Procedural 3D Audio for AR Applications

Angeliki Skandalou
John Jeremy Ireland

Supervisor:
Michael Rose

Kongens Lyngby 2017

Abstract

This is the first paragraph

THis is the second

THird paragraph of abstract

Four paragraphs is enough I guess

Acknowledgements

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

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

List of Figures

List of Tables

Contents

Abbreviations

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

AI Artificial Intelligence.

AR Augmented Reality.

BPF Band-Pass Filter.

BW Bandwidth.

CPU Central Processing Unit.

DAC Digital-to-Analog Converter.

DLL Dynamic-Link Library.

DTU Denmark's Technical University.

FBX Filmbox.

FEM Finite Element Method.

FFT Fast Fourier Transform.

FPS Frames Per Second.

GUI Graphical User Interface.

MUSHRA MUltiple Stimuli with Hidden Reference and Anchor.

Pd Pure Data.

UI User Interface.

VR Virtual Reality.

Nomenclature

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

CFM Collision force magnitude.

D Parameter corresponding to type of material.

K Kinetic energy.

M Matrix of modal data.

Q Parameter describing the damping of an oscillator.

ω Angular velocity.

d Damping parameter.

f Frequency.

Chapter 1

Introduction

Immersion and all these stuff that makes our thing good. Why we are doing it and what do we want to give to the community?

Audio in interactive projects like video games and VR/AR applications, plays a significant role for user immersion and realism. Visual and acoustic experiences are interconnected and lacking one of them spoils the whole experience.

The most difficult task is to produce realistic virtual sounds inside the application, difficult to distinguish them from the real ones. This can be achieved not only by playing back a realistic sound, but also by taking care of the environment effects and the context. For example, striking a nail on a board when it still vibrates from the previous struct, produces a different sound that gets added to the previous one [Cook, 2002].

Although some sounds like the soundtrack music or voices can be recorded and played back, sound effects need physically-based methods to synthesize them real-time, so as to be realistic and accurate. All objects vibrate when struck, even solid ones. It is something non-noticeable from sight, but it is capable of generating sound.

Stuff about 3d sound as well.

Stuff about sound in general and description of the thesis

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 the changes over time of it, 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].

In this thesis we present an audio design tool made for Unity®software platform, for physics-based sound synthesis in virtual environments.

To do describe how much better it is to have procedural audio than pre-recorded sounds and on top of that physics-based sounds!! WOW!

By pre-computing all necessary data, we are able to model a sound produced by a 3D model, very similar to the one that would be produced from a real-world one.

Why is our method better that others? (eg wavetable)? And why we think this is the future of the audio in video games?

Chapter 2

Theoretical Background

Short overview of the theory parts

This is a way to link to explanations Digital-to-Analog Converter (DAC)
THis is a todo: **To do** todo test (3)
THis is smth done:

2.1 State-Of-The-Art

Wildes and Richards (1988) defined the angle of internal friction (ϕ), a shape-invariant acoustical parameter that heuristically categorizes sounds into material categories, as

$$\tan(\phi) = \alpha/\pi f, \tag{2.1}$$

where $\alpha = \tau_e$ is the damping coefficient, with τ_e being the time for the vibration amplitude to decay to $1/e$ of its original value after the object is struck and f is the vibration frequency[Giordano and McAdams, 2006].

- Florens and Cadoz (The physical model: modeling and simulating the instrumental universe): the first to introduce modeling of surface vibrations with physical models
- Van den Doel (Scanning physical interaction behavior of 3D objects): robotic device to measure impulse response of an object being struck in different points
- O'Brien et al (Synthesizig sounds from Physically Based Motion): FEM (very accurate cause its based on physically measured data but very complicated to extract and require too much processing power)
- Raghuvanshi and Lin (Interactive sound synthesis for large-scale environments.): spring-mass system approximation
- Ren,Yeh,Lin (Example-Guided Physically Based Modal Sound Synthesis): one simple recording
- PHYSIS project: PHYsically informed and Semantically controllable Interactive Sound synthesis
- Van del Doel (FoleyAutomatic: physically-based sound effects for interactive simulation and animation): modal models derived from sound samples

- Brandon Lloyd et al. (Sound Synthesis for Impact Sounds in Video Games): short-time Fourier Transform

- Perry Coook (Real Sound Synthesis for Interactive Applications): a review of the work done on this field.

To do turn the bullets into text (5)

This thesis adopted the parameter extraction from recordings. We used everyday objects, making it easy for us to record sounds and model them on the computer. We also chose this method to make it easy for users of our product to extend the repository of physically-based impact sounds of objects.

2.2 How physical attributes affect sound

To do Note smwhere that we're using only vibrating solid objects (not liquids).Gaver[Gaver, 1993b] pg10 has info (6)

Impact sounds consist of an excitation that dampens over time. Hence, the amplitude of the oscillation depends only on the damping. On the other hand, scraping sounds consist of continuous supply of energy that adds to the amplitude value on top of the damping of the oscillation throughout the object interaction. In addition, the force of the interaction plays the most significant role for the amplitude of the oscillation. The stronger the force, the biggest it imposes the amplitude value to be - and the louder the sound [Gaver, 1993b].

Furthermore, the material of the interacting objects affects their vibrating oscillation. Damping factor is a material-specific attribute and the bigger it is, the faster the objects lose energy and thus the oscillation lasts shorter. For example, wood has way bigger damping factor than metal and this is why they produce a "thud" and a "ringy" sound respectively.

Moreover, the configuration of the object also affect its vibration. The size of it determines how high or low pitched sound it will produce. More specifically, the smaller the object the more high pitched will be the sound.

Table 2.1 shows the acoustic information affected by each physical attribute.

Source	**Effects on the Sound Wave**
Interaction	
Type	Amplitude, spectrum
Force	Amplitude, bandwidth
Material	
Restoring force	Frequency
Density	Frequency
Damping	Amplitude, frequency
Homogeneity	Amplitude, frequency
Configuration	
Shape	Frequency, spectral pattern
Size	Frequency, bandwidth
Resonating cavities	Spectral pattern
Support	Amplitude, frequency, spectrum

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

2.3 Modal Analysis

In this thesis we are using solid objects that are struck in different ways to produce sound. These ways could be falling on the floor or colliding with another object. The sounds produced can be impact, rolling or scratching sounds. When an object is struck, the forces applied cause deformations to it, emitting sound waves through the vibration of its outer surfaces [Van Den Doel et al., 2001].

Modal analysis studies the response of models under excitation. It uses the 3D model of an object to calculate its modal modes (vibration modes). There are multiple ways to do this, with the most accurate being FEM (Finite Element Method). The objective of FEM is to calculate the natural frequencies of a structure when it vibrates freely.

Another method for modal analysis is the "Exampled-guided", where data get extracted using example recordings of the objects being struck. Using a suitable algorithm it is easy to extract features from the recordings such as the fundamental frequency and its harmonics and the frequency peaks of the signal.

