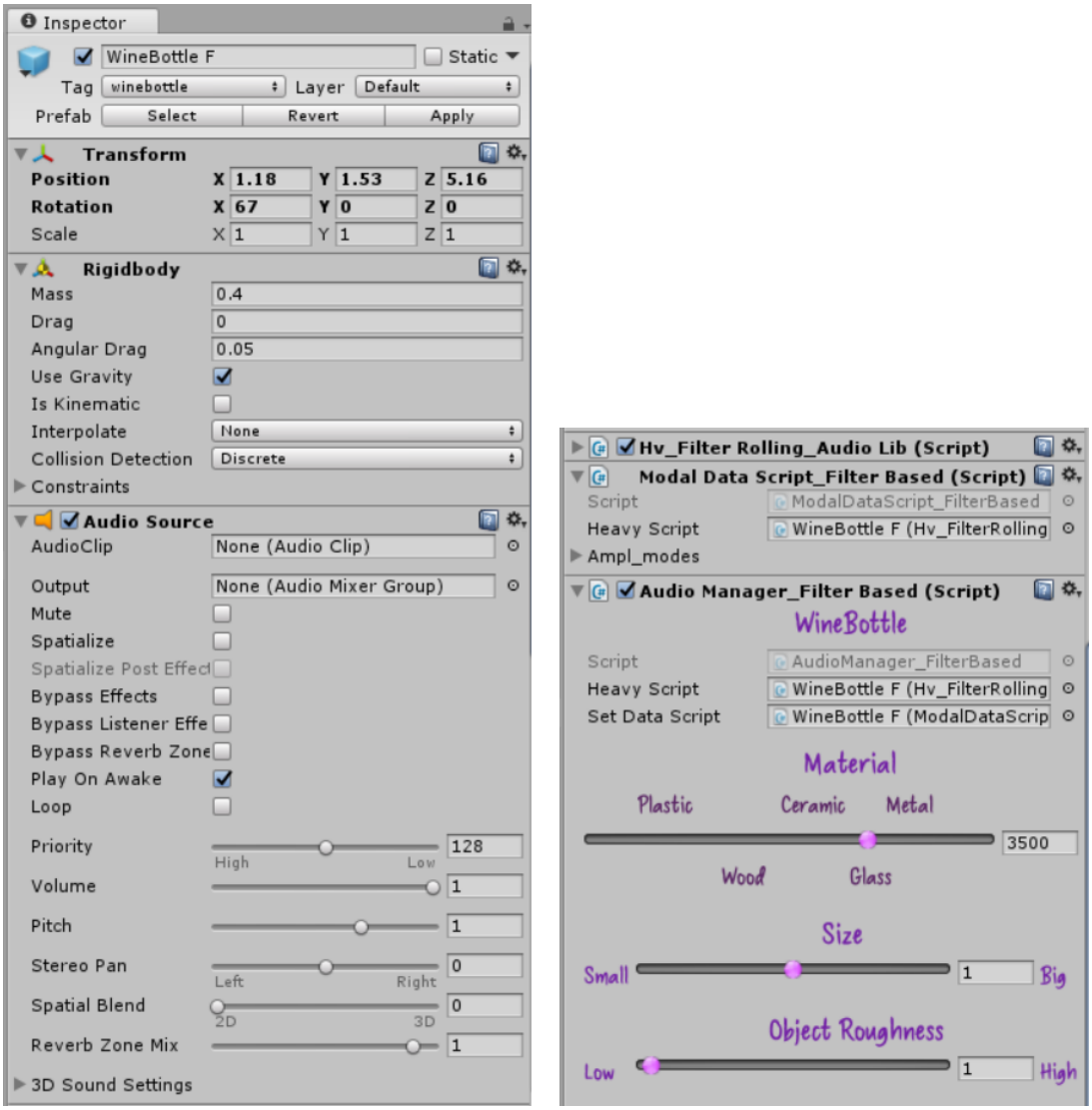


Figure 4.12: Designer can assign the audio manager from the menu bar.

There are a number of components that are necessary for a game object. The "Rigidbody" which gives physics characteristics, the "Audio Source" that activates the sound, the synthesis plugin that generates procedural audio, the "Modal Data Script" that assigns the correct modal data to the object and the "Audio Manager Script" which controls the audio events. Screenshots of an example Unity® inspector can be seen in figure 4.13
To do fix script names and take screenshots again (8)



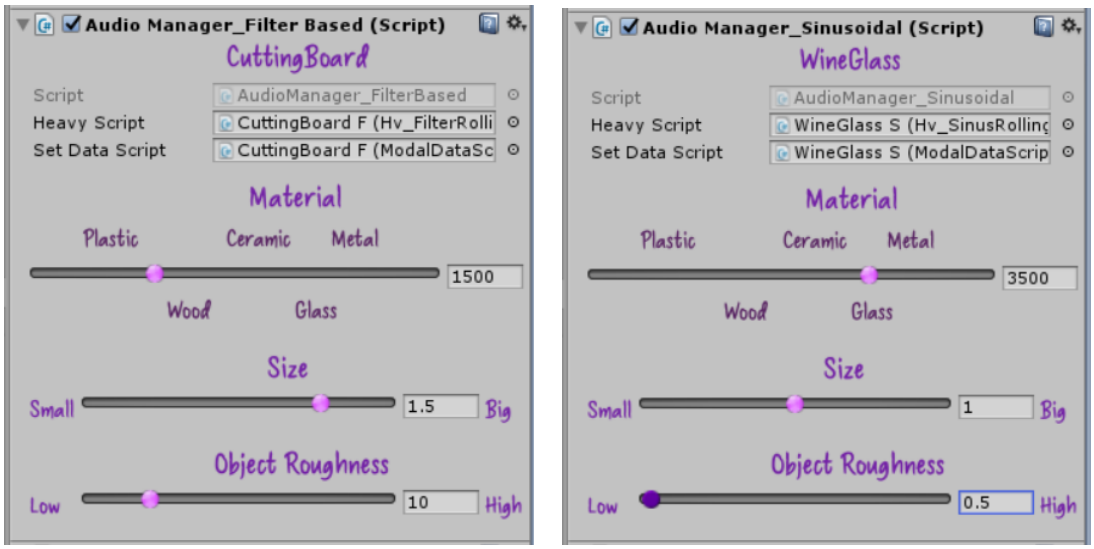
(a) Upper part **(b)** Lower part

Figure 4.13: An example inspector of a game object inside Unity® platform.

Since each set of modal data is related to one specific object, the tool is restricted to the objects available up to this time. Therefore, the designer has to assign a **"tag"** of one of the eleven objects available on the tool (cooking pot, cup, cutting board, jug, mortar, bowl, plate, rolling pin, wine bottle, wine glass or wok), if he uses his own objects. However, it is easy for everyone to contribute with new objects to the tool, by following the guide on appendix C.

Figure 4.14 shows the three parameter sliders which a designer can tweak. They are the same for both of the synthesis methods that were used in this thesis (figures 4.14(a), 4.14(b)). The first slider is the material selection. Although each object used in this thesis has a distinct type of material, designers are able to assign to them another material of the five available choises (plastic, wood, ceramic, glass, metal), or even blend of two of them. The second slider, concerning the size of the object, can be used when the designer wants to introduce a smaller or bigger object that the default ones, or even when he prefers the sound to be more high or low pitched. The slider allows changes to size from half up to double of the original object. The last slider adjusts the roughness of the object's surfaces. By tweaking

this, designers are able to chose surfaces of the objects to be smoother of rougher than the default ones. Below there is an extended description of the sliders.



(a) Filter Based **(b)** Sinusoidal

Figure 4.14: The custom inspector inside Unity® platform.

4.4.1 Assignment of Different Materials

Different materials can be assigned to the objects made for this thesis. The designer is able to choose between *plastic, wood, ceramic, glass and metal* by adjusting the corresponding slider on the interface (see figure 4.14).

Metallic or glass sounds are more "ringy" than wooden or plastic ones that are more "thud". We achieve those sounds by changing the **Q-factor** of the **band-pass filters** used in the pd patch. Q-factor indicates the power loss in the filter. The higher the Q the less power is lost, so the resonator vibrates longer as explained in equation 4.1 [Cory et al., 2006].

$$Q = 2\pi \frac{\text{maximum energy stored}}{\text{total energy lost per cycle at resonance}} \tag{4.1}$$

Using trial and error method, we came out with an average value of the Q-factor for each material and we use it as the default value on the tool. Those values are shown in the table 4.1.

Material	Average Q-factor
Plastic	1000
Wood	1500
Ceramic	3000
Glass	3500
Metal	4000

Table 4.1: Default values of Q-factor for each material in the tool.

4.4.2 Changing the Size

In an application, the same object can appear in different sizes. It is known that under the same excitation, the smaller the size of an object, the more high-pitched sounds it will produce, because the sound waves travel a smaller distance. Hence, we implemented a slider for the designer to choose the best sound that corresponds to the size of the object. We should note that the middle position of the slider (*size:1*) corresponds to the real object used for the data extraction.

4.4.3 Changing the Object Roughness

Another setting that sound designer is able to tweak is the object roughness. Even though our tool covers several different object materials, not every material has a uniform surface roughness when used in different objects. Adjusting a slider, the designer is able to choose a unique sound for every object of the same material.

4.5 Unity® Scripts

We used the Heavy [Enzien Audio, Ltd.,] compiler to convert the Pd patches described above into Unity® compatible C# code. However, we needed to implement several more scripts for actions like 1) Identify the type of the object and assign the corresponding modal data 2) Calculate the scale of the object if any 3) Detect which point of the object collided and assign the corresponding modal data 4) Calculate the impact force 5) Calculate distance traveled when rolling 6) Start an impact, rolling or scratching sound .

