The Velvet Cloud

- Infinite Sustain and Modulated Reverberation using velvet noise -

Master Thesis

Carmen Muñoz Lázaro/SMC201032

Aalborg University
Electronics and IT

Front page figure represents the first stage of the art work for the plugin which is finally discarded.

Electronics and IT
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
The Velvet Cloud: Infinite Sustain and Reverberation using velvet noise

Theme:
Digital Audio Reverb and Sustain plug-ins

Project Period:
Fall Semester 2020

Project Group:
SMC201032

Participant(s):
Carmen Muñoz Lázaro

Supervisor(s):
Stefania Serafin
Silvin Willemsen

Copies: 1

Page Numbers: 113

Date of Completion:
February 27, 2021

Abstract:

In this project two digital audio effect plugins (VSTs) are proposed to simulate an automatic infinity sustain and reverb respectively. Sustain audio effect algorithm is based on convolution with velvet noise. For reverberation, a Modulated Velvet Feedback Delay Network implementation will be used. The use of velvet noise enhances reverberation with a quick built of early reflections in comparison with standard Feedback Delay Networks. The Modulation approach produces interesting deep/psychedelic sounds which model unreal spaces. Both VSTs are thought to work together allowing the user to place them in her/his favourite configuration (e.g. Sustain in series with reverberation). A technical evaluation (impulse response, White Gaussian Noise response and T60 times) is performed regarding the different possible combinations of plugins. Also, a Two Alternative Forced Choice test and a survey about the VST characteristics are carried out for a more subjective sound and plugin evaluation. Results are that The Velvet Cloud plugins are unique, achieve a deep, psychedelic reverberation with micro-variations thanks to modulation with a background infinity sustain over musicians can play melodies on top.

Contents

Preface

This Master thesis describes the implementation of a digital modulated reverberation effect in conjunction with an automatic infinite sustain audio effect regarding its use on guitar. Both applied and combined in different forms, leads to a "dreamy" sound effect which tries to evoke, a flight over the clouds when the guitarist is playing chords and notes. The project has been developed in collaboration with Stefania Serafin and Silvin Willemsen at Aalborg University.

Virtual Sound Technology (VST) has fascinated me since I was 13 years old. Since I bought my first basic bundle of guitar plus a clean amplifier, I was much interested in getting those electric sounds out of my guitar. Since I had no money, I had to come up with an idea of getting that sound effect on a cheap and universal way; and like magic, there it is indeed when VSTs appeared in my life. With only VSTs, I produced my own CD with entirely digital audio effects processing. I was and am, so amazed by the capabilities this digital world gives us, universally, without purchasing power distinction, anywhere, anytime, that I really had a goal in mind; create my own VST and designing my own sounds.

This thesis is structured as follows:

Chapter 1 gives a brief introduction about how reverberation and sustain are tackled in the thesis. Also, the inspiration and goals that were initially set will be stated. Chapter 2 describes the state of the art of both audio effects implementation approaches as well as the plugins that can be found in the market followed by a light evaluation and discussion about them. After this review, in Chapter 3 comparisons and a discussion about the different implementation algorithm will be carried out. At the end of this chapter, the most suitable approaches in accordance with the Project Goals 1.5 are chosen to be used in this project. Chapter 4 and 5 describe the algorithms created based on the chosen implementation approaches for sustain and reverberation respectively. Then, Chapter 6 presents three different evaluations; a Technical Evaluation (based on Impulse Response (IR), White Gaussian Noise (WGN) response and Reverberation times (T60 times) within different frequency bands), a Two Alternative Forced Choice Evaluation, and a VST Evaluation of the plugin within a Digital Audio Workstation (DAW). Results of these

evaluations can be found at the end of Chapter 7. Chapter 8 discusses the results of the different evaluations. Lastly, Chapter 9 concludes the thesis and presents possible future work to be done on the final plugins.

Finally, I would like to thank heart-fully Silvin Willemsen for supporting me, specially during the hard times of real time implementation and supervising me thoughtfully, with eternal patience and kindly during the course of this project. Also, I would like to thank Stefania Serafin for her guidance and making very useful suggestions for the fulfillment of this project.

Aalborg University, February 27, 2021

Chapter 1

Introduction

The Velvet Cloud (TVC henceforth) is a digital compound/complex sound effect comprised of two distinct sound effects: Sustain and Reverberation. Independently, both can be merged in different ways (parallel or series) and its particular parameters can be tweaked separately. The common particularity of both effects is the usage of velvet noise for its DSPs achieving the high computational speed required for a real time plugin application.

1.1 Inspiration

Since the young years of the author, she was interested in how VSTs are developed, mainly, because they give the power (with little money) to produce a whole CD using only DSP and instruments. Also, as being a guitarist, the author was really interested in musical pieces of low complexity musically speaking while having an incredible sound richness and complexity and how that fulfills the atmosphere (listen to [62] for an example). Precisely, one of the sound effects that fills the environment such as no other effect does, is reverberation. Enchanted by how far this effect can be taken and amazed and inspired by the sound a DSP such as Strimon Big Sky [52] can achieve, TVC has followed this wake in the form of a digital VST plugin.

1.2 Sustain approach

Yet in the market, only a pair of sustain VST [22, 6] can be found. On the other hand, digital and analog standalone guitar effects that can act also as a sustain can be located [9, 51], although not many neither. In this project, sustain algorithms from the state of the art were studied to implement finally an automatic infinity sustain based on convolution with velvet noise to produce a VST.

1.3 Reverb approach

The field of artificial reverberation has been studied widely since the 60s [59]. Diverse algorithms for implementing digital reverb can be found in literature as well as many digital reverb plugins can be located in the current market. After analysing the different possible approaches, a Velvet Feedback Delay Network (VFDN) [11] was implemented as a starting point. In conjunction, a new approach (not found in scientific literature with respect to Feedback Delay Networks) was added to the VFDN; the use of Modulation. Finally, a Modulated Velvet Feedback Delay Network was implemented as a digital audio plugin to be used in any DAW.

1.4 Reverb and Sustain integration

In the early stages of this project, sustain and reverb effects were thought to be combined as one standalone plugin. Nonetheless, having them separately allowed the user to combine in her/his preferred configuration (e.g. signal flow follows first sustain and after reverb) at first glance and study the user preferences towards both plugins combination.

1.5 Project goals

The main goal of the project was to create a VST plugin effect which combines sustain and reverberation audio effects in one plugin, to be used on an electric guitar (this finally would be set up as 2 distinct plugins thought to be used together). Both effects, combined together as a system in distinct forms (parallel, series), was thought to achieve an eternal reverberation and heavenly atmosphere. The combination design of the effects is further explained in section 7.1.6.

1.5.1 System requirements

More in detail, TVC features are be comprised of:

Sustain

An infinite sustained sound was desired, preferably, without coloration or spectral smearing. Moreover, the effect should be triggered automatically; when input reaches a maximum (strumming a chord) a new chord/note onset must be sustained. Besides, it should be fast enough computationally speaking to work in real time without noticeable latency then having smooth transitions.

Reverb

A long lasting ending tail was sought to create an "abstract" or "space" atmosphere, which could not be feasible in a real physical place. Therefore, it was not of our main interest the reverberations whose aim is to reproduce the acoustics of a special room, plate or spring size with specific characteristics. Rather, the TVC goal was to recreate an unreal space, out of the world, like a cloud full of ringing sustained reverberation echoes.

GUI

To represent the possible settings of TVC, was desirable to have an attractive GUI which increased the expectancy of the participants during its evaluation. This GUI must be kept simple so anybody can (more or less) understand what to expect if switching/pressing buttons without technical language.

Open Source code

Code was released to the public as a thank you for the "free" VSTs that once were used by the author of this thesis.

Chapter 2

State of the Art

This chapter is divided into four sections. First, the state of the art (SOTA) 2 regarding algorithm implementations approaches both for sustain and reverb are described. Afterwards, the SOTA regarding digital sustain and reverb commercial plugins (VSTs) is described followed by a discussion.

2.1 Algorithms for implementation of Infinite Sustain

Not much research have been found regarding the sustain audio effect, as it is not much exploited in the plugin market neither. The main approaches used are:

2.1.1 Synthesis by modulation and pitch estimation

In [33] the author implements an algorithm to synthesize a guitar chord and sustains it by synthesis. It is based on the spectrum of the sample of the guitar played which will be sustained. It uses a sinusoidal model and partial amplitude modulation for the synthesis. This effect has no "automatic" mode as it is not triggered automatically when some condition is met (e.g. input level has reached a threshold). Instead, guitarist must press a pedal to sustain a chord. Consequently, the algorithm will search for the onset of that chord. For the conclusion, it is stated the latency makes this effect not very suitable for real time, having some glitches due to operations that takes much time to compute, such as the interpolation operation as stated by the author.

2.1.2 Additive and Granular Synthesis detecting a note onset

In the following work [37], some well-known digital effects are studied, including a method for infinite sustain. An additive and granular synthesis method is used when a note or chord is detected. Spectral composition analysis is used to detect

the note at two points. The mix of the two synthesis techniques and the two analysis times creates a sustained note with lots of timbrel variations. The mix between the synthesis techniques is controlled by tuning the volume for each one. See figure 2.1 for the effect parameters GUI.

Sadly, the proposed toolbox where the effect is hosted cannot be accessed to evaluate how well it performs.

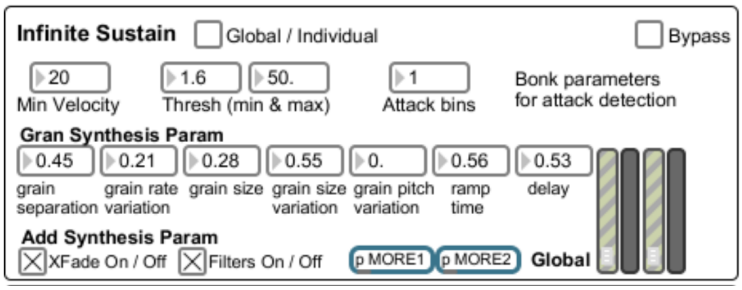


Figure 2.1: Options for Infinite sustain in [37].

2.1.3 Looping when detecting a note onset

In this method for obtaining an infinite sustain [36], onset detection is used to look for new musical notes. Once they decay to a steady state, the note is looped until a new onset comes. To loop the audio, pitch detection is performed to get the period of the signal so that phases are aligned. Refer to figure 2.2 for the workflow of the algorithm.

Although pitch detection algorithm performs well, it is often misidentifying the note by few Hz which is audibly noticeable. In the worst case, the error might get to an octave.

2.1.4 Convolution with Velvet Noise

In this new approach [8], sustain audio effect was achieved by Overlap-And-Add method. To freeze the audio signal, an input grain was windowed and convolved with velvet noise, which in essence is summing ramdomly time-shifted copies of a snippet sample with random sign. See figure 2.3 for the algorithm concept of sustain.

Results show a convincing preservation of the original spectrum through time. Fortunately, there was an available audio example of the proposed method so it could be evaluated by anyone ears [7] confirming the exposed results.

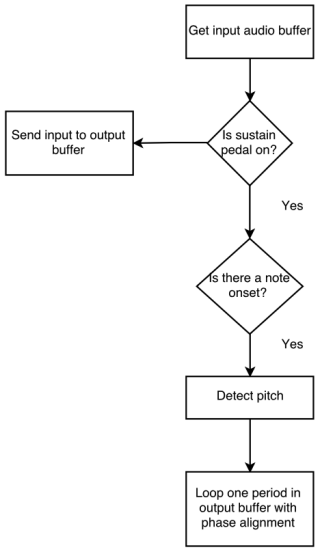


Figure 2.2: Flowchart of [36].

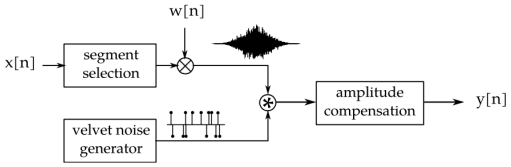


Figure 2.3: Flowchart of [8].

2.2 Algorithms for implementation of Reverberation

Compared to sustain, reverberation effect is a long studied field in the audio effects world since the 60s. Regarding its digital implementations, we can find the main following different models:

2.2.1 All-Pass filter and Comb filters

One of the earliest algorithms to tackle reverberation consist on using all-pass filters introduced by Schroeder and Logan [44], which ensures that the spectrum is not modified in frequency but does a change in phase. Afterwards, Moorer enhanced this first reverberation stage researching example architectures with delay line and filter parameter values [28]. Many improvements have been suggested since to this primary idea, like a nested all-pass structure to get a higher echo density, combining parallel comb filters followed by all-pass filters, or using sparse FIR filters to simulate early reflections [59]. Other techniques were preferred over this initial form of reverb to add the reflections complexity that lacks this old method. Although not really recent, some stills use it even for a real time implementation [63].

2.2.2 Convolution with Impulse Response

One of the most straightforward ways to implement reverberation is to convolve the audio signal with a specific impulse response. The impulse response can be recorded or modelled for a desired reverberation behaviour, which can be representative of a concrete physical space. The convolution operation is efficiently performed in the frequency domain although, the fast convolution works on big blocks of audio data which causes some undesirable latency for real time implementation [63]. Nevertheless, nowadays computers power is capable of running without much artifacts long filters up to several seconds [26], like the SpatialVerb reverb plugin does (refer to section 2.4.5). Hybrids solutions are developed for enhancing performance, for example, implementing the initial attack with a short FIR filter and the tail reflections with other reverberation algorithm. We can find works of reverberation modelling using this hybrid technique [55, 38] dating around the millennia. Notwithstanding, more modern works are still based in this implementation:

In [57], the late part of a room was modelled by dividing an impulse response into short segments and approximating each one as a velvet noise random sequence. The main drawback of this design is the computational efficiency compared to feedback delay networks (see section 2.2.4) or other reverb algorithms as discussed in the results of the paper. Although, velvet noise has been under studied recently in other studies [14] for the same purpose, and recently, in [17],

achieving a enormous reduction of the computational cost compared to direct convolution. Lastly, this method have been found not only to be used for emulation of rooms, but others like spring reverberators [31].

2.2.3 Ray-tracing technique

Also, we can find reverb effects whose impulse response was calculated as a result of using a ray-tracing technique [60]. In the ray-tracing method, a fixed number of rays is emitted in various directions with equal angles spherically from a given source point. With each bouncing, these rays loose energy according to the absorption coefficient of the different surfaces (see figure 2.4 for a representation). This technique is very useful for high-frequency sound propagation in complex architectures [2].

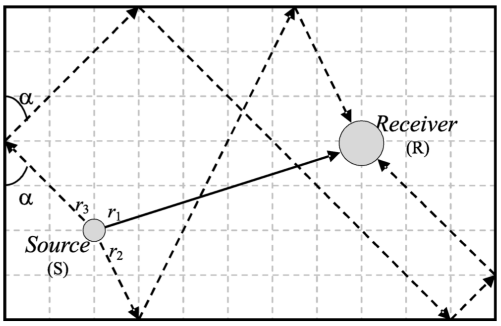


Figure 2.4: From [2], tracing three rays emitted from a source point in 2D environment.

2.2.4 Feedback Delay Networks

Proposed in 1982 by Stautner and Puckette [48], Feedback Delay Networks (FDNs) still one of most efficient reverb implementations. FDNs can be regarded as a series of delay lines interconnected in a feedback loop linked by a feedback matrix [63]. Several enhancements of the primary algorithm have been proposed along, such as using orthogonal matrix to control the quality of the reverb, increasing the FDN order and therefore the number of delay lines [59], using a different reverberation time for each frequency band [20], etc. Hereunder, we will consider some recent variations of the FDNs algorithm:

In [35], the user can control the FDN in real-time by adjusting its reverberation time in octave bands, the feedback matrix type and delay-line lengths showing that choosing too wide or a too narrow length range was disadvantageous to the synthesized sound quality. In the same line of study, [34] proposes having control over an accurate graphic equalizer as the attenuation filter of the FDN to reproduce

a reverb behaviour in a room. This option leads to more realistic artificial reverberation. To understand and analyze more accurate how FDN models artificial reverberation, a modal decomposition was resolved in [42]. Refer to figure 2.5 for a conceptual overview of modal decomposition.

Circulant and Elliptic FDN [40] have also been proposed, achieving more computational efficiency and versatility. Versatility was achieved by introducing the matrix eigenvalues which act on the distribution of frequency peaks, affecting color and smoothness of the reverberation. Other works also studies Circulant FDN [39] and the particular choices of the feedback coefficients to achieve a maximum echo density while reducing the time computation. Some of the drawbacks that FDN present is the initial slow build-up of echoes. This was tackled in [11] by convolution with velvet-noise, which have random coefficients at the input and output branches resulting in using half of the delay lines to obtain the echo density of a conventional FDN and therefore, saving computation time.