2.3.1 Data Extraction

Modal analysis is performed before modal synthesis, to extract the necessary data. Modal synthesis is the sum of damped oscillators each corresponding to a modal frequency, as it will be discussed further below. The data needed for synthesis and their origin are shown in the table 2.2.

Symbol	Description	Derivation
A_n	Initial amplitude	Modal analysis
d_n	Damping	Material properties
f_n	Modal frequency	Modal analysis

Table 2.2: Derivation of data used in modal synthesis.

Since every different point being struck produces different deformations on the object, we need matrices of size N (N being the number of struck points of the object). More specifically, we need a vector **f** of size **N** corresponding to the modal frequencies of every point, a vector **d** of size **N** corresponding to the damping ratios and a matrix **A** of size **NxK**, where K is the number of modal frequencies calculated in one point, which corresponds to the amplitudes of each mode in every point of the object. All the above gives the modal model which can be symbolized as $\boldsymbol{M = \{f,\ d,\ A\}}$ [Van Den Doel et al., 2001].

2.4 Modal Synthesis

In the modal synthesis part, using the data extracted above, we synthesize the struck sound corresponding to the object. When an object is struck, a sound envelope (**ADSR**) is produced. Starting with the *attack* which is the characteristic sound of the object, often named as "timbre", a number of high frequencies is produced which *decay* rather quickly, leaving space for the low frequencies which *sustain* for some time depending on the material of the object. At the end we have the *release* when sound stops producing. There are different ways to synthesize impact sounds, two of them being "Sinusoidal Additive Synthesis" and "Filter-based Modal Synthesis". The former uses exponential damping and the latter band-pass filters where the damping is the Q-factor of the filter.

2.4.1 Sinusoidal Additive Synthesis

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

The frequencies used for sound synthesis are the ones at which an object vibrates when struck and are called *resonant modes*. On excitation, the sine waves representing the mode vibrations peak to the designated amplitude and then start decaying over time. High frequency modes decay faster than low frequency ones [Lloyd et al., 2011], leaving a low-pitched residue named *"tail"*, especially when damping is low.

[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 leads to recognize the power of a sound synthesis model based on the sum of several sinusoidal modes. Figure 2.1 shows a model that enables to control the amplitudes and frequencies of a bank of sinusoidal oscillators.

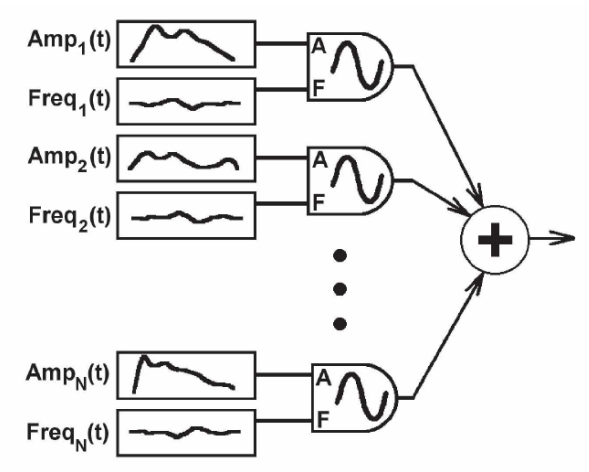


Figure 2.1: Sinusoidal Additive Synthesis Algorithm [Cook, 2002].

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

$$y_k = \sum_{n=1}^{N} A_{nk}\ e^{-d_n t}\ \cos(2\pi f_n t) \tag{2.2}$$

if $t >= 0$ and $y_k = 0$ if $t < 0$ [Van Den Doel et al., 2001].

2.4.2 Filter-based Modal Synthesis

Band-pass Filters

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

The passing range or "band" of frequencies is called **Bandwidth (BW)**. Defining as 0db the resonant peak, we can find the two cut-off frequencies ($f_{c_{\text{LOWER}}}$ and $f_{c_{\text{HIGHER}}}$) at -3dB. The

range between them is the bandwidth (equation 2.3). In figure 2.2 we can see the frequency response of a BPF. [AspenCore, Inc.,].

$$BW = f_{c_{\text{HIGHER}}} - f_{c_{\text{LOWER}}} \tag{2.3}$$

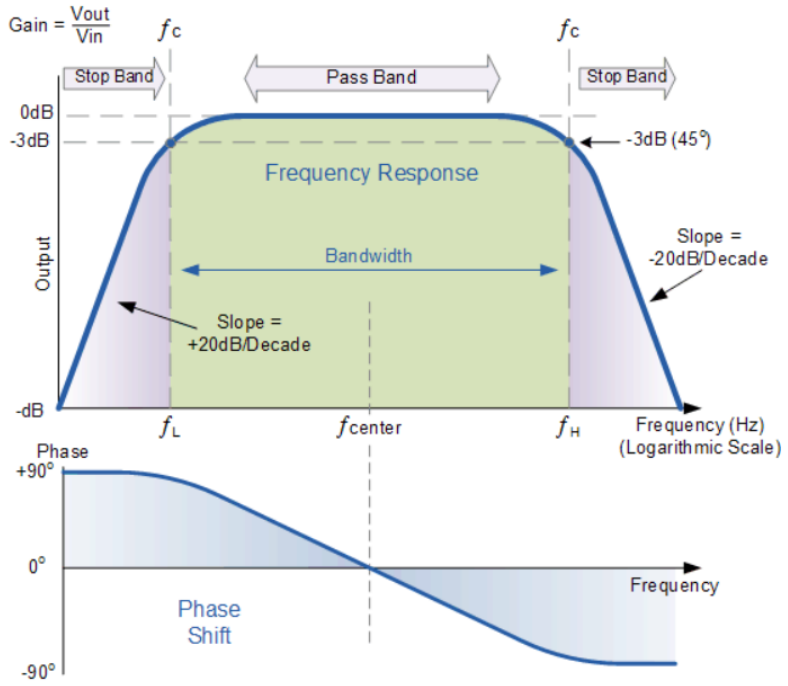


Figure 2.2: Frequency Response of a Band-pass Filter [AspenCore, Inc.,].

Synthesis

This method is also additive, since we are adding the outputs of a number of band-pass filters. To synthesize a sound using this method, we use as many filters as the modal frequencies. The filter takes as input an impulse, the center frequency which is the modal frequency and a **Quality factor (Q-factor)** which specifies the bandwidth of the filter. The Q-factor is calculated heuristically, depending on the material of the sound and is inversely proportional to the bandwidth ($Q = f_c/BW$), so the lower the Q-factor, the wider the bandwidth and vice-versa. Hence, more or less frequencies will be included in the audible range. We call the above structure a *resonator*, which also includes a multiplication with the corresponding amplitude, taken from the A matrix.