4.5.1 Scaling

As mentioned above, impact sound gets more high-pitched when an object is scaled down and vice versa. To achieve a realistic scaling when the designer uses the build-in scaling feature of Unity®, the tool calculates the size of the game object on start. More specifically, a *scaling average (avgScale)* is calculated, taking into account all three dimensions:

```
1 avgScale = local scale of x dimension + local scale of y dimension + local
    scale of z dimension / 3
```

This average is used as an adder to the *size parameter* described above. To avoid distortion in sound and to stay within the audible sound frequencies, we set a limit of adding 8.5, a number found heuristically. We consider this to be a good choice because Unity® uses *meter* as the default unit and since we are mainly focusing on everyday objects, we find it rare for someone to use objects more than 8.5 times its original size.

Then, the tool checks whether a scale-up or a scale-down was executed. In the first case, we perform a normalization to 1/10th of the average scale value and we add it to the pitch multiplier **To do** reference the description of the pitch multiplier (9). We apply it to the size slider value and then to the pitch multiplier which we use to re-set the modal frequencies. To be more precise, instead of applying the actual pitch multiplier added with the average scaling, we subtracted from 2 and then we use it ($2 - temp$ on the code below). This happens because we reversed the size slider. More specifically, the multiplier directly applied to the frequencies, increases them when it is bigger and decreases them when it is smaller. However, for convenient reasons, we wanted it to be the bigger the multiplier, the bigger the object and reverse. Since 2 is the biggest value, we normalized it to be the smallest one.

In the second case, we subtract the value from the pitch multiplier. We do not need to normalize the average scaling value, because it is already between 0 and 1. Afterwards,

we follow the same procedure as above, with one difference; instead of subtracting the new pitch multiplier from 2 we add it to 1. This happens because now the biggest value of the size slider is 1 -since above this it counts as a scale-up- and we still want the reversed value for the slider, so we subtract the subtracted value, making it a plus (+). If no scaling took place nothing happens.

```
IF average scale > 1 (scale-up) THEN
    DIVIDE average scale by 10 to normalize
    ADD the normalized value to the pitch multiplier
    STORE new pitch multiplier to the size slider
    CALL SetTheFreqs to re-set the modal frequencies
ELSE IF average scale < 1 (scale-down) THEN
    SUBTRACT average scale from the pitch multiplier
    STORE new pitch multiplier to the size slider
    CALL SetTheFreqs to re-set the modal frequencies
END IF
```

4.5.2 Excitation of Impact Sounds

This is were the tool detects a collision of an object with something else. The collision could be either with the ground, another object from the tool or just an object in the scene and it is identified by the program when two collider Unity® components attached to two different objects touch each other.

OnCollisionEnter function

OnCollisionEnter [Unity Scripting Reference,] is a Unity®'s build-in function, which gets called when an object enters a collision.

The first thing that happens when a collision takes place, is to identify what kind of object is the one that collided with something, and which part of the object collided. Hence, a function is called which detects the type of the object using the tag manager. Tag manager holds all available tags. Tags are used to group similar game objects and make them retrievable from scripts. After type of object is identified, the variable used for object perimeter (used for rolling sound) is assigned to its equivalent value and then the modal data (frequencies and amplitudes) that correspond to this type of objects are assigned to the variable used from the algorithm.

Moreover, the algorithm calculates the kinetic energy of the whole collision, using the equation 4.2 [Crowell, 2003]:

$$K = \frac{1}{2}mu^2, \tag{4.2}$$

where m is the mass of the object under test and u the magnitude of the relative velocity between the two colliding objects. In Unity® we can calculate the latter using the command: *Collision.relativeVelocity.magnitude* [Unity Scripting Reference,].The kinetic energy corresponds to the collision force magnitude and we use it to turn the volume of the sound up or down depending if the collision was strong or wear respectively.

Another parameter that the algorithm takes into account is whether an object collides with another modal object or not. By modal object we mean the object that our tool provides that come with modal synthesis sounds. In case we have a collision with a modal object, we enhance the collision force magnitude adding more force that depends on the materials of both of the participant objects. Namely, we calculate an average between the two quality factor normalized values, we multiply it by the collision force magnitude and add this to the initial collision force magnitude. We do this because from our experience with the objects,

we found out that when two high-pitched materials like glass or metal collide to each other, they produce a much more intense sound than when a high-pitched material collides with a low pitched like plastic or wood.

$$new\ CFM = CFM + CFM * \left(\frac{\frac{Qfactor_1}{5000} + \frac{Qfactor_2}{5000}}{2} \right) \tag{4.3}$$

where CFM : the Collision Force Magnitude.

To do see if it is necessary to describe the 3 different DACs here or just in the patch (10)

4.5.3 Excitation of Rolling and Scratching Sounds

OnCollisionStay function

Unity®'s build-in function *OnCollisionStay* [Unity Scripting Reference,] is called once per frame for as long as a collision keeps happening. Inside this function the program decides whether rolling or scratching takes place and calculates the corresponding velocity which stops the sound when it goes down to zero. The magnitude of this velocity is computed using the velocity vector of the object's *Rigidbody* (*rigibody.velocity.magnitude*). Unity®'s rigidbody is "the way of controlling an object's position through physics simulation" as noted in Unity®'s documentation API [Unity Scripting Reference,].

The decision whether an object is rolling or sliding on a surface is made using the **angular velocity** ($\omega = \frac{angular\ displacement}{time}$). Unity®'s rigidbody variable for angular velocity is a 3 dimensional vector, measured in radians per second [Unity Scripting Reference,]. The magnitude of it is calculated from the equation 4.4

$$\omega_{magnitude} = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{4.4}$$

where $\omega_x = \frac{\theta_x}{t}, \omega_y = \frac{\theta_y}{t}, \omega_z = \frac{\theta_z}{t}$ and $\theta_x, \theta_y, \theta_z$ the angular displacement of the x, y, z axis respectively and t the amount of time this displacement lasted. In Unity® it is referred to as (*rigidbody.angularVelocity.magnitude*) [Unity Scripting Reference,]. More specifically, when angular velocity magnitude is over 1, means that the object is rolling, otherwise it is not and providing that linear velocity is non-zero, means that the object is sliding on the surface.

To do why 1 is the border? maybe because (0.0,0.0,0.0) angular velocity gives 0.smth in magnitude (11)

To do add an Optimization section maybe? (12)

Chapter 5

Subjective Experiments

When working on a field where human perception plays a significant role, it is important to test whether your work makes sense for target users and if it solves successfully the problem that was designed for.

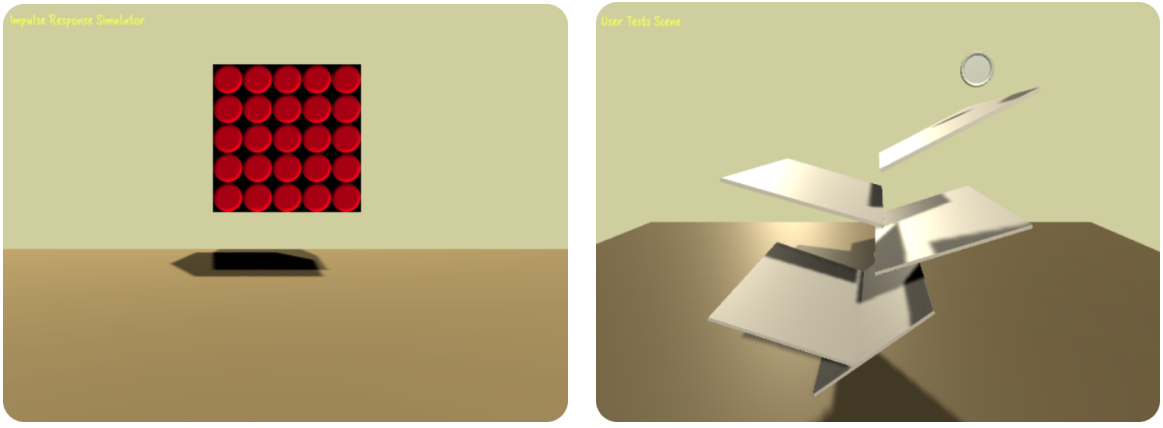
To examine the immersion of real-time produced and physics-based sounds on game players, we performed *MUSHRA*[Series, 2014] tests to people. Our aim was to answer the following questions: 1) Which of the two synthesis methods (sinusoidal and filter-based additive synthesis) is closer to reality? 2) Does physics-based synthesis make a sound more realistic and less boring? 3) Which is the range (in Q-factor values) of every material's sound? . The results of the experiments are used to verify the modal synthesis methods implemented in this thesis, choose the more qualitative one and increase our understanding of spectral characteristics in relation with the material of the object.

5.1 Preparation

For the experiments we used several audio files that we recorded inside the Unity® platform. We designed two special scenes for this purpose, because we wanted to test both the impulse response of the synthesis model and the sounds in real-life conditions -when an object is falling and colliding with other objects (figure 5.1).

In the first scene (figure 5.1(a)), we recorded the audio files for the first experiment. During the recording session, we "tagged" the cube as different objects (a process that assigns to the cube different modal data) and let it touch the ground without any bounce. We also repeated the process with an Audio Source and the recordings files, to achieve consistency of the stimuli.

In the second scene (figure 5.1(b)), we recorded the audio files for the second and third experiment. In this scene, objects fall, roll and scratch freely on rotated platforms, simulating a real room with obstacles.



(a) Impulse Response Simulator Scene. **(b)** Real-Life Conditions Scene.

Figure 5.1: The Unity® scenes designed to record the audio files used for the user tests.

Every sound in the audio files starts 1 second after participant presses play and ends half a second after no sound can be heard. They are recorded with 16-bit resolution and a sample rate of 44100Hz using Audacity® [Audacity Team,]. Stimuli was presented to the participants through a pair of *AKG K271* headphones, in a room with reduced external noise.

Table 5.1 gives details about all parts of the experiment.

Experiment	Scene	Description
1	Single impact (5.1(a))	Participants listen to impact sounds and choose which method (filter-based or sinusoidal additive synthesis) sounds closer to the real-world recording.
2	Slanting platforms (5.1(b))	Participants listen to sounds recorded using 3 different methods: real-world recording, filter-based and sinusoidal additive synthesis. They are asked to decide, for each of them, whether a sound variation per object area or a single sound per object sounds better in their opinion.
3	Slanting platforms (5.1(b))	Participants listen to sounds produced by 5 different objects (one of each material under observation: plastic, wood, ceramic, glass and metal). However, they listen to a range of sounds that derive from a continuous increment of a parameter that affects the material of the object. They are asked to choose the point where a change of the material took place.