In [57], a new approach using radiance transfer method facilitates the "manual" setting of FDNs parameters from prerecorded impulses responses to emulate room acoustics. The proposed solution links straightforward the FDN parameters to the geometry settings.

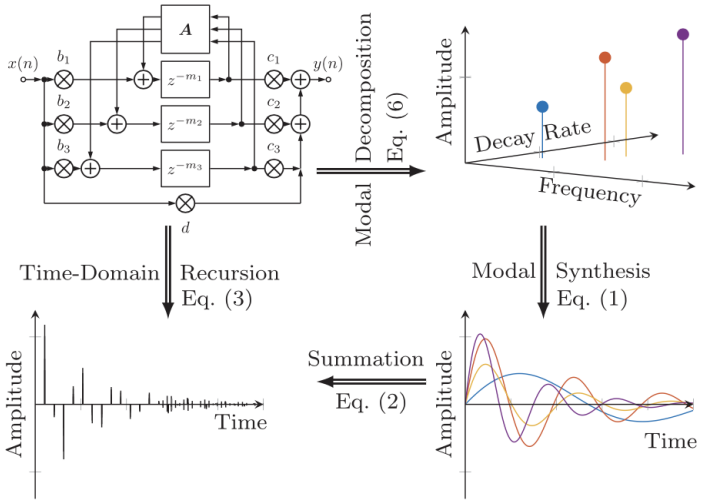


Figure 2.5: Overview of modal decomposition and synthesis of a FDN [42].

2.2.5 WaveGuide Networks

Wave Guide Networks (WGNs) model reverberation from a physical modeling point of view [59], following the scheme of interconnecting acoustic tubes. First introduced by J.O Smith [1], they are comprised of a set of bi-directional delay

lines connected by "scattering junctions". The practical implementation splits each bidirectional delay line into a pair of delay units, which makes it equivalent to a FDNs [19]. One remarkable point of WGNs is its correspondence to FDNs, as it can be used to obtain the FDN parameters based on the physics and geometry of a real acoustic space [40].

2.2.6 Other approaches

Really simple reverberations methods have been implemented too (but not recently). As an example, in [18], a really simple algorithm generates the reverberated samples by adding together several scaled and delayed input to produce a corresponding reverberated output. The algorithm boils down to the scheme shown in figure 2.6.

Results are unknown. Plane-wave synthesis method has been used in [47] to modify the spectral energy decay curve. The parameters of this electro-acoustic system controls the temporal properties of the reverberating sound field. Results show that the number of plane waves for a diffuse sound field was inconclusive. Other reverberation alternatives consider adaptive reverberation such as in [32], which in essence models a FDN depending on the audio harmonic characteristics. Also, in [61], it can be found algorithms for analog simulation of plate reverberation.

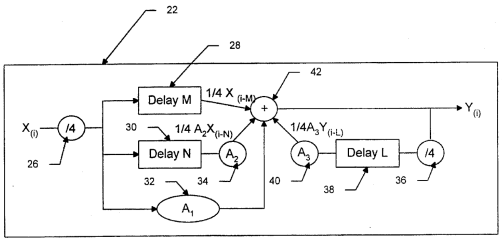


Figure 2.6: Diagram for [18] algorithm.

2.3 VSTs for Sustain

Apart from the purely digital plugins to obtain an infinite sustain, there can be also found devices that achieve similar results by making a string vibrate continuously using only analog means [54, 21, 30]. Also, guitar reverb effect pedals that count-in with an infinite sustain have been developed both in analog [51] and digital [9] processing (not remarkably cheap). There can be found some VSTs that resembles partly the sustain effect like this Compressor Sustainer Pedal plugin [15] but it was not an infinite sustain. Regarding VSTs purely dedicated to sustain there is not much offer in the market:

2.3.1 TIME FREEZER

TIME FREEZER (see figure 2.7) is a paid VST made by Marc Lingk in 2008 [22]. It promises to freeze any kind of audio in real time by pre-reading an audio file. The more outstanding tunable parameters for sustain of the plugin are:

- Max: Maximizes the output of the synthesized sound.
- Hold: to maintain the output ON.
- Denoiser: gives you the possibility of taking away the part of white noise of your source.
- Morph Time: defines how fast the transitions between the settings are going; from 1/100s to 10s.
- Frequency: if "resonance" is not set to zero, this will be the center of the band pass filter; the range is from 20 to 20.000Hz.
- Wave position: this allows you to browse in the loaded wave.
- Wave zoom: this is the size of the analyze window; the range is from 500 samples to 1.5 seconds at 44.100Hz.

Figure 2.7: TIME FREEZER GUI.

Evaluation

As there was not a free version of the plugin, videos of the effect working can be found [24] for reference. For a guitar player, it can be tedious to be continuously selecting which position of the sound audio file you want to select for the sustain while playing, as there is no automatic selection of the audio snippet that will

be "repeated" or synthesized to construct the sensation of infinite sustain. Also, loading the audio file previously to select the "wave position" we wanted to sustain did not seem a very good idea if wanting to implement a real-time sustain. The sound by itself on a guitar, sounds complex but did not recall at all a guitar [23], which is something the guitarist may want to hear too. A noteworthy thing to mention, was the built-in possibility of band-passing the sound.

2.3.2 Peak Sustainer

Peak Sustainer (see figure 2.8) is a high speed peak sustainer. It is free and only available for Windows. If the input surpasses the threshold value, the input filtered by the high pass filter will be sustained through time. Its more remarkable controls are:

- On/OFF button.
- Sustain: sustain threshold.
- Tone: input high-pass filter frequency.

Figure 2.8: Peak Sustain GUI with the used configuration.

Evaluation

From its impulse response and its WGN response we can see essentially that output is not noticeably modified as we can see from figures 2.9 and 2.10. As consequence of this, there is no measurable T60 (which will be 0 seconds).

For the sake of the evaluation, the effect has been also used over some guitar chords and it seemed it was virtually maximizing the input (only at the beginning) than rather sustaining it. It seemed not working for many notes (just acts on the first one) in time as we can see from figure 2.11. Outcomes clearly did not perform as a sustain effect is supposed to do.

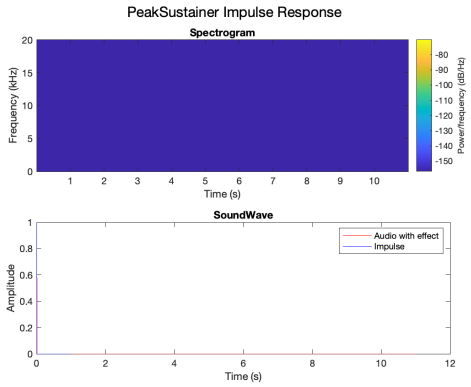


Figure 2.9: Peak Sustain spectrogram and amplitude evolution of WGN.

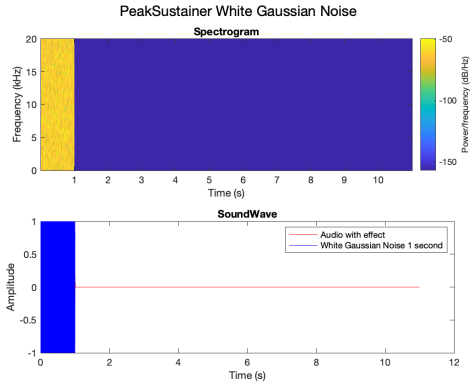


Figure 2.10: Peak Sustain spectrogram and amplitude evolution of an IR.

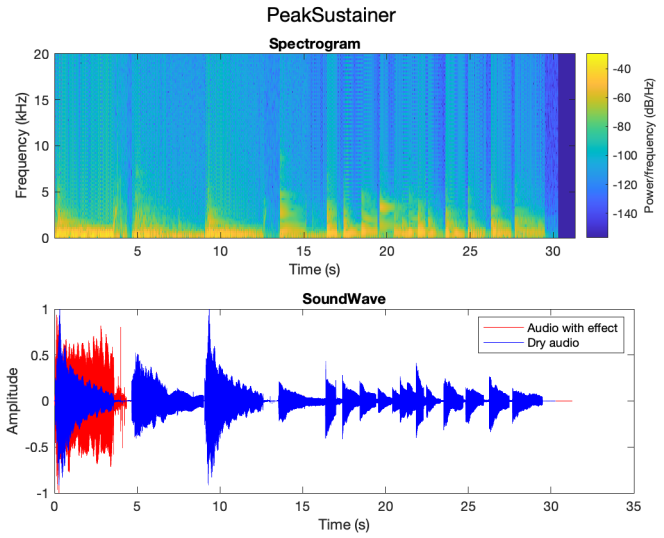


Figure 2.11: Peak Sustain spectrogram and amplitude evolution of some guitar chors.

Figure 2.12: Eos 2 GUI configuration (INFINITE button (orange) is ON).

2.3.3 Eos 2 (INFINITE Mode)

Developed by the AudioDamage, Eos 2 (see figure 2.31) [6] is the only analyzed reverb VST that comes with an infinite sustain built in. It can act as a reverb and a sustain. It has the main parameters others reverb effects have plus an special "INFINITE" button to hold reverb eternally (sustain). When Eos 2 VST is in "INFINITE" mode (see figure 2.12), it behaves as a sustain. This is what it is evaluated this section. There are no "sustain" parameters rather than ON/OFF.

Evaluation

As we can see from the impulse response in figure 2.13, the plugin perpetuated the sound without much coloration and without transients in amplitude behaving nicely through time. If input was WGN, the plugin tended to repeat and loop indefinitely the input and saturated the output homogeneously in all frequencies

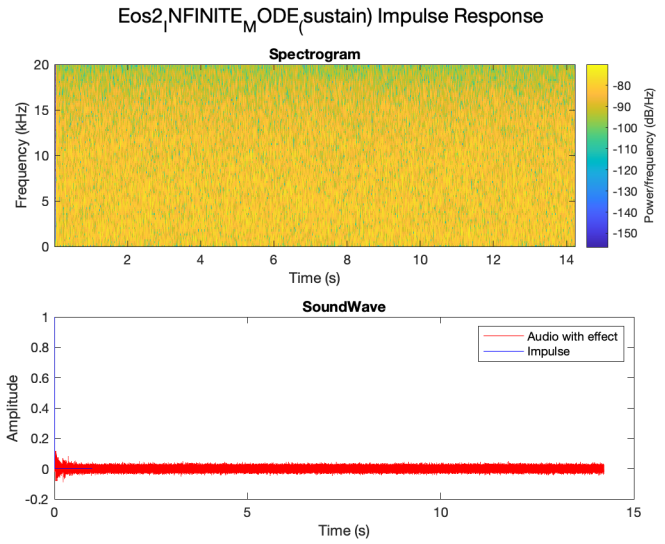


Figure 2.13: Eos 2 INFINITE MODE (sustain) impulse response.

as we can see from figure 2.14.

But when a real guitar played some chords and notes, an annoying behaviour arose when this sustain mode was ON. The algorithm just "accumulated" new incoming sounds making the output to be saturated, see figure 2.15.

This is not at all desirable for TVC, so a "trigger" or some sort of automatic sustain was finally developed as stated in the project goals 1.5.

VST Plugins for Reverb

Reverberation audio effect is a well exploited effect in comparison with sustain VSTs. There are plenty of different implementations whether paid and not paid in the market. To evaluate the plugins, IR, WGN response and measurement of T60 times will be calculated of a sound wave. In some cases, an informal evaluation by the author using as input a guitar playing a chord will be carried out too.

Calculation of T60 times

For calculation of T60 times, an impulse and white Gaussian noise have been set as input to the evaluated plugins. Output is translated into dBs using equation 2.1

$$SPL = 20 \cdot \log(p/P_o)[\text{dB}] \tag{2.1}$$

Where p is the sound pressure level, and P_o is the pressure reference level.

Afterwards, T60 is calculated as the time required for the sound to decay a level of 60 dBs. As reverberation does not behave identical in the different frequency

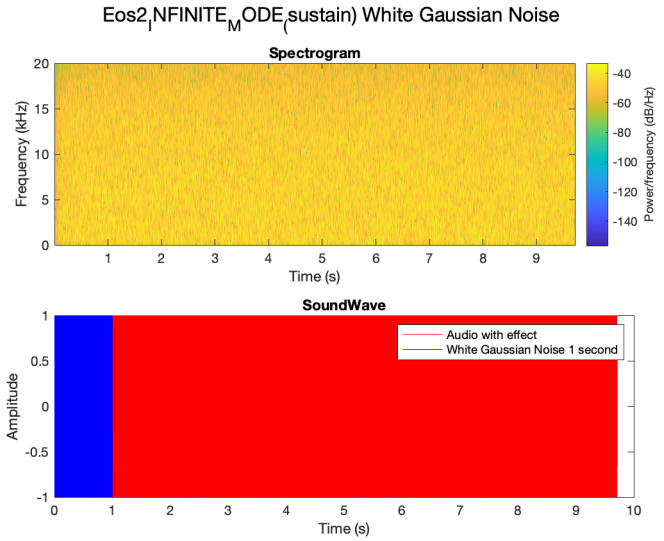


Figure 2.14: Eos 2 INFINITE MODE (sustain) WGN response.

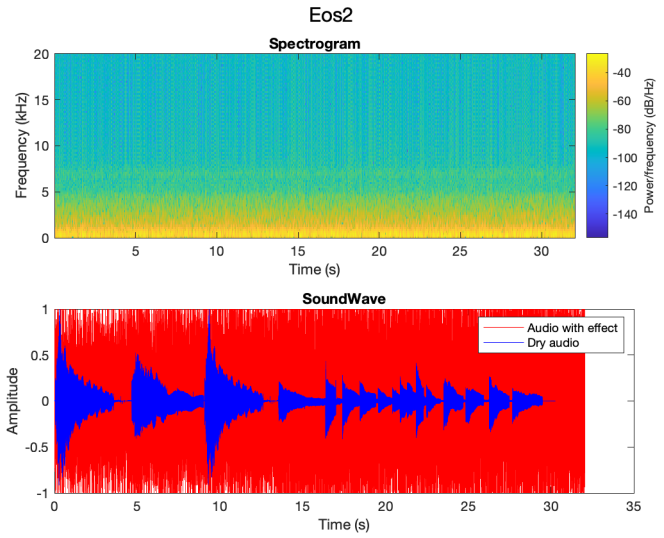


Figure 2.15: Eos 2 INFINITE MODE (sustain) saturating when a guitar is being played.

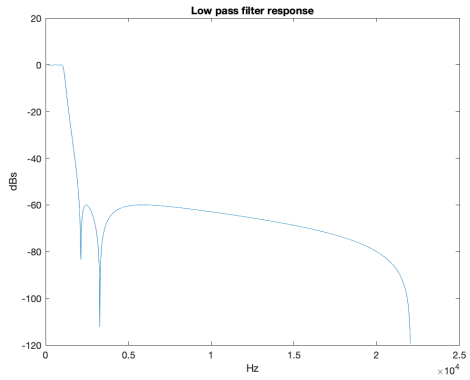


Figure 2.16: Low pass filter response with frequency cutoff at 1000 Hz.

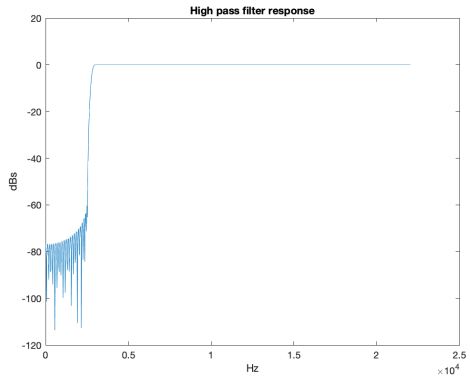


Figure 2.17: High pass filter response with frequency cutoff at 3000 Hz.

bands, T60 is also measured after filter the plugin output through a low pass filter and a high pass filter of cutoff frequencies 1000Hz and 3000Hz respectively (see figures 2.16, 2.17 for filter responses), over sounds containing a range of frequencies of 0Hz - 22500Hz.

Here, some of the most outstanding will be highlighted:

2.4 VSTs for Reverb

2.4.1 Valhalla Super Massive

Developed by the acclaimed ValhallaDSP audio effects producer, Valhalla Super Massive (see figure 2.18) [5] reverberation effect promises clouds of reverb and swelling waves of feedback among others. In their web page [4] it is said about a first stage of using Feedback Delay Networks and some algorithms that did not want to be "tamed". So then came the decision to switch into "algorithms to move

outside the realm of reverberation", but no clear information of the used algorithms is mentioned, being kind of hidden. Its most outstanding controls are:

- Mix: controls the balance of the input signal with the chorus.
- Width: Controls the width of the stereo image.
- Delay: sets the delay time in milliseconds.
- Wrap: adjust the delay length relative to the delay settings.
- Feedback: controls the amount of feedback around the delays.
- Density: Controls the density of echoes when feedback is turned up. 0
- Mod Rate: controls the rate of the delay modulation.
- Mod Depth: controls the depth of the delay modulation.

