Procedural 3D Audio for AR Applications

Angeliki Skandalou
John Jeremy Ireland

Supervisor:
Michael Rose

Kongens Lyngby 2017

Abstract

Audio that is consistent and in synchrony with the visual graphics in a game is of crucial importance. Prerecorded sounds lack the flexibility needed in a virtual environment. Procedural audio, on the other hand, generates highly dynamic audio that can handle unpredictable events, while solving the storage problem of portable devices.

Typical sound interactions within a game consist of impact, rolling and scratching sounds. Two different ways of synthesizing these sounds are examined in this thesis, both deriving from modal synthesis. After extracting the modal data from real world recordings, they are fed either into a number of damped oscillators or a number of band-pass filters. To achieve sound variation along the object's surface, several recordings that correspond to a different "sound area" on the object are used.

A UI consistent with Unity®'s system is implemented to achieve consistency and offer control over the synthesized sounds. Sound designers are provided with high level intuitive parameters which form a tool that generates physics-based procedural audio.

Based on perceptual tests, it has been noted that a single parameter enables to identify changes in material category and that sound variation along an object's surface is desirable and preferred over one single sound.

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 recommendations helped us to make the most out of this project and make it 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 Jørgen Rasmussen from the Acoustic Technology group at the Department of Electrical Engineering of DTU for giving us access to the anechoic chamber and providing the equipment to perform the necessary 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 Technical University of Denmark.

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.

SDK Software Development Kit.

UI User Interface.

VBAP Vector Base Amplitude Panning.

VR Virtual Reality.

WFS Wave Field Synthesis.

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, etc). 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, *Sinusoidal Additive Synthesis* and *Filter-based Modal Synthesis*, 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, synthesis techniques and 3D sound. 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 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.

Supplementary video: https://vimeo.com/222842969

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, Vector Base Amplitude Panning (VBAP) and Wave Field Synthesis have been developed to simulate 3D sounds on loudspeakers. Binaural techniques use Head-Related Transfer Function (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 Software Development Kit (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 "everyday 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 and 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 physics-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 VR and 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 for the latter, 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 or scratching/sliding on it. They can be modeled as successive impact forces, where one adds on top of the other. The distinction between rolling and scratching depends on the shape of the object and on the roughness of both surfaces.

This thesis focuses 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 reach other objects in the environment and get reflected and absorbed 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 only object-related interactions are examined and not sound propagation, below is explained in depth how each of the object's physical attributes affect sound.

2.3.1 Acoustic Effects of Sound Attributes

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

Furthermore, the configuration of the object also affects its vibration. The size of it determines how high or low pitched the produced sound will be. That is to say, 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 gets 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, certain modal parameters that are characteristic of the vibrating solid object and that will be fed as input need to be extracted. 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 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 spring-mass 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's vertices into masses and its 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 Digital Signal Processing (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, its outer surfaces vibrate, as explained in section 2.3.1. The object vibrates at specific frequencies, which are called resonant modes, and they 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 a sum of damped harmonic oscillators each corresponding to a modal frequency through additive synthesis [Bilbao, 2009]. This is the 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.

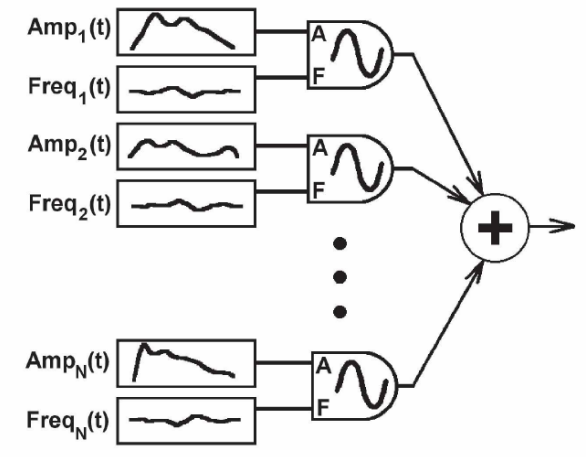


Figure 2.1: Sinusoidal Additive Synthesis Algorithm. Picture from [Cook, 2002].

2.5.2 Filter-based Modal Synthesis

This method is also additive, since the outputs of a number of band-pass filters are added (figure 2.2). To synthesize a sound using this method, as many filters as the amount

of modal frequencies chosen are used.

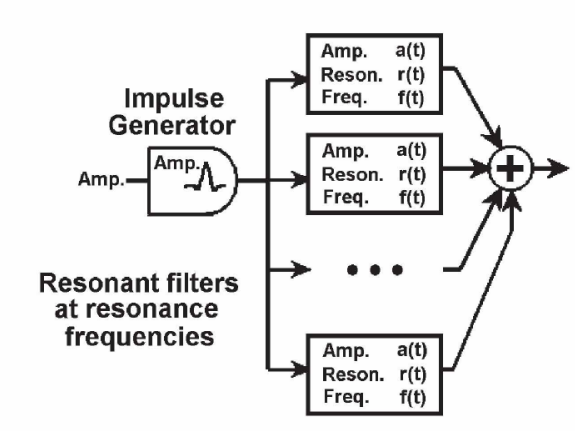


Figure 2.2: Filter-based Modal Synthesis Algorithm. Picture from [Cook, 2002].

Band-pass Filters

At this point we will give some basic description of the band-pass filter since it is the core aspect of this method. A Band-Pass Filter (BPF) takes a signal as input and lets 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 cascading a low-pass and a high-pass filter. In figure 2.3 the frequency response of a BPF can be seen.

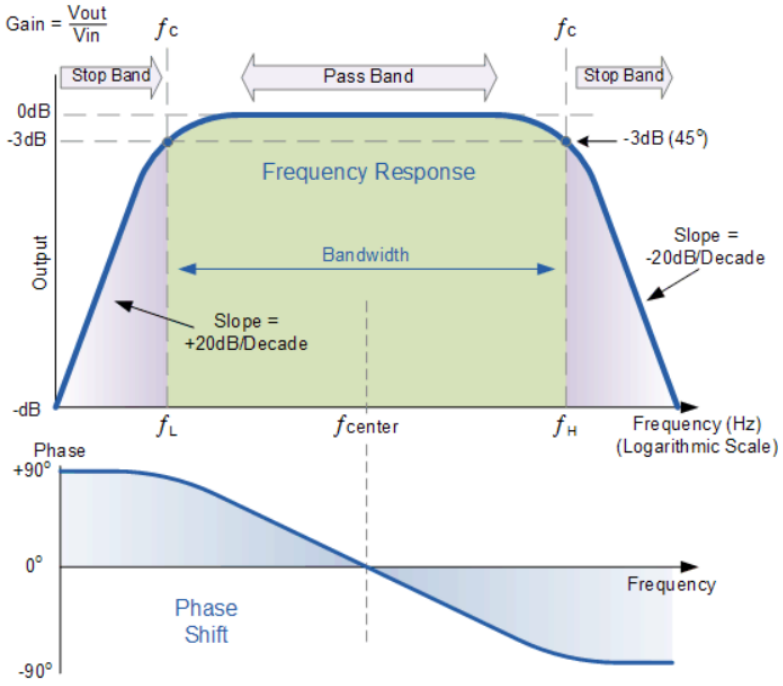


Figure 2.3: Frequency Response of a Band-pass Filter. Picture from [AspenCore, Inc.,].

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

difference between those two is the bandwidth

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

[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 of the filters. The Q is a dimensionless parameter 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.

2.6 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 a matrix of amplitudes of size $K \times N$ corresponding to K different resonant modes of each of the N points of the object is obtained. It is to note that the damping and frequencies 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 theoretically 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. However, 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, as proposed in [Lloyd et al., 2011].

2.7 3D Audio

Spatialized sound is important for 3D virtual worlds because it enables the localization of objects in combination with visual cues. Auditory cues are also essential to localize a sound source that is outside the field of view (i.e. a car driving closer but hidden behind a building corner). Studies show that accurate auditory information increases the sense of presence in a virtual environment [Larsson et al., 2002]. Different methods for rendering three-dimensional sound field for the listener's ears are covered in this section.

2.7.1 Binaural and Transaural Techniques

The aim of these methods is to recreate a wave field at the ears of the listener using most commonly headphones (binaural) or loudspeakers (transaural).

Head-Related Transfer Function

The goal of HRTF is to model, by the use of filters, the effects of resonances inside the ear and close to the head and upper body of sound propagation. To obtain these filters small microphones can be placed at the entrance of a listener's ear canals or on a dummy-head. HRTFs vary depending on the individual, due to morphological differences, and depending on the angle of thelistener's head [Zhang et al., 2013], thus they should be adapted for each listener to increase the quality [Funkhouser et al., 2002]. It is nowadays the most used technique for sound spatialization in combination with head tracking for VR and AR headsets [Oculus VR, LLC., , Microsoft,].

Transaural Stereo

This method makes use of a stereophonic setup of loudspeakers for binaural reproduction. As the signal emitted by the right speaker also reaches the listener's left ear and vice versa, filtering is needed to cancel the cross-talk. Its use is limited to desktop and suffers from a limited sweet-spot, where the binaural reproduction can be acceptable [Funkhouser et al., 2002].

2.7.2 Multi-channel Auditory Displays

These techniques use an array of loudspeakers positioned around the listening area to generate a 3D sound field. They are usually used for larger audiences and suffer from sweet-spot problems but on the other hand do not need complex HRTF filtering. VBAP aims at reproducing the auditory cues, utilized by the humans for accurate 3D localization, whereas the Ambisonics and Wave Field Synthesis (WFS) methods aim at reproducing the physical aspects of the sound field.

Vector-Based Amplitude Panning

Vector Base Amplitude Panning (VBAP) is a computationally efficient and accurate method based on auditory perception. By panning the sound sources, the perceived source location is shifted. The loudspeakers can be positioned in an arbitrary three-dimentional placement. However it does not allow the reproduction of elevation of the virtual sound source [Pulkki, 1997].

Ambisonics

Ambisonics is a spatial sound encoding method for surround and multi-speaker systems. It uses soundfield microphones that encode audio through an arbitrary number of channels that model the directional soundfield. First order ambisonics make use of four microphones, while accuracy increases with the increment of number of microphones used. Ambisonics has been extended to higher orders to offer more spatial resolution [Daniel et al., 2003].

Wave Field Synthesis

Wave Field Synthesis (WFS) is based on the Kirchoff integral theorem which states that any point of a wave front can be considered as a secondary source. This method requires a large amount (about 120-160 output channels) of loudspeakers, depending on the reproduction area, and processing to give accurate reproduction [Funkhouser et al., 2002].

2.7.3 3D Audio Discussion for VR/AR Applications

WFS and higher order Ambisonics are multi-user reproduction systems, which could be an interesting element for video games but due to their setup complexity do not seem to be an option for virtual environments. Other methods that make use of loudspeakers offer good spatialization results but are impractical for VR/AR applications as there should be consistency between the user's viewpoint in the virtual space and the virtual listening location. Additionally, sound reflections on the screen surfaces can engage acoustical problems. This is why binaural systems that use HRTF and which are incorporated into nowadays VR/AR headsets seem to be the best option to maintain consistency between the listener's head motion and the audio scene.

Chapter 3

Overview of Method and Tools Used

3.1 Method Overview

The aim of this thesis is to provide sound designers with an easy-to-use tool which enriches video games with procedural and physics-based audio instead of the use of prerecorded sounds. The tool enables to control contact sounds (impact, rolling and scratching) produced by vibrating objects through high level parameters that characterize the objects' (size, material and surface roughness). The challenge is to offer realistic real-time and event-based sounds in a game engine without high Central Processing Unit (CPU) usage.

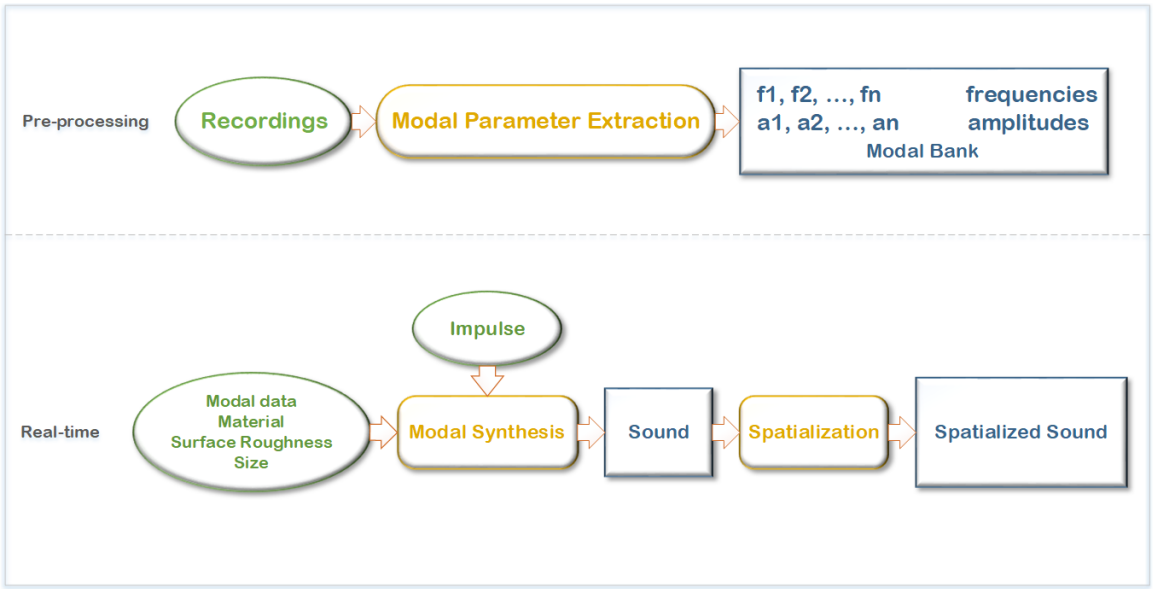


Figure 3.1: Tool overview.

To create this tool, the procedure described in figure 3.1 was followed. To extract the modal parameters of vibrating objects the "Example-guided" method described in section 2.4.3 was used due to its better integration within the audio pipeline as opposed to the rest of techniques presented in section 2.4. The first step was to find several everyday objects that were made of different materials (plastic, wood, ceramic, glass and metal). This is because a priority in this thesis is the synthesis of sounds based on material properties. To obtain sound

variations along the object's surface (see second method in section 2.6) the chosen objects were divided into areas that produced similar sounds when struck (e.g. bottle neck, rim of glass, etc). Between two and six sounds were recorded depending on the object. An example is presented in section 4.1 for illustration.

In the pre-processing stage, the recordings were then used to extract the data needed for the sound synthesis with the ChucK programming language [Wang,]. Some involved DSP algorithms that employ the Fast Fourier Transform (FFT) are used to capture the modal frequencies and gains specific of the object and which are present in the supplied audio clips. This process is explained in more depth in section 3.4.

As far as the modal synthesis of impact sounds is concerned, the two methods shown in section 2.5 have been implemented into Pure Data [Puckette,] patches (see section 4.3.1) to evaluate differences between the output sounds. Scratching and rolling sounds (see sections 4.3.2 and 4.3.3 respectively) also make use of the filter-based method.

3D models of the recorded real world objects were generated in Maya [Autodesk, Inc., b]. The models were then combined with their corresponding modal data and synthesis patches inside Unity® software [Unity Technologies,]. Heavy [Enzien Audio, Ltd.,] was used to generate audio plugins from the Pure Data (Pd) patches for better integration in the game engine and interactivity with the synthesizer.

At runtime, rigid-body physics events have been used to drive the synthesizer. Depending on the interaction (impact, rolling or scratching) the model is excited differently and produces the corresponding sound which is tightly coupled with the graphics. The final stage of the audio chain is the spatialization which is done with Unity®'s Audio Spatializer SDK [Unity Scripting Reference,].

3.2 Sound Discretization Scheme

The first step for the creation of this tool is to obtain the necessary data for the audio synthesis, through the use of the "Example-guided" method as mentioned in section 2.4.3. This method was applied instead of the FEM, as in [Director-O'Brien, 2001] and [O'Brien et al., 2002], or the spring-mass systems method as in [Raghuvanshi and Lin, 2006] because of its simplicity within an audio pipeline in a video game production. More specifically, the usage of recordings is an easier option for the targeted users (sound designers and game developers) than one that implies complex computations of vibrating structures. Additionally, Unity® seems to present some limitations for techniques that present sound variation depending on vertex collisions such as the FEM and spring-mass systems. The game engine is not able to provide the exact vertex that has collided.

To produce sound variations along the surface of an object, a 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. This method was tried during the development of this thesis, but was abandoned 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. Thus, recordings of the sound produced by everyday objects when hit in several locations were performed instead. The strategy used is explained in detail in section 4.1.

The objects recorded were chosen due to their simple shape and symmetry plus the fact that they are easy to find in most kitchens. A heuristic approach that assumes that nearby points produce almost the same sound was adopted. This lead to the division of the object into "sound areas" instead of calculating different modal matrices for each vertex of the model as in the FEM. Unofficial audio tests proved that this complexity-accuracy trade-off was acceptable to map the different sound variations along the struck object's surface.

Depending on the range of substantially different sounds produced by the object, more or less areas were chosen. A sample object and its division in areas is shown in figure 3.2.

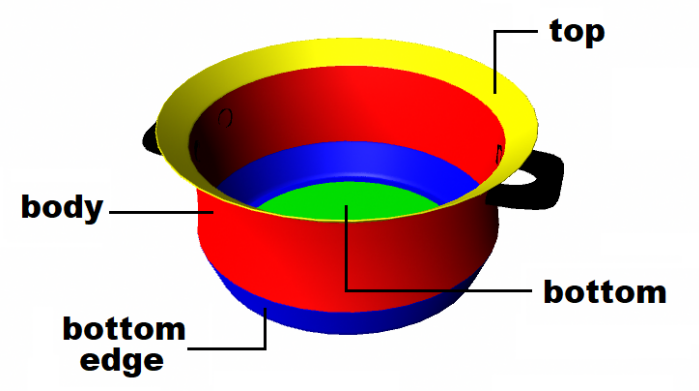


Figure 3.2: Division of an object into areas with similar sound.

3.3 3D Models

For coherence purposes the objects recorded were modeled so that the synthetic sounds match with the 3D model, the same way the recordings match with the real world objects. Hence, the dimensions and weight of the eleven objects were measured to model the objects. In table 3.1 the maximum dimensions and the weight of the objects are displayed. Maya Autodesk software was used to create the 3D models of the objects. They were exported as FBX® files [Autodesk, Inc., a] which is a format recognizable by Unity®.

Name	Dimensions(cm)	Weight(g)	Model
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.

3.4 Modal Data Extraction

The 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, independently of other sounds that may be playing at the same time.

ChucK was chosen for this stage of the work, because of its built-in functions to manipulate sound, like the FFT and windowing of input audio signals [ChucK Documentation,]. 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,]. The ChucK language was used to identify and extract the frequency modes and gains of the recorded audio files. The output provided by this code, therefore, delivers exactly the parameters needed for modal synthesis. The aforementioned course is the main reason behind the choice of this programming language.

When analyzing the recordings, one can identify that each object has a very high number of resonant modes. Although, most of them are inaudible and do not contribute significantly 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, ten modes were chosen as the sufficient amount for the analysis/synthesis. Afterwards, the algorithm having taken a recording as input, computes an histogram and identifies the modes. The frequencies where peaks occur are the modal frequency candidates. Depending on the number of modes chosen, the algorithm outputs the most relevant ones. Finally, the algorithm calculates after a normalization process the corresponding amplitude of each mode.