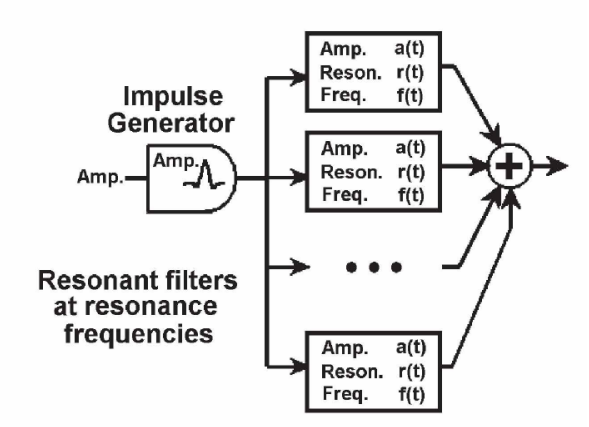


Figure 2.3: Filter-based Modal Synthesis Algorithm [Cook, 2002].

2.4.3 Sound Variations

Each point of an object produces a different sound when struck. This happens because of the different amount of excitation each resonant mode experiences. As mentioned above, during data extraction we get a matrix of amplitudes of size NxK corresponding to K different resonant modes of each of the N points of the object. This means that even though resonant frequencies are the same for each location of the specified object, each frequency's peak differentiated depending on the location.

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

1. The most accurate method is to perform FEM analysis on the object and distill as many amplitude matrices as the number of points consisting it. However, this method needs a lot of memory to store the data and computational power to access them.

2. Another less precise method is to separate the object into areas and compromise that each point belonging in the same area sound exactly the same. Thus, the amount of stored data decreases in a great deal when at the same time the sounding result is almost the same.

3. A third method is to store only one amplitude matrix and randomize the values for each impact sound, but this can lead to an unexpected behavior.

4. 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 changes on the pitch 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 [Lloyd et al., 2011].

2.5 3D Audio

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

Chapter 3

Method

To do maybe name this chapter smth else?? (7)

A combination of the methods described in Chapter 2 is proposed in the present study.

3.1 Overview of our tool

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

To do replace figure with a better one (8)

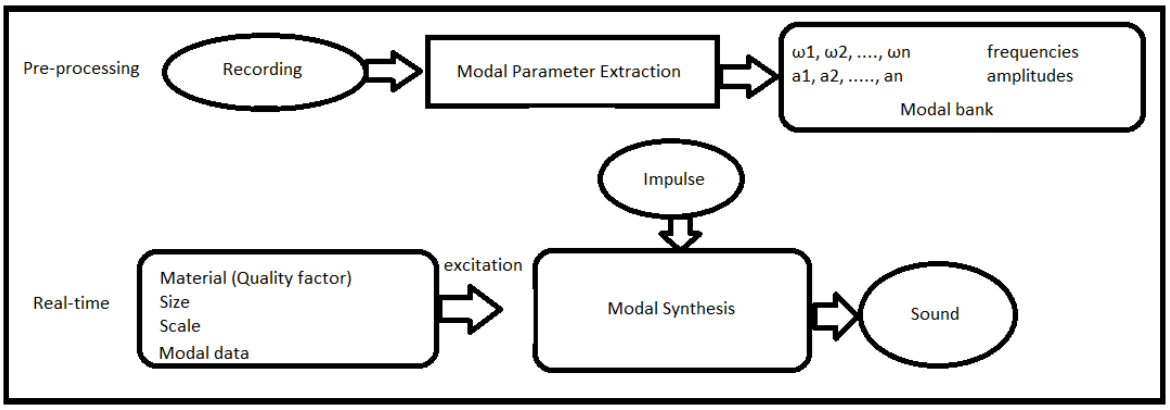


Figure 3.1: Sound Synthesis Procedure.

To obtain this tool, we followed the procedure described in figure 3.1. More specifically, the first step we carried out was to find several everyday objects, separate them into different areas that sound similar when struck and record impact sounds for each area. Afterwards, we used those recordings to extract data needed for sound synthesis. At the same time, we made a 3d model of every object we used. Next, we generated two different PureData patches, introducing two different synthesis methods.

Then, we combined the 3d models with their corresponding data and the synthesis patches inside Unity®software. This is where depending on the point where an object is struck, the corresponding frequencies and amplitudes are assigned to the patch and a sound is sent to the audio DSP chain.

3.2 Recordings

To obtain the necessary data we performed recordings of the sound produced of the object when being hit.

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

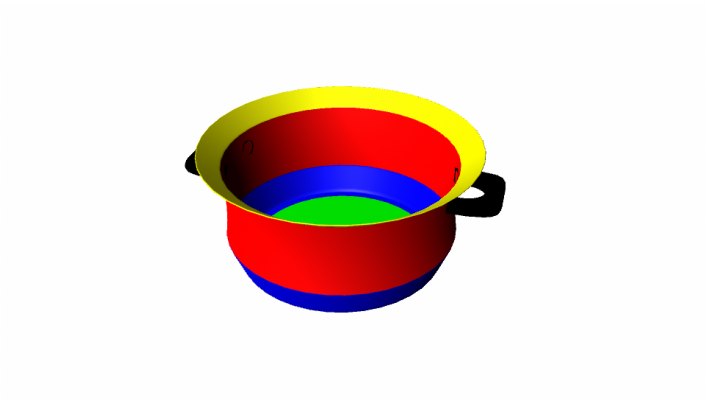


Figure 3.2: Division of an object into areas with similar sound.

3.3 3D models

To create the 3D models we used Maya Autodesk software [Autodesk, Inc., b] and exported them as FBX® files [Autodesk, Inc., a] which is a format recognizable by Unity® software [Unity Technologies,].

3.4 Chuck language

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

Modal features extraction code

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

From a FEM analysis one can find out that each object vibrates in a very high number of modes. Although, most of them are inaudible and do not contribute to the sound model. Thus, we can easily define *ten* to be the total number of peaks identified. After some trials, we found out that *five* peaks are already enough. However, the authors recommend to use twice the sufficient number, hence *ten* for our case. Afterwards, the algorithm having taken a recording as input, computes its histogram and identifies its peaks. The frequencies where peaks occur are the modal frequencies candidates. Depending on the numbers of peaks we

chose, the algorithm outputs a normalization between high and low peaks. Finally, the algorithm finds the maximum value of the signal on each peak, calculating the corresponding amplitude of each mode.

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

To do should we explain why you used FFT? (9)

3.5 PureData

Pure Data (pd) is another music programming language. It is open source and the main difference from ChucK language is that pd is a visual or "patcher" programming language, using objects instead of code, linked together to form a sequence [Puckette,].

Re-synthesis patch

For reasons that will be explained below, we had to attach only one pd patch to every 3D object in the demo. Therefore, all synthesized sounds (impact, rolling and scratching) are being synthesized under one main pd patch.

The main part of the re-synthesis patch is the resonator. Using band-pass filters it gives the impulse response of an object struck at a specified point. It also clips the signal to give more brightness and harmonics. **To do** develop more (10)

For optimization reasons, we lowered the number of modes down to 10 instead of 20 that we initially had, since the extra 10 did not add any useful information to the output sound. In addition, we clipped the range of frequencies to the human audible range of 20Hz to 20KHz.

3.6 Heavy Compiler

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

3.7 Unity®

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

The first part of the Unity implementation is the assignment of modal data to every different area of the object. This is done in $O(n)$ time for n modes.