Figure 2.18: Configuration used for Valhalla Super Massive.

Evaluation

As promised, the VST with its reverb knobs turned to the maximum, achieved a cloud of nearly eternal echoes (similar to an infinite sustain) and delays that created a fulfilled atmosphere. It was a reverb that did not model a real room or environment, but rather created abstract reverberations. From figures 2.19 and 2.20 we can see the spectrogram and amplitude evolution are maintained through time without much artifacts (there are no T60 as they do not decay in time). By informal listening, one can slightly hear a "buzzing" sound in frequencies although not visible in the spectrograms. It could be related to the MOD (modulation) parameter that can be controlled in its configuration, see figure 2.18.

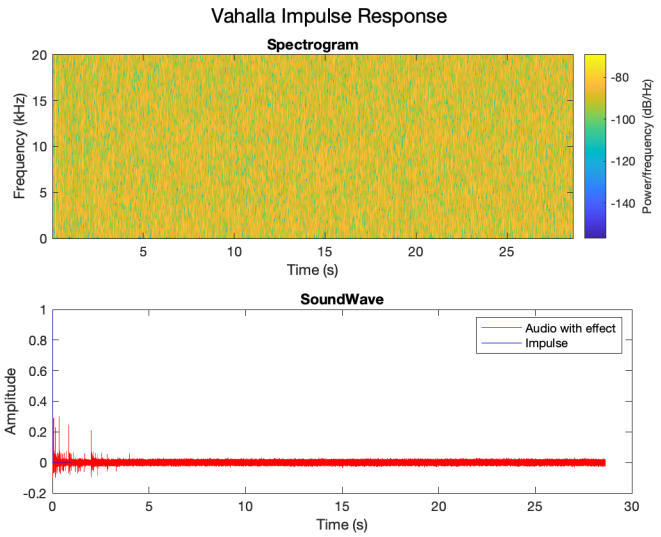


Figure 2.19: Valhalla Super Massive impulse response.

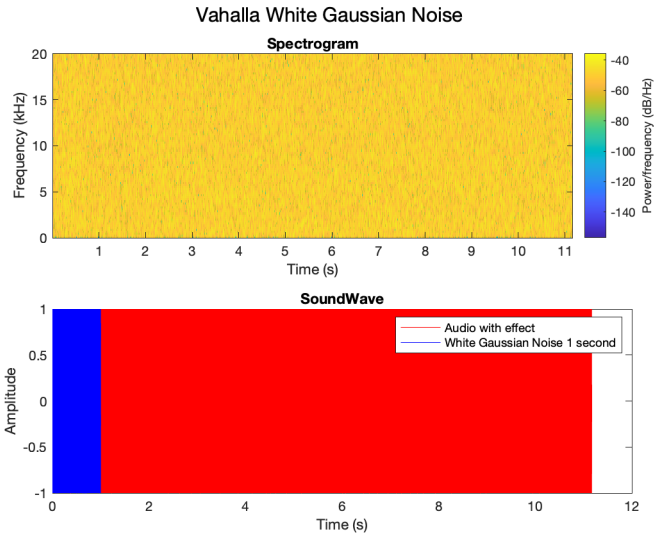


Figure 2.20: Valhalla Super Massive white gaussian noise response.

2.4.2 NeoVerb

Developed by the famous audio effects producer iZotope, NeoVerb (see figure 2.21) [16] reverb is AI powered and includes an intelligent Pre EQ section, which identify regions that have muddy harmonics that might create unwanted artifacts in your final reverb output. Also it includes a Reverb EQ includes specialized masking-reduction technology, and even a Reverb Assistant to select presets automatically. Its is comprised of three sections (early reflections, play tail and hall tail). Both "tail" engines apply the same parameters to different reverb algorithms. Unlike the Early Reflections, these engines generate myriad echoes of varying time and frequency as they bounce around in a space.Its most remarkable parameters are:

- Space: is a macro control that adjusts both the Time and Size parameters.
- Diffusion: how many reflections will arrive.
- Angle: strength of the initial reflections.

Figure 2.21: NeoVerb GUI configuration.

Evaluation

NeoVerb simply did what promised, kept a truly brilliant balance between the dry signal and the output reverb even if the user did not have much knowledge of reverb parameters. Having selected the preset which plays the longest decay tail (times set to maximum), the frequencies did not get just accumulated, but balanced in the whole dynamic range, producing a really pleasing reverb. It can be seen how big the T60 times are (see figure 2.24). Notice a higher decay for the high frequencies in figures 2.22 and 2.23, which gave a smoother decay sensation in the high range, such as real room would perform. Notice in figures 2.22, 2.23 and 2.24 the snippet of silence around second 14; this is due to the fact that the plugin was a DEMO version.

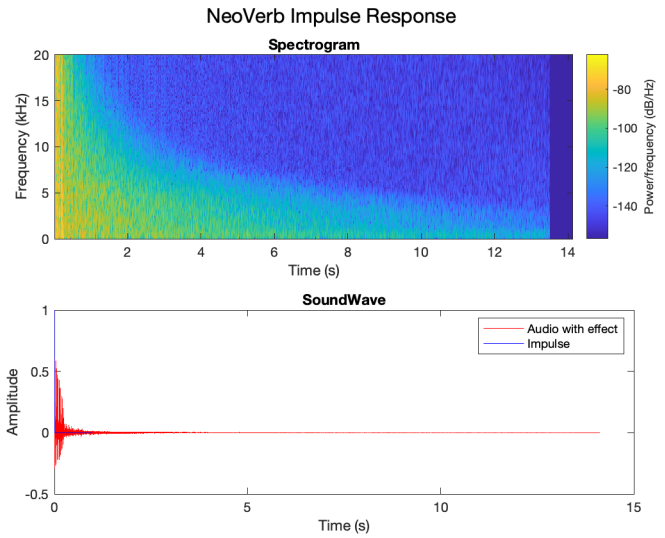


Figure 2.22: NeoVerb impulse response. Notice the silence around second 14 due to the demo version of the VST.

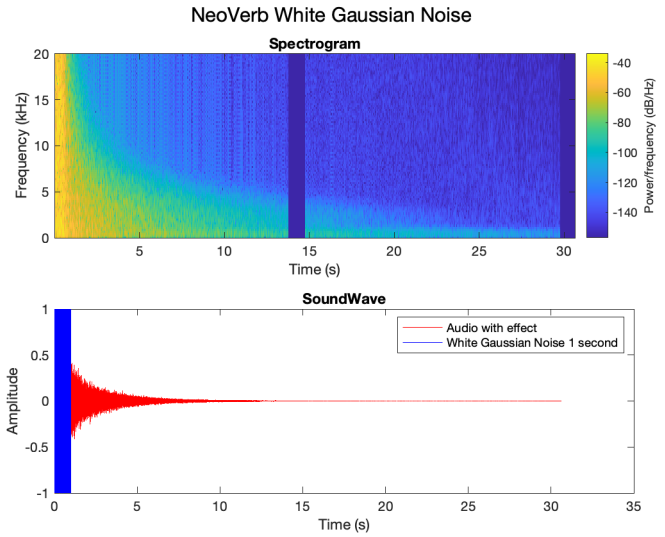


Figure 2.23: NeoVerb white Gaussian noise response. Notice the silence around second 14 due to the demo version of the VST.

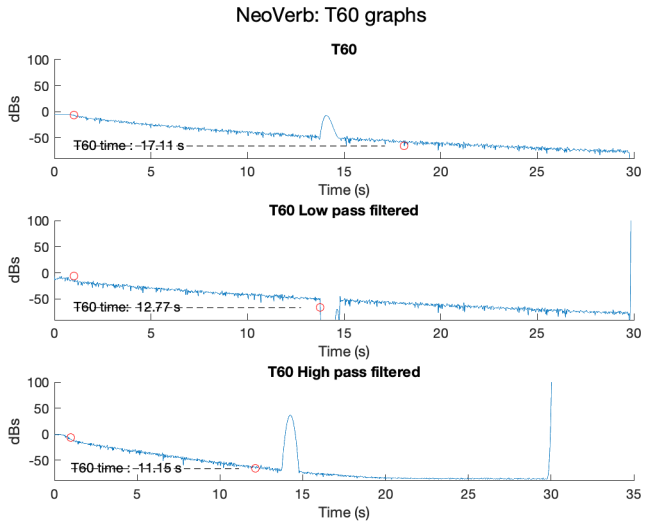


Figure 2.24: NeoVerb T60s graphs. Notice the silence around second 14 due to the demo version of the VST.

2.4.3 DreamScape

Developed by Minimal System Instrument. Unlike most of algorithmic reverbs, DreamScape (see figure 2.25) [13] does not separate its reverb into plate, hall, and room algorithms. Rather, it presents intuitive parameters to model your own personal reverb and design room reverbs, synth pad and delay effects, experimental reverb, filter effects among others.

Figure 2.25: DreamScape GUI configuration.

Evaluation

The plugin Impulse Response (IR) was very long and in some points it showed a periodic behaviour distributed as we can see from figure 2.26, easily noticing the "repetitions" of the impulse through time, which was not certainly pleasing to the ear. Although, not so important, as it tended to get diffused in time.

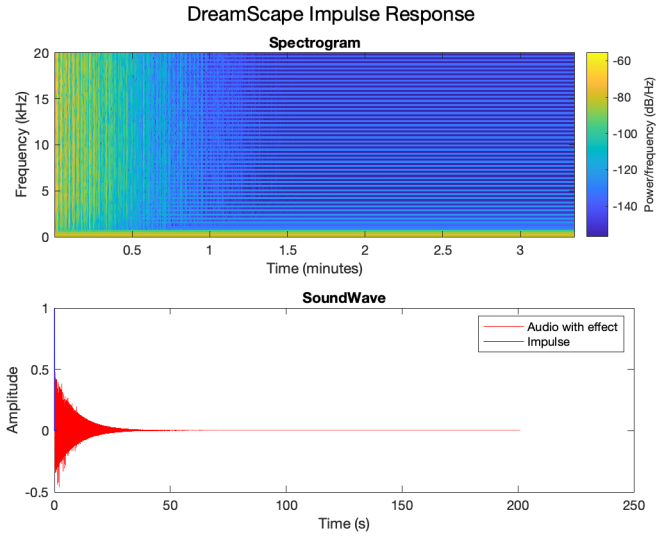


Figure 2.26: DreamScape impulse response.

In figure 2.27 with a less reverberating configuration we can notice this periodicity.

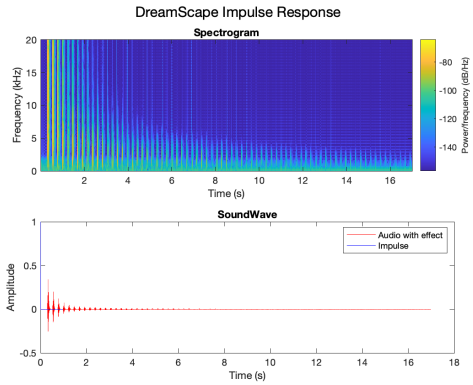


Figure 2.27: DreamScape impulse response with a less reverberating configuration.

From the WGN response, see figure 2.28, there was an equal decay in frequencies, modelling spaces which were not real. Actually, from the plugin T60s times

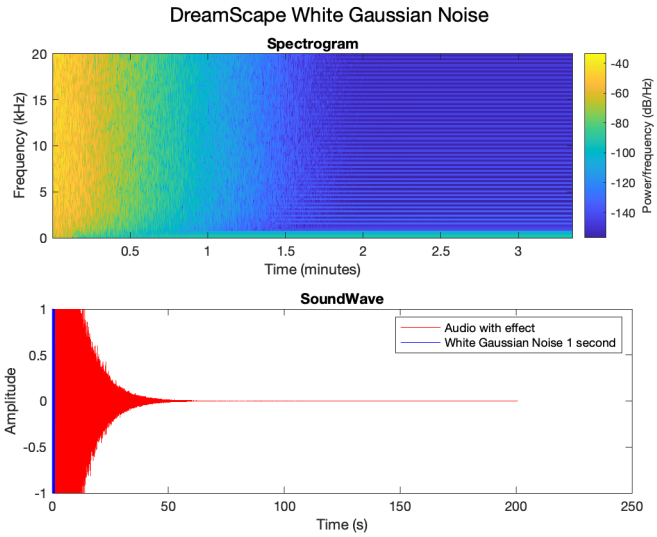


Figure 2.28: DreamScape WGN response.

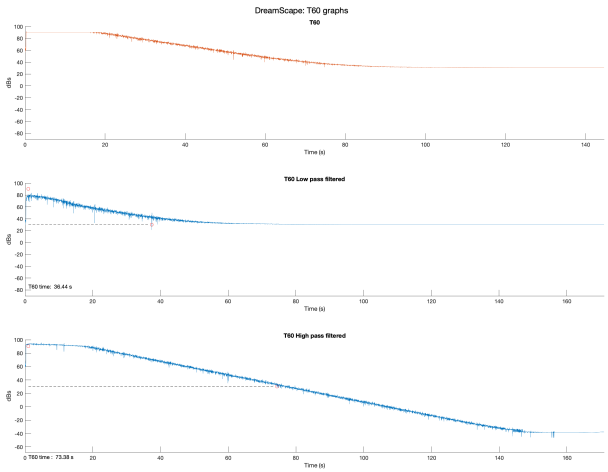


Figure 2.29: DreamScape T60 times. Notice top graph (in orange) to not have a T60 concrete value as output never reaches that difference.

(see figure 2.29) we can see incredibly high values.

From a more subjective point of view, some guitar chords and notes were played using the plugin effect (see figure 2.30). Output was embellished, but not as brightly reverberated as other VSTs accomplish. The use of convolution by impulse responses might be an issue.

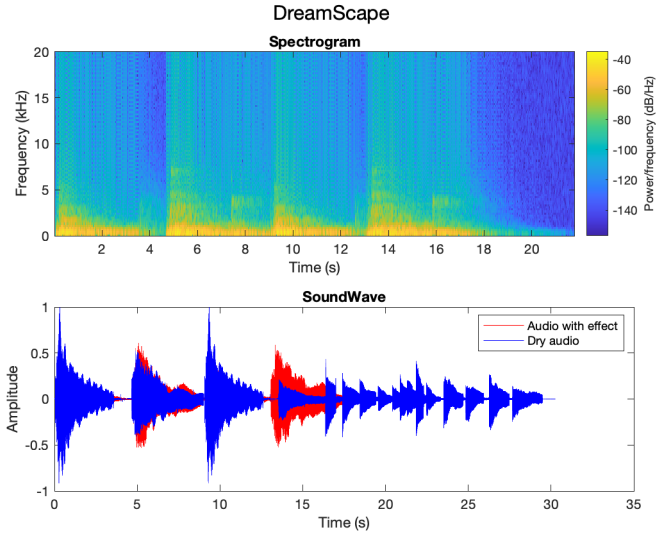


Figure 2.30: DreamSpace spectrogram and amplitude evolution.

2.4.4 Eos2

The reverberation effect from the plugin is what was evaluated in this section with the configuration that can been seen on figure 2.31 (notice INFINITE mode is OFF). Refer to section 2.3.3 for a description of the plugin.

Figure 2.31: Eos 2 GUI configuration (INFINITE mode OFF).

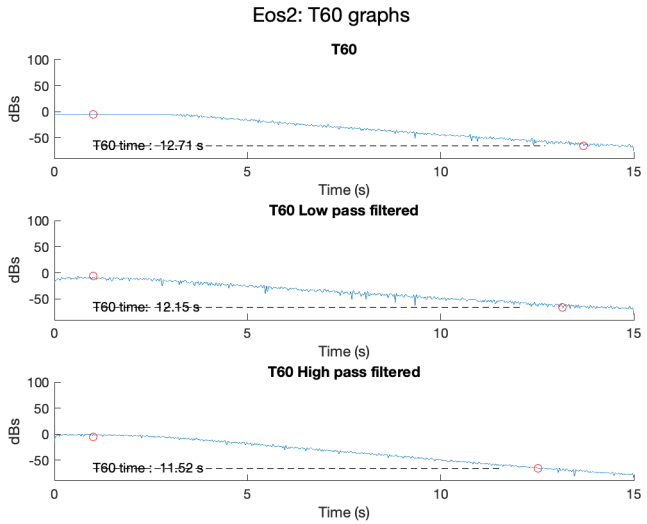


Figure 2.32: Eos 2 T60 times. INFINITE mode OFF

Evaluation

Its minimalist GUI did not need many more tricky parameters to achieve a conventional well-sounding reverb even when forcing the parameters to the maximum to achieve the greatest sounding "tail" as we can observe in the T60 graphs from figure 2.32. From the impulse response figure **??** and the white Gaussian noise figure 2.34, we can see that most frequencies decayed more or less equally in the different frequency band without much coloration. Remarkably, from the spectrograms of 2.33 and 2.34, it can be stated that high frequencies tended to slightly decay sooner.