3.5 Modal Synthesis Patches

Pure Data (Pd) is another music programming language. It is open source and the main difference with the ChucK language is that Pd is a visual or "patcher" programming language, using objects instead of code, linked together to form a sequence [Puckette,]. This software was chosen as our synthesis engine mainly because of the ability to compile the patches into C# code, as explained below in section 3.6. Another important reason for choosing it is the possibility of real-time parameter manipulation and easy testing during the implementation period.

All interaction sounds (impact, rolling and scratching) are 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.

3.6 Audio Plugins

Heavy is a compiler that generates audio plugins from Pd patches in interactive sound and music applications [Enzien Audio, Ltd.,]. In this thesis it is used to compile Pd patches into Unity® audio plugins. Heavy's interface is a website where users upload patches and then are able to download the corresponding plugins and put them into their applications. The plugins used consist of Dynamic-Link Library (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, float values can be sent 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) 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 encountered while 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.

3.6.1 Why Not OSC?

The most popular way to communicate between Unity® and Pd is the Open Sound Control (OSC) protocol. The reason why OSC was not chosen 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.

3.7 Game Engine

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

Unity® was chosen due to its ease of use, cross-platform publishing compatibility, intuitive collision events and because the authors have experience developing games with this game engine. Additionally, Unity® is the only game engine that is supported by Heavy.

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,].

3.8 Sound Spatialization

In this paper the sound spatialization is considered a separate process with respect to the sound synthesis. This approach was considered due to the Audio Spatializer SDK provided by Unity® which delivers the desired effect.

The Audio Spatializer SDK is a plugin that replaces the standard panner in Unity® by a more advanced one [Unity Scripting Reference,]. It makes use of HRTF filtering which is based on the KEMAR data set [MIT Media Lab Machine Listening Group,], an extensive library of HRTF measurements on a dummy head microphone by Bill Gardner and Keith Gardner at MIT Media Lab. The Audio Spatializer SDK changes how the audio is transmitted so that it seems like the listener is in a 3D environment. Depending on the position of the

listener in the virtual scene and the audio source, the spatializer adjusts the left and right channel gains to simulate a virtual sound field.

Chapter 4

Implementation

4.1 Recordings

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

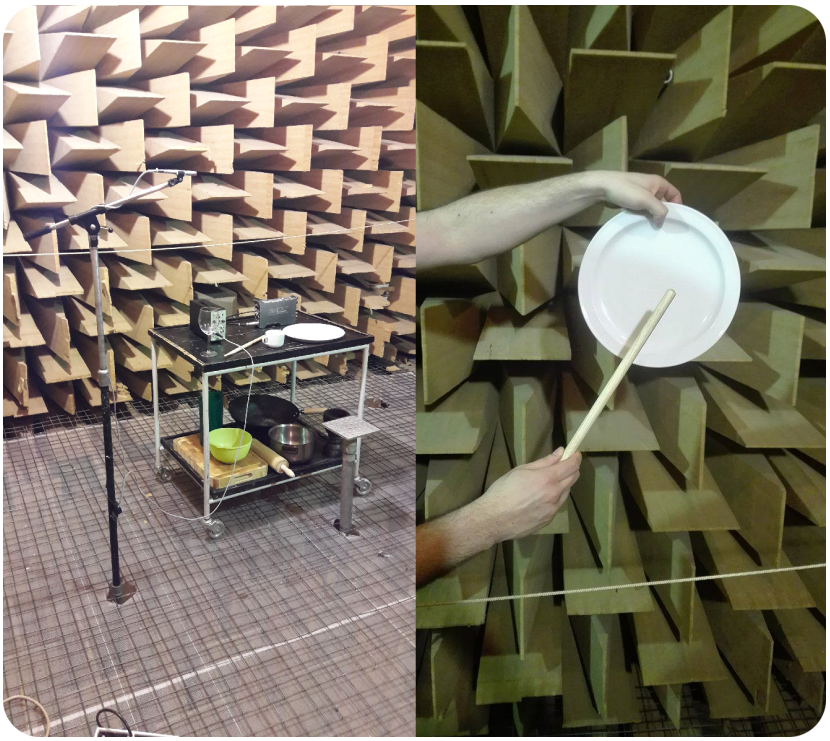


Figure 4.1: Picture of the setup for the measurements (left) and of a struck object (right).

To control the impact, the objects were hit by hand with a wooden drumstick (figure 4.1 right), 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 the ability to model them for the demo (as they should be simple enough). Secondly, there was a desire for the sounds to be familiar for the users who would perform the listening test. 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

The modes detector algorithm described in section 3.4 was used to extract 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" it was divided into.

Although, theoretically, each object can be modeled using one vector of frequencies and multiple vectors of amplitude data (one for each different vertex) it can be seen in figure 4.3 that this does not happen in practice. There are different reasons why this could be. During the measurement process certain factors such as the force applied when hitting the object or the way it is hold could affect the resulting sound. Holding the object prevents a free vibration behavior which could lead to the cancellation or significant reduction of some of the resonant modes' gains. In addition, the sound produced by the wooden stick when striking could interfere with the one of the object under study. Another factor could be related to the anisotropic material properties of the objects. As far as the audio algorithm is concerned there could probably be some improvement in the DSP calculations to obtain more accurate results. Additionally, to include all the resonant frequencies of one object, instead of the ones with the strongest peaks per area, would require a large frequency vector. This would take up a lot of memory space, without improving significantly the result.

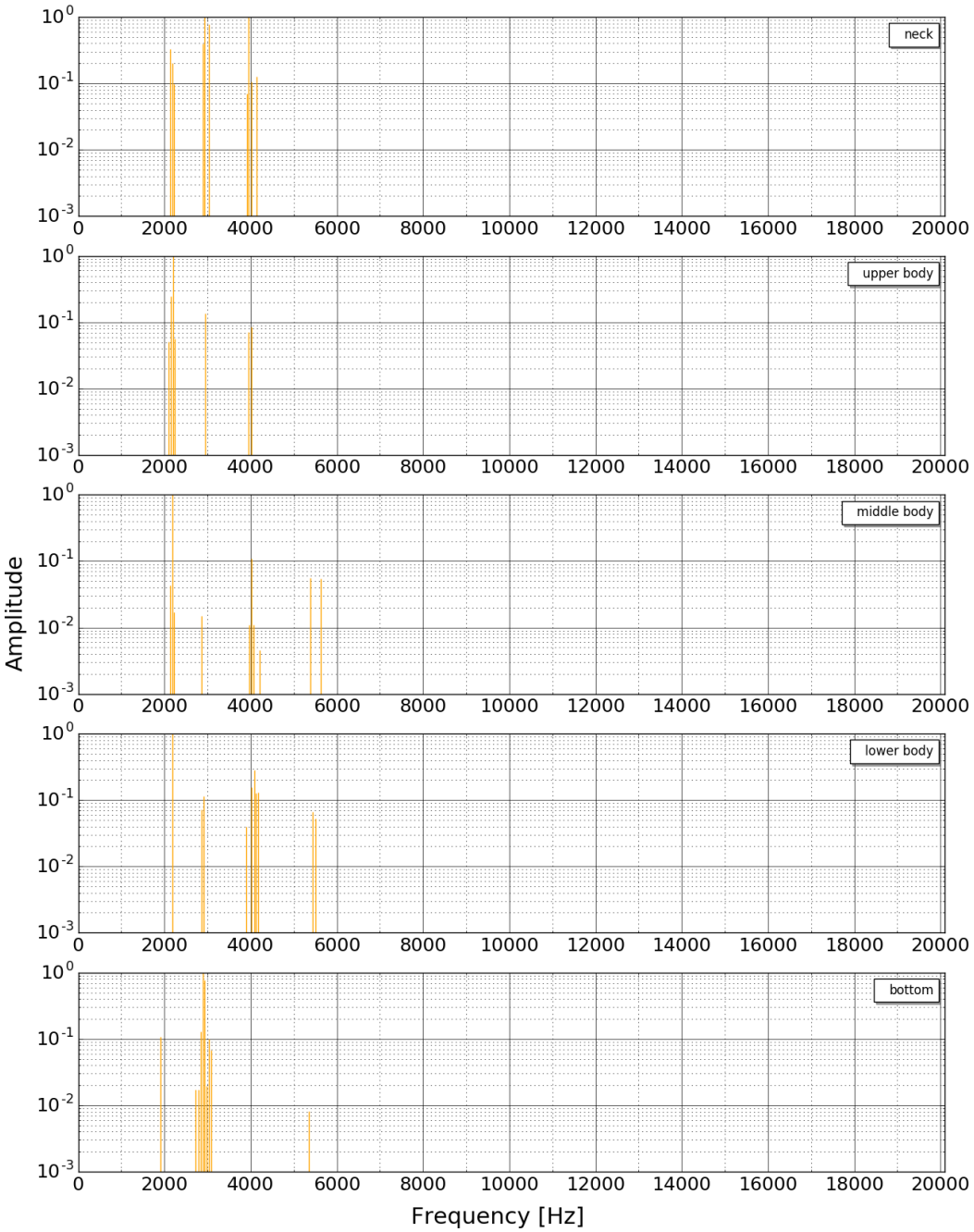


Figure 4.3: The frequencies and their corresponding peaks for each area of the wine bottle object.

Nevertheless, it can be noted that most of the frequencies are set just over 2 kHz and close to 4 kHz for different parts of the wine bottle although the bottom one presents some variations, probably due to the thickness of the glass in that specific area.

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.

Another C# script assigns, to every one of the objects, the modal parameters that were extracted in the analysis part (see section 3.4). This is done independently of the synthesis methods used below.

Two different types of force models are considered 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

This section describes 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. Therefore, these frequencies which are identified as f_n in the equation 2.1, can be fed 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.

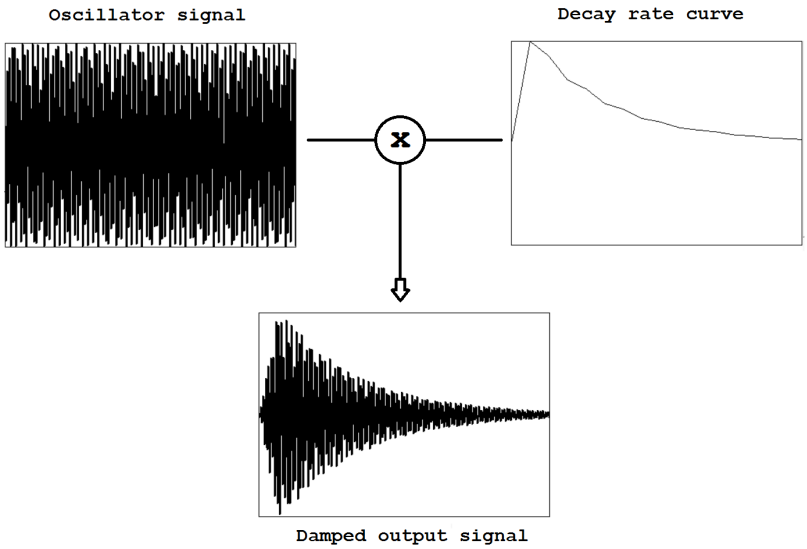


Figure 4.4: Diagram showing how the output partial is created from a 5000 Hz cosine wave and a decay rate curve with D = 0.005.

The expression $e^{-d_n t}$ which corresponds to the damping of every mode n is also translated into Pd. 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 is used in the patch, is to have $d_n = 2\pi f_n D$. By experimenting, it was 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 the damping is multiplied by the partial's initial amplitude A_n to obtain an amplitude envelope that varies over time and which is multiplied by the oscillator's signal. The output is what is called a *partial* which is illustrated in figure 4.4.

The final sound is produced by adding together the ten partials. The resulting signal is multiplied by the magnitude of the impact. For this, the kinetic energy is calculated

with Unity®'s physics components (see section 4.5.2). As described in [Farnell, 2010], before sending the signal to the Digital-to-Analog Converter (DAC), it is passed through a clipper. This gives richer harmonics and produces brighter sounds the harder the impact is [Aramaki et al., 2009b].

The patch, which is compiled in audio plugins, produces an impact sound whenever the *OnCollisionEnter* [Unity Scripting Reference,] 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 the magnitude of the collision is set and an event to excite the patch is sent. This is done by setting the value of t in equation 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 band-pass filters. Pd's band-pass 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.

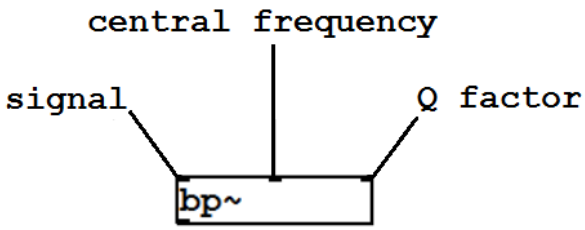


Figure 4.5: Pd's band-pass filter with its three inlets

The same way it is done in the previous method, all ten frequencies and amplitudes of the object are initialized. Every frequency f_n is sent into a band-pass 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 different material sounds can be obtained. Through experimentation values of Q that range from about 20 for plastic to 5000 for metal were found.

To cause the object to sound, an impulse signal that excites the filter is used. 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.

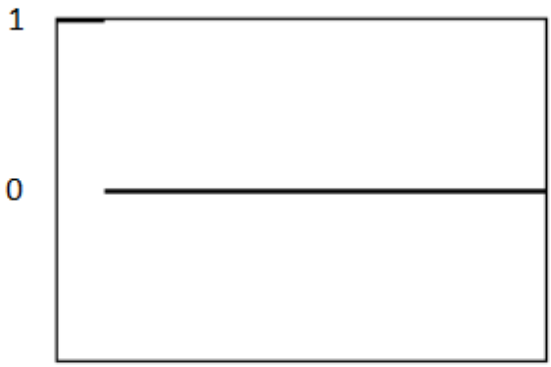


Figure 4.6: Impulse signal used to excite the bandpass filter.

Pd does not include a built-in impulse signal, thus it had to be constructed. The array shown in figure 4.6 was created to simulate a Dirac function which is commonly used to model impact events. For the purpose of this thesis the function is defined as

$$\delta(n) = \begin{Bmatrix} 1 \ , \ n = 0 \\ 0 \ , \ n > 0 \end{Bmatrix} \tag{4.1}$$

with n being the sample index and δ the Dirac function. The values of this array are read as the input signal. 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, the impulse array was set to size 441, which is the number of samples per 10 milliseconds and the input array was read 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

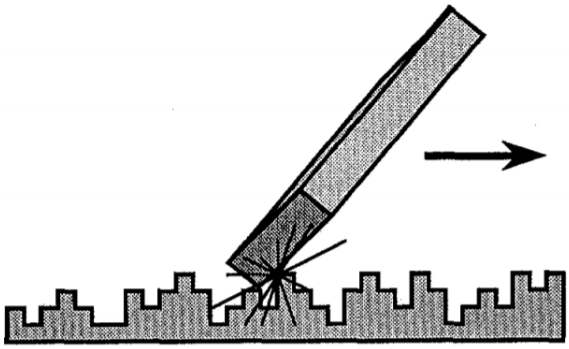


Figure 4.7: Scratching involves a multitude of micro-collisions against a contact area. Picture from [Gaver, 1993a]

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. The same modal parameters can then be used as in the impact methods described here above.

To produce scratching sounds, a similar technique to [Gaver, 1993a] and [Van Den Doel et al., 2001] who propose the use of filters is followed. The filter-based modal synthesis method is therefore chosen to model the resonator as seen in the previous section. The difference lies in the signal that excites the model. This is implemented by generating a noise impulse waveform, as seen in figure 4.8, that passes through the band-pass filters. This waveform is created 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 it was deduced that the higher the velocity and the Q, 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 scratching sound is triggered in Unity®'s *OnCollisionStay* [Unity Scripting Reference,] 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]. This condition is therefore added to differentiate between scratching and rolling.

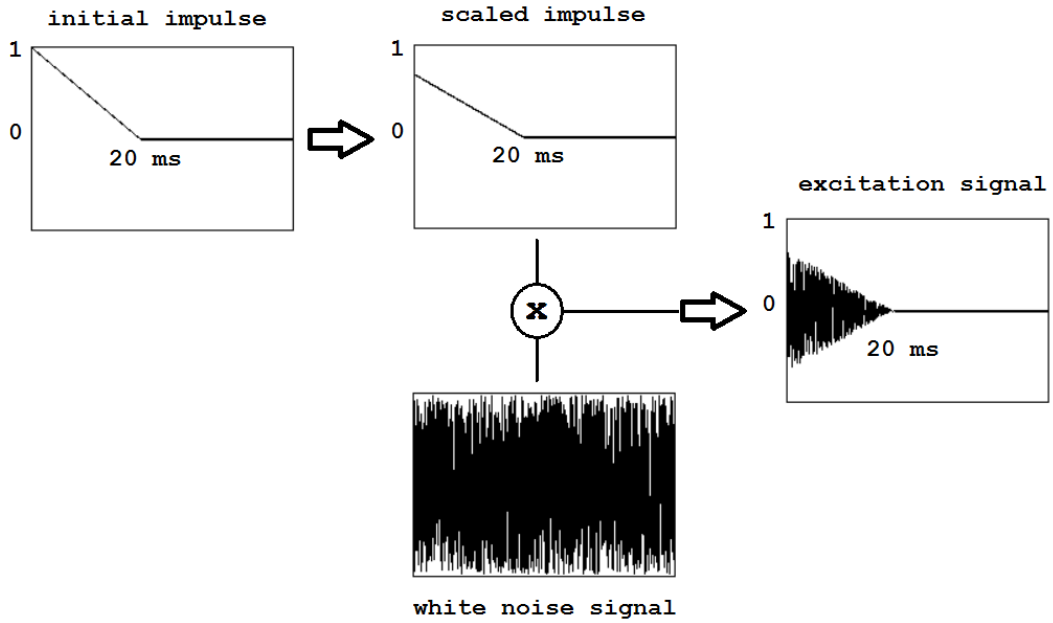


Figure 4.8: Diagram showing the process to get the excitation signal of the resonator to produce scratching sounds.

The authors, who's approach is followed, state that the center frequencies of the band-pass 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 the incoming filter frequencies are scaled 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. This study therefore focuses 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 the attention is directed to the sounds produced by the object's surface irregularities. For simplification a regular octagon that has received an impulsive force is used 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 the filter-based synthesis method is applied. 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.

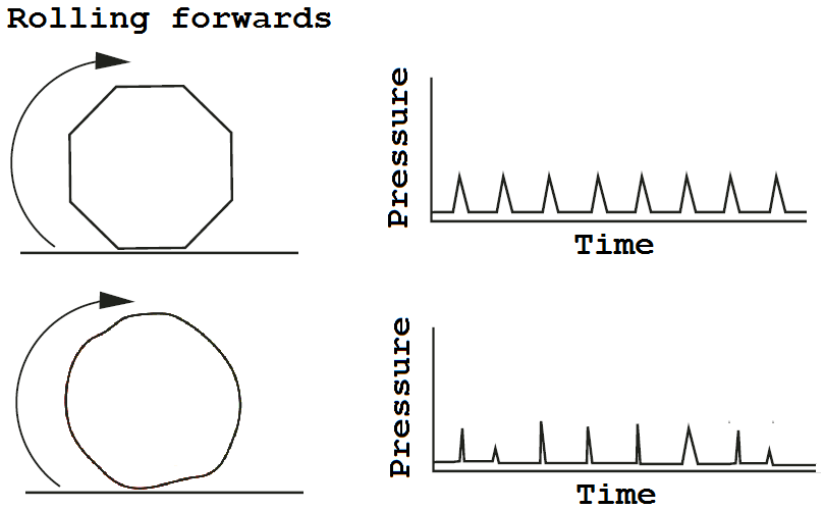


Figure 4.9: An octagon and a real spherical object pressure levels over time as they roll.