To do describe onaudiofilterread() (11)

3.8 Microsoft Hololens Emulator

Chapter 4

Measurements

4.1 Recordings

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

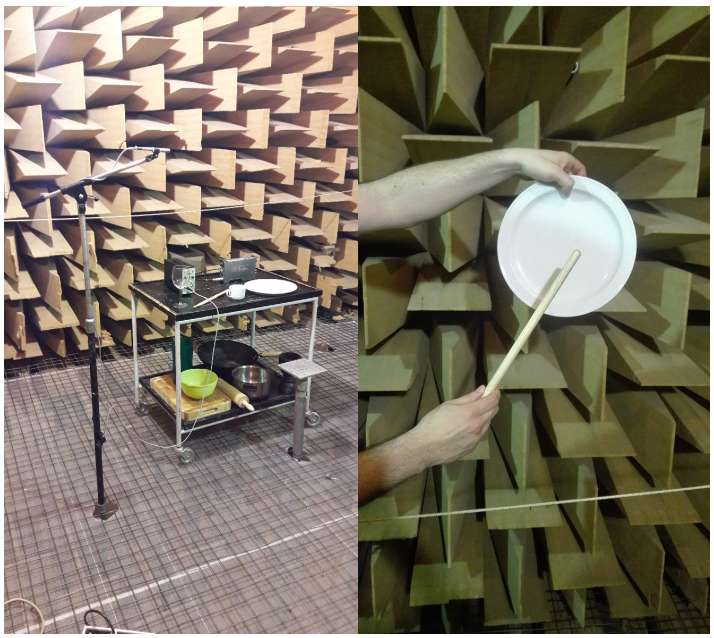


Figure 4.1: Picture of the setup for the measurements (left) and of a struck object (right).

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

Eleven objects of everyday life made of five different materials (plastic, wood, ceramic, glass and metal) were used for the experiment. The idea of choosing these objects came firstly from the need of owning them (to perform the recording) and our ability to model them for

the demo (as they should be simple enough). Secondly, since we wished to test the immersion of the synthesized sounds on users, we wanted to be sure that the sounds used are familiar to them. In figure 4.2 both real and modeled objects are shown.

(a) Real objects.

(b) 3D models of the objects.

Figure 4.2: The eleven objects used in the thesis.

4.2 User tests

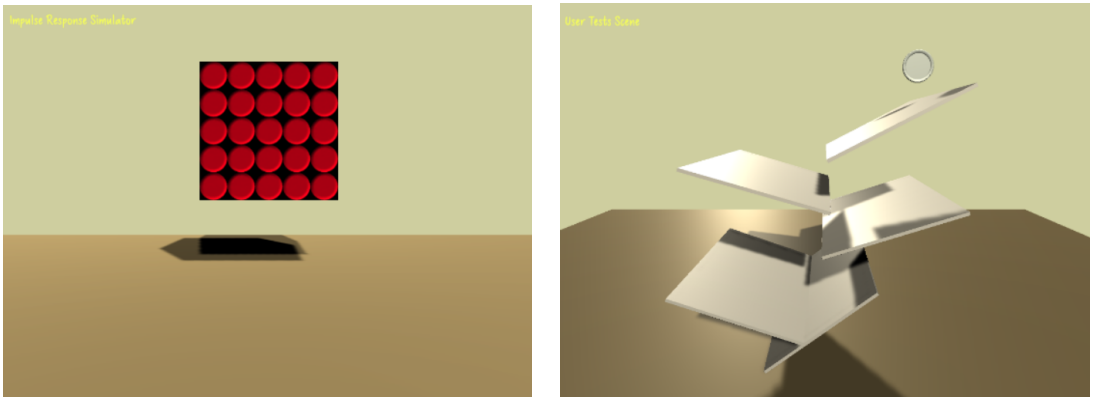
To do should we move this section to a separate chapter as Michael said? (12)

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

To examine the immersion of real-time produced and physics-based sounds on game players, we performed *MUSHRA*[Series, 2014] tests to people. Our aim was to answer the following questions: 1) Which of the two synthesis methods (sinusoidal and filter-based additive synthesis) is closer to reality? 2) Which is the range (in Q-factor values) of every material's sound? 3) Does physics-based synthesis make a sound more realistic and less boring? .

4.2.1 Preparation

For the tests we used several .wav files that we recorder inside the Unity®platform. We designed two special scenes for this purpose, because we wanted to test the sounds both in real-life conditions -when an object is falling and colliding with other objects- and the impulse response of the synthesis model (figure 4.3).



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

Figure 4.3: The Unity Scenes designed to record the .wav files used for the user tests.

In the first scene (figure 4.3(a)), we recorded the audio files for the first experiment. During the recording session, we "tagged" the cube as different objects (a process that assigns to the cube different modal data) and let it touch the ground without any bounce. We also repeated the process with an Audio Source and the recordings files, to achieve consistency of the stimuli.

In the second scene (figure 4.3(b)), we recorded the audio files for the second and third experiment. In this scene, objects fall, roll and scratch freely on rotated platforms, simulating a real room with obstacles.

4.2.2 Stimuli

We performed three different tests. On the first test, the stimuli was 44 pairs of impulse responses (one for each synthesis method) corresponding to different areas of the eleven materials and their corresponding reference sounds which were the recordings of the actual sounds produced by the physical objects.

On the second test, the stimuli consisted of 33 pairs of sounds that were produced by falling objects. From the pair, one sound is produced when objects are split into "sound areas" and each area produces a different sound and the other sound is produced when every point of the objects makes the same sound. This single sound was the one produced from the area of each object where the recording taken (section 4.1) was closer to the real sound of this object in our opinion.

The stimuli of the third test was 50 sounds coming from five different objects, one of each material under testing (plastic jug, wooden mortar, ceramic plate, glass bottle and metallic cooking pot). For each object, we used 10 different sounds that corresponded to a small variation on the Q-factor which changes the material of the object. More specifically, starting from a value of 600, we increased the Q-factor by 200 up to 4600, removed some sounds that were too similar with others and provided participants with the rest.

4.2.3 Procedure

Stimuli was presented to the participants through **To do** describe headphones (13), in a room with reduced external noise.

In the first test, participants were provided with a reference sound (the actual recording) and two testing sounds (the sinusoidal and filter-based synthesized sounds), and they were asked to choose the one sounding closer to the reference. In the second test, participants were given pairs of sounds where the first one was made of multiple sounds per area of the object and the second one was made by the same sound assigned to every area. They were asked to answer which of the two they prefered. Finally, in the third test, they were provided with sounds from the same object but with different materials assigned to it and they were asked to point out when a change in material happens.

Every sound file starts 1 second after participant presses play and ends half a second after no sound can be heard.

4.2.4 Participants

To do number of participants, age, gender, normal hearing, job (14)

4.2.5 Test Results

Chapter 5

Implementation

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

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

5.1 Impact Sounds

5.1.1 Sinusoidal Additive Synthesis

In this section we describe in depth how the Pd patch corresponding to the sinusoidal additive synthesis of impact sounds works. The patch attempts to translate equation 2.2 into the programming language of Pd. Some of its terms are referenced in the following explanation.