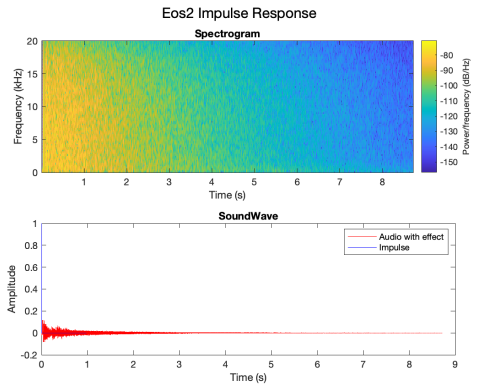


Figure 2.33: Eos 2 T60 times. INFINITE mode OFF

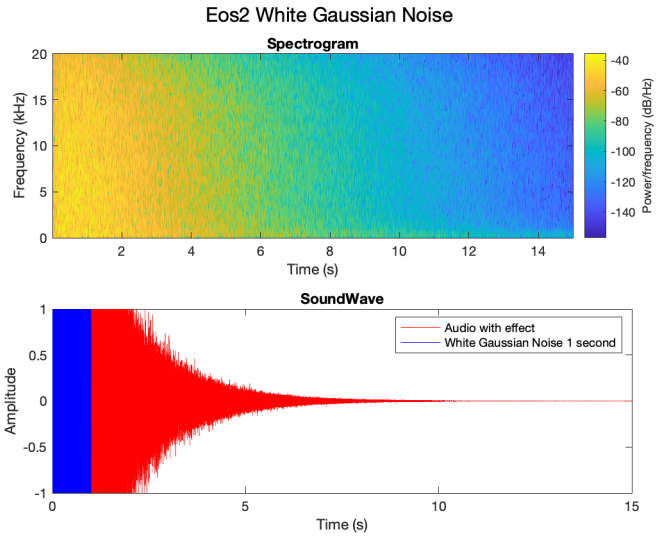


Figure 2.34: Eos 2 T60 times. INFINITE mode OFF

2.4.5 SpatialVerb

Developed by Daniel Werner, SpatialVerb (see figure 2.35) is a free plugin [60] which uses ray-tracing to generate the early impulse response. The late impulse response is governed by a Circulant Feedback Delay Network (CFDN) which presents low amount of ringing in the reverberation. There is no manual that explain how parameters are defined, but the main ones that can be found are:

- Size of the room.
- Location of two virtual source speakers.
- Two destination microphones.
- Feedback (of the FDN).
- Cutoff (of the FDN).
- WallAbsortion.
- Gain (of the FDN).

Evaluation

From the plugin impulse response, see figure 2.38, we can observe that there were not noticeable transients in the spectrogram performing a smooth decaying tail.

Figure 2.35: Spatial Verb GUI configuration.

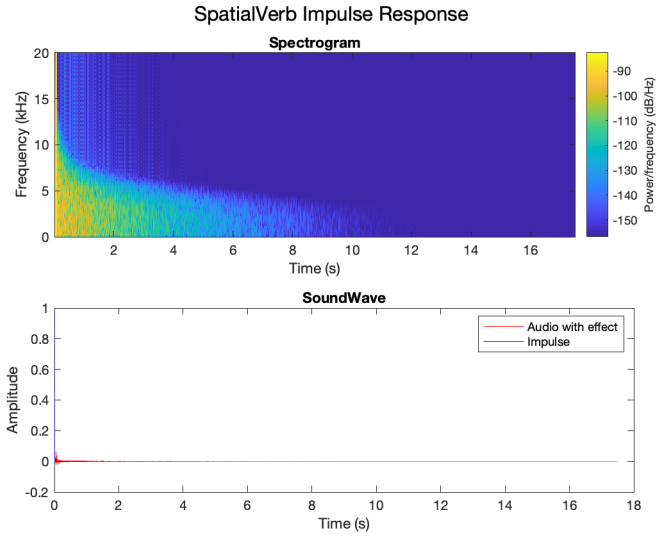


Figure 2.36: Spatial Verb impulse response.

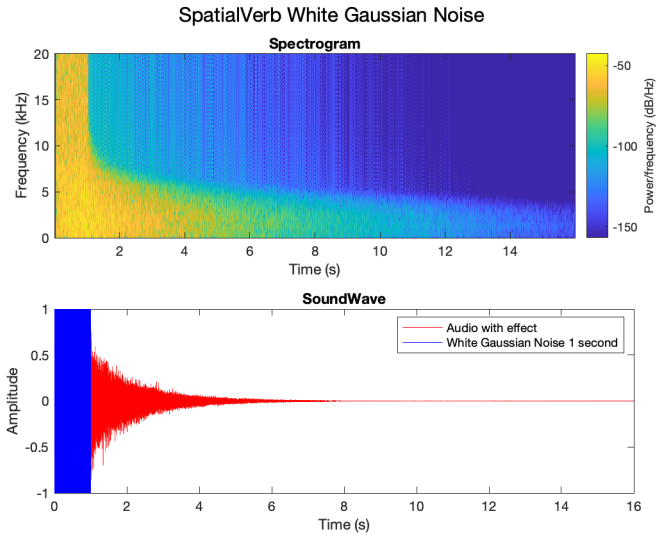


Figure 2.37: Spatial Verb WGN response.

High frequencies decayed soon, as we can see also from the WGN response in figure 2.37.

Spatialverb t60 times were within the range of 9-11 seconds, a lesser time compared to other plugins.

From a more subjective evaluation, playing some guitar chords and notes made sound get a little saturated when reverb parameters were set to the maximum, even with gain equals 0 (see figures 2.39, 2.40). It was not a really pleasant sound or reverb effect in essence.

2.4.6 FoG Convolver

Developed by AudioThing, FoG Convolver (see figure 2.41) [3] is a convolution plugin that applies the sonic character of an impulse response to another sound in real time. Its main parameters are:

- Dry: The amount of dry signal.
- Wet: The amount of wet signal.
- Pre Delay: Applies a delay in ms before the convolution.
- Gain: Increases or attenuates the volume of the Impulse Response.
- Pitch: Changes the speed/pitch of the Impulse Response. Also known as Size for generic convolution reverbs.

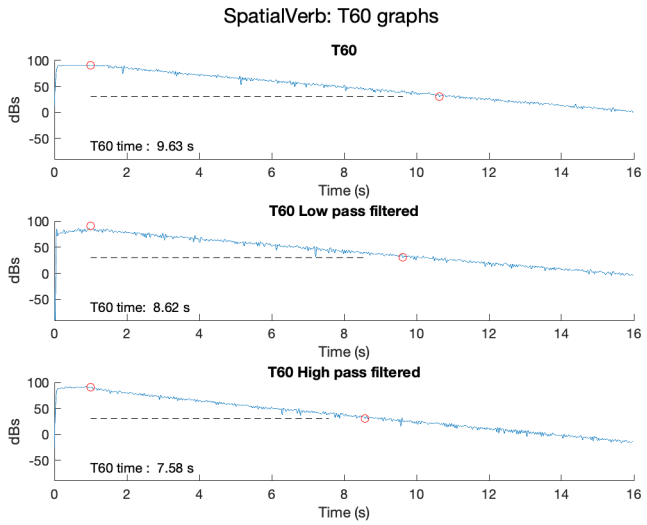


Figure 2.38: Spatial Verb T60 times.

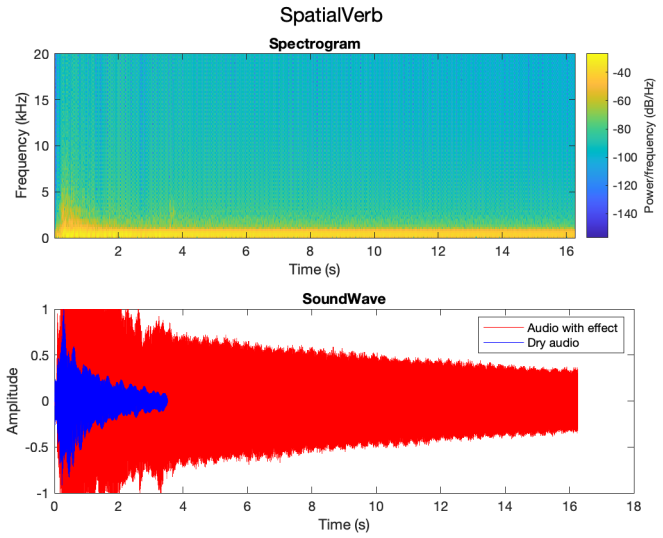


Figure 2.39: Spatial Verb soundwave amplitude evolution for one chord.

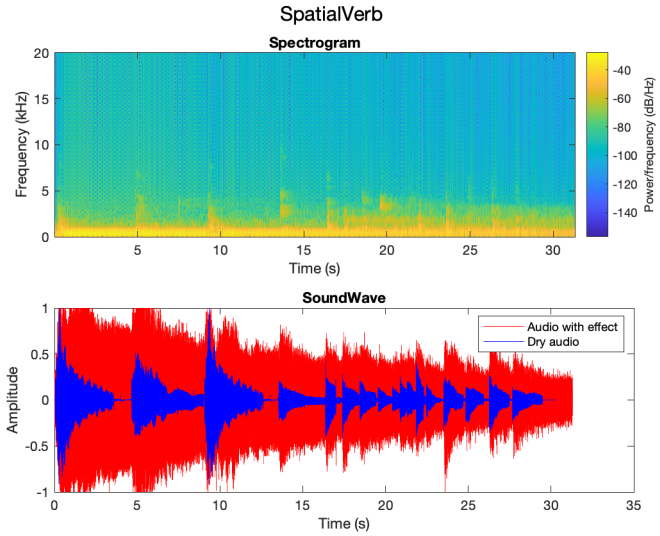


Figure 2.40: Spatial Verb soundwave amplitude evolution of chords and notes.

Figure 2.41: FoG Convolver GUI.

Evaluation

FoG Convolver is a nice VST if the goal was to model rooms rather than achieving a outer space reverb. It got saturated when reverb controls were set to maximum (refer to figures 2.45, 2.46). Its impulse response decayed very quick in time as we can see from figure 2.42. Same issue can be noticds from its WGN response 2.43.

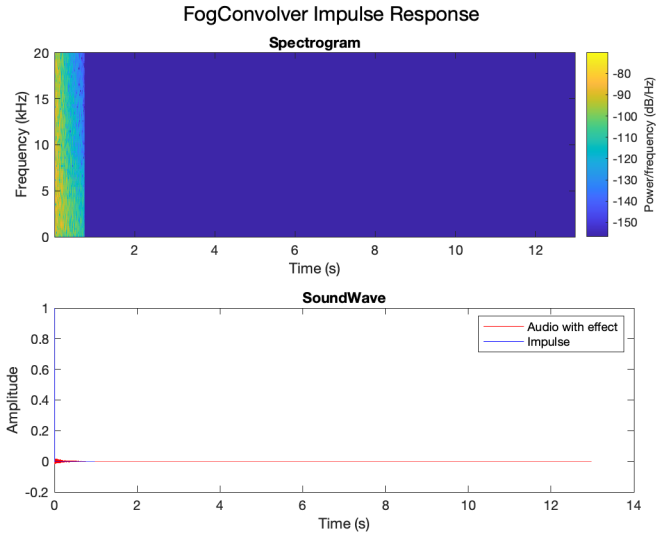


Figure 2.42: FoG Convolver impulse response.

As so, it presented short T60 times (refer to figure 2.44), far from NeoVerb or Valhalla VSTs T60 times.

We can also verify this short reverb subjectively by playing a guitar chord using the plugin (see figure 2.45).

2.5 Plugins Discussion

2.5.1 Sustain Plugins

With respect to sustain, little can be said rather than that Eos2 as said beforehand, "accumulative" infinite sustain saturated the incoming audio. TIME FREEZER was simply not really a real-time sustain in the sense of pre-loading an audio file before sounds comes out. PeakSustainer did not behave as the expected sustain effect. To conclude, little can be said about sustain VSTs rather than accumulating in a buffer incoming audio endlessly and playing it in a loop, will not work out nicely, as saturation may appear such as it does in Eos2. All in all, from the found sustain plugins, only one performs as so, although its behaviour as a little disturbing as it

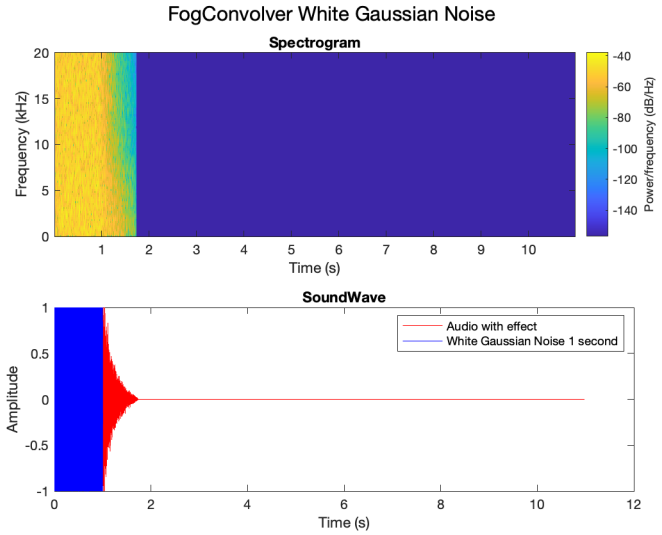


Figure 2.43: FoG Convolver WGN response.

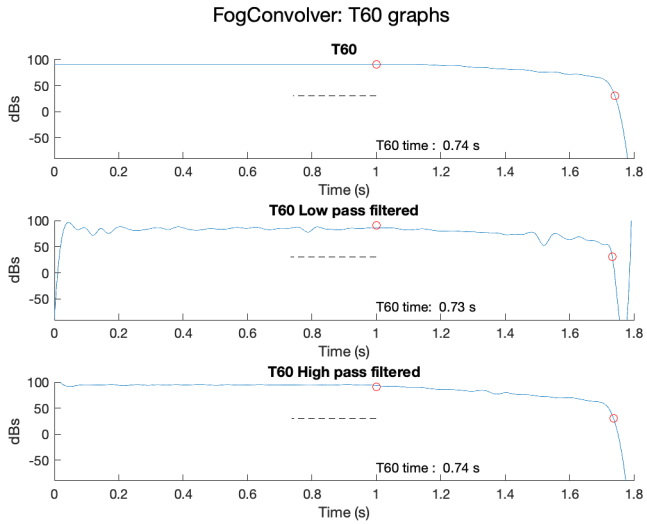


Figure 2.44: FoG Convolver T60 times.

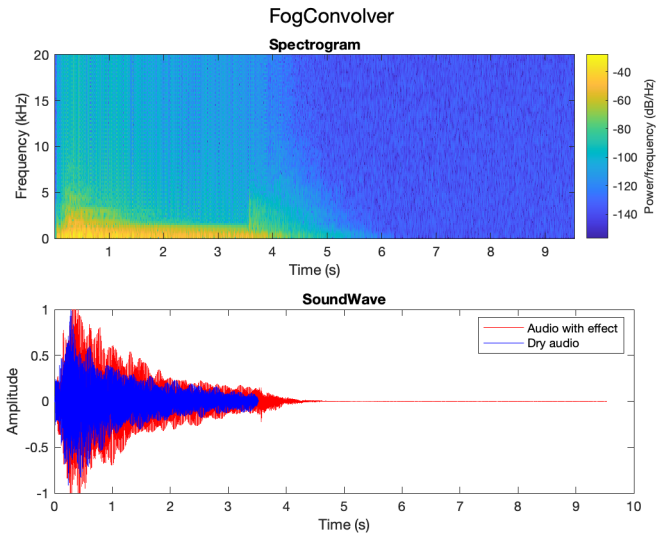


Figure 2.45: FoG Convolver soundwave amplitude evolution.

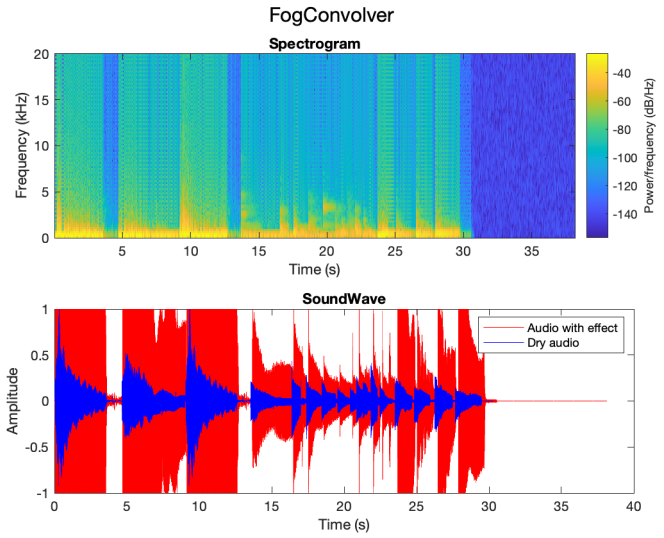


Figure 2.46: FoG Convolver soundwave of chords and notes amplitude evolution.

added more input audio to sustain on top of what was already being sustaining over and over leading to saturation.