Let us dive into the actual implementation of the latterly described model. It was previously mentioned that the amplitude of the impulse signals, used to excite the resonator, are different. With a heuristic approach, a sequence of eight multipliers that correspond to the prominence of the bumps along the object's surface contour is chosen. Additionally, a parameter that determines the object's surface roughness has been incorporated. 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. The amplitudes of the impulses for different roughness degrees can be compared in figure 4.10.

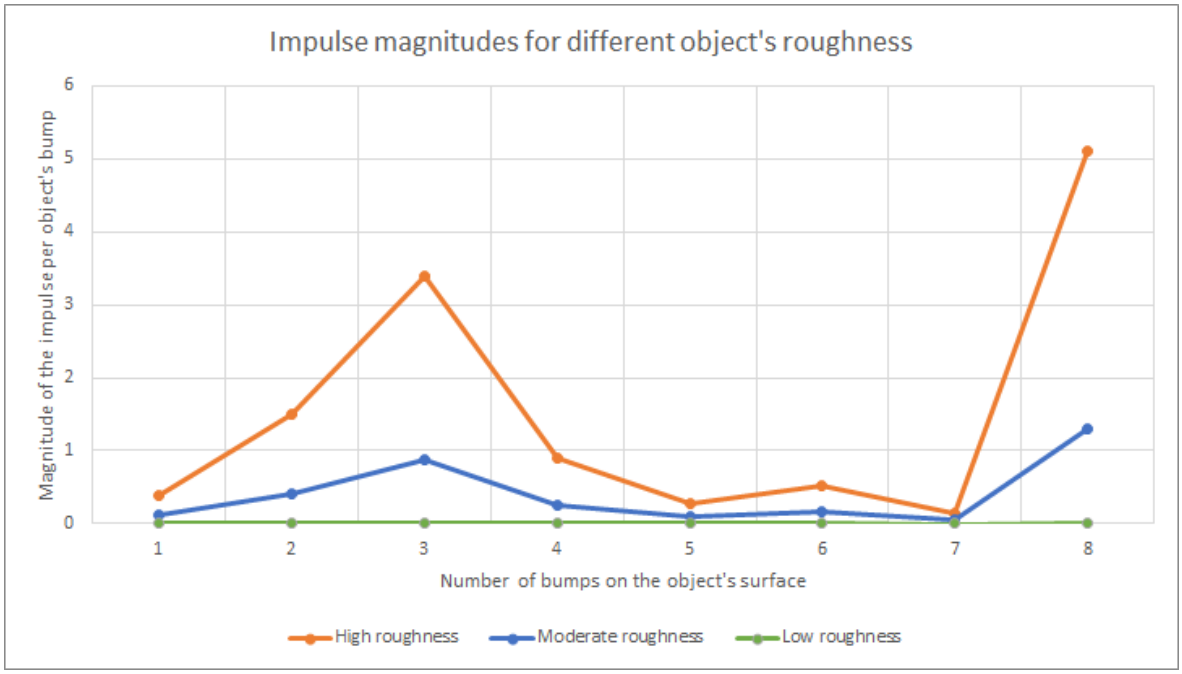


Figure 4.10: Graph showing the amplitude of the impulses for every bump on the object's surface for three values of the roughness parameter.

This paragraph looks 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, the surface's irregularities are considered 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 rolling friction forces are considered to be smaller compared to scratching friction [Mehta and Mehta,].

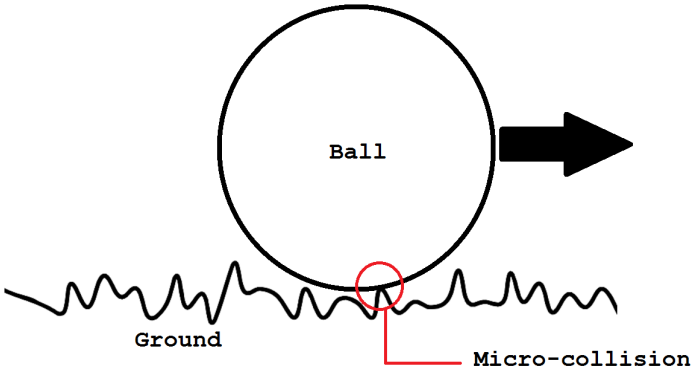


Figure 4.11: A smooth ball rolling over small ground irregularities.

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. The *Inspector* in Unity® is a window that shows up inside the platform, after selecting an object, a file, etc. and it displays all information relevant to it. A custom *GUISkin* is also used, 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 the pre-made prefabs that have been created (which correspond to the modeled kitchen objects) 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.

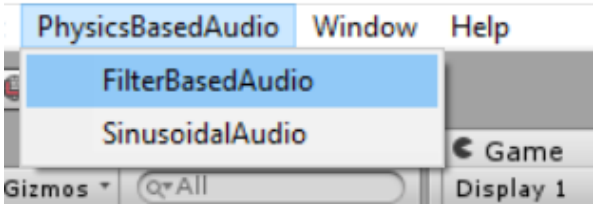


Figure 4.12: Designer can assign the audio manager from the menu bar.

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. An example of the whole Unity® inspector of a modal object can be seen in figure 4.13.

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*), or make his own if he uses his own objects. It is possible for everyone to contribute with new objects to the tool, by following the guide in appendix D.

Figure 4.14 shows the three parameter sliders, available for the designers to adjust the audio. They are the same for both of the synthesis methods 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 choices (*Plastic, Wood, Ceramic, Glass* and *Metal*), or even a 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 than the default ones, or even when he prefers the sound to be more high or low pitched. The last slider adjusts the roughness of the object's surface. By tweaking this, designers are able to choose surfaces of the objects to be smoother or rougher than the default ones. Changing this parameter is perceived when the object is rolling. Below is an extended description of the sliders.

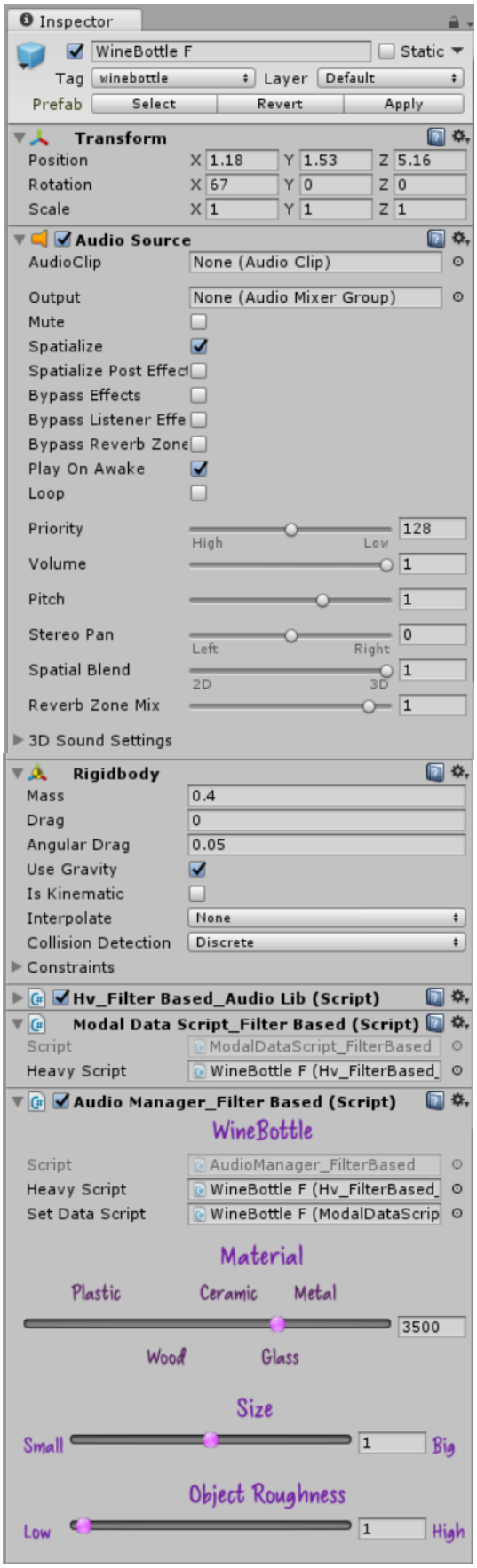
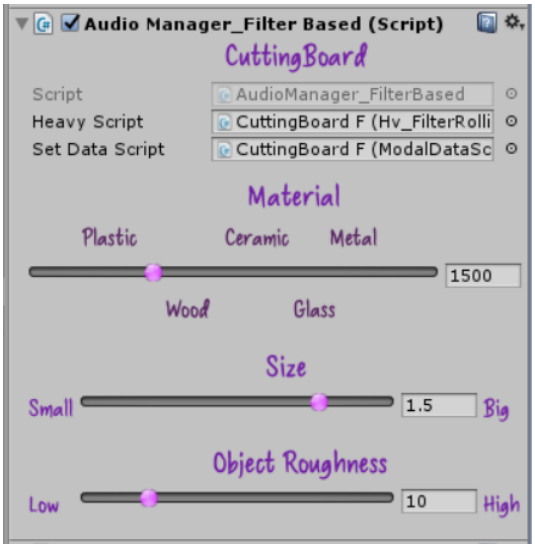
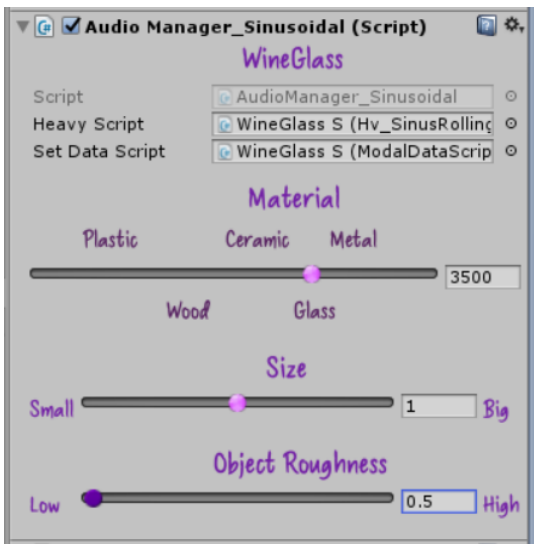


Figure 4.13: An example inspector of a game object inside Unity® platform.



(a) Audio manager of the filter-based modal synthesis method.



(b) Audio manager of the sinusoidal additive synthesis method.

Figure 4.14: The custom part of the inspector inside Unity® platform, with the parameters available to designers.

4.4.1 Changing the Material

Different materials can be assigned to the objects used in this tool. 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". Those sounds are achieved by changing the Q factor of the band-pass filters used in the PD patch. The Q 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 4.2 [Cory et al., 2006].

$$Q = 2\pi \frac{\text{maximum energy stored}}{\text{total energy lost per cycle at resonance}} \tag{4.2}$$

Using trial and error methods, an approximate value of the Q for each material is chosen and used as the default value in 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 [Gaver, 1993b]. Hence, a slider is implemented for the designer to choose the best

sound that corresponds to the size of the object. It is to note that the middle position of the slider (*size: 1*) corresponds to the real object used for the data extraction. The slider allows to size the object from a tenth of its original size to the double, to keep the frequencies in the audible range.

4.4.3 Changing the Surface Roughness

Another setting that the sound designer is able to tweak is the object's roughness. This parameter determines how irregular or "bumpy" the surface of the object is, therefore it affects rolling sounds as seen in section 4.3.3. By increasing the roughness of the object with the slider, the more noticeable the impacts caused by the micro-collision are. When the slider is set to the lowest roughness coefficient, this corresponds to a perfectly smooth surface which ideally will not produce any sound.

4.5 Unity® Scripts

Heavy [Enzien Audio, Ltd.,] compiler was used to convert the Pd patches described above into Unity® compatible C# code. However, several other scripts were implemented for the following actions: 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 sounds are more high-pitched when an object is scaled down and *vice versa*. To achieve a realistic scaling when the designer uses the built-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 = (scale of x dimension + scale of y dimension + scale of z dimension)
      / 3
```

This average is used as an adder to the *Size* parameter described above, which is then used as a multiplier for the resonant frequencies - here called "pitch multiplier". To avoid distortion in sound and to stay within the audible frequency range, a limit of 8.5 units for the maximum scaling is set, a number found heuristically. This is considered to be a good choice because Unity® uses *meters* as the default unit and since everyday objects are used, it is rare for someone to handle 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, a normalization to 1/10th of the average scale value is performed. The resulting scaling is applied to the size slider value, and to the pitch multiplier which is used to re-set the modal frequencies. To be more precise, instead of applying the actual pitch multiplier value added with the average scaling variable, this value is subtracted from 2 and is then used. This happens because the size slider is reversed. More specifically, the multiplier directly applied to the frequencies, increases them when it is smaller and decreases them when it is bigger. However, for convenient reasons, it was desired that the bigger the multiplier (right side), the bigger the object. Since 2 is the biggest value, it was normalized to correspond to the biggest one.

In the second case, the value from the pitch multiplier is subtracted. The average scaling value does not need to be normalized, because it is already between 0 and 1. Afterwards,

the same procedure is followed as above, with one difference; instead of subtracting the new pitch multiplier from 2 it is added to 1. This is necessary because now the biggest value of the size slider is 1 - since above this it counts as a scale-up - and the reversed value for the slider is wanted, so the subtracted value is subtracted, turning it into a plus value (+). If no scaling takes place nothing happens.

The whole scaling procedure is shown below:

```
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
```

Listing 4.1: Code for audio adjusting to object scaling

The tool, also, includes a function called *ScaleRealTime*, which can be called inside the game when a scaling takes place, for the pitch of the sound to scale respectively. However, the scaling of the pitch will change gradually and if the *GameObject* is moving during this change, undesirable sounds may occur. The difference from the previous function, which is called only in the beginning, is the condition of the *if* statement. The real time scaling function needs a previous scale float value as input to compare it with the new one.

4.5.2 Excitation of Impact Sounds

Impact sounds are excited whenever 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. The program identifies when two Unity® collider components attached to two different objects touch each other.

OnCollisionEnter Function

OnCollisionEnter [Unity Scripting Reference,] is a Unity®'s built-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*. The tag manager holds all available tags. Tags are used to group similar game objects and make them retrievable from scripts. After the type of object is identified, the object perimeter variable (used for rolling sounds) is assigned to its corresponding value. Then, the modal data (frequencies and amplitudes) that correspond to this type of object are assigned to the variable used from the algorithm.

Moreover, the algorithm calculates the kinetic energy of the whole collision, using the equation 4.3 [Crowell, 2003]:

$$K = \frac{1}{2}mu^2, \tag{4.3}$$

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® the latter can be calculated using the command: *Collision.relativeVelocity.magnitude* [Unity Scripting Reference,]. The kinetic energy

corresponds to the collision force magnitude and we use it to determine the output sound amplitude depending on whether the collision was strong or weak.

Another parameter that the algorithm takes into account is whether an object collides with another modal object or not; modal object meaning one of the predefined objects provided with the tool which comes with its modal synthesis data. In case a collision with another modal object occurs, the collision force magnitude is modified depending on the materials of both colliding objects. More specifically, an average between the two normalized quality factor values is calculated, it is then multiplied by the collision force magnitude and added to the initial collision force magnitude. This is done because the material of an object and thus its "hardness" (as seen in [Giordano, 2003]) affects the "loudness" of the produced sound as shown through experimentation in [Freed, 1990]. As a result when two hard materials like glass or metal collide with each other, they produce a much more intense sound than when a hard material collides with a softer one like plastic or wood.

$$new\ CFM = CFM + CFM * \left(\frac{\frac{Qfactor_1}{5000} + \frac{Qfactor_2}{5000}}{2} \right) \tag{4.4}$$

where CFM : the Collision Force Magnitude.

4.5.3 Excitation of Rolling and Scratching Sounds

OnCollisionStay Function

Unity®'s built-in function *OnCollisionStay* [Unity Scripting Reference,] is called once per frame as long as two rigidbodies are touching. 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 of whether an object is rolling or sliding along a surface is made using the angular velocity ($\omega = \frac{angular\ displacement}{time}$). Unity®'s rigidbody variable for angular velocity is a three dimensional vector, measured in radians per second [Unity Scripting Reference,]. The magnitude of it is calculated from the equation 4.5

$$\omega_{magnitude} = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{4.5}$$

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 the angular velocity magnitude is above 1, it means that the object is rolling, otherwise it is not and providing that the linear velocity is non-zero, it means that the object is sliding on the surface. Magnitude of 1 for the angular velocity was chosen as the threshold from trial and error tests in the Unity® editor, because a vector of $x, y, z = (0.0, 0.0, 0.0)$ for the three dimensions of the angular velocity, gives a magnitude value slightly bigger than zero.

4.5.4 Pseudo-code of the Procedure

```
1 CALL ScaleEverythingWithObject (described in code snippet 4.1)
2 IF collision detected THEN
3   RECEIVE object tag from tag manager
```