First of all the frequencies and amplitudes matching the ten modes of the object are initialized. We can therefore feed these frequencies, which we identified as f_n in the equation 2.2, into the different oscillators. In Pd, oscillators output a cosine wave which is equivalent to $cos(2\pi f_n t)$ from the equation which suits our purpose perfectly.

We also translate into Pd the expression $e^{-d_n t}$ which corresponds to the damping of every mode n. Gaver [Gaver, 1993a] states that for each partial the decay rates d_n are controllable through a parameter D which corresponds to a material and that a useful heuristic, that we use in our patch, is to have $d_n = 2\pi f_n D$. By experimenting we established that values of D range from approximately 0.0002 for metal to about 0.05 for plastic sounds, with glass, ceramic and wood sounds in between. The higher the damping the higher the values. Then we multiply the damping by the partial's initial amplitude A_n to obtain an amplitude envelope that varies over time and which we multiply by the oscillator's signal. The output is what we call a partial which is illustrated in figure 5.1.

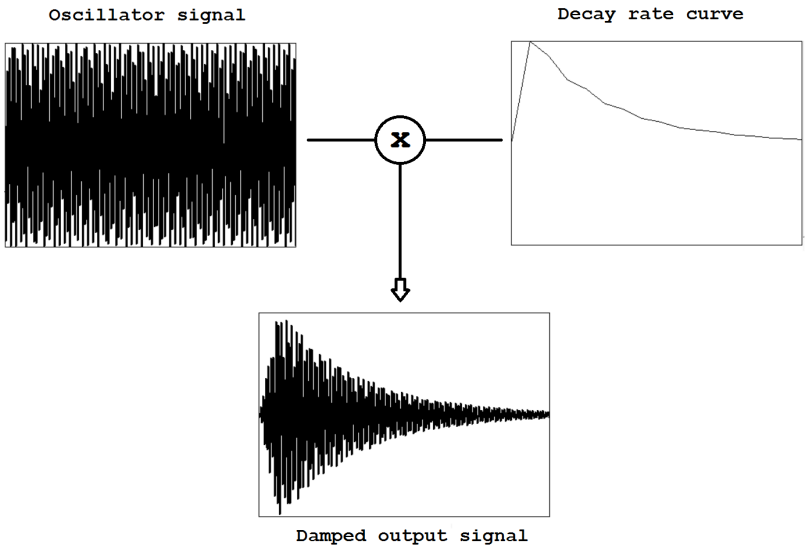


Figure 5.1: Diagram showing how the output partial is created from a 5000 Hz cosine wave and a decay rate curve with D = 0.005.

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

The patch produces an impact sound whenever the OnCollisionEnter method from Unity is called. This is done when the collider that has the script attached to it has begun to touch another collider. When this happens we set the magnitude of the collision and then send an event to excite the patch. This is done by setting the value of t in 2.2 to zero which increases over time making the sound decay.

5.1.2 Filter-based Modal Synthesis

This synthesis method is based on the utilization of a bank of ten bandpass filters. Pd's bandpass filters have three control inputs as seen in figure 5.2. 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.

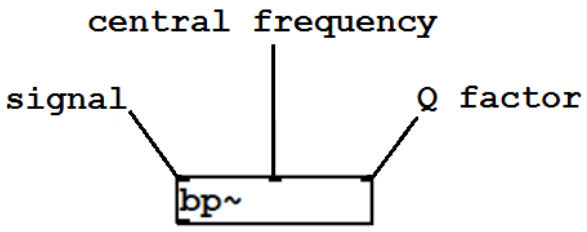


Figure 5.2: Pd's bandpass filter with its three inlets

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

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

To cause the object to sound we use a short impulse signal that excites the filter. The amplitude of the signal goes from 1 to 0 in 2 milliseconds as represented in figure 5.3. This impulse is multiplied by the value of the kinetic energy when the object impacts with another.

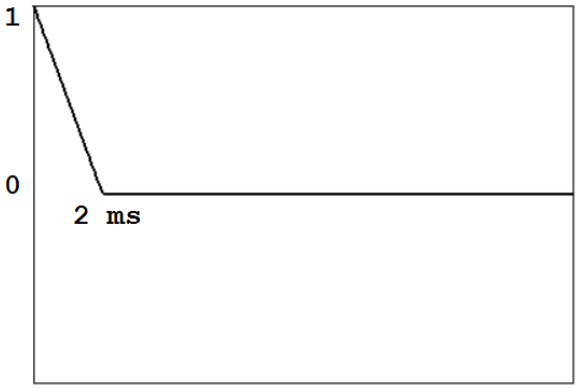


Figure 5.3: Impulse signal used to excite the bandpass filter.

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.

5.2 Scratching Sounds

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

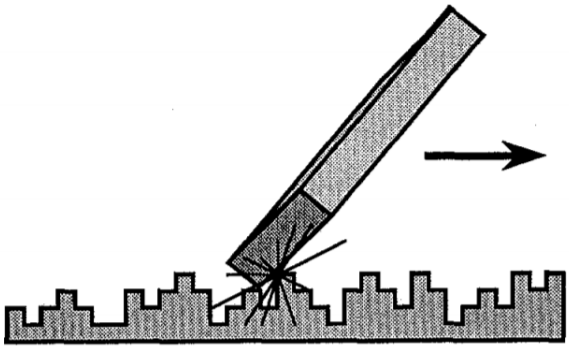


Figure 5.4: Scrapping involves a multitude of micro-collisions against a contact area. Picture from [Gaver, 1993a]

To produce scratching sounds we follow a similar technique to [Gaver, 1993a] and [Van Den Doel et al., 2001] who propose the use of filters. We therefore choose the filter-based modal synthesis method to model the resonator as seen in the previous section. The difference lies in the signal that excites the model. We implement this by generating a noise impulse waveform, as in 5.5, that passes through the bandpass filters. We create this waveform by having a simple impulse signal that is scaled relative to the velocity and material of the object and then multiplied by a white noise signal. From a heuristic approach we deduced that the higher the velocity and the Q factor, the higher the gain of the scratching sound. The length of the excitation signal depends on the time it took Unity to complete the last frame. The scraping sound is triggered in Unity's OnCollisionStay method when the object's collider is touching another one and with the condition that the angular velocity of the object is beneath a threshold. The angular velocity specifies the rotational motion of a rigid body [Sears and Zemansky, 1964]. We therefore add this condition to differentiate between sliding and rolling.