2.5.2 Reverb Plugins

Considering all the above and in respect to reverb, Valhalla Super Massive (VSM) was the most accurate VST to reproduce a long nearly-non-decaying reverb tail without coloration, which was part of this thesis goal. In comparison, NeoVerb had also a really long T60 while taking care of the harmonic evolution of the tail, attenuating higher frequencies first as a real room would do. Other VSTs simply did not perform as well in prolonging the reverb. Probably, the use of convolution may had some role in this issue. In conclusion, from what related before, the undisclosed algorithm used by VSM may not be only only a basic FDN, but an extra different way to face reverb (an hybrid, for example). This will be taken into account when deciding the approach for reverb implementation.

Chapter 3

Implementation Decisions

Based on this thesis Project Goals 1.5 our final election would be biased towards a reverb which fulfills the whole frequency spectrum while time evolves (does not lose a concrete frequency band over time), creating a rich atmosphere during a great period of time (subjectively rather than objectively). In respect to sustain, the election was based on a trustworthy method (that sustains the correct audio snippet) while at the same time maintained low coloration and had high computational efficiency. The decisions also mind the algorithms complexity, which must be suitable for a real time implementation (TVC must implement both effects, sustain and reverb, so complexity was added).

3.1 Sustain

Not many sustain algorithms have been developed as said beforehand, but among of those aforementioned, Convolution by Velvet Noise [8] was chosen as the starting point as being the most suitable due to its high speed, low computational complexity and convincing results 3.1. It is worthy to mention that this initial algorithm was versioned to achieve an automatic sustain as part of this project goals and part of it was rethought lessening the number of operations per cycle. Others were discarded as presented more computational time (pitch detection) to be executed, wrongly detected onsets and therefore, wrong notes to be sustained or noticeably audio latency which is not desirable for TVC.

3.2 Reverb

From audible and objective measurements (T60, amplitude evolution over time) from reverb VSTs, it had been decided that the technique of convolution by impulse response was not going to be applied. This is due to the short T60 times it

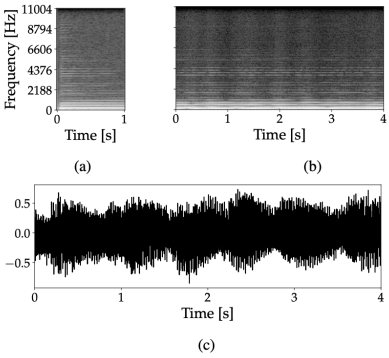


Figure 3.1: From [8], extrapolation of a guitar chord: spectrogram of the original excerpt (a), spectrogram of the extrapolated audio (b) and waveform (c).

presented and the high computation time required (due to the many operations it requires), causing sometimes audible artifacts. Adaptive reverbs [32], reverbs to model concrete physical spaces [57] or concrete frequency bands [35, 34, 42], novelty methods [47] and methods comparable to FDN (digital waveguide networks) [40] are discarded as they are not directed towards our target reverb. This leaves us among some of different FDN versions [40, 39, 11] that have been found. FDN by adding velvet noise [11] filters is a variation technique which creates a high echo density and especially reduces computation time (which was also promised in other papers [39]), and moreover, it solves the initial slow build-up of echoes. Besides, this study comes with available audible samples of the work [10], point that was missing in other papers [39, 40, 32] to verify how good sounds the reverb to the ear.

Therefore, final decision was to use as a starting point a FDN approach with Velvet Noise [11] for reverberation, not without mentioning that a new state-of-the-art advance was finally implemented on top of it (Modulated delay networks). The decision of choosing FDN was also supported by how well Valhalla Super Massive VSTs performed in the reverberation modelling and its probable usage of FDNs too. Additionally, the election was also backed by the large recent works on reverberation that have been based on FDNs and its variants.

Chapter 4

Sustain Implementation

Oriented by the work of D'Angelo [8] mentioned in section 2.1.4, the following implementation develops his main idea redesigning some parts of the algorithm and implementing an automatic version of the sustain.

The real-time plugin was developed using the Audio System Toolbox in the MATLAB environment and full code can be found in the author's Github respository [25]. When developing real time audio plugins in MATLAB, audio is proccessed in "chunks". The input is then processed in separated chunks of a standard size of 1024 samples, such as DAWs treat the incoming audio input (see figure 4.1). Reverb implementation described in chapter 5 was developed using this software

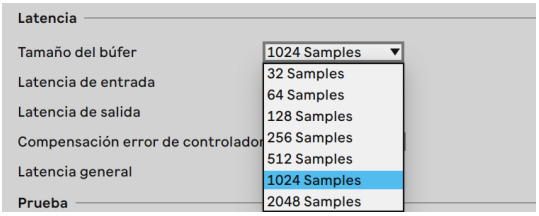


Figure 4.1: Input buffer size options for Ableton Live.

too.

4.1 Summary of the algorithm

For a complete picture of the signal flow of the algorithm, mind figure 4.2.

As aforementioned, audio data comes in "chunks" of 1024 samples. Then, the procedure is the following:

The incoming input audio chunk will be used to fill up a convolving snippet

buffer (it is the audio snippet to be sustained through time). After, this buffer is filled up and full, a set of positions of this buffer will be summed and multiplied by the correspondent signs (i.e, convolution with velvet noise). The positions and sign values are given by the generated velvet noise. The result is then multiplied by a gain (WET) and summed to the DRY signal. This constitutes the output of the system.

At some point when guitarist may strum the guitar, reaching a level higher than a threshold, a new snippet of audio will be selected and sustained through time.

A condition to be held before taking this new snippet of audio to sustain is that all values of an input chunk must have had a lower value than another threshold called *SustainThresholdReady*, as we are now ready to take another audio snippet to be sustained.

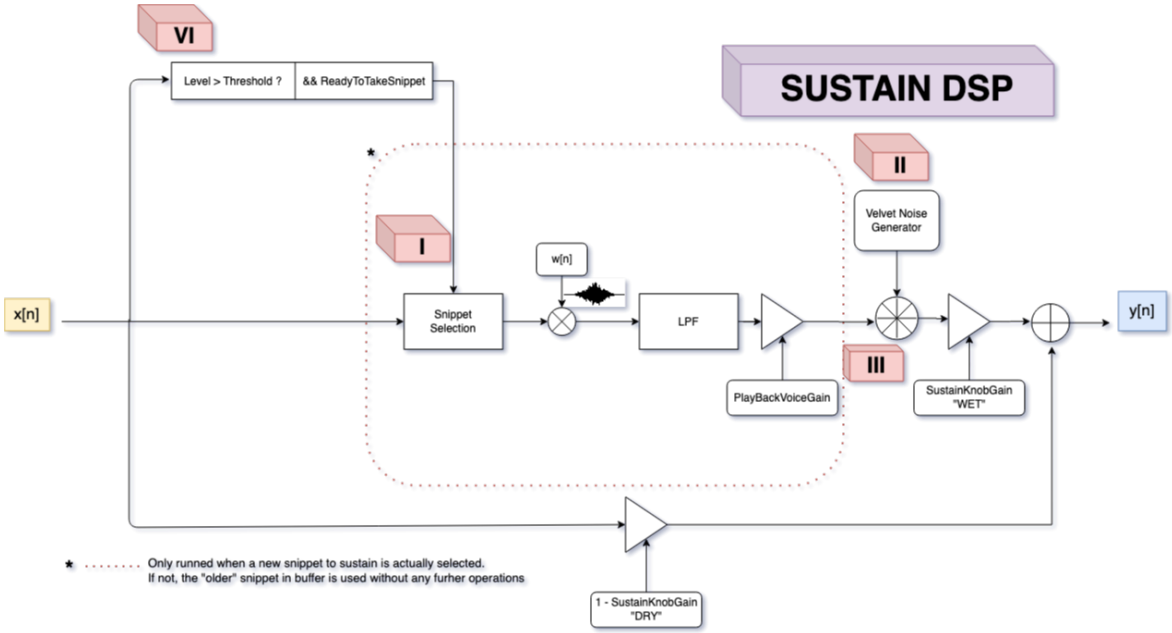


Figure 4.2: Overview of sustain algorithm signal flow. Red blocks refer to the UML processes in figure 4.3.

For a UML style of the algorithm, mind figure 4.3.

The real time implementation is divided in four sections which refers to the processes numbers I-IV in figures 4.3 and 4.2 and will be further explained hereunder:

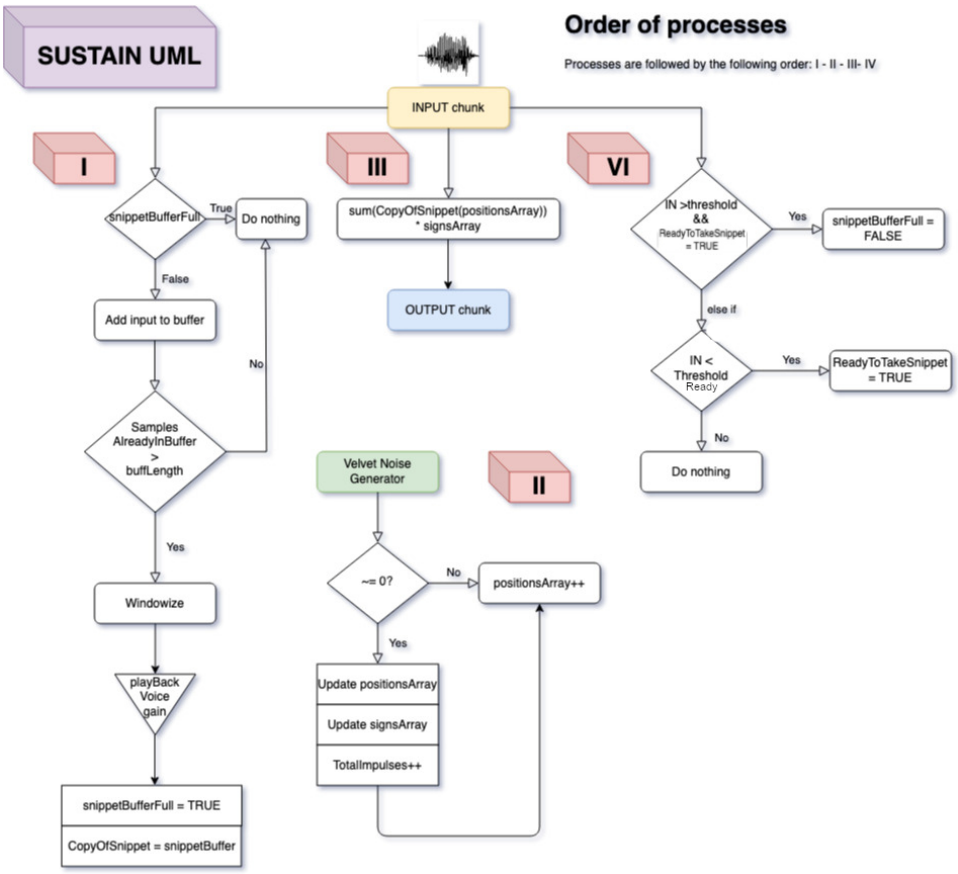


Figure 4.3: Overview of sustain algorithm UML. Roman numbers refer to sections 4.2, 4.3, 4.4 and 4.5.

4.2 I. Need to update convolving snippet?

First, the code starts by looking if *IsSnippetBufferFull*. There are two circumstances why this variable/toggle may be set to "false" (then, a new convolving snippet must be taken). One is that the plugin is been first initialized, so any incoming input will be used for filling the convolving snippet buffer and therefore be used as the sustained audio. The other is a double condition which will be further developed in the last section of sustain implementation 4.5.

Then, if *IsSnippetBufferFull* is false, the input audio chunk is used to fill up the *inSnippetBuffer* (of 30ms length). As the input chunks are smaller than the *inSnippetBuffer* (1024 samples as a standard), several times this filling up should be carried out.

```
1     % If buffer is NOT full (ie: initialization or gain > threshold)
2  if p.IsSnippetBufferFull == false % you want to "convolve" with this
       incomming snippet
3
4          % We put the chunk in the buffer
5     p.inSnippetBuffer = putVectorInBufferV4(in_mono, p.inSnippetBuffer,
          p.BufferLength, p.SamplesAlreadyInBuffer + 1 );
6
7          % Update how full is buffer
8     p.SamplesAlreadyInBuffer = p.SamplesAlreadyInBuffer + length(in_mono
          );
9
10         % If buffer is completed with the chunks,
11     if p.SamplesAlreadyInBuffer > p.BufferLength
12
13             % It is full,
14         p.IsSnippetBufferFull = true;
15         disp("bufferfull");
16
17             % windowize it,
18         p.inSnippetBuffer = p.inSnippetBuffer .* p.WindowsStatic;
19
20             % Lowpass filter it
21          p.inSnippetBuffer = filter(p.b, p.a, p.inSnippetBuffer);
22
23             % Duplicate, actually, we will use the copy for convolution
24          p.CopyOf_inSnippetBuffer = p.inSnippetBuffer .* p.
               playbackVoiceGain; % InSnippetBuff will be as high as
```

```
            playbackVoiceGain max
25
26          % and reinit
27          p.SamplesAlreadyInBuffer = 0;
28      end
29  end
```

Precisely, when *SamplesAlreadyInBuffer* is bigger than *BufferLength*, the state of *IsSnippetBufferFull* is set to true, as it is indeed full. Afterwards, a Welch parabolic window (has an exceptionally low computational cost [8]) is applied to the buffer followed by a low pass filter (Butterworth order 3 with a cutoff frequency of 5000Hz) and a multiplication by *playbackVoiceGain*. Finally, a copy of this buffer is made which will be the actual convolving snippet with which we will work further on.

Refer to figure 4.4 for an example of a *snippetBuffer* filled.

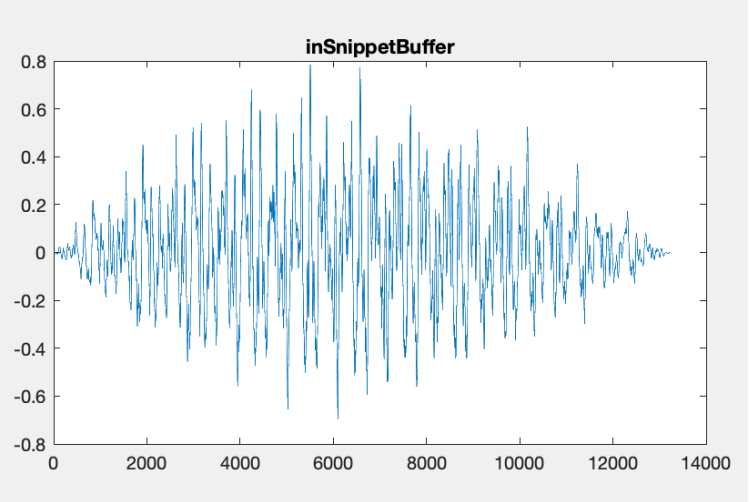


Figure 4.4: The full insnippetBuffer.