```
  SET object perimeter
  RECEIVE struck area from collider
  SET corresponding modal data
  APPLY to audio plugins
  IDENTIFY type of the other colliding object
  CALCULATE collision force magnitude
  SEND bang to excite impact sound
IF collision continues THEN
  CALCULATE velocity of object
  IF scratching
    SEND scratching sound duration
  ELSE
    IF started now THEN
      INITIALIZE values
    CALCULATE distance traveled
    IF new surface irregularity found
      CALCULATE time passed
      SEND rolling time duration
      SEND bang to excite rolling sound
      INCREMENT irregularity index
    IF done whole cycle THEN
      INITIALIZE values
```

Listing 4.2: Code of the whole procedure

Chapter 5

Subjective Experiments

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

To examine the quality of physics-based synthetic sounds on listeners, *A*-*B* and MUltiple Stimuli with Hidden Reference and Anchor (MUSHRA) [Series, 2014] tests were performed. The aim is to answer the following questions: 1) Which of the two synthesis methods (sinusoidal or filter-based additive synthesis) offers the best results? 2) Does sound variation make an audio cue more appealing? 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, study the listener's reaction to sound variation and increase our understanding of spectral characteristics in relation with the material of the object. The interface of the tests is displayed in appendix B.

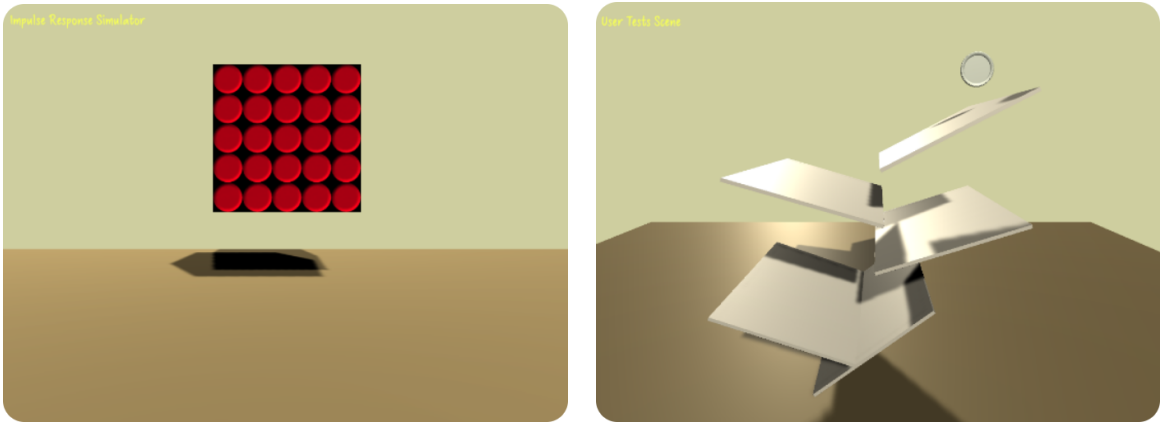
5.1 Preparation

The experiments use several audio files that were recorded inside the Unity® platform. Two special scenes were created 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)), the audio files for the first experiment were recorded. During the recording session, the cube was "tagged" with different object names (a process which implies assigning different modal data) and let it hit the ground without bouncing. The process was repeated adding an *Audio Source* component to the cube with the anechoic chamber recordings to achieve consistency of the stimuli.

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

Every sound in the audio files starts one second after the 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.



(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.

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	Rotated platforms (5.1(b))	Participants listen to sounds recorded using three 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	Rotated platforms (5.1(b))	Participants listen to sounds produced by five 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

Three different tests were performed. 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. The participant is then asked to choose which of the two synthesized sounds is closer to the reference and to what extend. The goal of this experiment is to gather information about the quality of the two methods and address which one of the two is best.

In the second test, the stimuli consists of 33 trials that contain sounds produced by falling objects. This time each trial includes two sounds and no reference. One of the sounds corresponds to an object that has been split into "sound areas" and struck in different locations. The second sound corresponds to an object that does not present sound variations along its surface and that is also struck several times. This single sound was chosen instead of other recordings corresponding to the object because it was the closest to the real sound (authors' opinion). Participants were asked which of the two sounds they preferred and to what degree. This test provides the results for sound variation versus repeating the same sound over and over.

The stimuli of the third test are sounds coming from five different objects, one of each material tested in this thesis (plastic jug, wooden mortar, ceramic plate, glass bottle and metallic cooking pot). For each object, ten different sounds per synthesis method are used. Each trial includes ten sounds that correspond to the same event, but with a small variation of the Q for every sound. More specifically, starting from a value of 1000, the Q was increased by 200 up to 4600. Some sounds that were too similar with others were removed to decrease the total size and length of the test. The participants were asked to choose the sounds where, in their opinion, a change in the material happened. This test was done to validate the chosen values for the Q.

In the first two experiments, both the sequence of trials and the conditions (which method is A and which is B) are randomized. However, in experiment number three, an increase of the Q factor was desired and thus the order of the sounds matters. Hence, only the sequence of the trials were randomized.

5.3 Participants

Eleven adult volunteers participated in this test which lasts about half an hour. They were five women and six men, aged 25 to 61 who stated having normal hearing. They have different backgrounds, two of them were musicians and the rest were not. An attempt to extract different results for the two groups would not give qualitative outcomes due to the reduced number of individuals in one of them.

5.4 Test Results

The participants' responses were obtained in ".txt" format and processed using Python programming language and the following results were acquired.

5.4.1 First Experiment

Results of the first experiment are shown in figure 5.2. The results were divided into smaller sub-graphs, one for each of the eleven objects, to examine the participants' preferences with respect to the synthesis methods per object. More precisely, their opinion on which of the two methods is closer to the recording of the real-world object (used as reference).

Participants were asked to move a slider between the integer values -3 and 3. Positive values correspond to a preference for the filter-based method, negative ones to the sinusoidal method and zero values mean that both methods sound the same. Possible answers are

displayed in table 5.2. Every different tick of the x-axis of figure 5.2 corresponds to a different "sound area" of the object. From left to right the areas go from the top to the bottom.

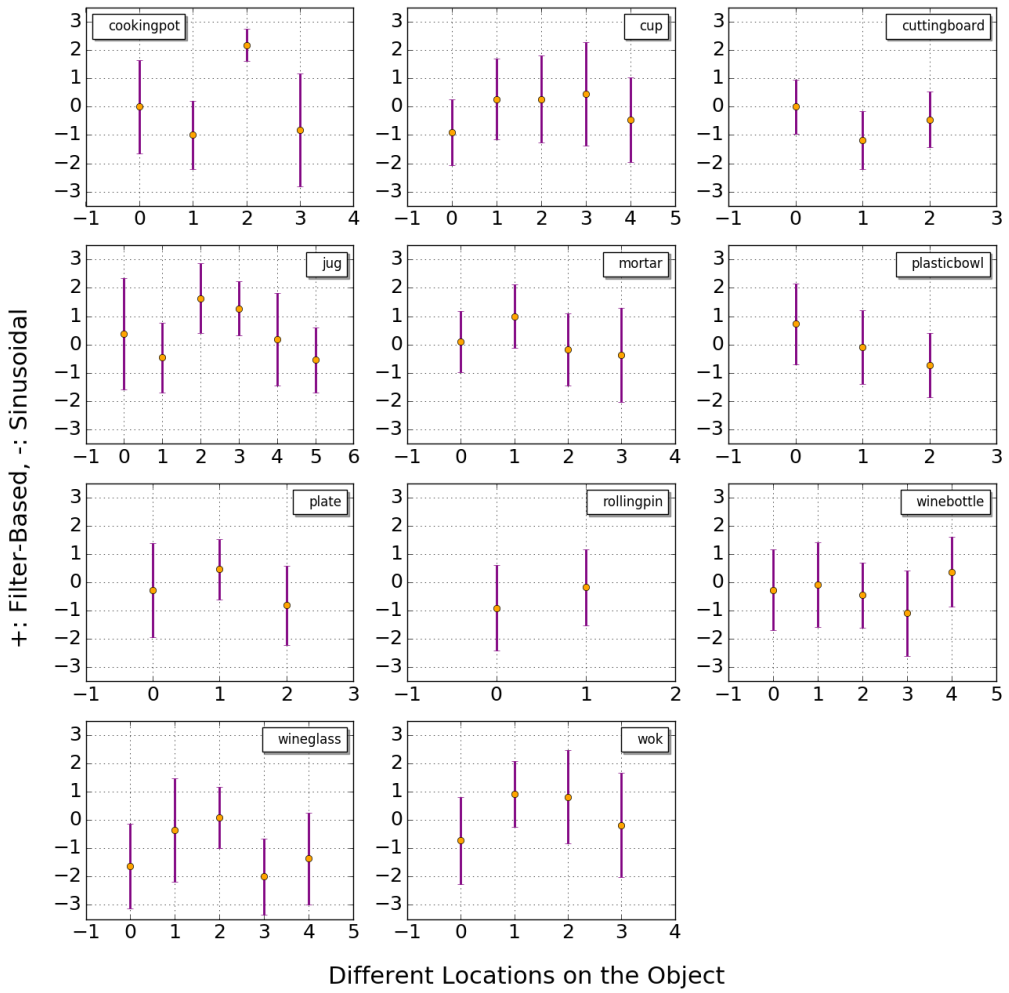


Figure 5.2: The mean values and standard deviations per location for all objects. Positive values give a preference to the filter-based method, while negative ones give preference to the sinusoidal method. Participants choises were 0: both sound the same, 1/-1: the chosen is slightly better, 2/-2: the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

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.

This experiment examines which of the two synthesis methods - filter-based modal synthesis or sinusoidal additive synthesis - used in this thesis is more accurate. From figure 5.2 it can be seen that there is no universal answer for all objects. More specifically, there is a tendency for the sinusoidal method for the glass (bottle and wine glass) and the wooden (cutting board, mortar and rolling pin) objects. On the other hand, metal objects (cooking pot and wok) and plastic ones (jug and bowl) seem to sound more realistic with the filter-based method. Finally, for the ceramic objects (cup and plate) both methods have similar results. It is important to notice that ceramic material lays, also, in the middle of the Q values. Those results are summed up in table 5.3.

Plastic	Wood	Ceramic	Glass	Metal
Filter-based	Sinusoidal	Both	Sinusoidal	Filter-based

Table 5.3: Preferred synthesis method per material.

5.4.2 Second Experiment

For the second experiment the results were divided into sub-graphs for every one of the objects. They are presented in figure 5.3. This experiment examines the participants' preferences between an impact sound that varies depending on the struck location and a single impact sound per object.

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 5.4: Possible answers for the 2nd experiment.

Participants were asked to use a slider similar to the previous test, with possible answers shown in the table 5.4. Positive values correspond to a preference for sound variation, while

negative ones correspond to a preference for having a 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.

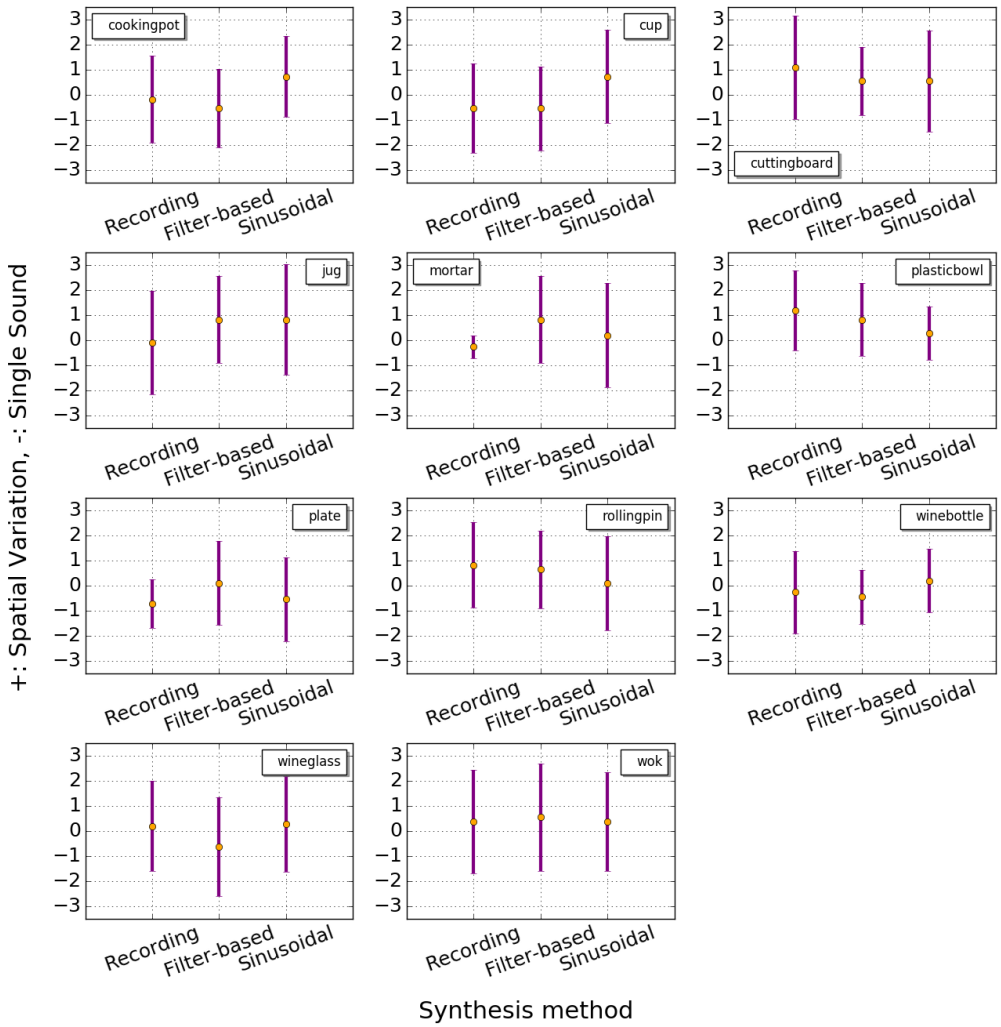


Figure 5.3: The mean values and standard deviations per method for all objects. Positive values show a preference for sound variation, while negative ones show a preference for one single sound per object. Participants' choises were 0: both sound the same, 1/-1: the chosen is slightly better, 2/-2: the chosen is better and 3/-3: the chosen is much better.