Table 5.1: Overview of the experiments.

5.2 Stimuli

We performed three different tests. In the first test, the participant listens to 44 impulse responses, corresponding to different areas of the eleven materials. Each trial of the test includes a reference sound, which is the recording of the actual sound produced by the physical object and the two different synthesized sounds, corresponding to the two examined methods. He is then asked to choose which of the two synthesized sounds is closer to the reference and whether it is very or less close. The goal of this experiment was to gather information about the quality of the two methods and addresses the best of the two.

In the second test, the stimuli consists of 33 trials that contain sounds produced by falling objects. Each trial includes one sound that is produced when object is split into "sound areas" and each area produces a different sound and a second sound that is produced when every point of the object makes the same sound. This single sound was chosen to be the one produced from the area of each object where the recording taken (section 4.1) was the closest to the real sound in our opinion. Participants were asked which of the two they preferred and how much in a scale from "A is the same as B" to "A is much better than B". This test provided us with the immersion results of sound variation within one object.

The stimuli of the third test are sounds coming from five different objects, one of each material under testing (plastic jug, wooden mortar, ceramic plate, glass bottle and metallic cooking pot). For each object, we use 10 different sounds per synthesis method. Each trial includes 10 sounds that correspond to the same event, but with a small variation on the Q-factor for every sound. More specifically, starting from a value of 1000, we increased the Q-factor by 200 up to 4600, removed some sounds that were too similar with others to decrease the size of the test and provided participants with the rest. They were asked to choose the sounds where, in their opinion, a change in the material happened. We performed this test to validate the chosen values for the Q-factor.

In the two first experiments, both the sequence of trials and the conditions (which method is A and which is B) are randomized. However, in experiment number three, we wanted to keep an increasing habit on the Q-factor, otherwise it would not make sense. Hence we randomized only the sequence of the trials.

5.3 Participants

To do number of participants, age, gender, normal hearing, job (13) The participants are three adult volunteers. Zero women and three men, aged 25 to 30. They stated normal hearing. They have different backgrounds, namely a Digital Signal Processing (DSP) engineer, a mathematician and a movie director.

5.4 Test Results

The participants' responses were processed using python programming language and the following results were acquired.

5.4.1 1st Experiment

Results of the 1st experiment are shown in figure 5.2. We divided the results into smaller sub-graphs, one for each of the eleven objects, to examine the participants' preference in the synthesis methods per object. Particularly, their opinion on which of the two resembles more the original recording of the real-world object (used as reference).

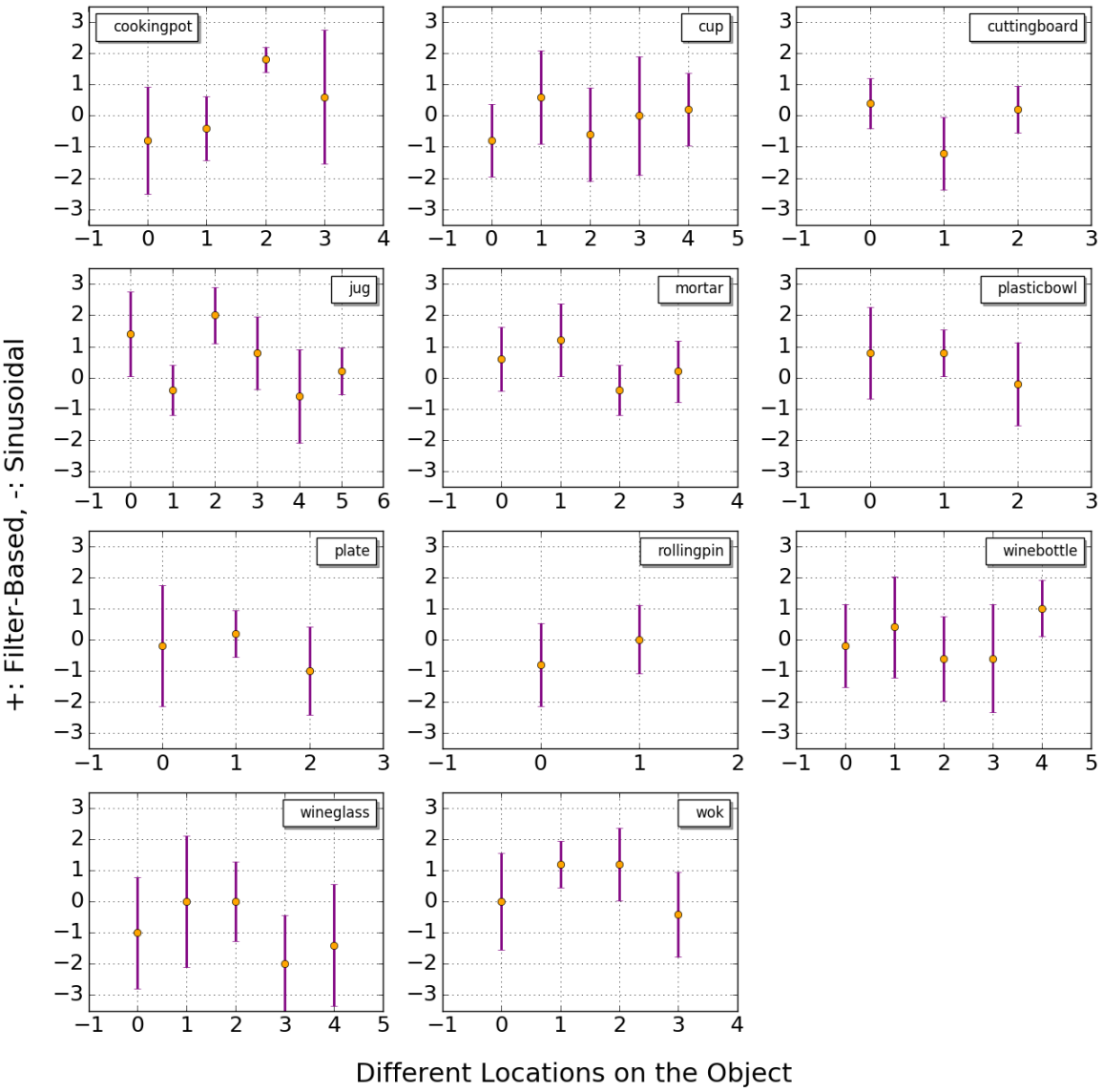


Figure 5.2: The mean values and standard deviations per location for all objects. Positive values give a preference to the filter-based method, while negative ones give preference to the sinusoidal method. Participants choises were 0: both sound the same, 1/-1: the chosen is slightly better, 2/-2: the chosen is better and 3/-3: the chosen is much better.

Participants were asked to move a slider between the integer values -3 and 3. Positive values correspond to a preference in the filter-based method, negative in the sinusoidal method and zero values means that both methods sound the same. Possible answers are displayed in table 5.2. Every different tick of the x-axis corresponds to a different "sound area" of the object. From left to right the areas go from the top to the bottom.

Slider value	Method	Amount of Preference
3	Filter-based	Much better
2	Filter-based	Better
1	Filter-based	Slightly better
0	Both sound the same	
−1	Sinusoidal	Slightly better
−2	Sinusoidal	Better
−3	Sinusoidal	Much better

Table 5.2: Possible answers for the 1st experiment.

With this test we wanted to answer the question of which of the two synthesis methods - filter based or sinusoidal additive synthesis - used in this thesis is more accurate. From figure 5.2 we can see that one universal answer for all objects does not exist. For example, we can see a tendency for the filter based method for the plastic (jug and plastic bowl) and the metal (cooking pot and wok) objects, but there is no distinct result for the rest of the objects.

To do check if those still are valid after adding more participants. (14)

5.4.2 2nd experiment

For the second experiment we also divided the results into sub-graphs per object. They are presented in figure 5.3. This experiment examines the participants' preference between a sound variation on impact sounds depending on the hit point and a single impact sound per object.

Participants were asked to use a slider similar to the previous test, with possible answers shown in the table 5.3. Positive values correspond to preference in sound variation, while negative ones correspond to preference in single sound per object. The three different ticks on the x-axis correspond to the actual recording, the filter-based and the sinusoidal method respectively.

Slider value	Sound Variation	Amount of Preference
3	Yes	Much better
2	Yes	Better
1	Yes	Slightly better
0	Both sound the same	
−1	No	Slightly better
−2	No	Better
−3	No	Much better

Table 5.3: Possible answers for the 2nd experiment.