4.3 II. Velvet noise new sample

Secondly, a new sample of velvet noise is generated. Velvet noise is a type of noise constituted by -1's, 0's and 1's. It is a sparse noise sequence which uses as few non-zero values as possible [58]. In a velvet noise sequence (VNS), the sign and position of each impulse (1 or -1) are randomized, but they still remain within a given interval. The pulse density is the average impulses per second, is defined as:

$$N_d = F_s / T_d \tag{4.1}$$

Where N_d is the impulse density, F_s is the sampling frequency and T_d is the average distance of impulses. See figure 4.5 for a velvet noise example. A sample of velvet

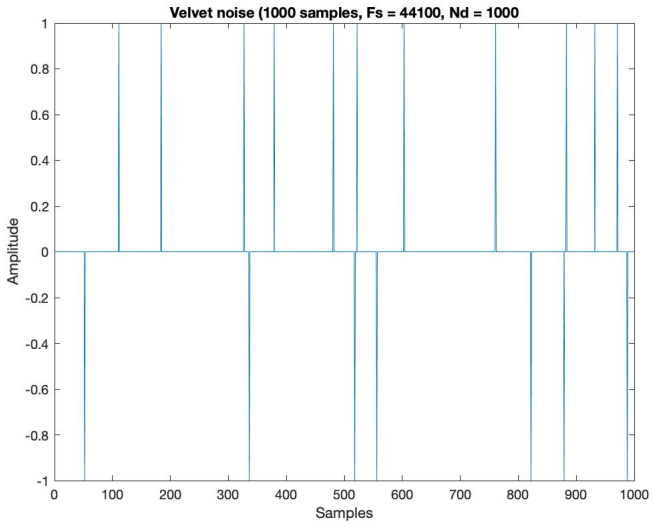


Figure 4.5: 1000 samples of Velvet Noise

noise is easily calculated with the following code:

```
function [VnoiseSample] = getVnoiseSample(Fs, Nd)
% Generates one sample of Velvet noiseat density Nd per second (Fs)

VnoiseSample = 0;

    if rand < Nd/Fs
        VnoiseSample = round(rand) * 2 - 1;
    end

end
```

Afterwards obtaining a velvet noise sample, if it is an impulse, three variables must be updated:

- The number of ***TotalImpulses*** (contains how many impulses are in the VNS) has increased, then add 1.
- ***signsArray*** (contains the impulses) must update its first position value (which corresponds to the present, now) to the value of the velvet noise impulse sign.
- ***positionsArray*** (contains the location of the impulses) must add the position of this impulse at this present time (which is now, which corresponds to 1).

```
            % Velvet noise for this time
        p.Vn = getVnoiseSample(p.Fs, p.Nd);

            if p.Vn  ~= 0 % then it is an impulse, update buffers

                % Update total impulses
             p.TotalImpulses = p.TotalImpulses + 1;

                % Put the Vn at this present position
             p.signsArray = [p.Vn ; p.signsArray(1 : end-1) ];
             p.positionsArray = [1; p.positionsArray(1 : end-1) ];
                 % Now is 1

            end
```

As time has passed by one sample, *positionsArray* advances one, which is equivalent to summing "1" to all the positions in the array.

```
        % We avance one, one delay for the postision of impulses
    p.positionsArray = p.positionsArray + 1;
```

If the last value of *positionsArray* is bigger than our *WindowsLength*, then we assign its *signsArray* and *positionsArray* to 0 (so it does not grow eternally) and we reduce the number of *TotalImpulses* by one, as this impulse has "fallen out" from out window (refer to figure 4.6).

```
        if p.positionsArray(p.TotalImpulses) > p.WindowsLength
                  % If an impulse position has reached the end
             p.signsArray(p.TotalImpulses) = 0; % set its sign to
                 zero
             p.positionsArray(p.TotalImpulses) = 0; % set its
                 position to zero

                % Update total impulses
            p.TotalImpulses = p.TotalImpulses - 1;

        end
```

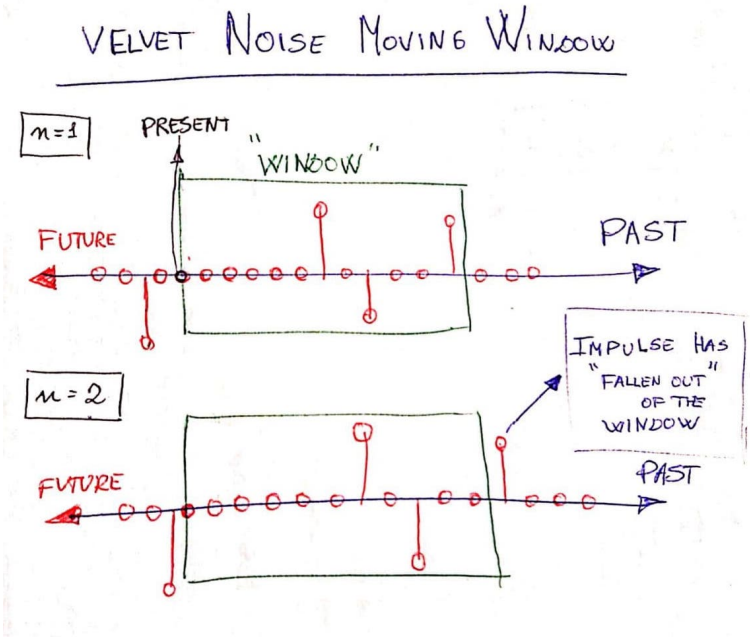


Figure 4.6: Depiction of how an impulse "falls out" from the window.

4.4 III. Convolution

To make the illusion of the audio been sustained, we convolve *inSnippet* by the generated velvet noise created in the previous section 4.3, which in essence is summing concrete samples of this snippet continuously through time. As convolution is an operator which needs many underlying operations, the same outcome can be achieved by accessing these locations of this snippet determined by the velvet noise, multiply them by the impulse signs, and finally and summing them all. This summation is the output sample, which is called *playbackVoice*, as it is a play-back of our *inSnippet*.

```
            % Multiply the impulses by the COPY of the Input snippet
        p.playbackVoice = sum( p.CopyOf_inSnippetBuffer(p.
            positionsArray(1:p.TotalImpulses)) .* p.signsArray(1:p.
            TotalImpulses));

        out_mono(n,1) =  p.playbackVoice(1, 1);
```

Subsequently, a simple dry/wet knob is implemented to model how much of the output signal belongs to the effect and how much from the original source. Finally, this is basically the output.

```
            % Mix Sustain & Input
        out = p.sustainKnobGain .* out_mono + (1 - p.sustainKnobGain)
            .* in_mono;
```

4.5 IV. Need a new convolving snippet?

At some point in time, the guitarist may want to strum the guitar and hear that chord sustained while she/he plays some improvised notes. Therefore, the code must analyze when is the suitable moment to perform a "reload" of the *inSnippet-Buffer* and sustain a new audio snippet. The condition to be satisfied for being allowed to take a new convolving snippet is that all values of the input audio chunk were below *SustainThresholdReady* in the past. If this condition is met, the toggle *ReadyToTakeSnippet* is set to true, as we are ready to take a new snippet to be sustained. Thereupon, we will in fact take a new snippet to be sustained when any sample of the input audio chunk has surpassed the threshold *SustainThreshold*. Consequently, *IsSnippetBufferFull* will be set to "false" and in the next algorithm main loop iteration, this will be handled by the algorithm discussed in section 4.2. These dynamic thresholds are required and need to be set by the user as instruments do not have same output level, and not every guitar player plays the same "loudness" or wants to trigger sustain at same levels.

```
        %% Check conditions for getting a new convolving snippet

    if any((in_mono) > p.SustainThreshold) && p.ReadyToTakeSnippet ==
        true

                p.IsSnippetBufferFull = false;
                disp("Take new snippet");
```

```
7                p.ReadyToTakeSnippet = false;
8
9       elseif any((in_mono) < p.SustainThresholdReady ) == true
10
11              % We are ready to take a new snippet
12          p.ReadyToTakeSnippet = true;
13          p.FirstRise = 0;
14          disp("ReadyToTakeSnippet");
15
16      end
```

4.6 About design parameters and implementation decisions

There are some parameters decisions specified previously without further explanation not to overcrowd the commentary of the code implementation. Thereupon, some clarifications follow:

Buffer size

The *inSnippetBuffer* is 30ms long. This is a little smaller value than the recommended value in [8]. Although, several values have been used for the length of this snippet, resulting in 30ms as a good value to work with low computational demand and nice sound.

Low pass filtering usage

If a guitar is once strummed, if the first audio cues are the ones to be sustained, it will be resulting in a striking sound, as the sustained audio is filled with lost of different frequencies of nearly the whole spectrum of a high amplitude (early beginning of a guitar strum). Rather than that, we would like not to sustain the first cue of the strum, but rather, the "tail" or where the sounds tends to fade off, as it is a more pleasant sound to hear through the time, with less amplitude and mainly low frequency content. Then, first idea could be "waiting" for that tail to happen, and then take it as the convolving audio snippet. There is a clear problem in doing that; while you wait for that "tail" to happen, another sound is being sustained and the guitarist wants to know where the sustain will be triggered, as she/he actually cannot predict when this tail will happen with high precision. And precision is a must when playing with an instrument; then this is not feasible at all. Then, the

audio characteristic of a sound fading off could be studied and tried to be reproduced without the waiting time. When a guitar is strummed, the first frequencies to decay in amplitude quickly are in a medium-high range. Then, one can think about a low-pass filter followed by some sort of amplitude control to simulate this behaviour. This simple extra adds sustain a more natural sounding and warmness without much further computation or complexity. This operation is carried out in the second code snippet showed in section 4.2, in line 12 .

Multiplication with playbackVoiceGain

As the reader may have seen from sustain implementation in III.Convolution section 4.4, "summing" randomly samples of an audio snippet may lead to saturation, as output can become really loud (due to summing many high volume samples). In [8], this is tackled using two VU envelope followers, which involves more operations and therefore less computation speed. Another way to face this problem, is just simply multiplying the taken sustained snippet once by some gain rather than being analyzing every time which is the incoming volume level so as to leverage the volume level of the sustained snippet. This approach solves the problem real quick without much more needed operations.

The procedure of storing velvet noise

From found math expressions of velvet noise definitions [11], one can regard implementing it such it is proposed. However, when running a real-time audio plugin in MATLAB, one cannot code a varying-size vector containing the impulses and positions of the VNS (the number of total impulses are within a range, but not the same number of impulses may be in a VNS and other even with the same impulse density). As the number of impulses may vary within a VNS and other with same characteristics due to their random nature, the position and sign array will vary its size depending on the iteration. That is the reason why the implementation of velvet noise 4.3 is coded as such using non-variable size vectors filled with zeros.

4.7 Tunable parameters and GUI Implementation

As stated in the Project Goals 1.5, TVC is requested to have an attractive GUI, so it is beautiful and not really complex. Therefore, the selected turnable parameters in the GUI are:

- **SustainThreshold**: when input signal SURPASSES this limit, input will be sustained.

- **SustainReady**: when input signal is BELOW this limit, ready to sustain again.
- **Cloud Volume**: gain for the sustained audio snippet.
- **Velvet Touch**: the number of impulses per second a sustained grain will have.

The standard GUI that MATLAB show for this plugin can be seen on figure 4.7:

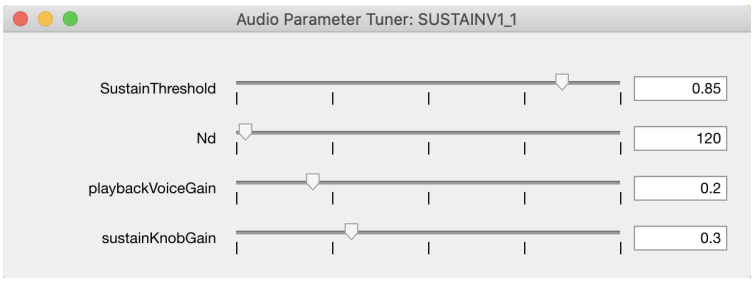


Figure 4.7: MATLAB default GUI for the Sustain effect

As figure 4.7 is not really an attractive user interface, a more colourful still minimalist GUI was designed with a "manual" within itself (see figure 4.8):

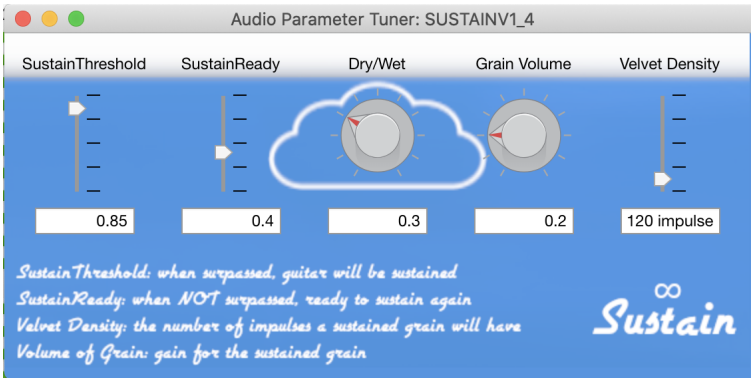


Figure 4.8: First version of sustain effect GUI

In despite of the implemented modifications of figure 4.8, still GUI did not look professional enough and parameters were preferred to be intuitive rather than using technical notation. Then, metaphors with clouds are used for naming the knobs and faders. Consequently, a redesigned GUI (see figure 4.9) was decided as final version of the GUI.

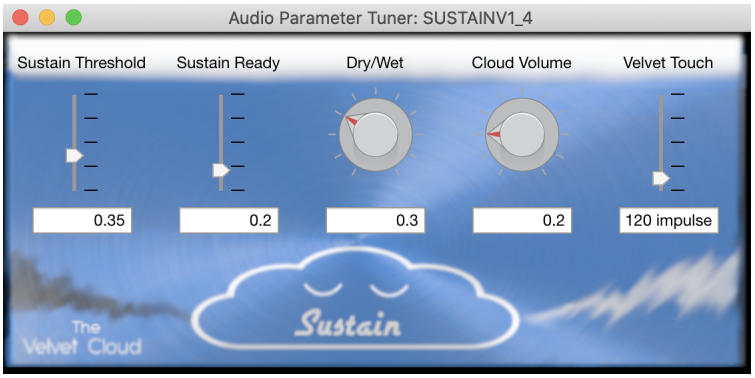


Figure 4.9: Final version of sustain effect GUI

Chapter 5

Reverberation Implementation

Oriented by the work in [11] of Velvet Noise Delay Networks, the following implementation developed its main idea adding the concept of "Modulated FDN" (MFDN) to achieve a different reverberation sound. MFDN is a new concept in FDN of which there is not much specific reference about it. Actually, no specific scientific mention to MFDN have been found in reference to reverb. Although, it could be found some references about a "wanted time variation" to break undesirable repetitions in late reverberation [46] to resemble "propagation path delays to vary over time". Also, in [53] a series of paid videos could be found where a basic FDN was using a fractional delay for modulating the delay lines (all at the same time). Likewise, [5] had a knob to control a general modulation of the delay, albeit not each delay line specifically. Still, in technical literature or scientific papers the concept of MFDN cannot be found yet as a studied reverb field. In MFDN, each delay line is modulated with some specific rate and amplitude, expanding the modulation possibilities. A similarity regarding the time variance of FDN, can be found on [43], where the feedback matrix are time-varying, although delay lines are untouched.

The real-time plugin was developed using the Audio System Toolbox in the MATLAB environment and full code can be found in the author's Github repository [25]. For the sake of brevity, FDN order 4 code implementation is referred henceforward.

5.1 Summary of the algorithm

For a complete work flow of the algorithm, mind figure 5.1.

A standard FDN consists of a set of delay lines interconnected through a feedback matrix $\boldsymbol{A}$ forming a loop (that is why it is called "feedback" delay network) [20]. The feedback matrix (also called "scattering" matrix) defines the recirculating gains for each delay line. If this matrix is orthogonal, a lossless prototype is ob-

tained, and the output of one delay is redistributed among to the input of all delay lines.

The output *y(n)* of the MFDN, for an input *x(n)*, is as follows:

$$y(n) = \sum_{i=1}^{N} c_i(z) s_i(n) \tag{5.1}$$

$$s_i(n + m_i + amp_i \cdot rate_i(n)) = \sum_{j=1}^{N} A_{ij} g_j s_j(n) + b_i x(n) \tag{5.2}$$

where b_i and c_i are the input and output coefficients, respectively, A_{ij} is the feedback matrix element, g_i is the attenuation gain, and s_i are the output states of each delay line. Notice that delay lines delays are being modulated by amp_i and $rate_i(n)$.

The transfer function of the FDN where b and c input and output gains vectors are replaced by $b_i(z)$ and $c_i(z)$ sparse VNS filters is:

$$H(z) = \frac{Y(z)}{X(z)} = \mathbf{c}(z)^T (\mathbf{D}_m(z)^{-1} - \mathbf{A})^{-1} \mathbf{b}(z) \tag{5.3}$$

where $\boldsymbol{b}$ and $\boldsymbol{c}$ are vectors containing the input and output gains, $\mathbf{Dm}(z)$ = diag($G_1(z)z^{m1}, G2(z)z^{m2}, ..., G_N(z)z^{m_N}$ and $\mathbf{A}$ is the feedback matrix.

For an overview of the algorithm in an UML style, refer to figure 5.2.

$$H(z) = \frac{Y(z)}{X(z)} = \mathbf{c}(z)^T (\mathbf{D}_m(z)^{-1} - \mathbf{A})^{-1} \mathbf{b}(z) \tag{5.4}$$

5.2 I. Generation of Velvet Noise

Velvet Noise Sequences (VNS) (as many as the FDN order) are generated only once, when the plugin is initialized. Unlike the Sustain Implementation described in Chapter 4, these VNS are set up as vectors rather than being generated on a sample by sample basis. Furthermore, there is another distinction; the VNS have the option to be exponentially decaying. The advantage of using decaying VNS instead of standard VN is that it can be optimized to have a practically flat magnitude response and prevent the smearing of transients [11]. This is at the cost of having some late impulses which may not contribute to the echo density, as they may become inaudible. See figure 5.3 for an example of decaying velvet noise.