The aim of this experiment is to prove whether sound variation for objects within a game are desirable and whether it makes a difference. By choosing the positive values, participants would prove that having spatial variation for impact sounds is more appealing than just having one repeated sound. Although the results do not provide a clear conclusion, the participants' answers lean towards this belief. In other words, sounds from seven out of eleven objects were chosen in the positive range. In addition, the rest of the objects' sounds have an average of

about zero, which means that both examined sounds are very similar. This might have been caused due to the lack of rotation during the fall of the object on the platforms, especially for the cooking pot and the plate. That is to say that there was probably not enough variation on the recorded sounds or the sequence of sounds was not the most appropriate, which might have influenced some of the participants answers.

5.4.3 Third Experiment

Figure 5.4 shows the results of the third experiment. This one was held to validate the chosen values of the Q factor that corresponds to each material. For the purpose of this thesis, one value per material that sounded closer to reality was chosen. Those selected values are shown in table 4.1.

Participants heard a range of materials assigned to the same object every time and were asked to move a slider if a change was noticed. The four changes that they had to identify are: plastic to wood, wood to ceramic, ceramic to glass and glass to metal.

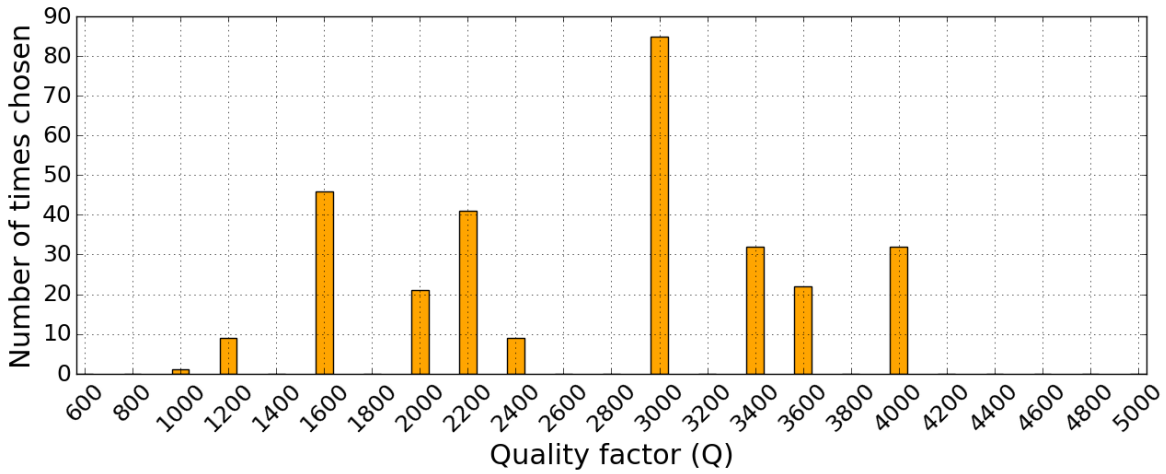


Figure 5.4: The chosen values of the quality factor that correspond to material change of the same object.

By studying figure 5.4, it can be seen that the values chosen by the participants in the experiment are similar to the chosen ones by the authors. The first change (plastic to wood) is chosen to be at value 1600, while our choice is 1500. Then, there is a change around value 2200 which does not correspond to any of the examined materials. The second change (wood to ceramic) was chosen by the majority of the participants to be at value 3000 (the same as our choice). Afterwards, the change from ceramic to glass is split between the values 3400 and 3600, when 3500 is our chosen value. Finally, the last change (glass to metal) lays on value 4000, which corresponds to our choice. As far as the extra change noticed by the participants is concerned, opens a discussion on whether a sixth material to be placed between wood and ceramic in the Q range should be included.

Chapter 6

Analysis

6.1 Results

This section makes an assessment of the final product and its results by pointing out what went well and what could be improved.

6.1.1 Evaluation

Overall, the work of this thesis proved successful. Procedural audio is used to replace prerecorded sound effects in a virtual environment. The sounds are generated using modal synthesis and based on physical models which are excited by different interactions driven by physics events in the game engine. The tool implemented enables the sound designer to control the synthesizer by choosing different materials for every object, the roughness of its surface and size.

Sounding Objects

The tool provides ready-made Unity® prefabs and their corresponding audio components attached. The sounds produced by the proposed prefabs are object-specific, meaning that the modal data they have assigned matches their physical properties. The effect of this is a sound which is tightly coupled with its corresponding audio source. However, one can use the modal data of a certain object for a different shaped object, if it fits their needs.

A drawback that entails from the aforementioned is that incorporating new objects in the game scene, that can benefit from the tool, is not a straightforward task. To ease the process a detailed guide that walks through the steps to add new objects and obtain their modal parameters is available in the appendix D.

Sound Variation

Since objects were separated into different sounding areas, it was easier to assume that all materials are homogeneous and isotropic. This assumption decreases the realistic aspect of sound variation along an object's surface which would ideally produce a different sound in every location. The higher resolution of the object's surface the more accurate will the sound on its surface be. This however results in higher calculations and higher amount of necessary recordings for every one of the areas. Therefore, a compromise is needed between the amount of "sound areas" and the realism of the varying sound. Section 3.2 explained how

the objects were divided into areas with a heuristic approach. The sounds generated by the objects within a virtual scene are compelling from the point of view of the authors and thus validated this approach.

Synthetic Sounds vs Recordings

Regarding the quality of the synthesized sounds it can be said that in general, sounds that present high damping are less suitable for modal synthesis than sounds with low damping.

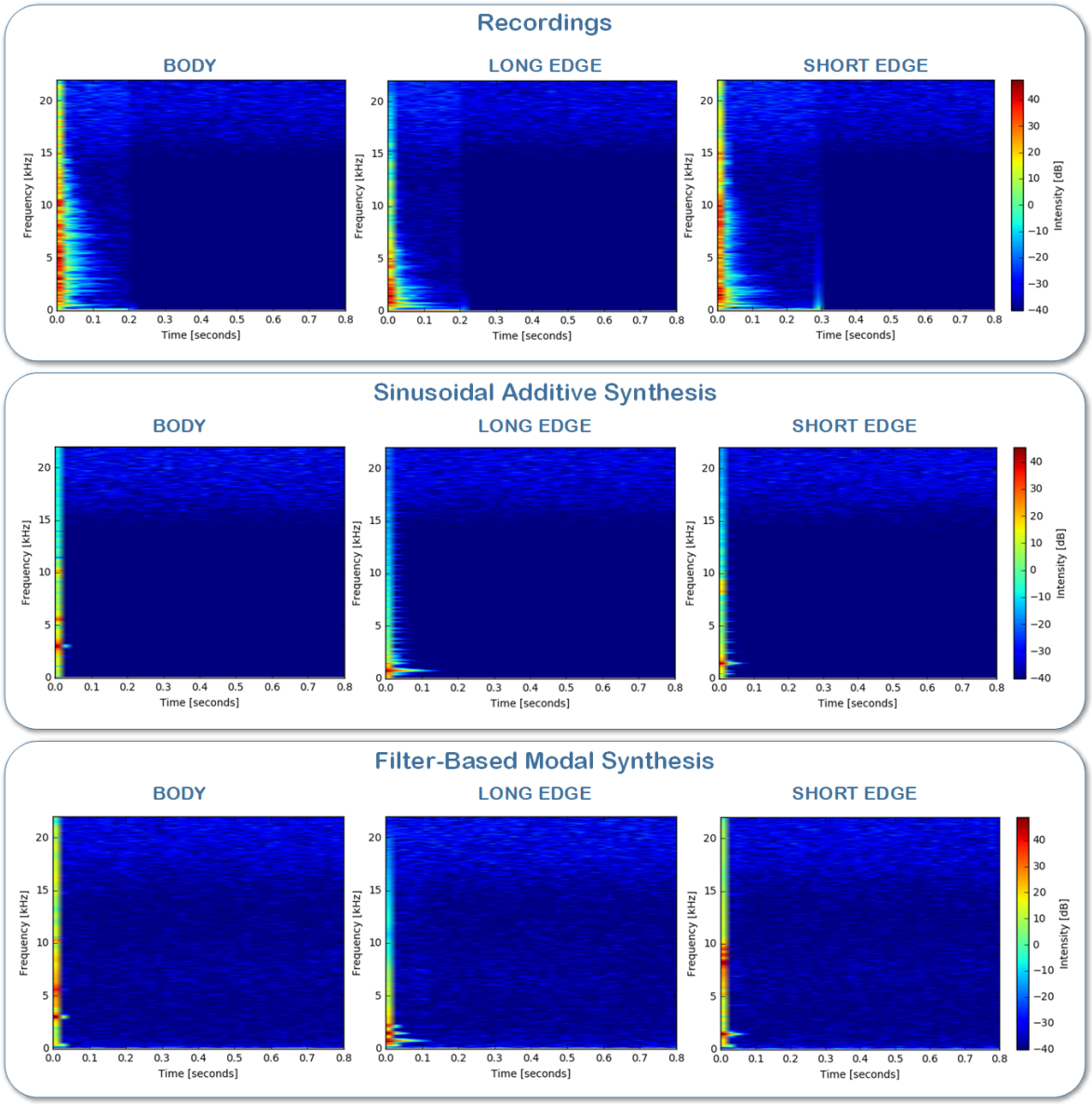


Figure 6.1: Spectrograms of a cutting board object's real-wolrd recordings and synthesized impacts, using sinusoidal additive synthesis.

Sounds produced by highly damped materials such as plastic or wood present spectrograms where information is spread in a wide frequency range. In this thesis the ten most prominent modes were chosen, which leads to loss of information and thus a decrease in the richness of the sound. Nevertheless, it was possible to synthesize sounds coming from these

types of objects. For example, the wooden cutting board, used in this study, is broad in frequency domain as seen in the first row of figure 6.1. However, the synthesized sounds' spectrograms show that our method is able to keep the most important information despite some frequency components loss.

On the other hand, sounds coming from low damped materials such as glass or metal present very clear frequency peaks. As seen in the spectrograms of the wine glass recordings in the appendix C, few peaks stand out among the rest. For this reason most of the information can be synthesized within ten modes. It can be seen on the spectrograms corresponding to the synthetic sounds of the wine glass that the prominent peaks are present. This is validated, also, through unofficial auditory perception tests.

Without leaving the realm of low damped materials, it is worth mentioning that in some cases metallic sounds cause undesired distortions. More specifically, for Q factor values over 3100, a distorted sound can be heard when the object is hit in different areas in a short time-lapse. The origin of this problem seems to be the change of the modal frequencies (as certain areas have different modes than others) that are fed into the filters while the previous sound is still "ringing".

Other inaccuracies in the synthetic sounds may derive from both the recording and parameters extraction phase. This thesis does not calculate the residual sound, namely the differentiation between the real-world sound and the synthesized sound [Ren et al., 2013]. Therefore, information that concerns mostly the beginning of the sound (when the collision happens) is not calculated and is said to be perceptually relevant to identify the sounding object [Freed, 1990]. However, the reason it was not included was precisely because the noise produced by the striker (a drumstick in this case) was not desired in the synthesized sound. This is because in the game scene the characteristics of the striking object are unknown. Another aspect that could have affected the synthetic sounds is again the sound of the drumstick that may have interfered in the data extraction phase.

Using procedural audio in games is much more convenient and less expensive as far as storage space is concerned compared to the use of audio files. In this paper the plugins corresponding to the filter-based synthesis method occupy 523 kB while the ones for the sinusoidal method are 772 kB. The total memory needed to store the recordings of the struck objects is 5.74 MB. The space saved with the synthesis techniques is indisputable. Moreover, procedural audio is adaptive to several different scenarios without further change or additional memory needs, as it is flexible and dynamic. For example, with just a slider, size adjustment is achieved. Adding new sounds is less time consuming and it is better for mobile platforms that are limited in both storage space and computational power.

Controlling Parameters

The tool itself is easy-to-use with the already existing objects as one can control sound properties through high level parameters that are intuitive for the user. Those controllers which are explained in section 4.4 are neither *weak* nor too *strong*. As stated in [Jaffe, 1995], weak parameters affect the result so little, that there is a possibility the designer will adjust them to arbitrary values as he is "wandering in the dark". 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. In this thesis certain functions were used to offer coherent sliders that allow an efficient control.