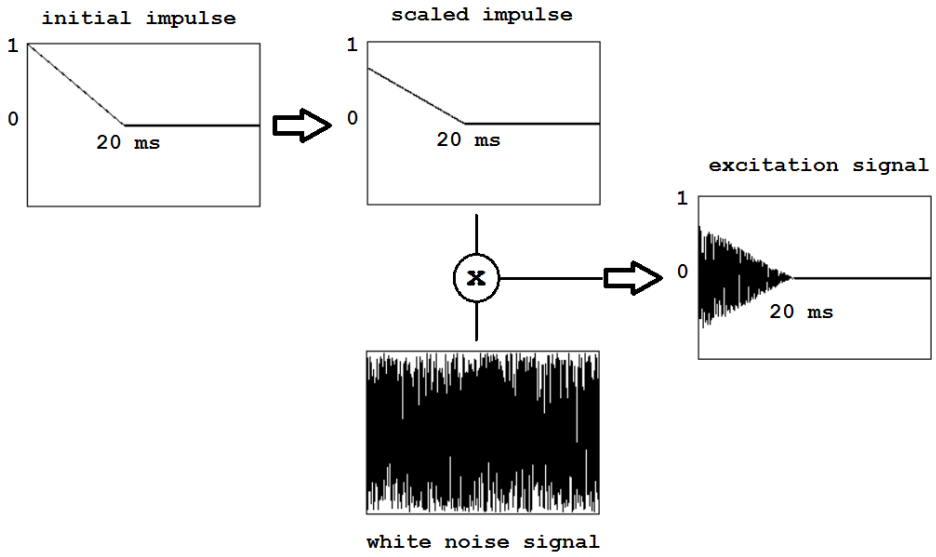


Figure 5.5: Diagram showing the process to get the excitation signal of the resonator to produce scratching sounds.

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

5.3 Rolling Sounds

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

First we aim our attention to the sounds produced by the object's surface irregularities. For simplification we take a regular octagon that has received an impulsive force and therefore rolls along a plane. Every time one of the vertices impacts the ground, energy is lost to heat and sound. Thus, as the octagon rolls, a pattern of eight impulses is created for every rotation

as seen on figure 5.6. To create these impact sounds we use the filter-based synthesis method. The filters are excited by a succession of eight impulses of different amplitudes as no real object has a perfect geometry. The outcome results in a pattern of quasi-periodic audio cues distributed differently over time depending on the speed of the rolling object. See figure 5.6. [Houben et al., 1999] and [Rath, 2003] suggest that this periodicity which originates from the asymmetry of the object, enables listeners to distinguish between sliding and rolling sounds.

Explain script somewhere

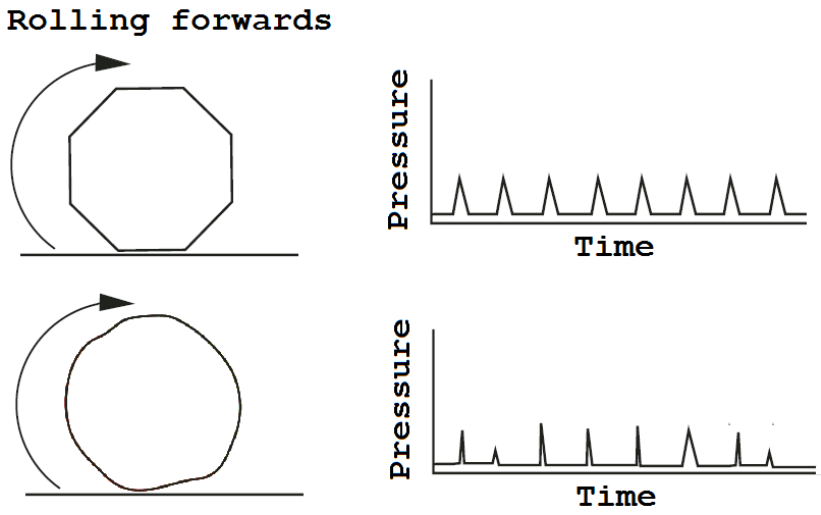


Figure 5.6: An octagon and a real spherical object pressure levels over time as they roll.

Let us dive into the actual implementation of the latterly described model. We mentioned that the amplitude of the impulse signals used to excite the resonator are different. With a heuristic approach, we choose a sequence of eight multipliers that correspond to the prominence of the bumps along the object's surface contour. Additionally, we have incorporated a parameter that determines the object's surface roughness. This parameter, which can be adjusted by the user, scales the values of the eight multipliers. The lower the value of the parameter the smoother the object's surface. In other terms, the smoother the object's surface, the smaller and more homogeneous the amplitude of the impulses are. We can compare the amplitudes of the impulses for different roughness degrees in figure 5.7.

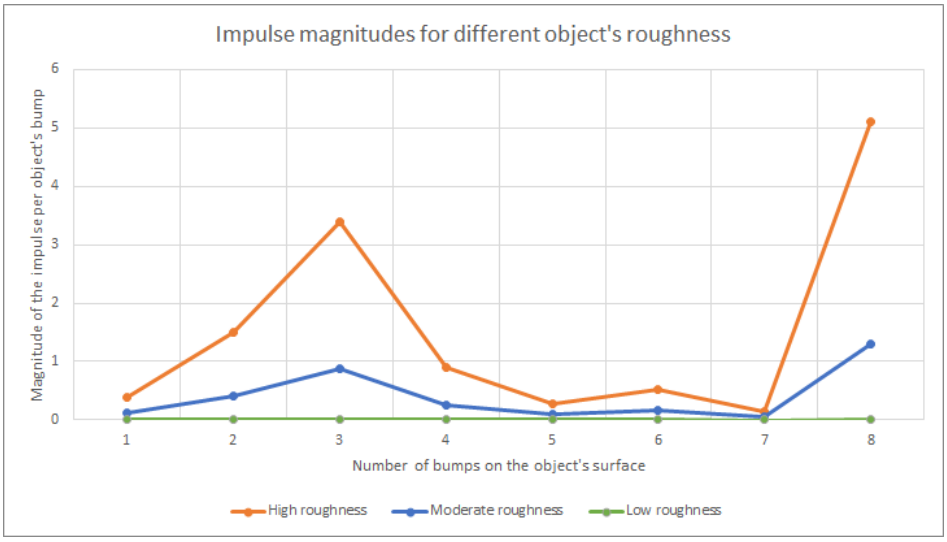


Figure 5.7: Graph showing the amplitude of the impulses for every bump on the object's surface for three values of the roughness parameter.

We now look into the sounds produced due to the irregularity of the ground. As pointed out in [Van Den Doel et al., 2001], even a smooth ball rolling on a rough surface produces sound. This paper and [Rath, 2003] indicate that this is due to the small constant collisions of the ball with the surface's asperities. The bigger the ball, the less the surface details are "felt". In our implementation we consider the surface's irregularities to be very small compared to the radius of the rolling object. See figure 5.8. This suggests that our model for uneven ground is similar to our previously explained scratching model as [Van Den Doel et al., 2001] proposes. The difference is that the gain of the output signal is lower for the rolling as we consider that rolling friction forces are smaller in comparison with scratching friction [Mehta and Mehta,].