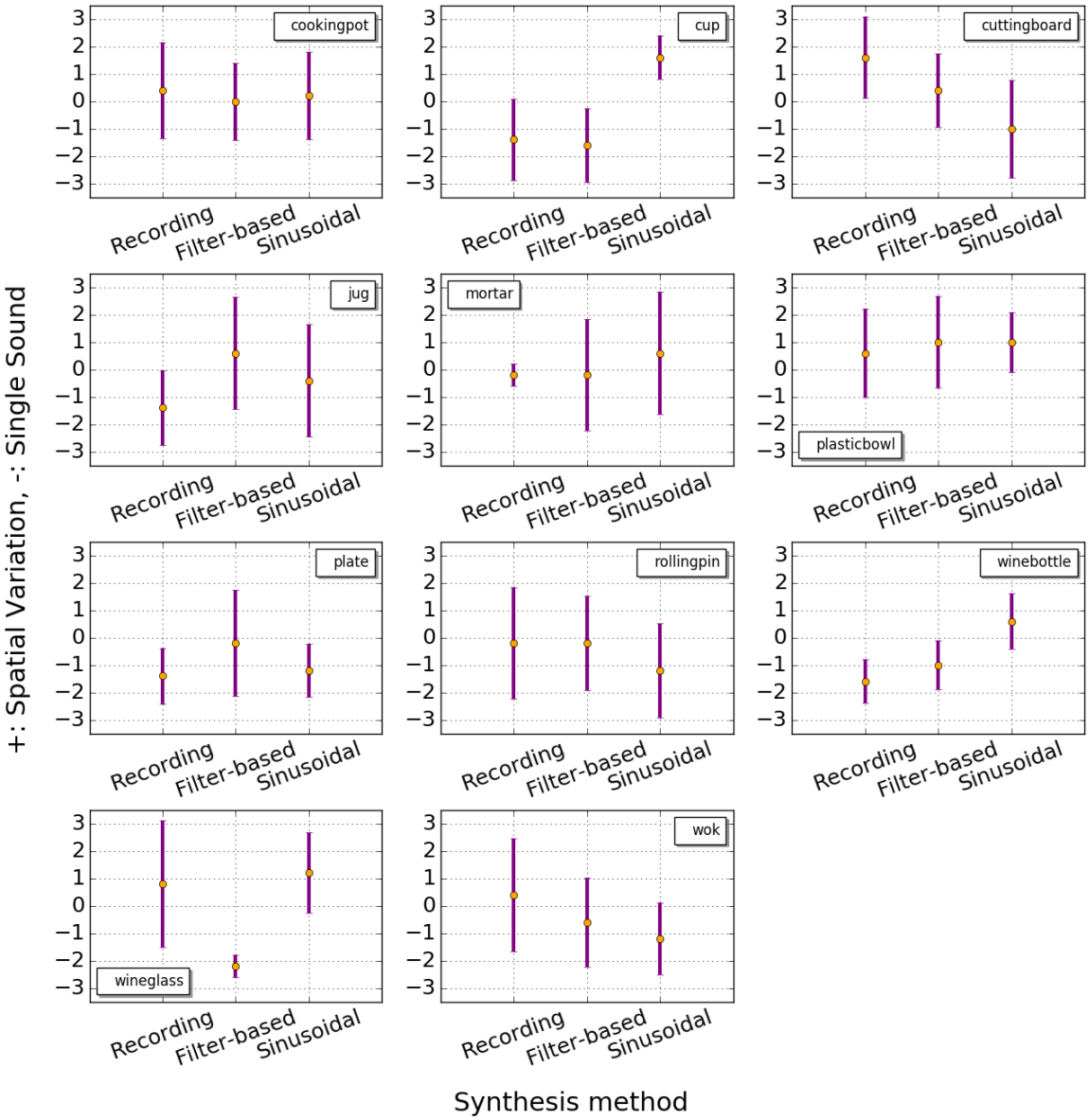


Figure 5.3: The mean values and standard deviations per location for all objects. Positive values give a preference to the filter-based method, while negative ones give preference to the sinusoidal method. Participants choises were 0: both sound the same, 1/-1: the chosen is slightly better, 2/-2: the chosen is better and 3/-3: the chosen is much better.

The aim of this experiment is to prove whether sound variation in game sound effects is desirable and makes a difference. By choosing the positive values, participants would prove that having spatial variation in impact sounds benefits the immersion of the player.
To do decribe the graph (15)

5.4.3 3rd experiment

To do check if it comes out that we cannot transform a given object's material just with damping (maybe we need to change the spectr

Figure 5.4 shows the results of the third experiment. This one was held to validate the chosen values of the Q-factor that correspond to each material. For the purpose of this thesis,

we chose one value per object that sounded closer to the specified material, in our opinion. Those selected values are shown in table 4.1.

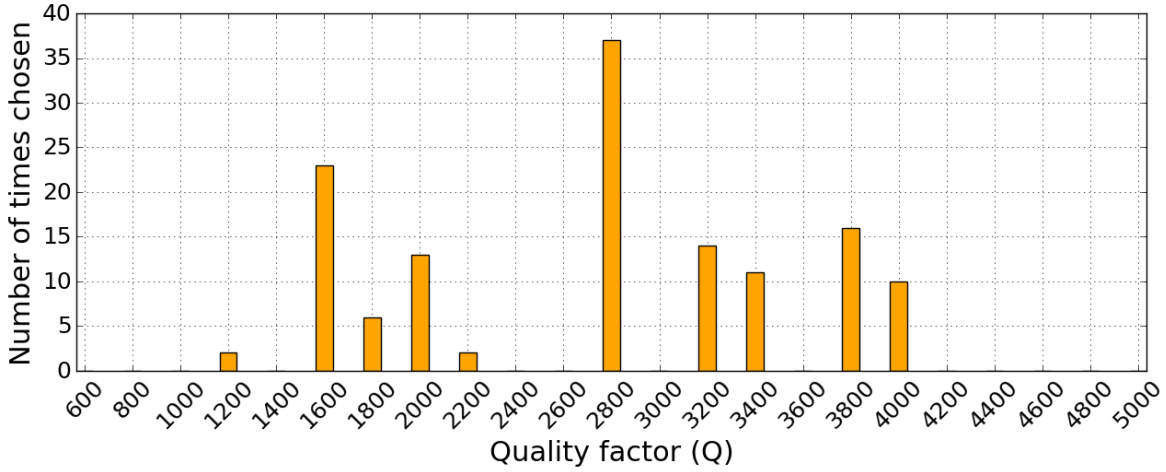


Figure 5.4: The chosen values of the quality factor that correspond to material change of the same object.

By studying figure 5.4, we can see that the values chosen from the experiment participants do not vary a lot from ours (see table 4.1).

Chapter 6

Analysis

6.1 Results

6.1.1 Which Synthesis Method Is Better?

6.1.2 Evaluation

Overall, the work on this thesis proved successful. We managed to replace pre-recorded sound effects with procedural audio. In addition, we made this audio physics-based and influenced by the context of the impact. One big challenge that was achieved is the ability to choose different material for every object, which extends our tool for imaginary game scenarios. **To do** talk about why metals sound bad (17)

An important drawback is the difficulty to extend the tool with new objects, but we include a detailed guide of how to do it in appendix C. However, we managed to make it easy-to-use with the already existing objects, by implementing high level controllers that are easily understandable from a designer. Those controllers, that are explained in section 4.4, are neither *weak* nor very *strong*. As explained in [Jaffe, 1995], weak parameters affect the result so little that there is a possibility that the designer adjusts it to a random, undesirable value. On the other hand, when even a minuscule tweaking of a very strong parameter induces a big change, the designer will probably find it difficult to choose the value he wants.

We tried to keep the UI similar to the Unity®'s UI, so game developers can use the tool without further need of sound design knowledge. It is as easy to assign procedurally generated, physics based sound to objects as if you assign a texture to them. Therefore, with this tool we are not only targeting sound designers who desire more realistic sound effects, but also game developers with knowledge of the Unity® Editor and no further sound design knowledge.

6.1.2.1 Bugs

- you have to press apply on prefabs
- the procedure of putting the freq and ampl data in
- it is object specific (not a bug actually, more like a drowback)
- lack of acoustical richness that might characterize synthetic signals [Giordano and McAdams, 2006]
- some objects are not modal

6.1.3 What did we do new?

Combined all three major sounds inside a game or an application (impact, rolling, scratching) and made them available and ready-to-use to developers.

Bridged university with industry by implementing a tool inside the game engine.

6.2 Discussion

6.2.1 Types of games that it can be used

This tool can be used for development of all sorts of games that include object interaction. They can vary from indoor AR applications to open world environment games running in consoles.

6.2.2 Why our work can be used in VR/AR?

Virtual and augmented reality are becoming more and more widespread technologies. There are multiple applications where they can prove to be very useful both in everyday life and entertainment. Focusing on AR, our thesis' goal is to develop a framework for sounds produced by object interactions. We designed the sounds to be realistic and event-based, so they can adapt into an AR environment where a big portion of the objects are indeed real.

6.2.3 CPU demands

Synthesizing sounds for applications real-time instead of using a mass of prerecorded clips is a good solution to the storage problem of nowadays portable and limited in memory devices. On the other hand, it is challenging and requires a lot of CPU performance when usually audio is restricted to a low limit and most of it is given to graphics, physics and artificial intelligence (AI) [Lloyd et al., 2011].

In our tool, however, when profiling a demonstration of a wine bottle rolling down a number of oblique platforms (seen in figure 5.1(b)), using the *Profiler Window* of Unity® Editor we can see that except for the initialization at the beginning where scripts consume some CPU power, the rest of the demonstration remains stable in performance and around $100 fps$ (figure 6.1). This performance test was held on a laptop with 4 Intel® Core™ i7-6700HQ CPUs running at 2.60GHz, each with 2 hardware threads.

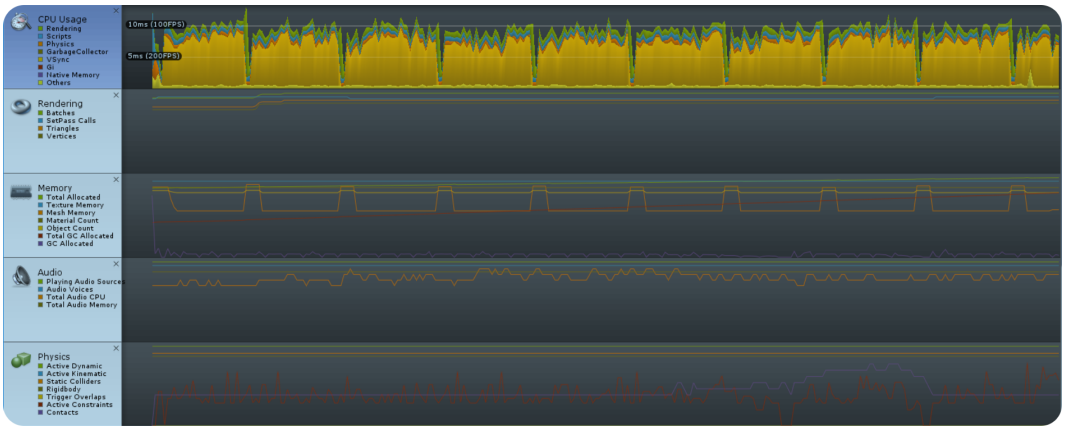


Figure 6.1: The profiler view of Unity® Editor when a wine bottle rolls down a number of platforms.