Using the provided Unity® prefabs as reference, their data can be modified to reflect variations of the object's scaling, surface roughness and material properties. Those three parameters were chosen as they are physical and perceptual sound descriptors [Aramaki et al., 2009b]

and thought to be intuitive enough for a non-expert user. Unity® offers the possibility to use one dimensional sliders which game developers are used to implementing for game designers to tweak parameters that affect the game. This is why it made sense to use these kind of sliders to control the synthetic sounds. Moreover, the parameters chosen *can* be translated into a one dimensional slider. Despite this, and as commented here below, two dimensional controllers could have given a greater control over certain parameters such as the object's roughness or the object material.

Figure 6.2 displays waveforms of a rolling bottle with different surface roughness values. As the roughness parameter is increased, the sound generated by the bumps along the surface becomes more audible. It is noticeable by looking at the waveforms that the rougher the surface, the louder are the synthetic sounds derived from the micro-collisions. This evidence matches with the graph 4.10 from section 4.3.3 that shows how the amplitude of the impulses corresponding to the bumps increases with the roughness of the object. An improvement to customize the surface irregularity of the object would be to implement a two dimensional graph that maps the object's contour profile with a curve (similarly to the curves in 4.10). This way the user could fine-tune the roughness of the object with a more visual controller.

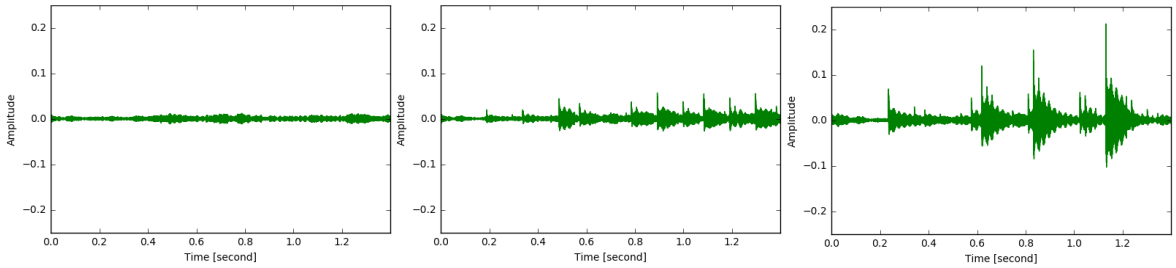


Figure 6.2: Waveforms of a bottle object with surface roughness variation (from very smooth to very rough), rolling on a platform.

Figure 6.3 shows spectrograms of a bottle falling over several platforms with different materials assigned to it, starting from plastic on the left and ending with metal on the right. Although the impact sounds are produced at the same moment of time, it can be seen that the "tails" of the individual impacts last longer towards the rightmost spectrogram which corresponds to metal. This proves the fact that the lower the damping (as for metallic sounds) the longer the sound lasts. In the context of this thesis a one dimensional slider was sufficient to determine the material of an object as it is based on one parameter which is the damping. However, some studies [Aramaki et al., 2009a] categorize materials depending on more than one parameter (global damping and frequency-relative damping) which would benefit from two dimensional pads to control the sound

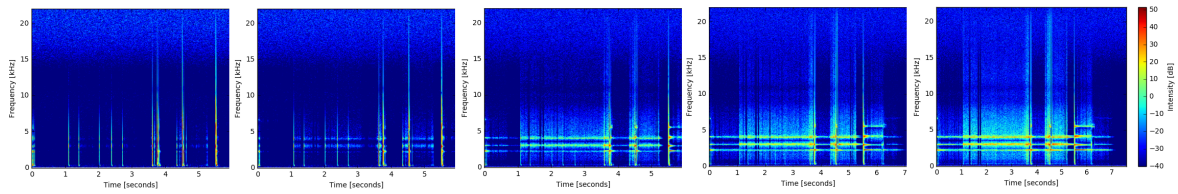


Figure 6.3: Spectrograms of a bottle object with material variation (plastic, wooden, ceramic, glass and metallic respectively).

Spectrograms of the synthesized sound produced by a struck cup with different sizes (from smaller to bigger) can be seen in figure 6.4. Resonant frequencies are inversely propor-

tional to the size, thus the pitch goes down when the object gets bigger. The Unity® slider here fits perfectly as size can be expressed as a one dimensional parameter.

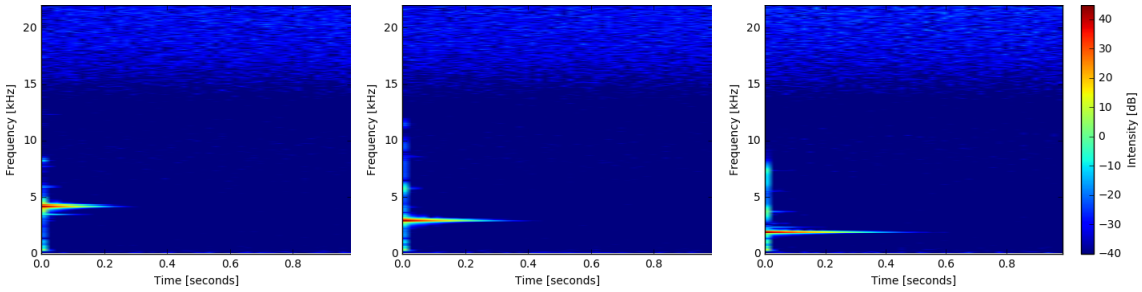


Figure 6.4: Spectrograms of a cup object with size variation (smaller, reference size and bigger respectively).

All controllers offer the possibility to be modified at run-time which is an advantage as changes in sound can be experimented on the go. Despite this, some small disturbances can be heard when controlling the *Size* slider as it modifies the central frequencies of the filters or the oscillators as a sound is already "ringing".

Integration of the Tool

The tool's UI has been implemented similar to Unity®'s UI so that game developers feel familiar with the controls and can use it without having to learn how to use a third party audio solution. The goal is to assign a procedurally generated sound to an object in the game scene as easily as one can apply a material texture. Therefore, not only sound designers who desire more realistic sound effects are targeted, but also game developers who are used to the Unity® editor and have no further sound design knowledge. Some game developers were consulted regarding the integration of the tool within the game engine and endorsed this approach.

As a side note, it is important for the designer to remember to press the *"Apply"* button, on the prefab, every time a change is made, otherwise it will be undone.

6.1.2 CPU Demands

Synthesizing sounds for applications in real-time instead of using a large bank of prerecorded clips is a good solution for the data storage problem. This is particularly a critical issue for standalone devices such as the HoloLens headset [Microsoft,] which presents storage and memory restrictions. Audio systems require sufficient CPU to operate when usually they have to compete with other systems such as graphics, physics and artificial intelligence (AI) [Lloyd et al., 2011].

In the tool presented in this paper 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, it can be seen that except for the initialization stage where scripts consume CPU power, the rest of the demonstration remains stable in performance and at around $100fps$ (figure 6.5). This performance test was held on a laptop with 4 Intel® Core™ i7-6700HQ CPUs running at 2.60GHz, each with 2 hardware threads.

When testing the limits of the tool, it was found that for the filter-based method a total of 16 objects can be active at the same time before the audio gets distorted. The corresponding number for the sinusoidal method is 12 objects. This could be due to the fact that the latter

method completes more calculations when computing the amplitude envelopes of each impact sound. Those numbers were obtained on the laptop mentioned above, using the *Unity® Audio Profiler Window* and represent the amount of *Playing Audio Sources* without the *Total Audio CPU* exceeding 100% (see figure 6.6).

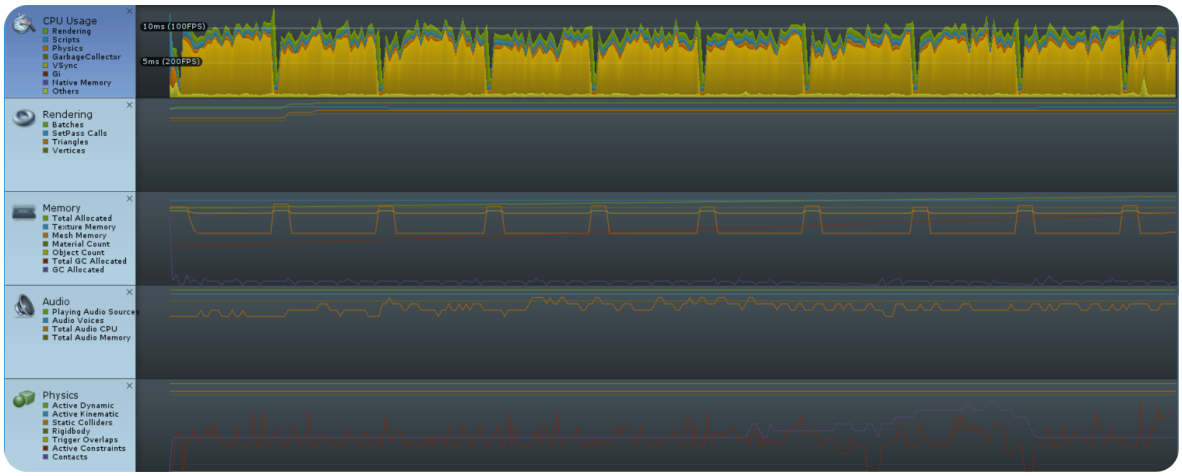


Figure 6.5: The profiler view of Unity® editor when a wine bottle rolls down a number of platforms.

Overall, most of the calculations happen at the start of the application, leaving only the identification of the object and its corresponding data to be sent to the audio chain in real-time. Therefore, it can be said that procedural audio does not consume more calculation power than is provided for audio in game and application productions.

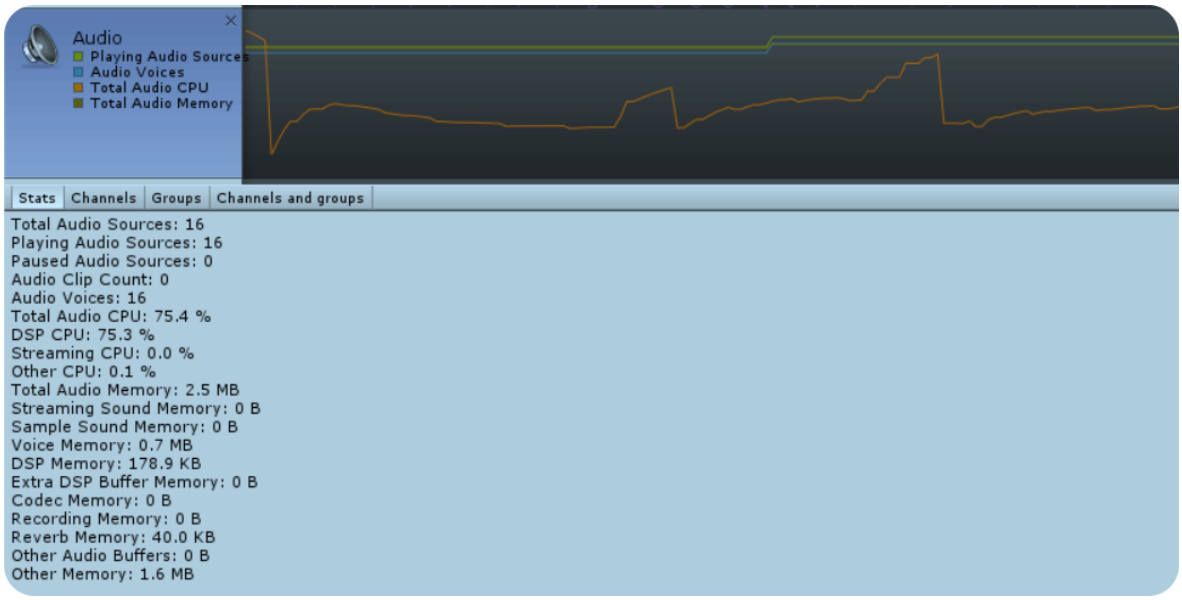


Figure 6.6: The audio profiler view of Unity® editor when a total of 16 objects are enabled in the scene using the filter-based method for audio synthesis.

The following analysis could have been incorporated also in 6.1.1 as it deals with comparing synthetic sounds with recordings. Here, more precisely, the audio CPU usage is commented. The audio profiler of a wine bottle rolling down some slopes and impacting them

making use of recordings can be seen on figure 6.7. Comparing it to figures 6.8 and 6.9 which correspond to the profiler for filter-based and sinusoidal sounds respectively, it can be said that these two latter methods are slightly more CPU demanding regarding audio as the *Total Audio CPU* curve presents higher values. However if the overall CPU usage profiler is observed it can be seen that there are no big differences with the recordings. It can be stated then that procedural audio is not a problem for CPU usage compared to recordings.

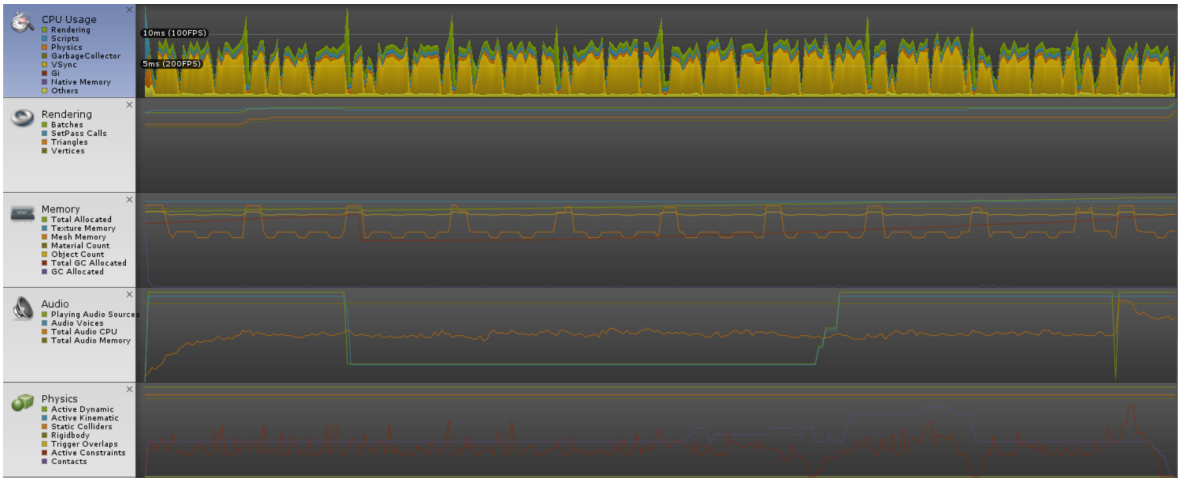


Figure 6.7: The audio profiler view of Unity® editor when a wine bottle rolls and impacts some slopes using recordings.

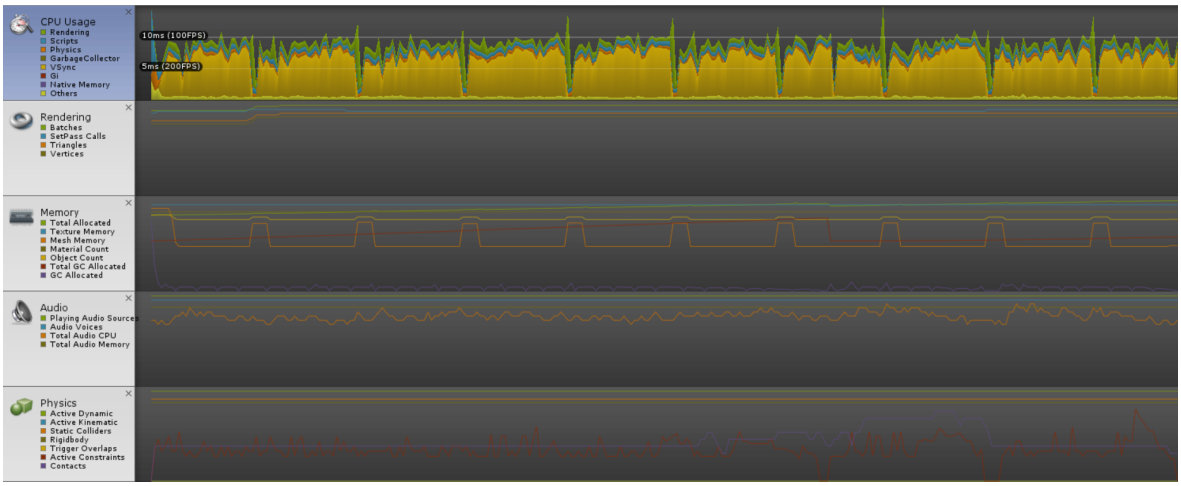


Figure 6.8: The audio profiler view of Unity® editor when a wine bottle rolls and impacts some slopes using the filter-based method.

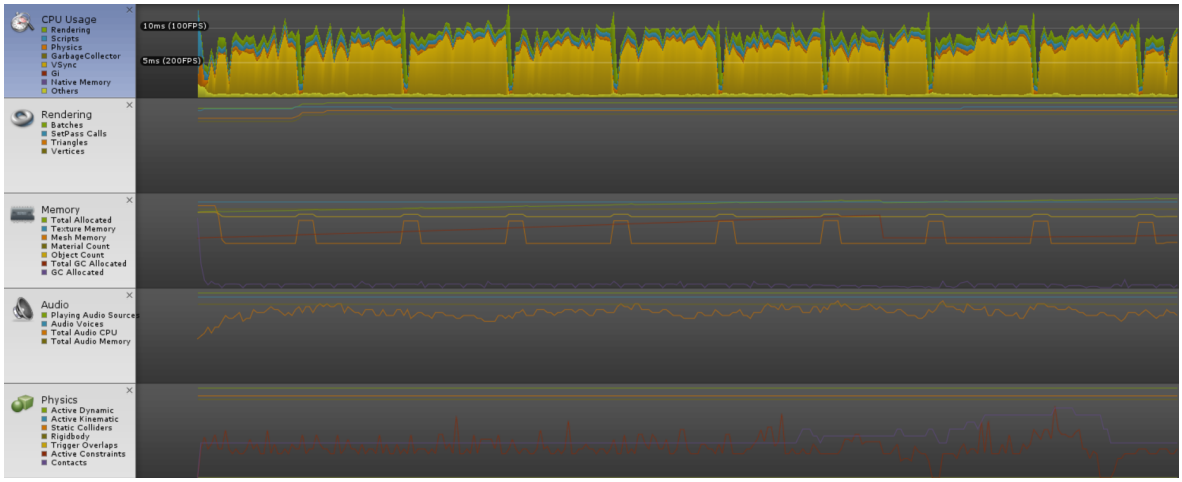


Figure 6.9: The audio profiler view of Unity® editor when a wine bottle rolls and impacts some slopes using the sinusoidal method.

6.1.3 Which Synthesis Method Is Better?

After analyzing the results of the listening experiments from section 5.4, it can be stated that there is not much difference between the two methods used to synthesize modal sounds. Each object, with its characteristic physical attributes, gives slightly different results. However, a division per material is possible to arise as seen in table 5.3 although no clear conclusions can be extracted from it. A synthesis method does not seem to give better results over another, depending on the damping of the material.