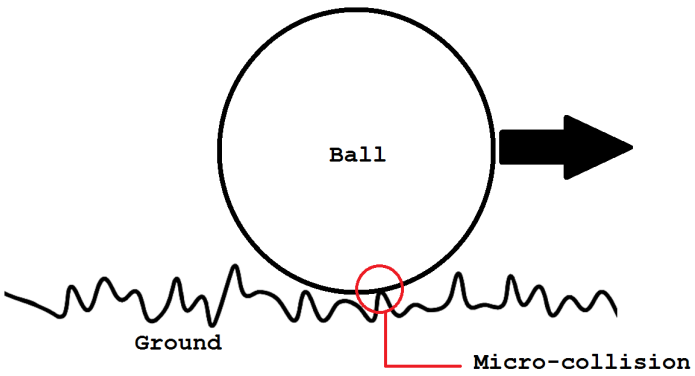


Figure 5.8: A smooth ball rolling over an small ground irregularities.

5.4 Sound Variations

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

One method is to have only one amplitude matrix (corresponding to only one point of the object) and randomize the values every time a collision occurs. During the development of this thesis we tried this method, but we abandoned it rather quickly due to undesirable

sounding results. In other words, every other sound, when an object was for example bouncing on a surface, was exceptionally different from the previous one even though the two collision locations were very close.

Although the last method described in section 2.4.3 of chapter 2 appears to be the best solution, we did not implement it in this thesis due to the inability of Unity®to give us data about the exact point of the object that participated in the collision.

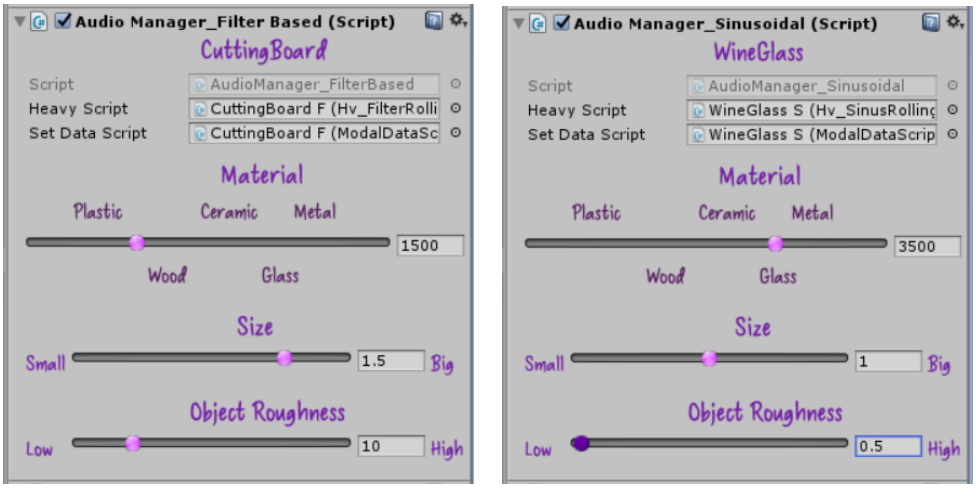
Hence, we decided to carry on with another method where we separated each object into similar "sound areas" and simplified the calculations by assuming that each area produces the exact same sound when struck. Therefore, during measurements, we conducted one recording for each one of these areas, ending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

5.5 User Interface

Here we will describe the *user interface (UI)* of the tool, where sound designers are able to choose the sound they prefer for every object.

The UI is made inside Unity®, by programming a custom inspector. *Inspector* in Unity®is a window that shows up after selecting an object, a file etc inside the platform and it displays all information relevant to it.

We also used a custom *GUISkin*, which is a set of settings about the Graphical User Interface (GUI).



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

Figure 5.9: The custom inspector inside Unity platform.

When the designer wants to assign the procedural audio component on a Unity game object, 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 5.10.

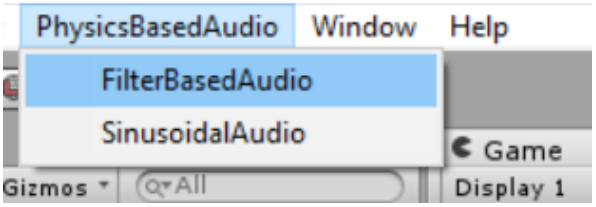


Figure 5.10: Designer can assign the audio manager from the menu bar.

Since every sound is produced from a corresponding real world 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, hug, mortar, bowl, plate, rolling pin, wine bottle, wine glass or wok). However, it is easy for her to contribute with new objects to the tool, by following the guide on appendix B.

5.5.1 Assignment of Different Materials

Different materials can be assigned to the objects made for this thesis. The designer is able to choose between *plastic, wood, ceramic, glass and metal* by adjusting the corresponding slider on the interface (see figure 5.9).

Metallic or glass sounds are more "ringy" than wooden or plastic ones that are more "thud". We achieve those sounds by changing the **Q-factor** of the **band-pass filters** used in the pd patch. Q-factor indicates the power loss in the filter. The higher the Q the less power is lost, so the resonator vibrates longer as explained in equation 5.1 [Cory et al., 2006].

$$Q = 2\pi \frac{\text{maximum energy stored}}{\text{total energy lost per cycle at resonance}} \tag{5.1}$$

Using trial and error method, we came out with an average value of the Q-factor for each material and we use it as the default value on the tool. Those values are shown in the table 5.1.

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

Table 5.1: Default values of Q-factor for each material in the tool.

5.5.2 Changing the Size

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

5.5.3 Changing the Object Roughness

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

5.6 Unity Scripts

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

5.6.1 Scaling

As mentioned above, impact sound gets more high-pitched when an object is scaled down and vice versa. To achieve a realistic scaling when the designer uses the build-in scaling feature of Unity®, the tool calculates the size of the game object on start. More specifically, a *scaling average* is calculated, taking into account all three dimensions (equation 5.2.

$$avgScale = (transform.localScale.x + transform.localScale.y + transform.localScale.z)/3; \tag{5.2}$$

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

Then, the tool checks whether a scale-up or a scale-down was executed. In the first case, we perform a normalization to 1/10th of the average scale value and we add it to the pitch multiplier **To do** reference the description of the pitch multiplier (15). We apply it to the size slider value and then to the pitch multiplier which we use to re-set the modal frequencies. To be more precise, instead of applying the actual pitch multiplier added with the average scaling, we subtracted from 2 and then we use it ($2 - temp$ on the code below). This happens because we reversed the size slider. More specifically, the multiplier directly applied to the frequencies, increases them when it is bigger and decreases them when it is smaller. However, for convenient reasons, we wanted it to be the bigger the multiplier, the bigger the object and reverse. Since 2 is the biggest value, we normalized it to be the smallest one.

In the second case, we subtract the value from the pitch multiplier. We do not need to normalize the average scaling value, because it is already between 0 and 1. Afterwards, we follow the same procedure as above, with one difference; instead of subtracting the new pitch multiplier from 2 we add it to 1. This happens because now the biggest value of the size slider is 1 -since above this it counts as a scale-up- and we still want the reversed value for the slider, so we subtract the subtracted value, making it a plus (+). If no scaling took place nothing happens.

```
1 IF average scale > 1 (scale-up) THEN
2     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
```

5.6.2 Excitation of Impact Sounds

This is were the tool detects a collision of an object with something else. The collision could be either with the ground, another object from the tool or just an object in the scene and it is identified by the program when two collider Unity®components attached to two different objects touch each other. Below we will describe what is done inside the Unity's build-in function *OnCollisionEnter* [Unity Scripting Reference,], when an object enters a collision.