When testing the limits of the tool, we found out that in Filter-based method a total of 16 objects can be active at the same time before audio gets distorted. The corresponding number for the Sinusoidal method is 12, since more calculations take place. Those numbers were obtained on the laptop mentioned above and using the *Unity® Audio Profiler window* and represent the amount of **"Playing Audio Sources"** without the **"Total Audio CPU"** exceeding 100% (see figure 6.2).

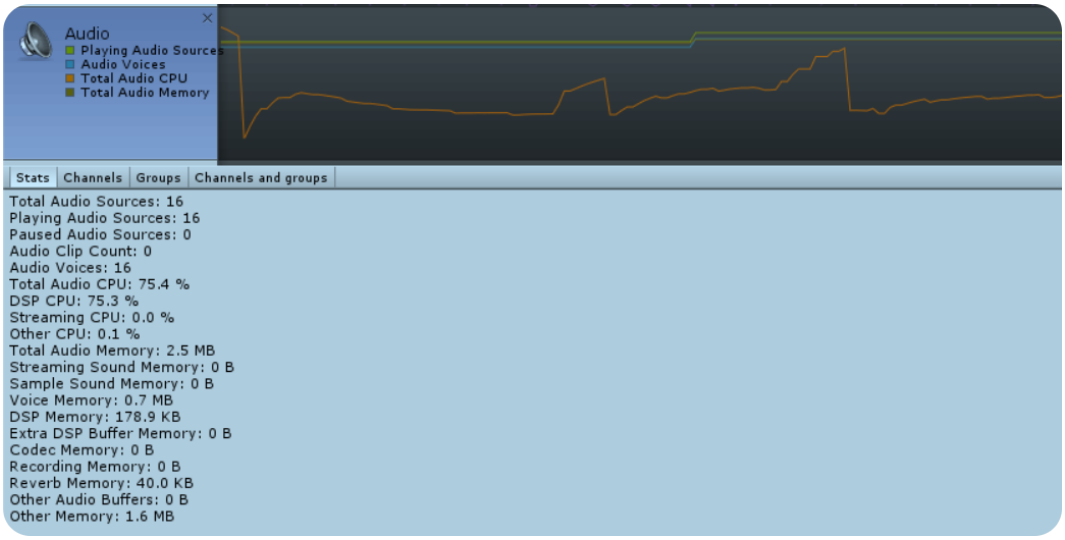


Figure 6.2: The audio profiler view of Unity® Editor when a total of 16 objects are enabled in the scene using the filter-based method for audio synthesis.

6.2.4 How can we improve our work?

- take into account the environment (reverberation etc)
- make objects destructible
- randomize initial phase so peaks of the sine wave don't line up and distort the sound (saturation)
- spatialize at the same time with the synthesis
- radiation and propagation (Interactive Acoustic Transfer Approximation for Modal Sound has a method) Even though the recordings used to extract the modal data include information about the radiation and propagation of the sound in the environment, this thesis did not examine it.
- use only one recording and interpolate for all areas
- level of detail, using a masking threshold of e.g. 15dB as used in MEASUREMENTS OF PERCEPTUAL QUALITY OF CONTACT SOUND MODELS

6.2.5 Pros and Cons of Procedural Game Audio

pros
more adaptive (flexible/dynamic).
physics based.
possibility of alternations without extra memory.

less memory space.
no need for time consuming sound library development.
easy to create new sounds.
better for mobile platforms.

cons
usually bad quality (not realistic - you can tell it's synthesized).
more processing power.
additional code requirements.
not everyone is familiar with this sound design approach.
not every sound is easy to be produce procedurally.

Chapter 7

Conclusion

This is the conclusion
4-5 paragraph approx

Appendix A

Pure Data Patches

To do make the patches more readable and replace the pics (18)

A.1 Filter Based Additive Synthesis

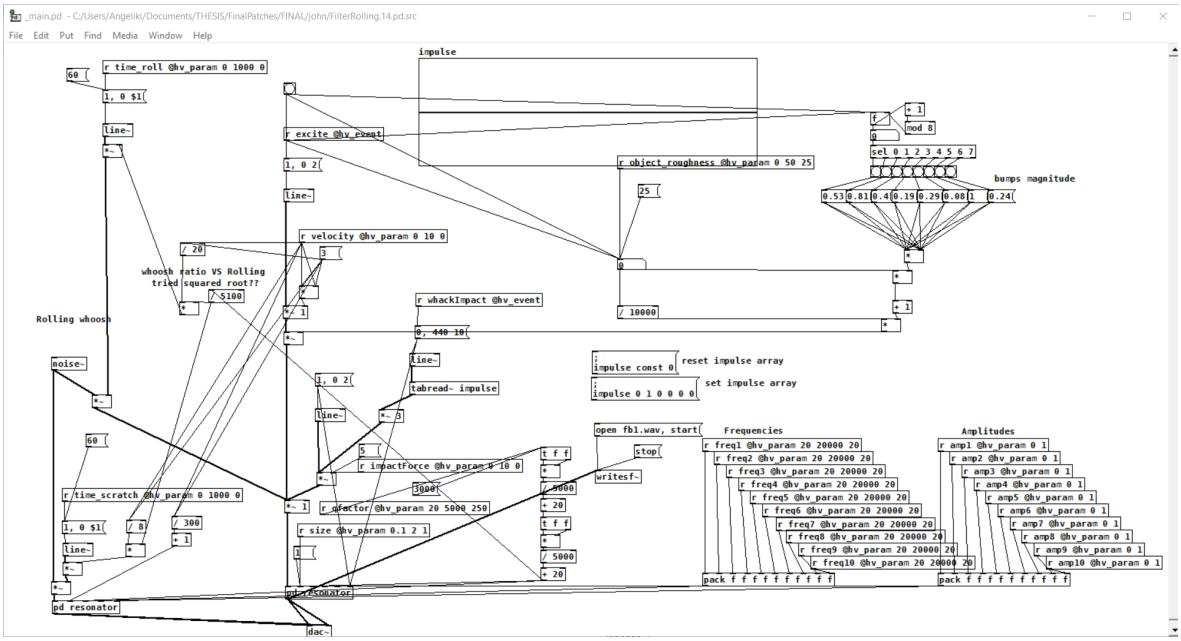


Figure A.1: The main Pure Data patch for the filter-based additive synthesis.

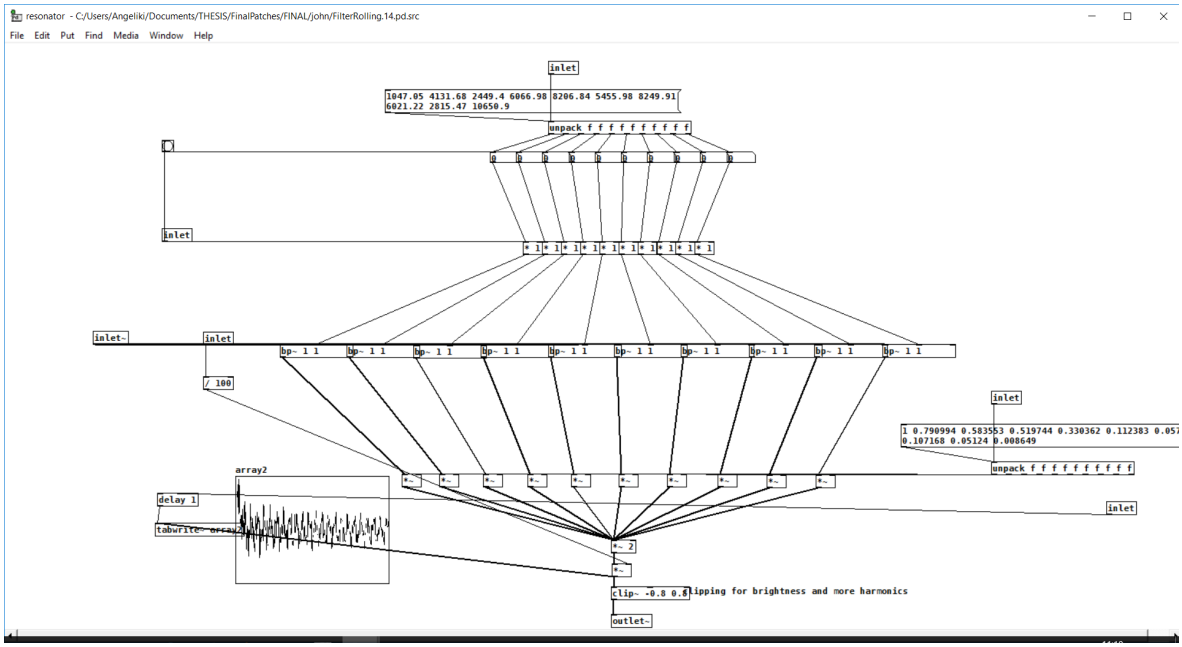


Figure A.2: The resonator Pure Data patch for the filter-based additive synthesis.