Refer to equations 5.5, 5.6, 5.7 for the signs, positions of the impulses and decaying impulses of a decaying VNS calculation respectively:

$$s(m) = 2 \cdot \text{round}(r_1[m]) - 1 \tag{5.5}$$

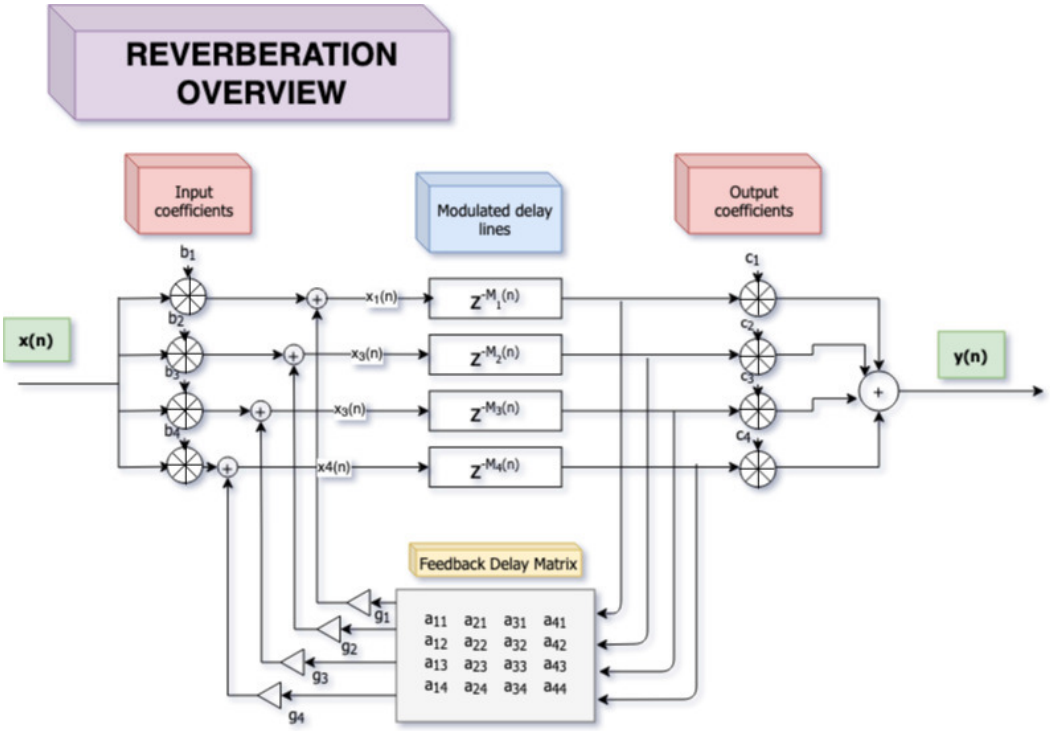


Figure 5.1: MFDN overview of order 4.

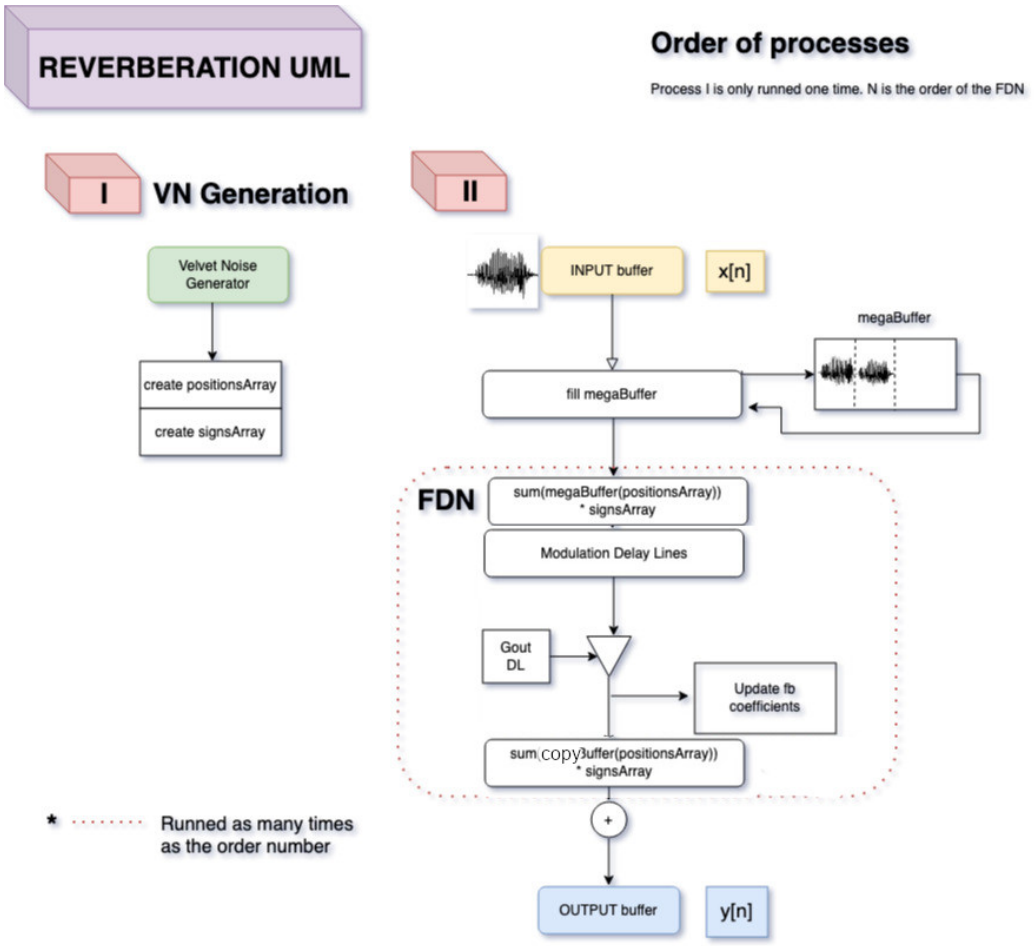


Figure 5.2: Velvet Noise Modulated Feedback Delay Network (VNMFDN) UML.

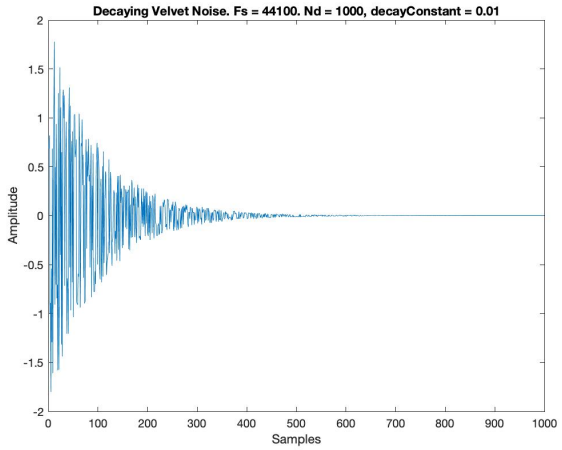


Figure 5.3: Decaying Velvet Noise with $F_s = 44100, \alpha(decayConstant) = 0.01, Nd = 1000$.

$$k(m) = \text{round}[mT_d + r_2[m](T_d - 1)] \tag{5.6}$$

$$se(m) = e^{-(\alpha m)} \cdot s(m) \cdot r_3(m) \tag{5.7}$$

Where r_1 is a sequence of random numbers uniformly distributed in the range from 0 to 1, r_2 is also a random noise sequence with same characteristics, r_3 is a vector of random numbers uniformly distributed in the range from 0.5 to 2 (as range used in [11]) and T_d is the average distance of impulses.

The following code is used for the generation of VNS (notice k to be the *arrayPositions* and se the *signArray* of decaying impulses):

```
1
2  function [vn, se, k, NumberOfImpulses ] = V3vNoiseGeneratorPAPERvelvet(
       Ls,Fs, Nd, DecayConstant)
3
4  vn = zeros(Ls, 1); % <PREALLOC>
5
6  % Grid size
7   Td = Fs / Nd; %Nd: The pulse density, or the average number of nonzero
        impulses per secon
8
9  % How many impulses in this VN?
10 NumberOfImpulses = floor(Ls / Fs * Nd );
11 m = (1:NumberOfImpulses)';
12
13 % Positions at where impulses are located?
14 k = floor( m * Td + rand(NumberOfImpulses, 1) * (Td - 1));
15
16     % make sure our locations are less than NoiseSamples size
17 k = k(k < length(vn));
18
19 % Sign (-1,1) for each impulse
20 sign = 2 * round(rand(NumberOfImpulses,1)) - 1;
21
22 % Assing the sign to those positions
23 vn(k) = sign(1:length(k));
24
25     %% Computation of Se
26 % r3(m) is a random gain between 0.5 and 2.0
27 r3 = rand(NumberOfImpulses,1) * 1.5 + 0.5;
```

```
28
29 se = exp(-DecayConstant .* m) .* sign .* r3;
30
31      % Make sure we are not taking more
32 se = se(1:length(k));
33
34 end
```

Thereupon, the loop of the real time implementation starts (II. block of figure 5.2). It is divided into five sections which will be further explained hereunder.

5.3 Update Megabuffer with Input chunk

As FDN is a reverberation effect, we need a buffer which "keeps track" of the past to use it as a playback (the actual reverberation). This is called in TVC the *megaBuffer*. *megaBuffer* will be filled up with the incoming input audio chunks using function *putVectorInBufferV4* after initializing the input and output vectors:

```
1                % Fill megaBuffer putVectorInBufferV2(in,buffer,
                     bufferLength, n)
2            p.megaBuffer = putVectorInBufferV4(in_mono, p.megaBuffer, p
                 .MegaBufferSize, length(in_mono) + p.InputLengthHolder);
                  % Write locations
```

```
1  function [buffer] = putVectorInBufferV4(in,buffer, BufferLength, n)
2  %putVectorInBuffer Puts a whole vector into a buffer from position N to
3  %N+length(in)
4
5  % BUFF = [0 0 0 0 0]';
6  % N = 3;
7  % IN = PAST <[ 1 2 ]'>PRESENT
8  % PUTIN = [1 2 0 0 0];
9
10 indexC = mod(n-1,BufferLength)+1;
11 indexL = indexC+length(in)-1; % Last position
12
13     if indexL < BufferLength
14
15         buffer(indexC: indexL,  1) = in;
16
17     else
```

```
18
19          buffer(indexC: BufferLength,  1) = in(1:BufferLength-indexC+1);
20          buffer(1: abs(indexL - BufferLength),  1) = in((BufferLength-
                indexC+2): end);
21
22      end
23
24  end
```

5.4 Input convolution with velvet noise

A convolution will be performed for each delay line by accessing the *megaBuffer* at k positions (each one corresponds to each VNS, and there are as many VNS as the FDN order). This is achieved by multiplying the sample values at these positions with their correspondent decay impulses and finally summing them all (per each delay line). Afterwards, the feedback coefficients are summed, which will constitute the input to the delay lines.

```
1   % OutB
2               % outB1
3           p.outB1 = sum(p.megaBuffer(  kCalculatorV3(p.InputLengthHolder
                +  n, p.k1', p.kLengthINPUT, p.MegaBufferSize)   ).* p.se1);
4               % outB2
5           p.outB2 = sum(p.megaBuffer(   kCalculatorV3(p.InputLengthHolder
                 +  n, p.k2', p.kLengthINPUT, p.MegaBufferSize)  ).* p.se2);
6               % outB3
7           p.outB3 = sum(p.megaBuffer(  kCalculatorV3(p.InputLengthHolder
                +  n, p.k3', p.kLengthINPUT, p.MegaBufferSize)  ).* p.se3);
8               % outB4
9           p.outB4 = sum(p.megaBuffer(  kCalculatorV3(p.InputLengthHolder
                +  n, p.k4', p.kLengthINPUT, p.MegaBufferSize)  ).* p.se4);
10
11      % inDL: Sum outB1 with fb (feedback coeffs) for each delay line
12
13                  % Combine input with feedback for respective delay
                        lines
14              inDL1 = p.outB1 + p.fb1; inDL2 = p.outB2 + p.fb2; inDL3 = p
                    .outB3 + p.fb3; inDL4 = p.outB4 + p.fb4;
```

As we are using buffers, we need a function that translates any position (positions greater than the buffer) to some positions that we actually can relate of our buffers (e.g. if bufferSize is 100 and we want to access position 125, the actual acccessed position would be 25). This is what is performed in *kCalculatorV3* function:

```
function [k] = kCalculatorV3(n, k, lengthK, lengthBuffer)


aux = n-k ;

if any(aux <= 0)

   pos =  (aux <= 0);
   pos = sum(pos);
   pos = (lengthK - pos + 1) : lengthK ;
   aux(pos) = aux(pos) + lengthBuffer;

end


if any(aux > lengthBuffer)

   pos =  (aux > lengthBuffer);
   pos = 1:sum(pos);
   aux(pos) = aux(pos) - lengthBuffer;

end


k = aux;

end
```

5.5 Modulated Delay Lines

Each delay line has its own buffer so the past can be accessed in order to perform the modulation operation. The delay (accessed past position) is determined by the *d* value of that delay line plus the amplitude (*amp*) multiplied by the (*rate*). The maximum delay is given by summing the amplitude (*amp*) to the delay value (*d*). If the result of the summation is bigger that the buffer, (suppose we want to access

past position 1000, but our buffer is 750), then, modulation will be accessing position 250 of our buffer. This modulated delays works similar like a flanger effect would do.

```
1        % outDL: Parallel output delay lines
2           [outDL1, p.buffer1] = modDelay(inDL1,p.buffer1, p.Fs, n + p
                .idx_process * length(in_mono) , p.d1, p.amp1, p.rate1);
3           [outDL2, p.buffer2] = modDelay(inDL2,p.buffer2,p.Fs, n + p.
                idx_process * length(in_mono) , p.d2,p.amp2,p.rate2);
4           [outDL3, p.buffer3] = modDelay(inDL3,p.buffer3,p.Fs, n + p.
                idx_process * length(in_mono) , p.d3,p.amp3,p.rate3);
5           [outDL4, p.buffer4] = modDelay(inDL4,p.buffer4,p.Fs, n + p.
                idx_process * length(in_mono) , p.d4,p.amp4,p.rate4);
```

5.6 Feedback Coefficients

The output of each delay line is multiplied by the feedback matrix coefficients (*a11, a12 ... a44*). Thereupon, the outputs of the feedback matrix are multiplied by each line gain to constitute the updated feedback coefficients (*fb*) which will be used in the next iteration. They will be used in next main loop iteration (see code of section 5.4).

```
1         % fb: Calculate feedback coefficients entering the matrix (
              including crossover)
2           p.fb1 = p.g1*(p.a11*outDL1 + p.a21*outDL2 + p.a31*outDL3 +
                p.a41*outDL4);
3           p.fb2 = p.g2*(p.a12*outDL1 + p.a22*outDL2 + p.a32*outDL3 +
                p.a42*outDL4);
4           p.fb3 = p.g3*(p.a13*outDL1 + p.a23*outDL2 + p.a33*outDL3 +
                p.a43*outDL4);
5           p.fb4 = p.g4*(p.a14*outDL1 + p.a24*outDL2 + p.a34*outDL3 +
                p.a44*outDL4);
```

5.7 Output convolution and output mix

Same procedure as in section 5.4 is followed: convolution with velvet noise (a different velvet noise from the one calculated for input coefficients). Subsequently, a simple dry/wet knob is implemented to model how much of the output signal

belongs to the effect and how much from the original source.

```
        % OutC
                    % outC1
                p.outC1 = sum(p.copy1_of_megaBuffer(  kCalculatorV3(p.
                    InputLengthHolder +  n, p.k5', p.kLengthOUTPUT, p.
                    copy_of_megaBuffer_Size)   ).* p.se5);
                    % outC2
                p.outC2 = sum(p.copy2_of_megaBuffer(   kCalculatorV3(p.
                    InputLengthHolder +  n, p.k6', p.kLengthOUTPUT, p.
                    copy_of_megaBuffer_Size)  ).* p.se6);
                    % outC3
                p.outC3 = sum(p.copy3_of_megaBuffer(  kCalculatorV3(p.
                    InputLengthHolder +  n, p.k7', p.kLengthOUTPUT, p.
                    copy_of_megaBuffer_Size)  ).* p.se7);
                    % outC4
                p.outC4 = sum(p.copy4_of_megaBuffer(  kCalculatorV3(p.
                    InputLengthHolder +  n, p.k8', p.kLengthOUTPUT, p.
                    copy_of_megaBuffer_Size)  ).* p.se8);

                % Combine parallel paths
            out_mono(n,1) = p.OutputGain * 1/4 *(p.outC1 + p.outC2 + p.
                outC3 + p.outC4); % Mono, I know

                        % Wet/dry: Mix reverb & Input
        out_mono = p.Wet .* out_mono + (1 - p.Wet) .* in_mono;

        out = [out_mono out_mono]; % Here you are your stereo
```

5.8 Design parameters decisions

There are some parameters decisions specified previously without further explanation not to overcrowd the commentary of the code implementation. Thereupon, some clarifications follow:

Choice of delay lines

d values are chosen randomly among prime numbers keeping a minimum distance of 400 samples and are uniformly distributed. This is intended to avoid clustering of echoes. Although there is not much consensus about the delay line values [41], their values, as suggested by Schroeder and in [45, p. 111], are chosen to be be mutually prime.