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

Moreover, the algorithm calculates the kinetic energy of the whole collision, using the equation 5.3 [Crowell, 2003]:

$$K = \frac{1}{2}mu^2, \tag{5.3}$$

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®we can calculate the latter using the command: *Collision.relativeVelocity.magnitude* [Unity Scripting Reference,].The kinetic energy corresponds to the collision force magnitude and we use it to turn the volume of the sound up or down depending if the collision was strong or wear respectively.

Another parameter that the algorithm takes into account is whether an object collides with another modal object or not. By modal object we mean the object that our tool provides that come with modal synthesis sounds. In case we have a collision with a modal object, we enhance the collision force magnitude adding more force that depends on the materials of both of the participant objects. Namely, we calculate an average between the two quality factor normalized values, we multiply it by the collision force magnitude and add this to the initial collision force magnitude. We do this because from our experience with the objects, we found out that when two high-pitched materials like glass or metal collide to each other, they produce a much more intense sound than when a high-pitched material collides with a low pitched like plastic or wood.

$$new\ CFM = CFM + CFM * \left(\frac{\frac{Qfactor_1}{5000} + \frac{Qfactor_2}{5000}}{2} \right) \tag{5.4}$$

where CFM : the Collision Force Magnitude.

To do see if it is necessary to describe the 3 different DACs here or just in the patch (16)

5.6.3 Excitation of Rolling and Scratching Sounds

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

The decision whether an object is rolling or sliding on a surface is made using the **angular velocity** ($\omega = \frac{angular\ displacement}{time}$). Unity's rigidbody variable for angular velocity is a 3 dimensional vector, measured in radians per second [Unity Scripting Reference,]. The magnitude of it is calculated from the equation 5.5

$$\omega_{magnitude} = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \tag{5.5}$$

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 angular velocity magnitude is over 1, means that the object is rolling, otherwise it is not and providing that linear velocity is non-zero, means that the object is sliding on the surface.
To do why 1 is the border? maybe because (0.0,0.0,0.0) angular velocity gives 0.smth in magnitude (17)

To do add an Optimization section maybe? (18)

Chapter 6

Results & Discussion

6.1 Results

6.1.1 Which Synthesis Method Is Better?

6.1.2 Did we manage to achieve what we wanted?

- do we think we are successful?
- did we manage to do what we wanted?
- did we prove what we wanted to prove?
- is it really a useful tool?
- who is it targeting as a user?

6.2 What did we do new?

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

6.3 Discussion

6.3.1 Types of games that it can be used

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

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

6.3.3 CPU demands

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

In our tool, however, when profiling a demonstration of a wine bottle rolling down a number of oblique platforms (seen in figure 4.3(b)), using the *Profiler Window* of Unity Editor we can see that except for the initialization at the beginning where scripts consume some CPU power, the rest of the demonstration remains stable in performance and around $100 fps$ (figure 6.1). This performance test was held on a laptop with 4 Intel® Core™ i7-6700HQ CPUs running at 2.60GHz, each with 2 hardware threads.

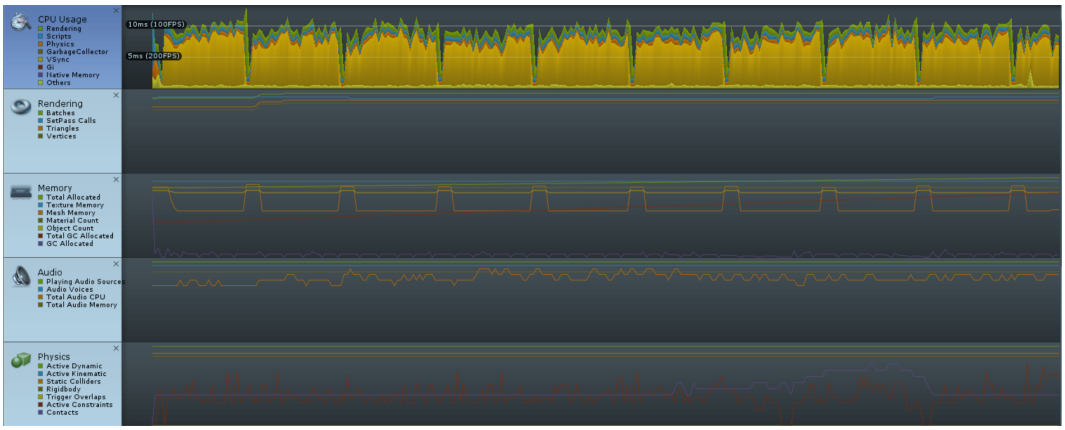


Figure 6.1: The profiler view of Unity Editor when a wine bottle rolls down a number of platforms.

To do demo with a lot of objects and check if it needs to reduce quality (19)

6.3.4 Bugs

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

6.3.5 How can we improve our work?

- take into account the environment (reverberation etc)
- make objects destructible
- randomize initial phase so peaks of the sine wave don't line up and distort the sound (saturation)

6.3.6 Pros and Cons of Procedural Game Audio

Chapter 7

Conclusion

This is the conclusion
4-5 paragraph approx

Appendix A

Results of tests to users

Appendix B

User Guide to our product

Bibliography

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

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

[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].

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

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

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

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

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

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

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

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

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

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

[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.

[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].

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

[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., 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.

[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.

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

To do...

- ☐ 1 (p. v): name of the recording guy
- ☐ 2 (p. 1): describe how much better it is to have procedural audio than pre-recorded sounds and on top of that physics-based sounds!! WOW!!
- ☐ 3 (p. 3): todo test
- ☑ 4 (p. 3): this is done
- ☐ 5 (p. 4): turn the bullets into text
- ☐ 6 (p. 4): Note smwhere that we're using only vibrating solid objects (not liquids).Gaver[Gaver, 199 pg10 has info
- ☐ 7 (p. 9): maybe name this chapter smth else??
- ☐ 8 (p. 9): replace figure with a better one
- ☐ 9 (p. 11): should we explain why you used FFT?
- ☐ 10 (p. 11): develop more
- ☐ 11 (p. 11): describe onaudiofilterread()
- ☐ 12 (p. 14): should we move this section to a separate chapter as Michael said?
- ☐ 13 (p. 16): describe headphones
- ☐ 14 (p. 16): number of participants, age, gender, normal hearing, job
- ☐ 15 (p. 25): reference the description of the pitch multiplier
- ☐ 16 (p. 26): see if it is necessary to describe the 3 different DACs here or just in the patch
- ☐ 17 (p. 27): why 1 is the border? maybe because (0.0,0.0,0.0) angular velocity gives 0.smth in magnitude
- ☐ 18 (p. 27): add an Optimization section maybe?
- ☐ 19 (p. 30): demo with a lot of objects and check if it needs to reduce quality