A.2 Sinusoidal Additive Synthesis

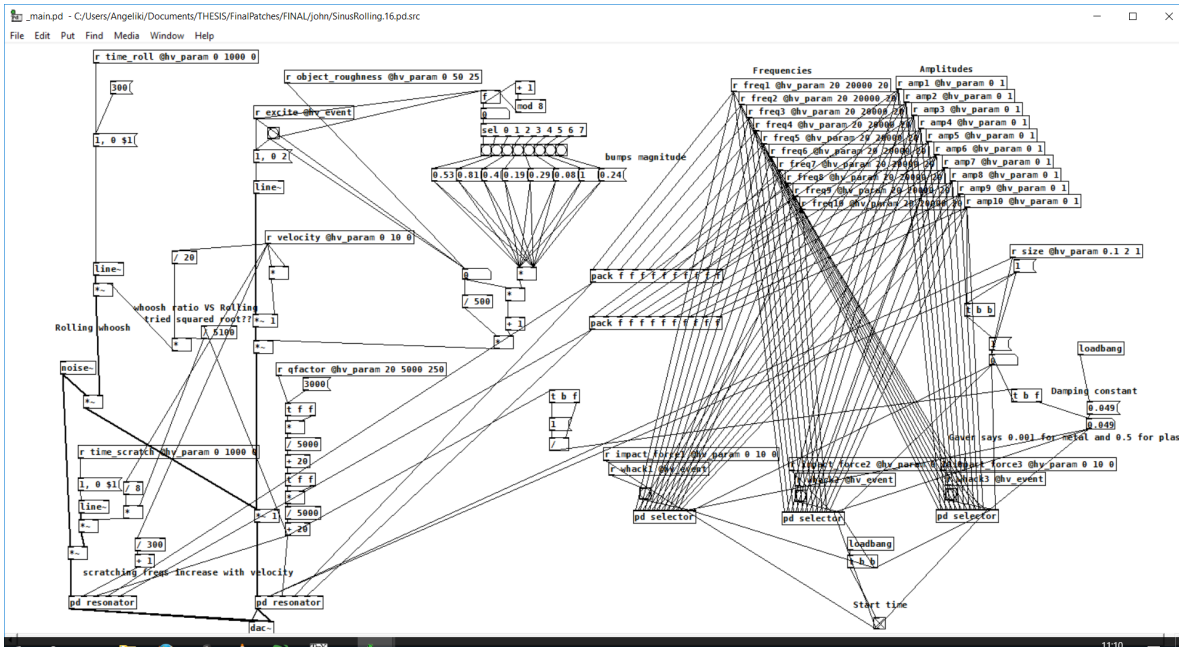


Figure A.3: The main Pure Data patch for the sinusoidal additive synthesis.

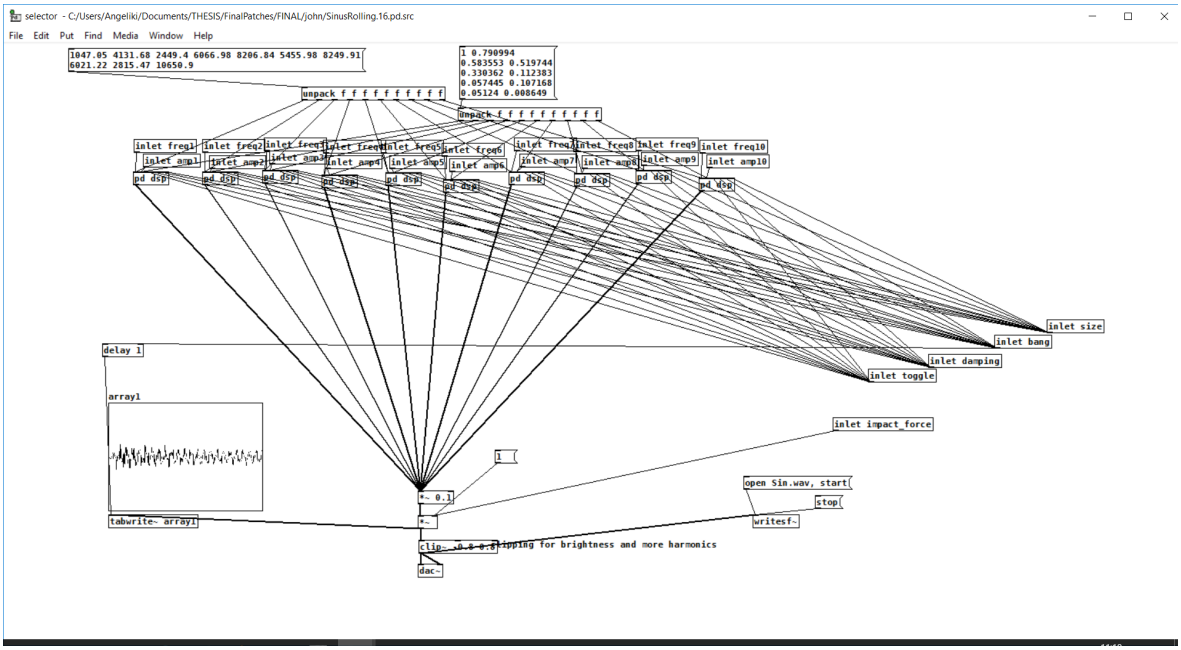


Figure A.4: The selector Pure Data patch for the filter-based additive synthesis.

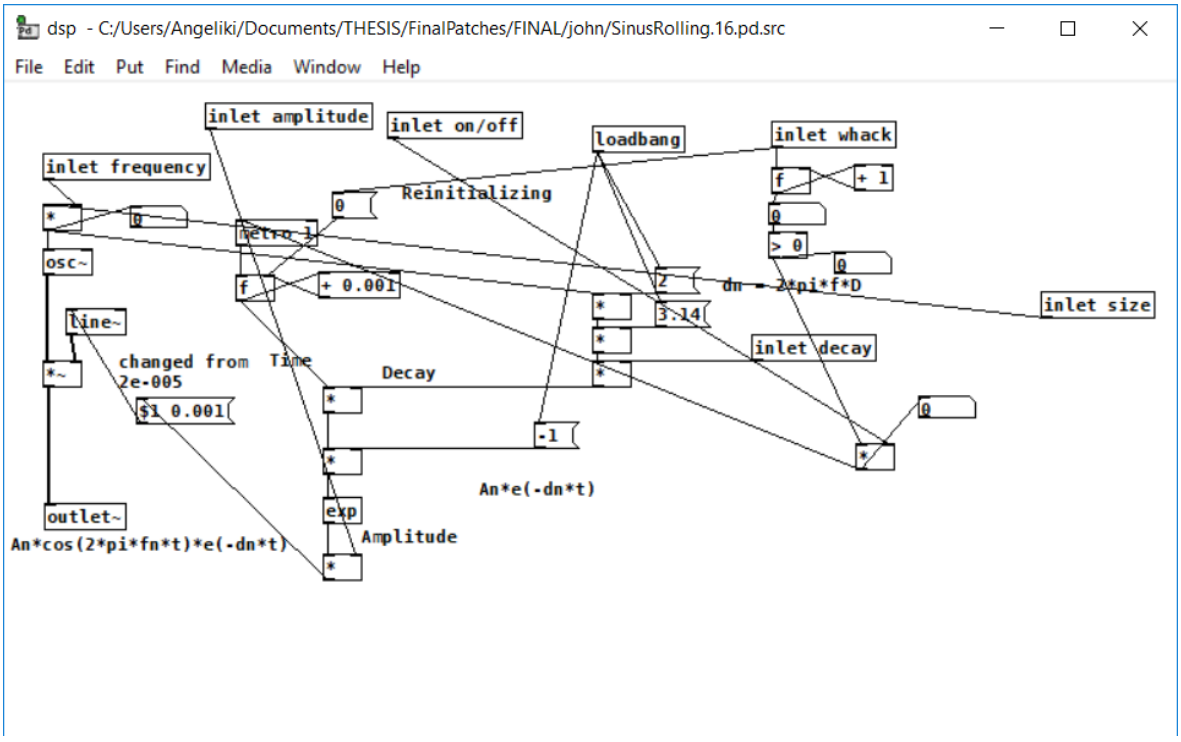
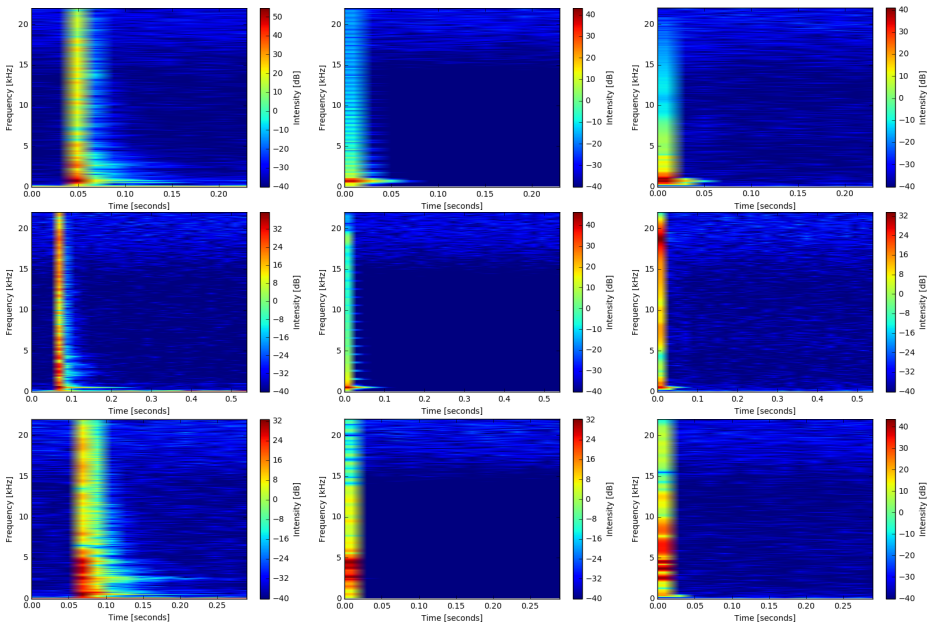


Figure A.5: The dsp Pure Data patch for the filter-based additive synthesis.