In appendix C, the spectrograms for one object of each material examined in this work are shown. Although there are differences, both synthesis methods follow similar behavior in the frequency and time domain to the recordings.

From a more practical point of view, the sinusoidal synthesis method integrates less well in an environment in which multiple sounds can be triggered in a short lapse of time. This method uses an amplitude envelope that changes over time to simulate the sound decay. The problem arises when a second sound is triggered while the previous one's "tail" has not yet finished "ringing". This leads to the abrupt stop of the first sound in detriment of the second. To correct this, three envelopes have been implemented so that up to three sounds coming from the same object can be played at the same time. On the other hand, this issue does not occur with the filter-based method which is able to handle several impulses at a time. For this same reason the filter-based method is chosen for rolling and scratching sounds as these imply numerous micro-impacts in a short time lapse. Therefore, this method integrates better in interactive applications which demand a high degree of versatility as far as contact sounds are concerned. Moreover, the filter-based synthesis allows more spatialized audio sources to be active before being distorted than the sinusoidal method, as seen in section 6.1.2. It can be therefore said that the filter-based method is not very different to the sinusoidal one as far as sound quality is concerned, but overall it is more suitable for interactive applications due to its flexibility and integration.

6.2 Discussion

In this section, an overall discussion about the tool's usability is made and future work directions are considered.

6.2.1 What Is New?

The main novelty of the tool presented is that it enables the generation and intuitive manipulation of physics-based procedural sounds within a game engine. This is based on the goal to bridge university research with industry, which often do not progress hand in hand. Additionally, a study between two different sound synthesis methods, namely *Sinusoidal Additive Synthesis* and *Filter-based Modal Synthesis* is carried out. The sounds produced by these methods were tested by performing audio perception experiments on a group of participants to find out whether one of them offers better results than the other. The tool offers the possibility to synthesize three kind of contact sounds (impact, rolling and scratching) that can be controlled through high level parameters. All of this being available in a package that is easily integrated in Unity® and ready-to-use for game developers.

6.2.2 Tool Applications

This tool can be used for the development of all sorts of games that include collisions and interaction with physical objects. They can vary from indoor AR applications, VR games or simulations, to open world environment games running on consoles. However, in its current form it is recommended to be used for indoor games, since all prefabs available are every day objects found in a kitchen. Designers are highly advised to use it for AR applications, in which the synthesized sounds' realism is compared to the sounds of the real world.

6.2.3 Why Can It Be Used In AR/VR?

Virtual and augmented reality are becoming more and more widespread technologies. There are multiple applications where they can prove to be useful both in everyday life and entertainment. Focusing on AR, this thesis' goal is to develop a framework for sounds produced by the interaction of objects. The sounds are designed to be realistic and physics-based so they can adapt into an AR environment where real sounds and virtual ones coexist. Furthermore, thanks to the memory storage solution that procedural audio entails, it is possible to use our tool on portable devices with small storage space.

VR devices, although they do not have storage problems since they are connected to PCs to work, they, as well as AR, require flexible sounds that can adapt to any virtual scenario. In addition the tool allows sound spatialization, which suits both technologies that simulate three dimensional worlds.

6.2.4 Future Work

The implemented tool is able to transfer an object's physical attributes from recordings to synthesized sounds. However, certain changes could be made to improve the accuracy of the synthesis and its cost from a computational point of view.

To improve the realism of the sound produced by a vibrating object in a virtual scene, sound radiation should be taken into account. As described in [Corbett et al., 2007], not only the location of the excitation point controls the timbre of the sound, but also the location of the listener. The radiation, namely the energy transferred to the listener's ear, depends on his location and it is calculated through the *acoustic transfer function*. Adding this component would enhance significantly the feeling of immersion in the virtual environment.

Another improvement that could be done is the early spatialization of the sound, in the synthesis process. More specifically, instead of dividing the process into two steps, namely the synthesis of the monophonic sound and then the spatialization, another option would be to do it at the same time as seen in [Verron, 2010], using the location of the object and

listener. This approach can be accomplished by cascading the band-pass filters of the filter-based method with spatial filters and it would save calculation power for sounds that are very far away from the listener. The concept of dynamic level of audio detail is interesting for the creation of sounds that would present a variable cost as stated by Farnell [Farnell, 2010]. Similarly to the compression of audio, like with the MP3 format, less resources would be used as the sounding object moves further from the listener.

Furthermore, some future work regarding sound propagation in rooms and occlusion produced by the environment could be done. Some papers have studied the sounds produced by objects that break into smaller pieces which could be another direction to explore [Zheng and James, 2010].

Chapter 7

Conclusion

A sound designing tool has been presented in this study, where two different synthesis algorithms are implemented. The goal is to achieve physics-based procedural audio in a virtual environment in the Unity® game engine which can be manipulated through intuitive controls.

A number of everyday objects which are separated into different "sound areas" are recorded by striking each one of them at these different surface location. 3D models of these objects are created, matching their shape and dimensions. By analyzing the recordings with DSP audio algorithms, it is possible to extract the modal data necessary for the sound synthesis stage. This data is fed into the modal synthesis model which is driven by physics events from the game engine. This process transfers successfully the sound properties of the real world objects to their corresponding virtual ones but has some limitations. Further extension of the tool is possible by adding more types of objects while following the same procedure.

Three different kinds of contact sounds have been synthesized; impact sounds and continuous contact sounds, namely rolling and scratching. The tool's UI, with high level adjustable parameters, enables the designers to customize the sounds according to their needs. A metallic object can be transformed into a wooden one just by adjusting a slider. In addition, having the original objects' sizes as reference, designers are able to scale the produced sound together with the object. Finally, the surface's roughness can be fine-tuned to match the desired sound of a rolling object.

Listening experiments performed to several individuals did not favor noticeably one synthesis method over the other. The methods presented better results for certain materials and *vice versa*. However the filter-based method was proven to integrate better within an interactive scene thanks to its flexibility. Moreover, it was confirmed through user testing that sound variation along an object's surface is desirable and is preferred over just a single sound per object. Finally, the distinction of the different materials for a range of values of the Q factor was identified well from the participants. Hence, the use of a one dimensional slider to shift from one material to another was justified.

Although the extraction of data from recordings offers computational efficiency, it is impossible to include the whole sound information without alterations. Thus, the synthesized sounds do not sound exactly like the references, but their source is highly recognizable. In addition, the drumstick used for striking the objects produces a sound as well, which interferes with the desired sound. Another aspect that can improve the recording process is to try to approximate as much as possible the system to a free vibration when holding the struck object in the air.