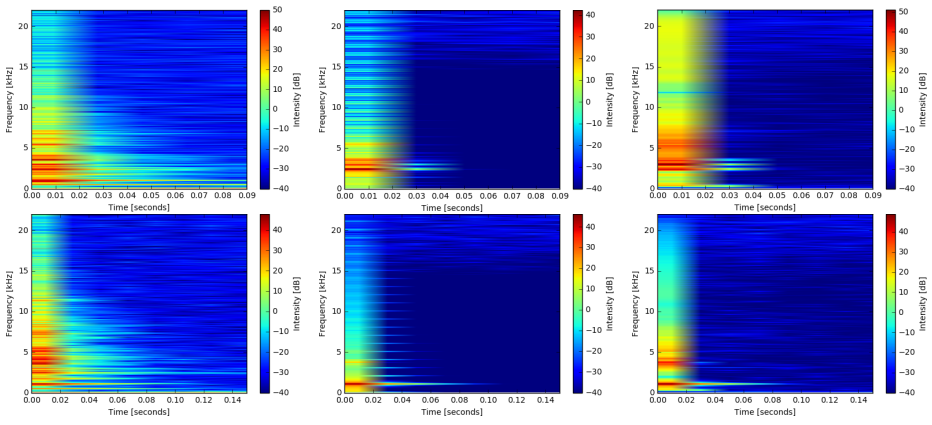
Appendix B

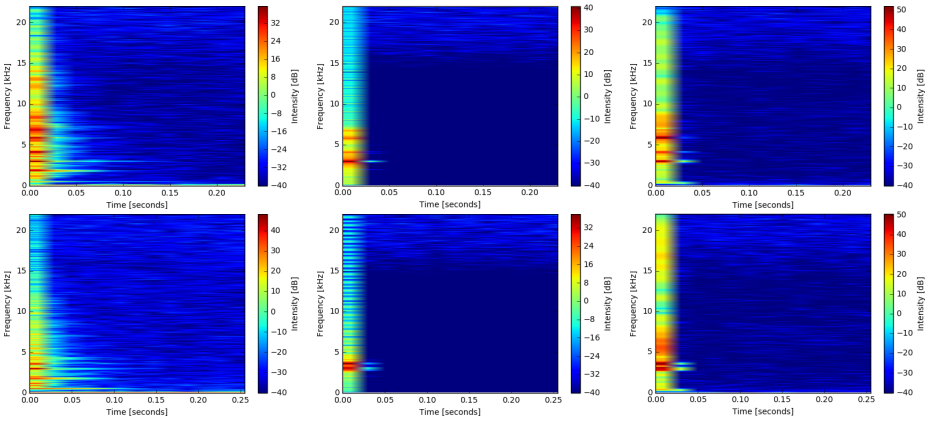
Spectrograms

Plastic bowl

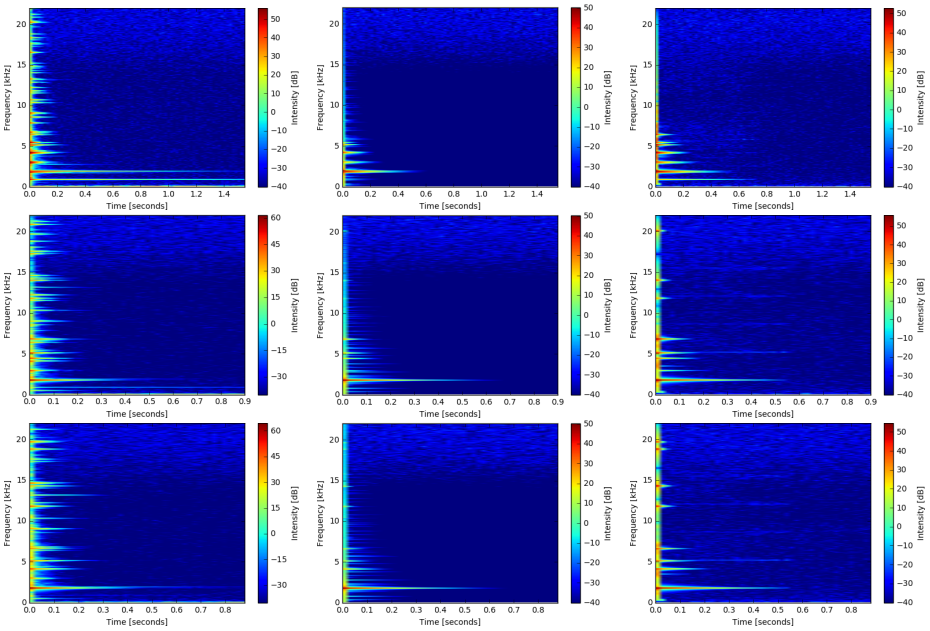


Wooden mortar

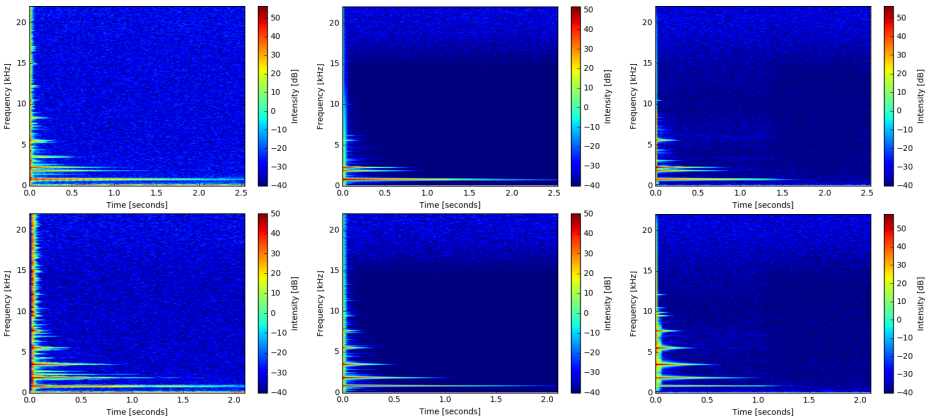


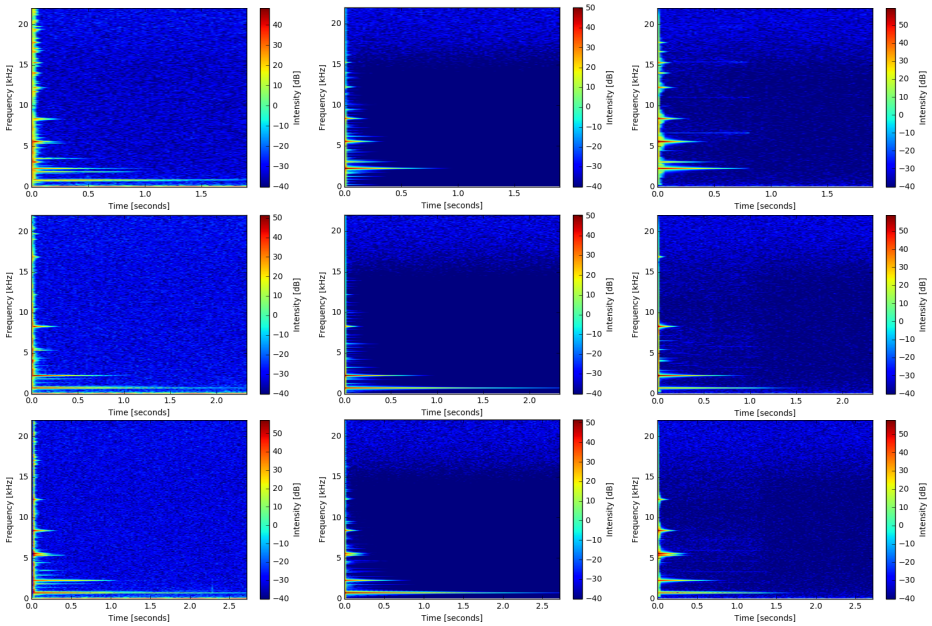


Ceramic plate

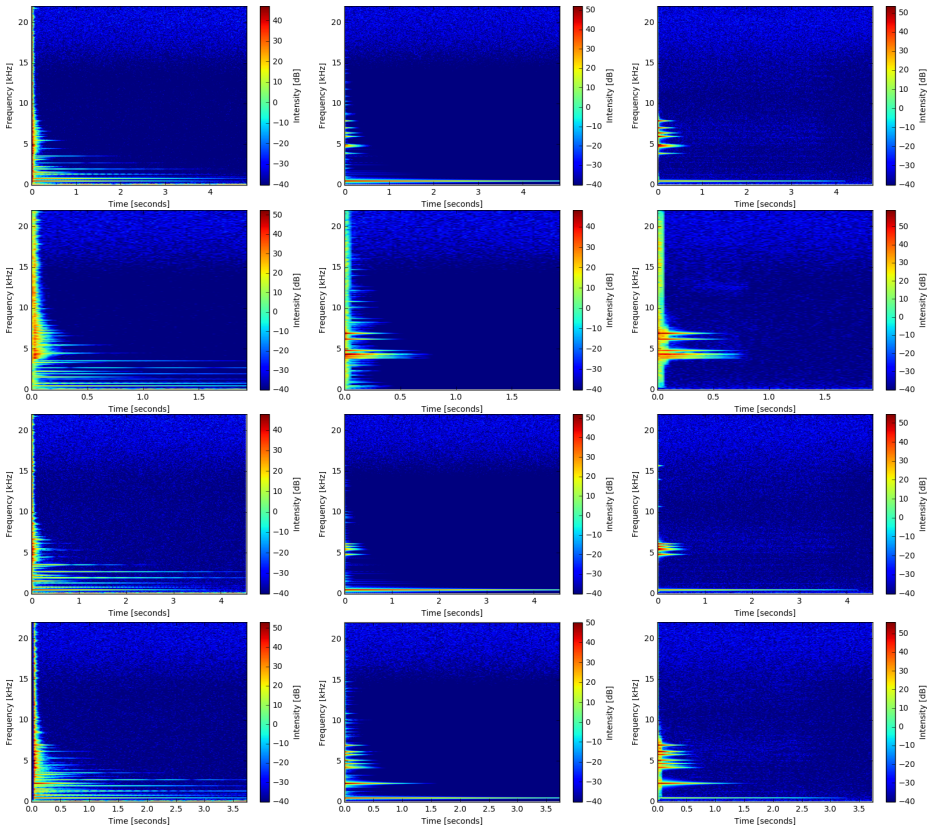


Wine glass





Metallic wok



Appendix C

User Guide to our product

- Record impact sounds
- Put audio files into the data extraction algorithm
- Take the output data and put them into unity
- Make an FBX® model of your object with the same dimensions and put it into a Unity® scene as game object
- Assign the audio manager to the game object
- Tag the game object with the corresponding tag
- Adjust the material, size and object roughness sliders

Bibliography

[Aramaki et al., 2011] Aramaki, M., Besson, M., Kronland-Martinet, R., and Ystad, S. (2011). Controlling the perceived material in an impact sound synthesizer. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(2):301–314.

[Aramaki et al., 2009] Aramaki, M., Gondre, C., Kronland-Martinet, R., Voinier, T., and Ystad, S. (2009). Thinking the sounds: an intuitive control of an impact sound synthesizer. Georgia Institute of Technology.

[AspenCore, Inc.,] AspenCore, Inc. Passive band pass filter. http://www.electronics-tutorials.ws/filter/filter_4.html. [Online; accessed 28-April-2017].

[Audacity Team,] Audacity Team. Audacity. http://www.audacityteam.org/. [Online; accessed 12-June-2017].

[Audiokinetic Inc.,] Audiokinetic Inc. Wwise®. https://www.audiokinetic.com/products/wwise/. [Online; accessed 14-June-2017].

[Autodesk, Inc., a] Autodesk, Inc. Fbx file format. https://www.autodesk.com/products/fbx/overview. [Online; accessed 2-May-2017].

[Autodesk, Inc., b] Autodesk, Inc. Maya Autodesk (version 2016). https://www.autodesk.com/products/maya/overview. [Online; accessed 20-April-2017].

[Bilbao, 2009] Bilbao, S. (2009). *Numerical sound synthesis: finite difference schemes and simulation in musical acoustics*. John Wiley & Sons.

[Bonneel et al., 2008] Bonneel, N., Drettakis, G., Tsingos, N., Viaud-Delmon, I., and James, D. (2008). Fast modal sounds with scalable frequency-domain synthesis. In *ACM Transactions on Graphics (TOG)*, volume 27, page 24. ACM.

[ChucK Documentation,] ChucK Documentation. A. kapur, a. tindale, p. davidson, a. misra, r. fiebrink, g. wang. http://chuck.cs.princeton.edu/doc/. [Online; accessed 31-May-2017].

[Cook, 2002] Cook, P. R. (2002). *Real Sound Synthesis for Interactive Applications*. A. K. Peters, Ltd., Natick, MA, USA.

[Cory et al., 2006] Cory, D., Hutchinson, I., and Chaniotakis, M. (Spring 2006). 6.071j Introduction to Electronics, Signals, and Measurement. Massachusetts Institute of Technology: Mit OpenCourseWare. https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

[Crowell, 2003] Crowell, B. (2003). *Conservation laws*, volume 2. Light and Matter.

[Director-O'Brien, 2001] Director-O'Brien, J. F. (2001). Synthesizing sounds from physically based motion. In *ACM SIGGRAPH 2001 video review on Animation theater program*, page 59. ACM.

[Doel, 1998] Doel, C. P. v. d. (1998). *Sound synthesis for virtual reality and computer games*. PhD thesis, University of British Columbia.

[Enzien Audio, Ltd.,] Enzien Audio, Ltd. Heavy (version r2017.02). https://enzienaudio.com/. [Online; accessed 2-May-2017].

[Farnell, 2007] Farnell, A. (2007). An introduction to procedural audio and its application in computer games. In *Audio mostly conference*, pages 1–31.

[Farnell, 2010] Farnell, A. (2010). *Designing sound*. Mit Press.

[Gaver, 1993a] Gaver, W. W. (1993a). How do we hear in the world? explorations in ecological acoustics. *Ecological psychology*, 5(4):285–313.

[Gaver, 1993b] Gaver, W. W. (1993b). What in the world do we hear?: An ecological approach to auditory event perception. *Ecological psychology*, 5(1):1–29.

[Giordano and McAdams, 2006] Giordano, B. L. and McAdams, S. (2006). Material identification of real impact sounds: Effects of size variation in steel, glass, wood, and plexiglass plates. *The Journal of the Acoustical Society of America*, 119(2):1171–1181.

[Houben et al., 1999] Houben, M., Hermes, D., and Kohlrausch, A. (1999). Auditory perception of the size and velocity of rolling balls.

[Jaffe, 1995] Jaffe, D. A. (1995). Ten criteria for evaluating synthesis techniques. *Computer Music Journal*, 19(1):76–87.

[James et al., 2006] James, D. L., Barbič, J., and Pai, D. K. (2006). Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 987–995. ACM.

[Klatzky et al., 2000] Klatzky, R. L., Pai, D. K., and Krotkov, E. P. (2000). Perception of material from contact sounds. *Presence: Teleoperators and Virtual Environments*, 9(4):399–410.

[Larsson et al., 2002] Larsson, P., Vastfjall, D., and Kleiner, M. (2002). Better presence and performance in virtual environments by improved binaural sound rendering. In *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio*. Audio Engineering Society.

[Lloyd et al., 2011] Lloyd, D. B., Raghuvanshi, N., and Govindaraju, N. K. (2011). Sound synthesis for impact sounds in video games. In *Symposium on Interactive 3D Graphics and Games*, pages PAGE–7. ACM.

[Mehta and Mehta,] Mehta, V. and Mehta, R. *S. Chand's Principles Of Physics For XI*. S. Chand Publishing.

[O'Brien et al., 2002] O'Brien, J. F., Shen, C., and Gatchalian, C. M. (2002). Synthesizing sounds from rigid-body simulations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 175–181. ACM.

[P. Cook,] P. Cook, J. S. Physics-based sound synthesis for games and interactive systems. https://www.kadenze.com/courses/physics-based-sound-synthesis-for-games-and-interactive-systems-iv. [Online; accessed 2-May-2017].

[Pai et al., 2001] Pai, D. K., Doel, K. v. d., James, D. L., Lang, J., Lloyd, J. E., Richmond, J. L., and Yau, S. H. (2001). Scanning physical interaction behavior of 3d objects. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 87–96. ACM.

[PHYSIS,] PHYSIS. Physically informed and semantically controllable interactive sound synthesis. https://sites.google.com/site/physisproject/home/. [Online; accessed 14-June-2017].

[Picard-Limpens, 2009] Picard-Limpens, C. (2009). *Expressive sound synthesis for animation*. PhD thesis, Université Nice Sophia Antipolis.

[Pruvost et al., 2015] Pruvost, L., Scherrer, B., Aramaki, M., Ystad, S., and Kronland-Martinet, R. (2015). Perception-based interactive sound synthesis of morphing solids' interactions. In *SIGGRAPH Asia 2015 Technical Briefs*, page 17. ACM.

[Puckette,] Puckette, M. Pure data programming language. http://puredata.info/. [Online; accessed 2-May-2017].

[Raghuvanshi and Lin, 2006] Raghuvanshi, N. and Lin, M. C. (2006). Interactive sound synthesis for large scale environments. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 101–108. ACM.

[Rath, 2003] Rath, M. (2003). An expressive real-time sound model of rolling. In *Proceedings of the 6th" International Conference on Digital Audio Effects"(DAFx-03), London, United Kingdom*.

[Ren et al., 2010] Ren, Z., Yeh, H., and Lin, M. C. (2010). Synthesizing contact sounds between textured models. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 139–146. IEEE.

[Ren et al., 2013] Ren, Z., Yeh, H., and Lin, M. C. (2013). Example-guided physically based modal sound synthesis. *ACM Transactions on Graphics (TOG)*, 32(1):1.

[Sears and Zemansky, 1964] Sears, F. W. and Zemansky, M. W. (1964). University physics.

[Series, 2014] Series, B. (2014). Method for the subjective assessment of intermediate quality level of audio systems. *International Telecommunication Union Radiocommunication Assembly*.

[Smith III, 2011] Smith III, J. O. (2011). *Spectral audio signal processing*. W3K publishing.

[Takala and Hahn, 1992] Takala, T. and Hahn, J. (1992). Sound rendering. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 211–220. ACM.

[Unity Scripting Reference,] Unity Scripting Reference. Unity technologies. https://docs.unity3d.com/ScriptReference/index.html. [Online; accessed 26-May-2017].

[Unity Technologies,] Unity Technologies. Unity® software (version 5.5.1f1 personal). https://unity3d.com/. [Online; accessed 2-May-2017].

[Van Den Doel et al., 2001] Van Den Doel, K., Kry, P. G., and Pai, D. K. (2001). Foleyautomatic: physically-based sound effects for interactive simulation and animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 537–544. ACM.

[Van den Doel and Pai, 1998] Van den Doel, K. and Pai, D. K. (1998). The sounds of physical shapes. *Presence: Teleoperators and Virtual Environments*, 7(4):382–395.

[Van Den Doel and Pai, 2003] Van Den Doel, K. and Pai, D. K. (2003). Modal synthesis for vibrating objects. *Audio Anectodes. AK Peter, Natick, MA*, pages 1–8.

[Verron, 2010] Verron, C. (2010). *Synthese immersive de sons d'environnement.* PhD thesis, Aix-Marseille 1.

[Wang,] Wang, G. Chuck programming language. http://chuck.cs.princeton.edu/. [Online; accessed 2-May-2017].

[Wildes and Richards, 1988] Wildes, R. P. and Richards, W. A. (1988). Recovering material properties from sound. *Natural computation*, pages 356–363.

To do...

- ☐ 1 (p. v): name of the recording guy
- ☐ 2 (p. 2): write stuff about 3d sound??
- ☐ 3 (p. 11): explain HRTF
- ☐ 4 (p. 13): replace figure with a better one
- ☐ 5 (p. 15): should we explain why we used FFT?
- ☐ 6 (p. 16): develop more?
- ☐ 7 (p. 17): keep it or not?
- ☐ 8 (p. 28): fix script names and take screenshots again
- ☐ 9 (p. 31): reference the description of the pitch multiplier
- ☐ 10 (p. 33): see if it is necessary to describe the 3 different DACs here or just in the patch
- ☐ 11 (p. 33): why 1 is the border? maybe because (0.0,0.0,0.0) angular velocity gives 0.smth in magnitude
- ☐ 12 (p. 33): add an Optimization section maybe?
- ☐ 13 (p. 37): number of participants, age, gender, normal hearing, job
- ☐ 14 (p. 39): check if those still are valid after adding more participants.
- ☐ 15 (p. 40): decribe the graph
- ☐ 16 (p. 40): check if it comes out that we cannot transform a given object's material just with damping (maybe we need to change the spectral content)
- ☐ 17 (p. 43): talk about why metals sound bad
- ☐ 18 (p. 49): make the patches more readable and replace the pics