To conclude, physics-based procedural audio aims to increase realism and the sense of presence in virtual worlds especially with the advent of AR and VR applications. Audio events are tightly coupled with graphic events and complement one another, producing a compelling experience for the user. Despite the numerous studies carried out in this thrilling field, work still needs to be done to ease the process of incorporating physics-based procedural audio in audio pipelines for interactive productions such as computer games or AR/VR.

Appendix A

Pure Data Patches

A.1 Filter-based Modal Synthesis

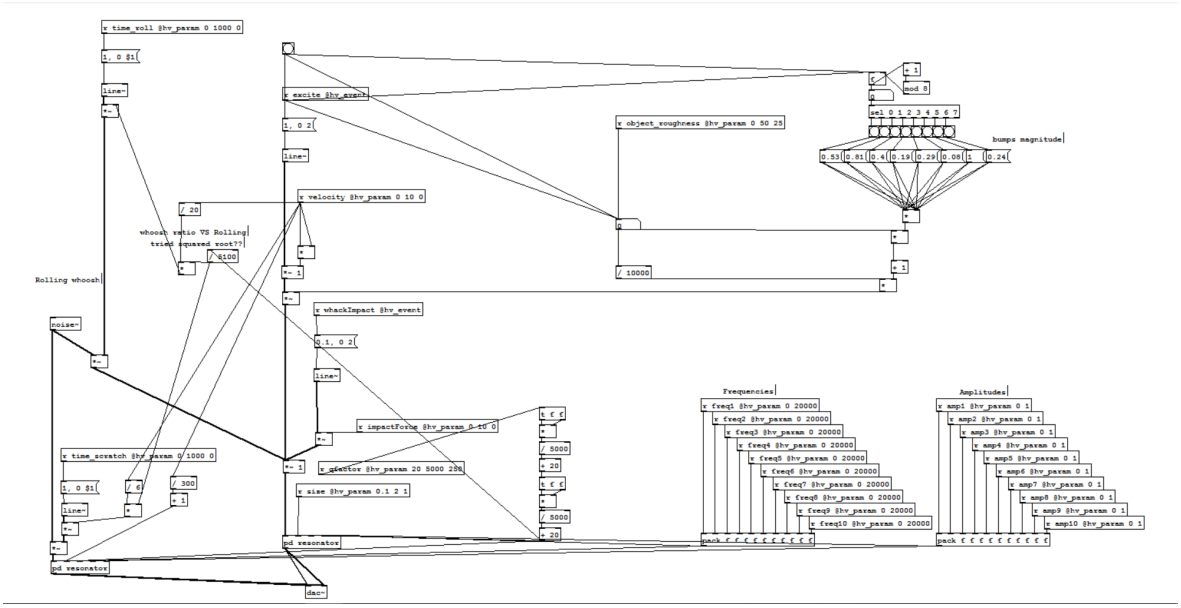


Figure A.1: The main Pure Data patch for the filter-based modal synthesis.

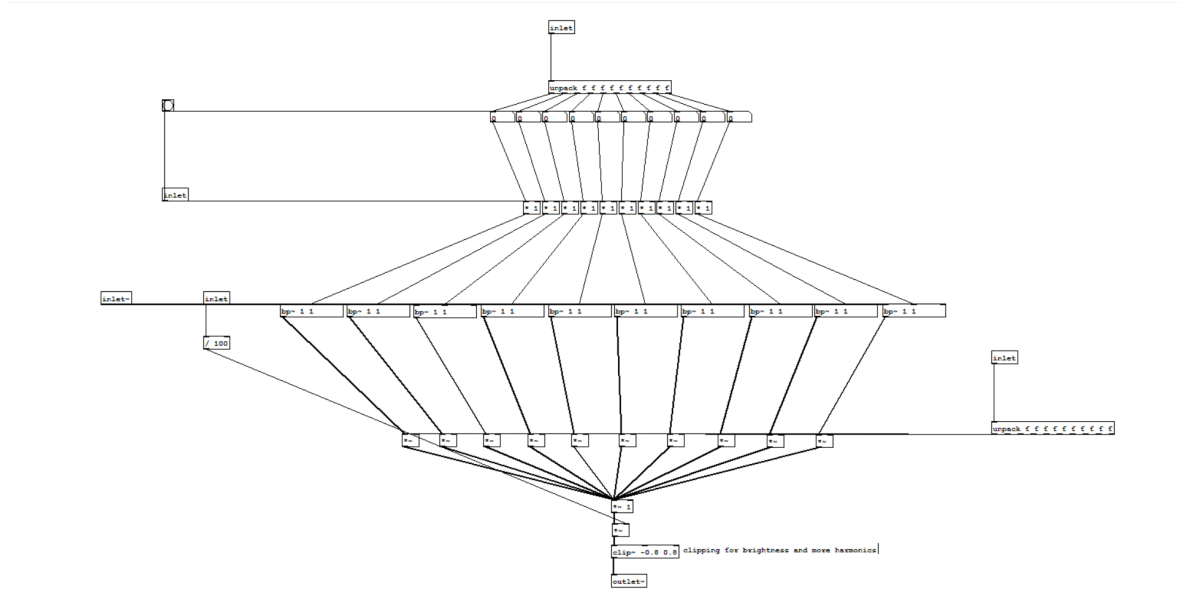


Figure A.2: The *resonator* Pure Data patch for the filter-based modal synthesis.

A.2 Sinusoidal Additive Synthesis

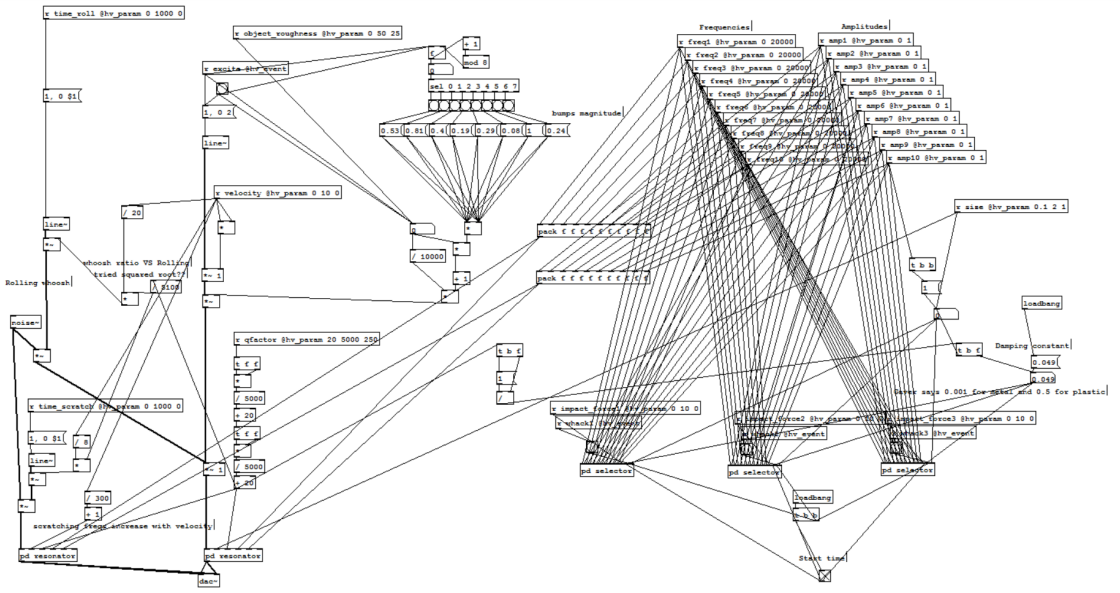


Figure A.3: The main Pure Data patch for the sinusoidal additive synthesis.

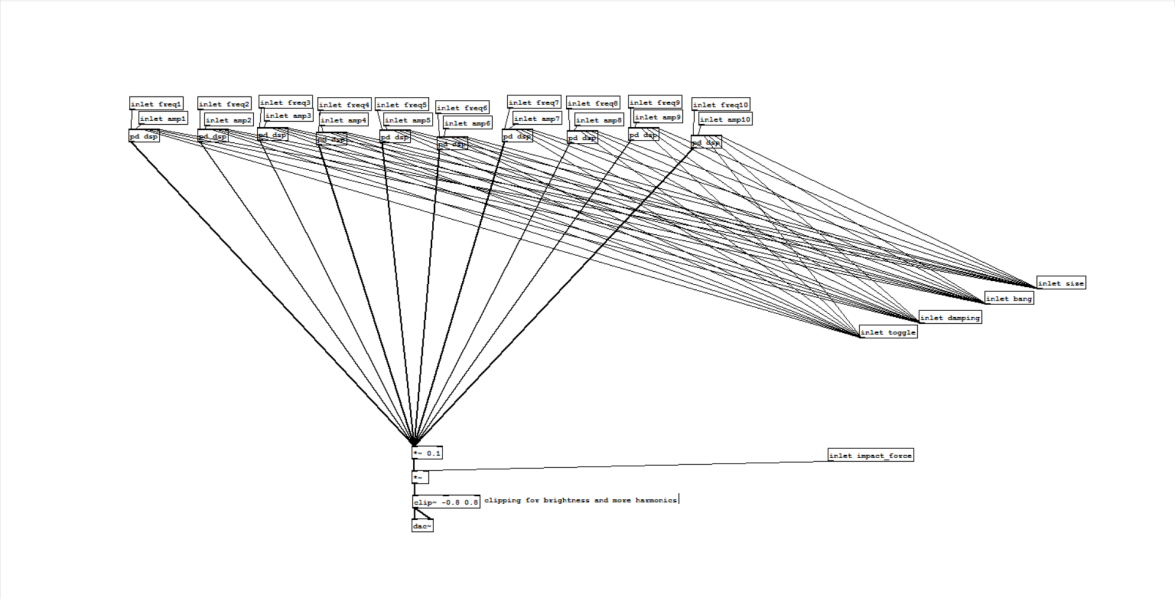


Figure A.4: The *selector* Pure Data patch for the sinusoidal additive synthesis.

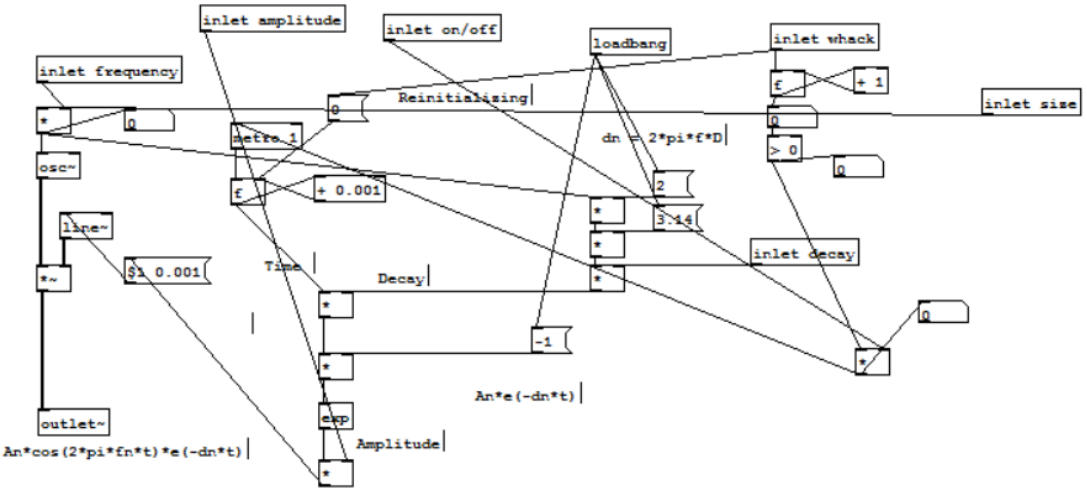


Figure A.5: The *dsp* Pure Data patch for the sinusoidal additive synthesis.

Appendix B

Interface of the Subjective Experiments

The perceptual tests on people use the *A-B* and MUSHRA interface. Through instructions, participants were asked to choose the appropriate file for each test, fill the names of the tester and the test subject and choose a number of options, different for each part of the test (figure B.1).

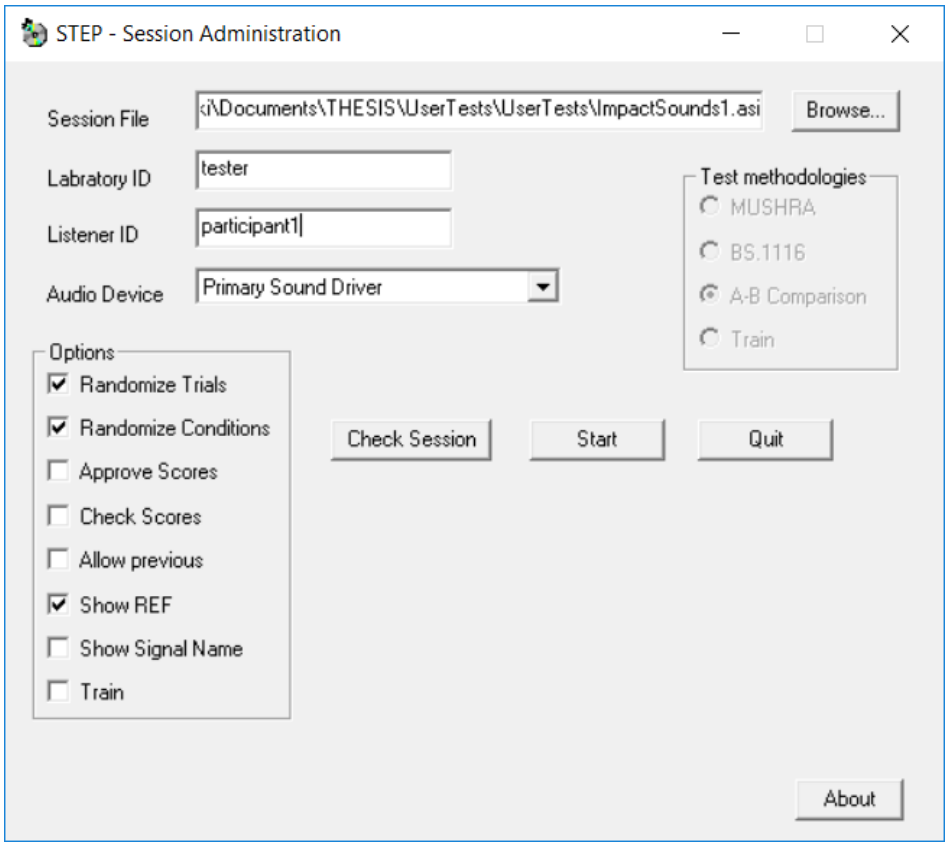


Figure B.1: Settings interface.

First Experiment

The first experiment is an A-B test, where participants are encouraged to choose the closer to the reference between two sounds. All sounds come from struck objects and the reference sound is a real-world recording, while A-B options are synthesized sounds one for each synthesis method. The test was split in two parts, to reduce the fatigue of participants. The options required for this part are the "Randomize Trials", which makes the trials show up in different order every time, the "Randomize Conditions", which randomizes the possibility an audio file to be assigned as an "A" or "B" option, and the "Show REF", which enables the reference sound button to appear.

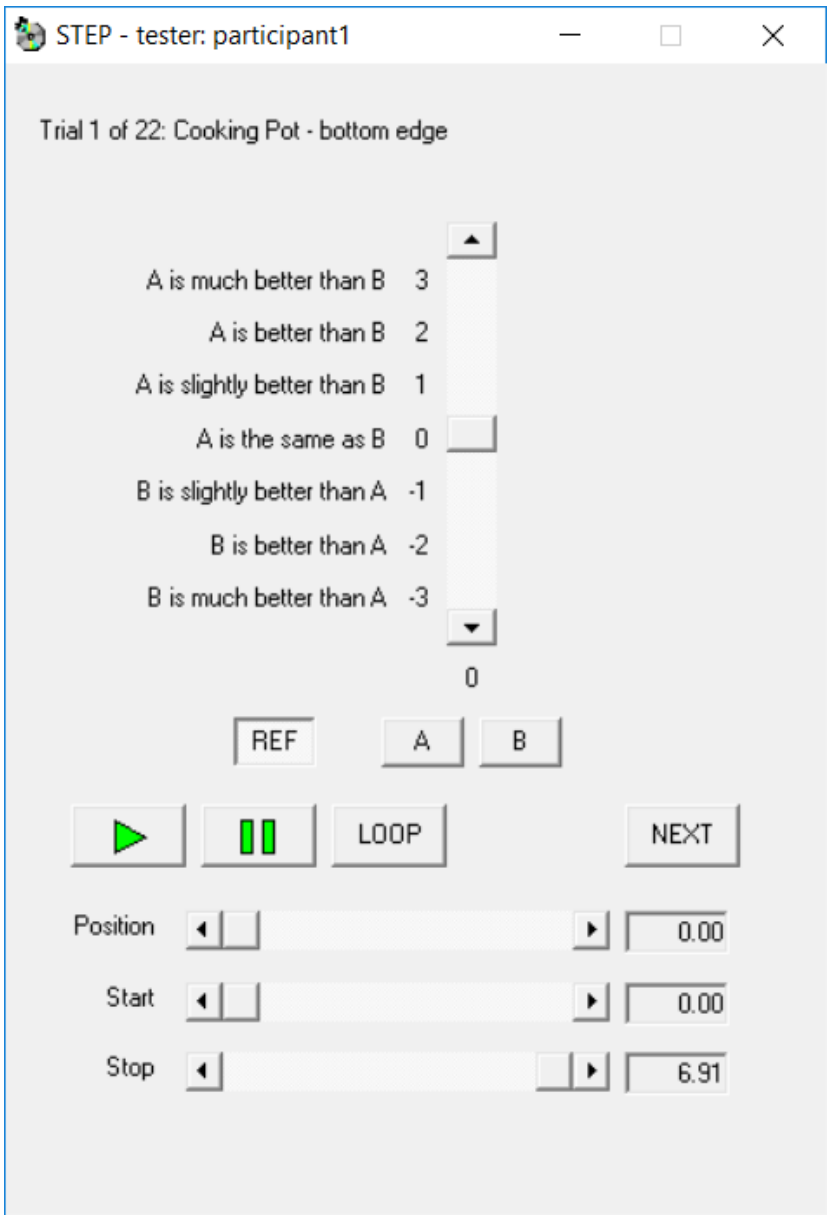


Figure B.2: The interface of the first experiment.

After pressing "Start", the window shown in figure B.2 appears. Participants are prompted to press the "Play" button and the "REF", "A" and "B" buttons, each corresponding to one sound. They are, also, able to loop through the sounds instead of pressing

"Play" every time they want to hear a sound again, or to shorten the sound length using the sliders on the bottom.

After making up their mind, they are encouraged to position the vertical slider in the appropriate position and press "NEXT" to hear the next trial.

Second Experiment

The second experiment is, also, an A-B test and is split in two parts; recordings and synthesized sounds. Participants listen to objects falling and colliding on inclining platforms with and without sound variation. The difference from the previous test is that there is no reference sound, since participants are asked to choose which sound they consider more qualitative. Hence, the options needed for this part are only the "Randomize Trials" and "Randomize Conditions".

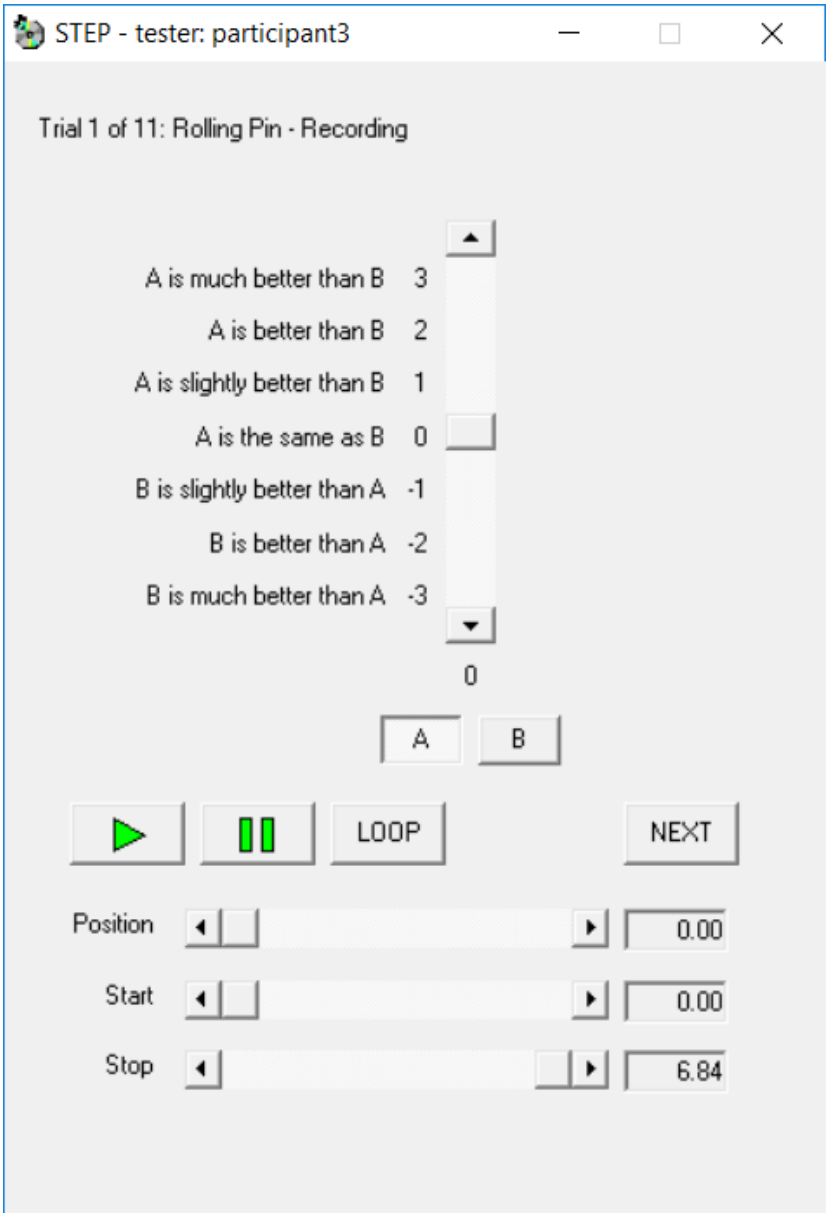


Figure B.3: The interface of the second experiment.

The interface of this test (figure B.3) is similar to the previous one, except for the "REF" button. Participants were asked to follow the same procedure as before.

Third Experiment

The third experiment in a MUSHRA test, where participants listen to ten different sounds produced by the same object falling on inclining platforms, while a range of different materials is assigned to it. The only option required for this part is "Randomize Trials", since the same increasing range of values for Q is necessary. Participants are encouraged to listen the ten sounds and decide where a change in material happened. Then, they are asked to bring the corresponding slider down to zero.

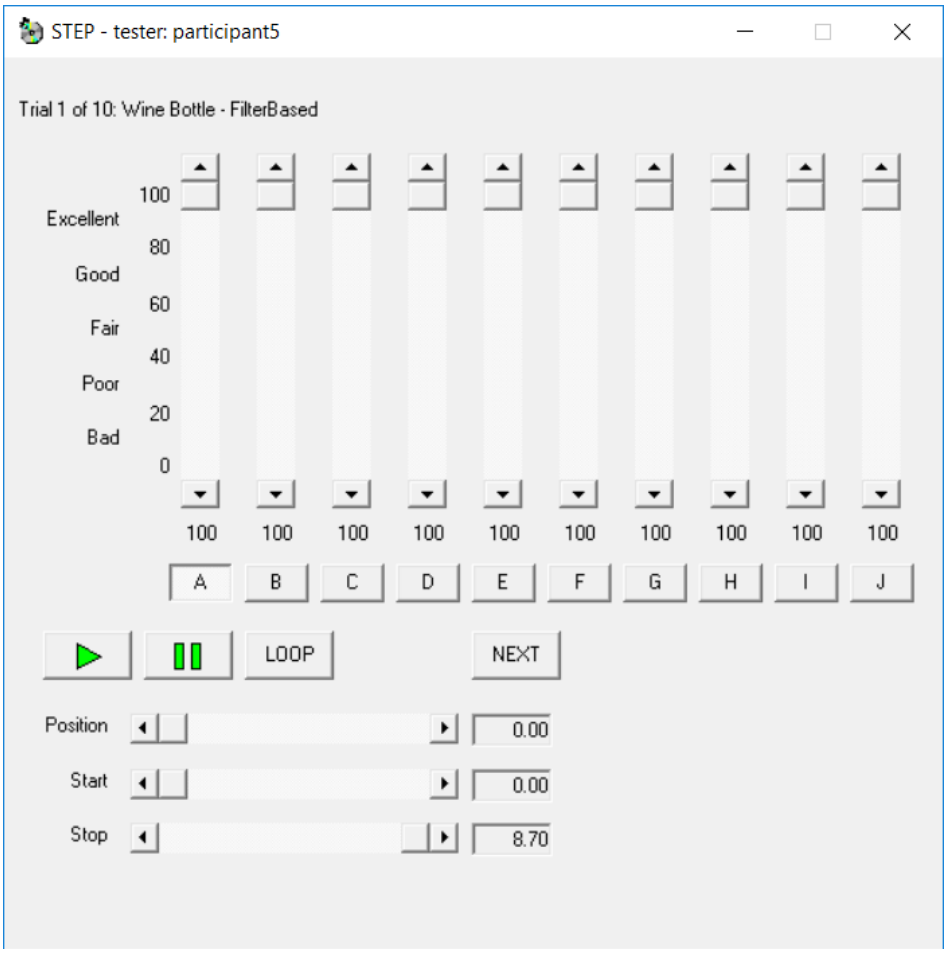


Figure B.4: The interface of the third experiment.

The interface of the third part of the experiment is displayed in figure B.4, where participants were asked to ignore the scaling of the slider and adjust it either at the top when no change in material was noticed, or at the bottom when it was noticed.

Appendix C

Spectrograms

Spectrograms of one object from each material used in this study. Each graph represents sound coming from one "sound area" of the object, either from the real-world recording or from the two synthesis methods (*Sinusoidal Additive Synthesis* and *Filter-based Modal Synthesis*).

Plastic Bowl

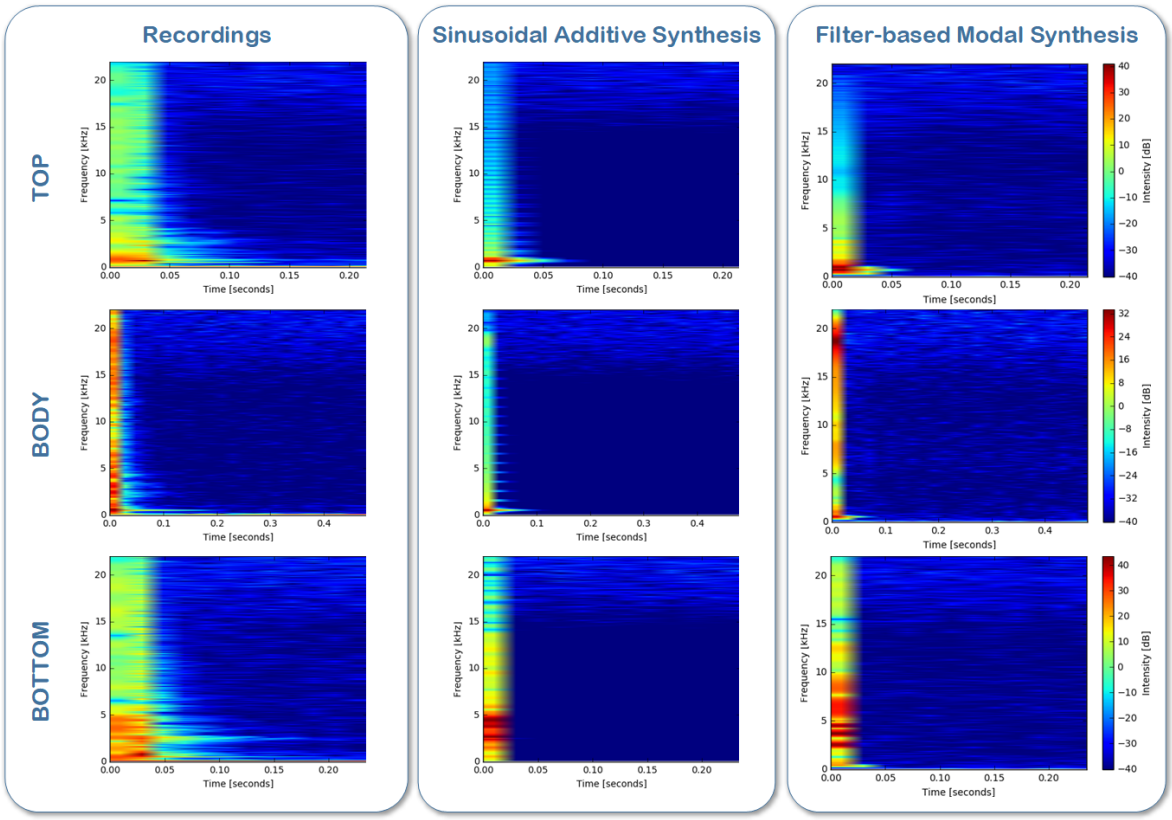


Figure C.1: Spectrograms of recordings and the two synthesis methods for the plastic bowl.

Wooden Mortar

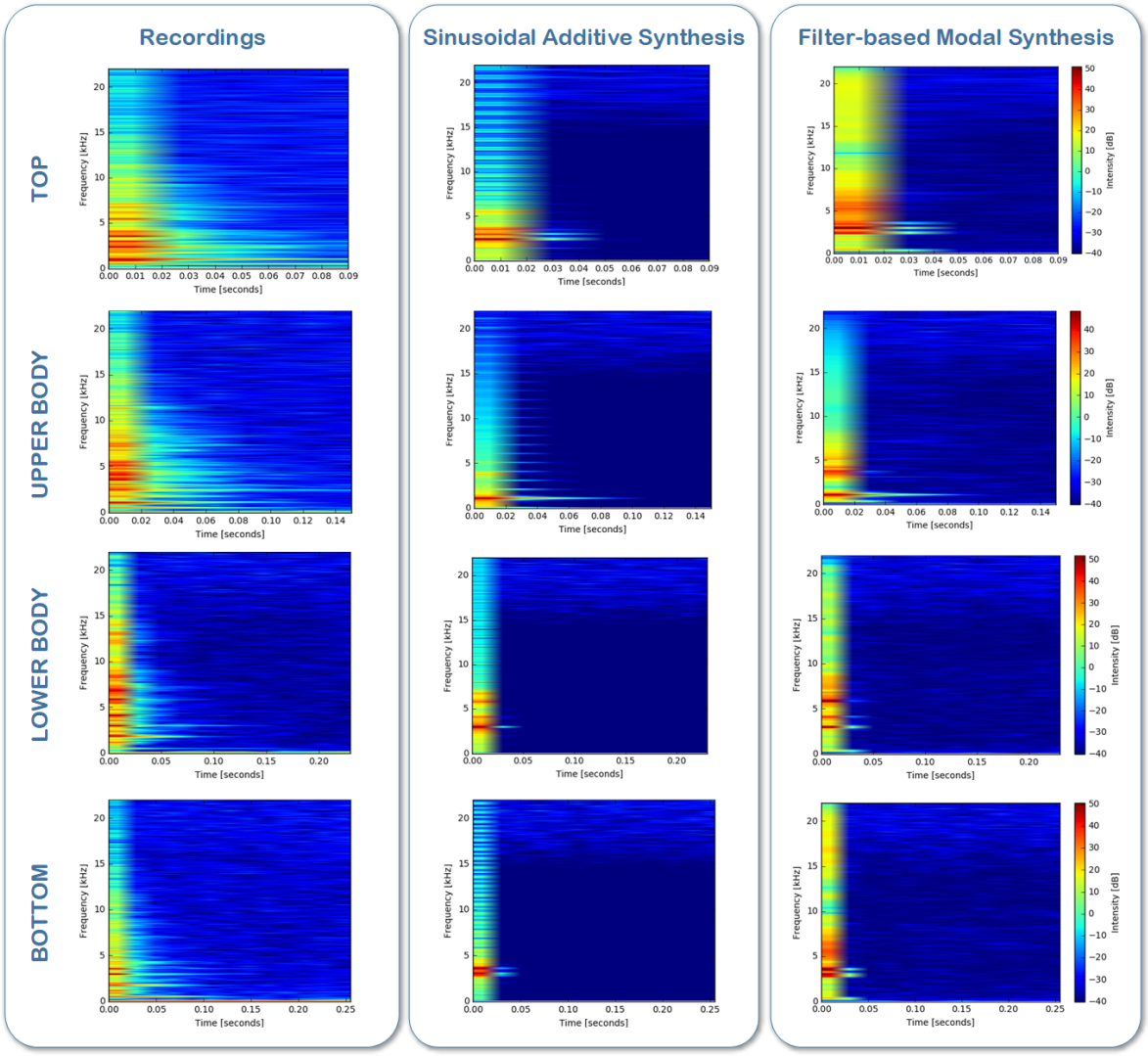


Figure C.2: Spectrograms of recordings and the two synthesis methods for the wooden mortar.

Ceramic Plate

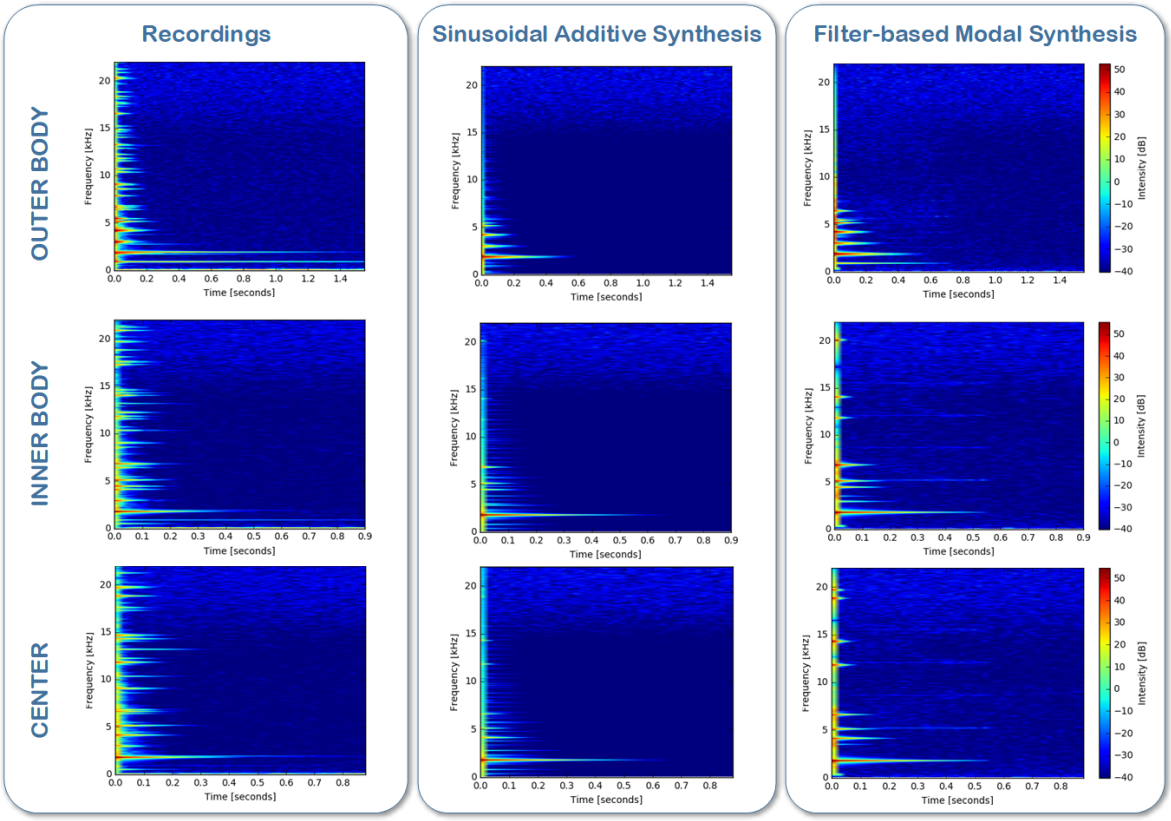


Figure C.3: Spectrograms of recordings and the two synthesis methods for the ceramic plate.

Wine Glass

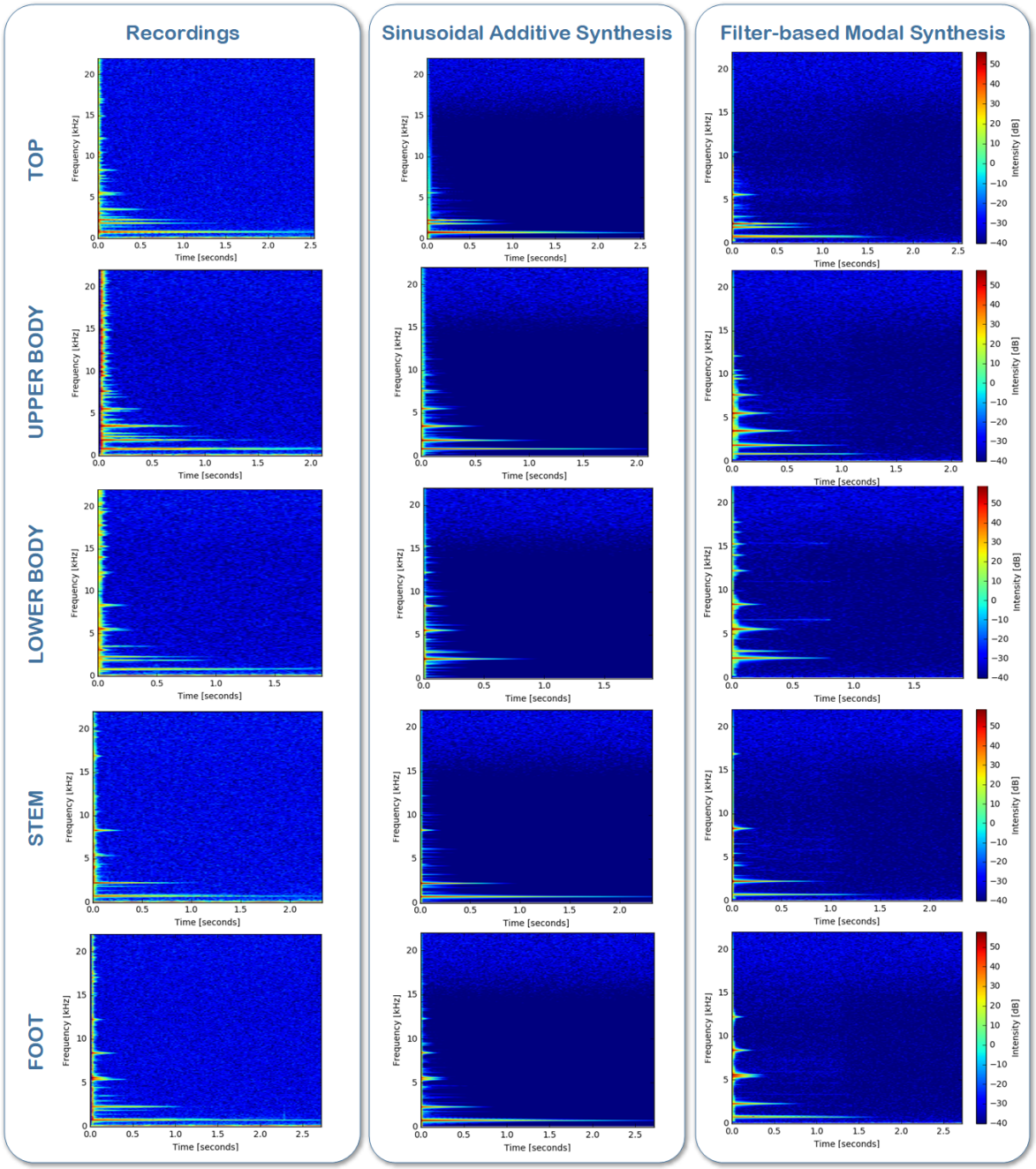


Figure C.4: Spectrograms of recordings and the two synthesis methods for the wine glass.

Metallic Wok

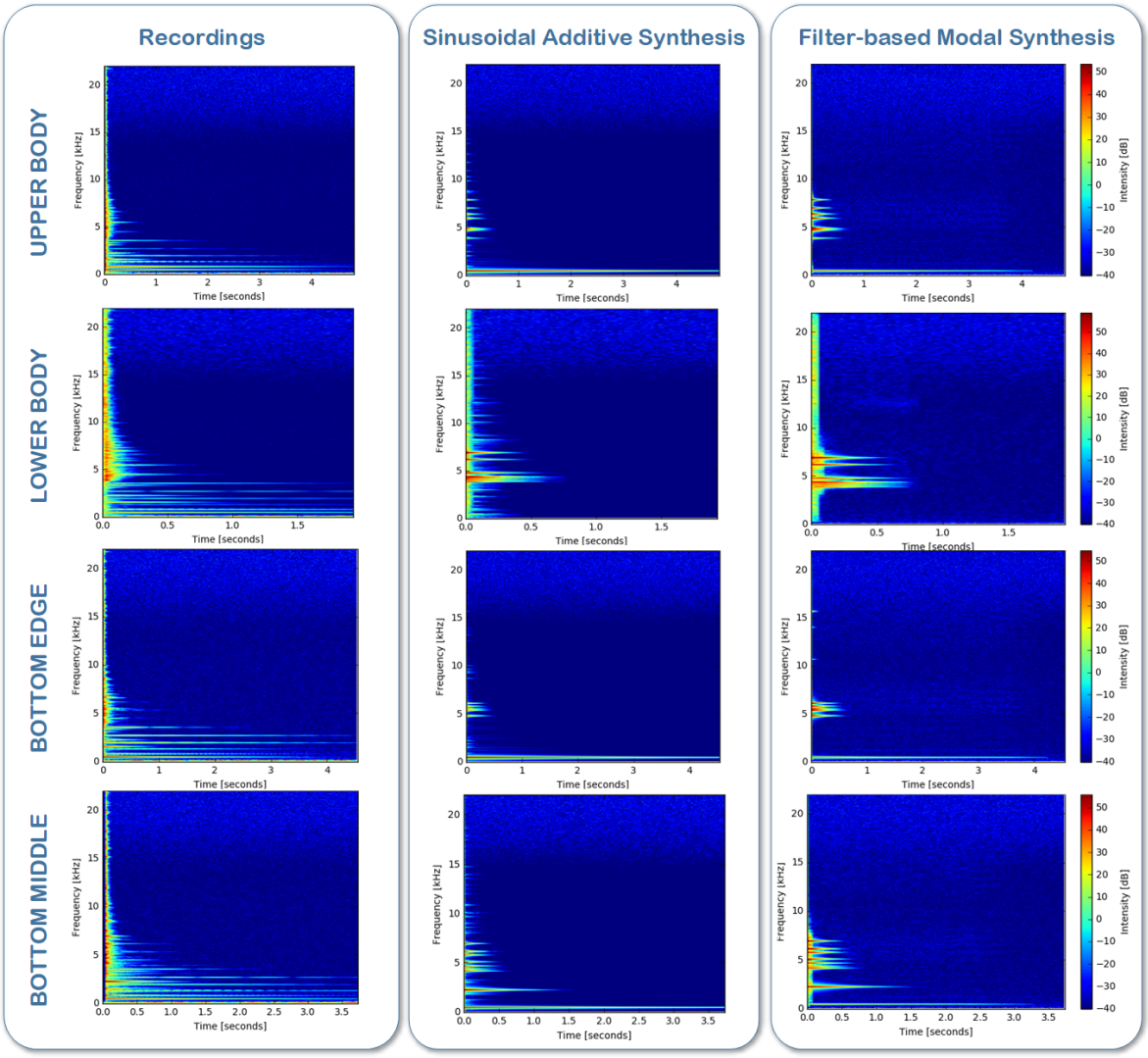


Figure C.5: Spectrograms of recordings and the two synthesis methods for the metallic wok.

Appendix D

User Guide for Tool Extension

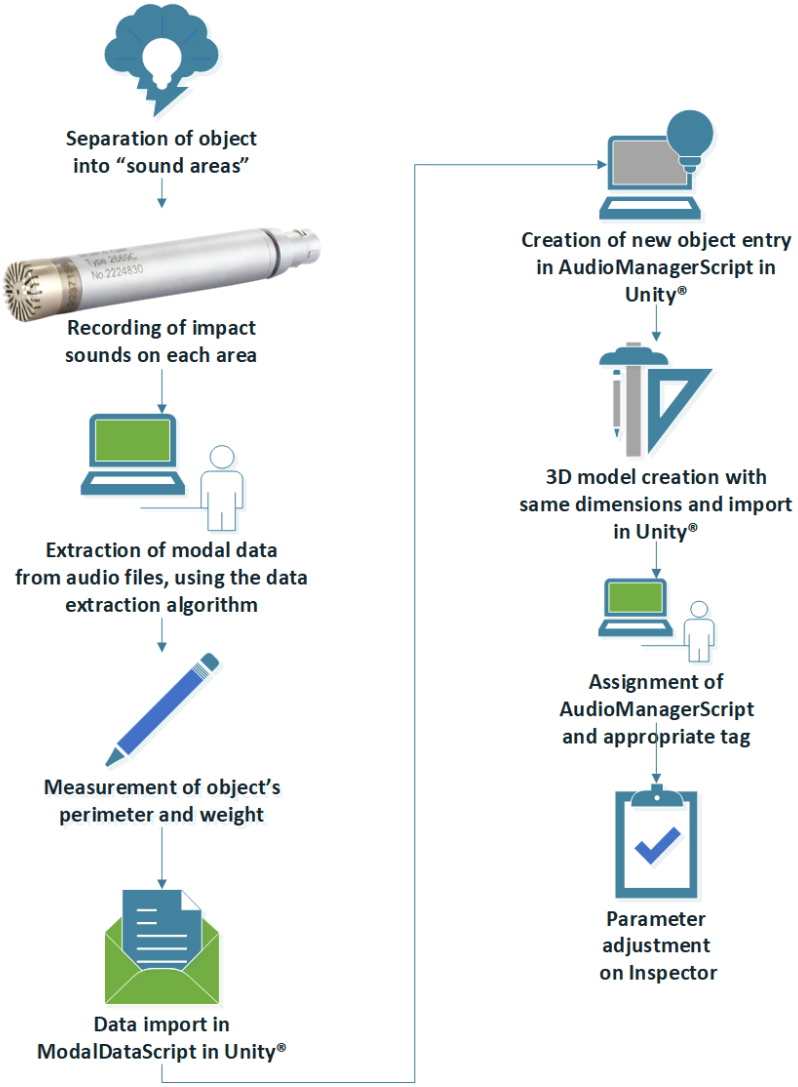


Figure D.1: Workflow for tool extension.

Bibliography

[Aramaki et al., 2009a] Aramaki, M., Besson, M., Kronland-Martinet, R., and Ystad, S. (2009a). Timbre perception of sounds from impacted materials: behavioral, electrophysiological and acoustic approaches. *Computer Music Modeling and Retrieval. Genesis of Meaning in Sound and Music*, pages 1–17.

[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., 2009b] Aramaki, M., Gondre, C., Kronland-Martinet, R., Voinier, T., and Ystad, S. (2009b). 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.

[Corbett et al., 2007] Corbett, R., Van Den Doel, K., Lloyd, J. E., and Heidrich, W. (2007). Timbrefields: 3d interactive sound models for real-time audio. *Presence*, 16(6):643–654.

[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.

[Daniel et al., 2003] Daniel, J., Moreau, S., and Nicol, R. (2003). Further investigations of high-order ambisonics and wavefield synthesis for holophonic sound imaging. In *Audio Engineering Society Convention 114*. Audio Engineering Society.

[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.

[Freed, 1990] Freed, D. J. (1990). Auditory correlates of perceived mallet hardness for a set of recorded percussive sound events. *The Journal of the Acoustical Society of America*, 87(1):311–322.

[Funkhouser et al., 2002] Funkhouser, T., Jot, J. M., and Tsingos, N. (2002). Sounds good to me! computational sound for graphics, virtual reality, and interactive systems. *SIGGRAPH 2002 Course Notes*.

[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, 2003] Giordano, B. L. (2003). Material categorization and hardness scaling in real and synthetic impact sounds. *The sounding object*, pages 73–94.

[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.

[Microsoft,] Microsoft. Windows mixed reality. https://developer.microsoft.com/en-us/windows/mixed-reality/. [Online; accessed 19-June-2017].

[MIT Media Lab Machine Listening Group,] MIT Media Lab Machine Listening Group. Hrtf measurements of a kemar dummy-head microphone. http://sound.media.mit.edu/resources/KEMAR.html. [Online; accessed 20-Juin-2017].

[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.

[Oculus VR, LLC.,] Oculus VR, LLC. Oculus vr. https://developer.oculus.com/documentation/. [Online; accessed 19-June-2017].

[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].

[Pulkki, 1997] Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. *Journal of the Audio Engineering Society*, 45(6):456–466.

[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.

[Zhang et al., 2013] Zhang, W. et al. (2013). Measurement and modelling of head-related transfer function for spatial audio synthesis.

[Zheng and James, 2010] Zheng, C. and James, D. L. (2010). Rigid-body fracture sound with precomputed soundbanks. *ACM Transactions on Graphics (TOG)*, 29(4):69.