Choice of feedback matrix

The most common matrices used for FDN are unilossless. Inside this category, we can find different matrix types: Hadamard [19], Householder [19], identity matrices [27] and orthogonal. Hadamard is finally chosen as it turns out to prolong more the reverberation in MFDN in comparison with Householder. Identity matrix was rejected to use as it reduces the missing the complexity other matrices achieve in comparison to its simplicity (converts the FDN to a parallel set of comb filters).

Amplitudes and rates values for modulation

At some amplitude and rates values, the MFDN behaves as a flanger or chorus, which is not the expected comportment of a reverb effect. Then, using informal hearing, the highest values (just before turning the MFDN into a flanger) of amplitude and rates were selected. These highest values conform the "Extreme" mode for the reverb plugin. The slightest modification of amplitudes and rates in which an user can experiment a clear difference in the quality of sound forms the "Light" mode for modulation (see figure 4.9 noticing "Cloud Brightness" drop-down list).

Convolutions with velvet noise

As stated in the introductory paragraph of this chapter 5, a convolution with velvet noise is performed in the input and in the output of the FDN. this increases echo density in the beginning of the impulse response with low computational cost, with better results (faster echo density growth) than doubling the number of used delay lines [11].

Using a decaying velvet noise

Standard velvet noise cause frequency coloration in comparison of using a decaying velvet noise sequence which have a more flat spectrum [11]. Also, output is

more pleasant confirmed by an informal listening by the author.

5.9 Tunable parameters and GUI Implementation

As stated in the Project Goals 1.5, TVC is requested to have an attractive GUI, so it is beautiful and not really complex. Therefore, the selected tunable parameters in the GUI (see figure 5.4) are:

- **Cloud Height**: toggles between high feedback gains values (heaven) and standard values (sky).
- **Cloud Brightness**: selects a modulation preset. There are five different options of modulation (light, medium, high, ultrahigh and extreme).
- **Cloud Density**: standard dry/wet knob.
- **Output Gain**: to amplify output.

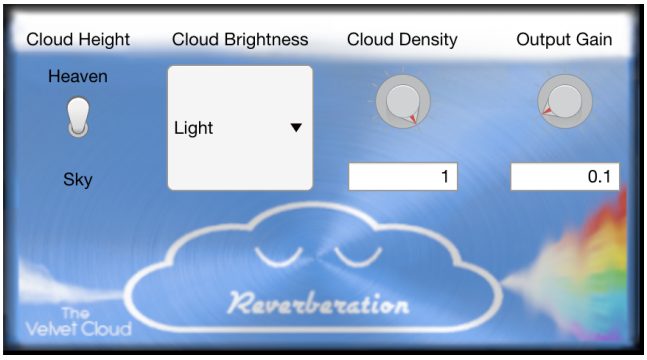


Figure 5.4: GUI of reverb effect

Full control GUI

The aforementioned GUI in figure 5.4 presents some limitations when most of the FDN parameters can be actually be under user control in real time. So, not this to limit the options to the user who wants it all, a "Full Control" version of the reverb VST for thee different orders was created (see figures A.1,A.2 in Appendix A for orders 8 and 16) having the following parameters:

- **FeedbackGain**: feedback gain of the line.
- **Delay**: samples delayed in line.
- **Modulation Speed**: how fast is modulation in line.
- **Modulation Range**: range of samples modulation per line.

Chapter 6

Evaluation

This chapter describes three different types of evaluations to assess the results of the plugin. First of all, individual evaluations (impulse response (IR), White Gaussian Noise (WGN) response and T60 measurement, with white Gaussian noise) were carried out over sustain effect and reverberation effect. For the reverberation effect, also different levels of modulation (low, medium and high) were set to see how affects the sound. Subsequently, different combinations of sustain and reverb were be arranged to measure the system response to impulse and WGN. Based on these combinations, the best plugin arrangement was chosen to make the second test where people actually evaluate how they perceive the plugin using a two-alternative forced choice test. Lastly, a third evaluation was made to evaluate how experienced users of VSTs like the plugin.

6.1 Technical Evaluation

For the technical evaluation, the following experiments were carried out:

- **Sustain Responses:** impulse response and WGN response.
- **Reverberation Responses:**

  FDN4: No modulation, medium and high modulation. For each one, IR, WGN response (in amplitude and spectrogram both) and T60 times per each).

  FDN8: same as FDN4.

  FDN16: same as FDN4.

- **Arrangement of Effects:**

  Series: Sustain + Reverberation: WGN response (in amplitude and spectrogram).

Series: Reverberation + Sustain: WGN response (in amplitude and spectrogram).

Parallel: Reverberation // Sustain: WGN response (in amplitude and spectrogram).

6.2 Two-alternative forced choice audio test Evaluation

For the comparison audio test and considering the coronavirus pandemic, an online evaluation was preferred. For that purpose, PsyToolkit [50, 49] has been used for the online TVC. With this survey platform [50, 49] one can upload audio files in a fixed audio quality, which is an important feature in tests such as this. For the test method, Two-Alternative Forced Choice (2AFC) [12] was chosen and used for measuring the subjective experience of a person of how pleasant or how she/he likes/prefers a sound in comparison with another. After some personal and musical questions [29], 2 different sound snippets (of different configurations, FDN4 No modulation and FDN8 high modulation for example) were presented (regard figure 6.1). Afterwards choosing the most pleasant, the participant is asked to motivate her/his decision or choice in few words (this is for each pair of audios). A

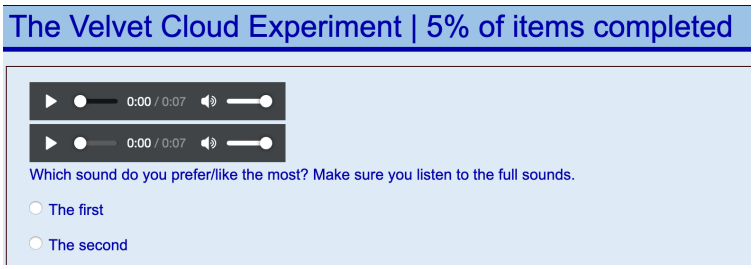


Figure 6.1: Screenshot of the Comparison audio test.

total of 36 pairs of audio snippet were to be compared by participant as it can be noticed from figure 6.2. To motivate participation, the Sustain plugin was offered

Possible sound combinations	FDN4_NOmod	FDN4_Medium	FDN4_High	FDN8_NOmod	FDN8_Medi	FDN8_High	FDN16_NOmod	FDN16_Medium	FDN16_High
FDN4_NOmod									
FDN4_Medium	x								
FDN4_High	x	x							
FDN8_NOmod	x	x	x						
FDN8_Medium	x	x	x	x					
FDN8_High	x	x	x	x	x				
FDN16_NOmod	x	x	x	x	x	x			
FDN16_Medium	x	x	x	x	x	x	x		
FDN16_High	x	x	x	x	x	x	x	x	
			Tota pairs to listen to (sum(X)):		36				

Figure 6.2: Matrix of possible sounds combinations (pairs) to listen to.

free after completing the test.

6.3 VST Evaluation

For the participants who have had previous experience with VSTs, the following test about the plugin appearance and performance was made. As stated in the Project Goals 1.5, an attractive GUI is desirable for the project. Therefore, there were some questions regarding this issue too.

The following questions among others were asked to participants using the VSTs in their preferred DAW:

Regarding appearance:

- What did you think The Velvet Cloud UI/design when you first saw it?
- Anything in design that can be improved?

Regarding preferred performance:

- Make a screenshot of your favourite plugin configuration
- What is your least favorite part about The Velvet Cloud?
- If you could add any feature(s) what would it/they be?

Further information:

- If you could add a question to this questionnaire, what should it be? What would your answer be to that question?
- Any additional comments?

The European Survey web service [56] was used to gather information for easiness (uploading screenshots from participants) and to make the questionnaire more attractive, see figure C.2 in Appendix B for a reference.

6.4 Preliminary Evaluation of code efficiency

As the reader may have noticed, the reverb implementation 5 code could be written in distinct ways so as to speed up the DSP, such as hard-coding functions, invoking loops rather than calling a function several times in multiples lines, or fitting variables into a matrix. Those techniques are very common ones to be used in general programming or coding for saving computational time. However, it turns out completely the opposite in this specific case.

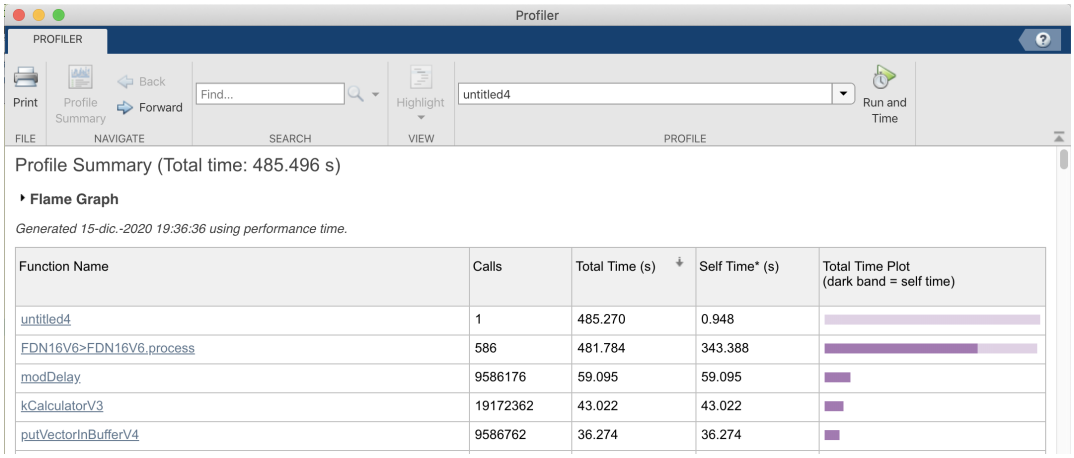


Figure 6.3: Screenshot of MATLAB Profiler tool.

Using MATLAB Profiler tool (refer to figure 6.3) and checking out which were the code lines that were consuming the most time, one finds out that these approaches were even counterproductive, increasing CPU time.

If we clicked on a specific function, it could be observed how much time a code line took to be processed, as we can regard in figure 6.4.

Notice that in figure 6.4, inside the red square (the line code that actually lasted the most time to be computed) was an access to a position of a vector. This may be a reason why MATLAB ran slower the code if we for example, merged several variables into a matrix, and we accessed specific positions of the matrix rather than each variable separately in this specific case.

Some changes (thought to save time) in code that were carried out were:

- Variables (ie: k1, k2, k3 ..., and se1...) were merged into a matrix of vectors: (k(:,1), k(:,2)....se(:,1), ...).
- FOR loops are created for computing OutB OutC.
- modDelay was invoked within a loop.
- fb1,fb2 ...fbn and g1, g2, ...gn were transformed into vectors (e.g. fb(1), fb(2) ... fb(N)).
- *copy1ofmegaBuffer* and the others buffers were merged into a huge matrix.

Surprisingly, none of them reduced the computation time. Maybe, using another audio processing tool like JUCE, which is based on C++ programming language (a lower level language) and much better suitable for real time applications could led to a computing speed improvement.

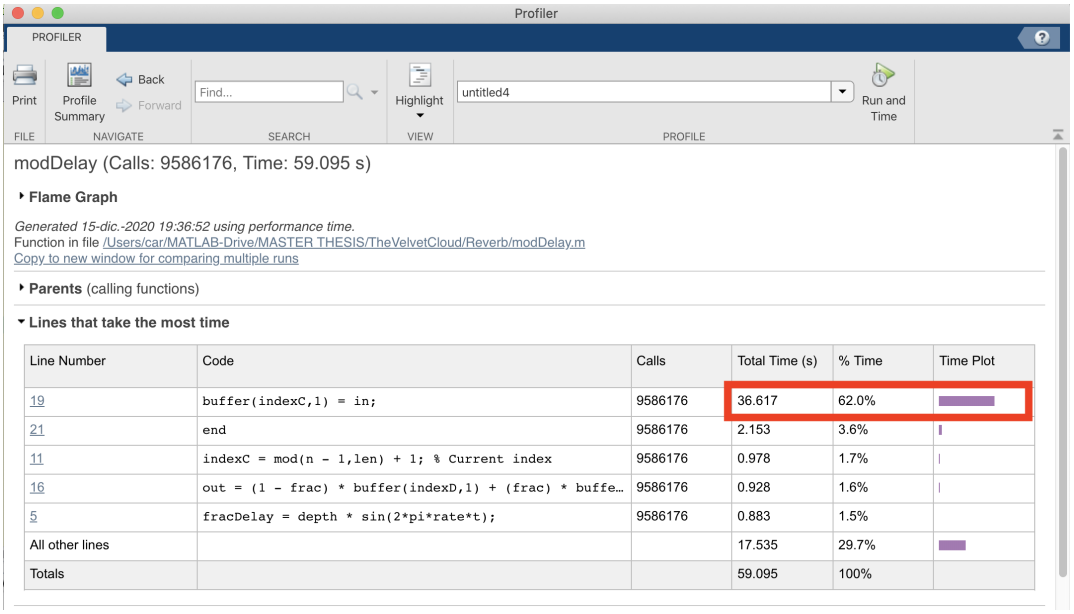


Figure 6.4: Screenshot of MATLAB Profiler within *modDelay* function. In a red square, the line that takes the most time to compute is highlighted.

Chapter 7

Results

This chapter presents the results of the three different evaluations carried out in chapter 6.

7.1 Technical Evaluation Results

7.1.1 Sustain Responses

Here, the sustain audio effect will be exposed.

Impulse response

When setting an impulse as input to the sustain effect, it let it pass without really sustaining it as we can see from figure 7.1; silence is coming as an output.

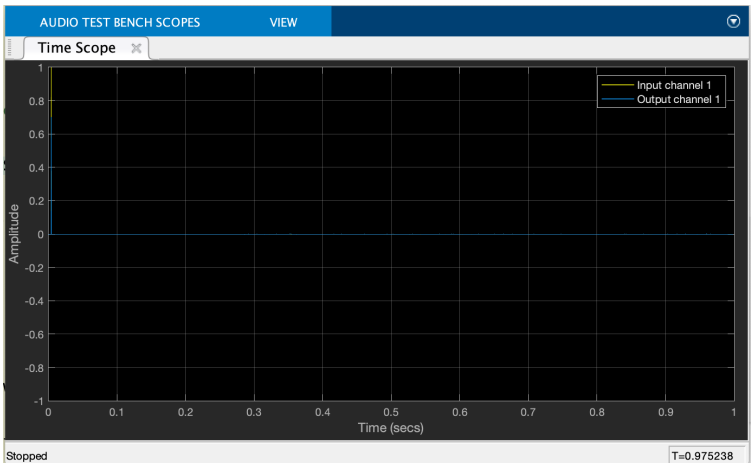


Figure 7.1: Sustain impulse response

White Gaussian Noise response

When WGN is used as an input, noise was low-passed and amplitude is decreased as we can see from figure 7.2.

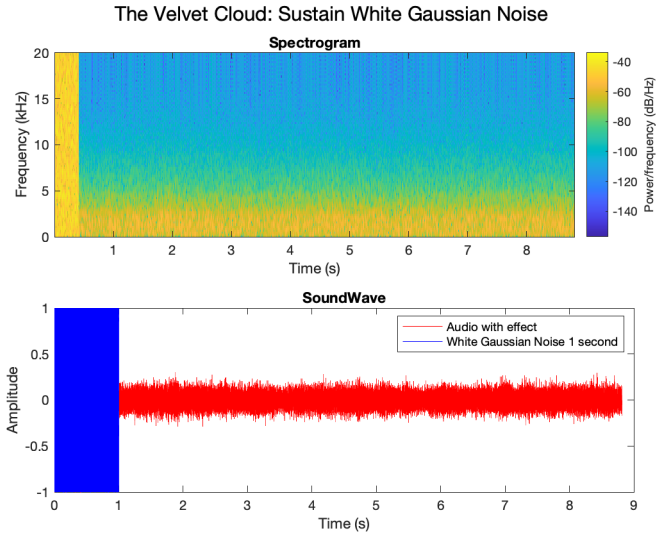


Figure 7.2: WGN sustain response

7.1.2 Reverberation Responses

Based on the different FDN chosen orders (4, 8 and 16) the reverb audio response will be analyzed. The distinguishable characteristic of this reverb implementation was the possibility of being modulated. Thereupon, the three different orders will be analyzed without modulation, and with a type of modulation. As modulation possibilities are infinite, some amplitude and rate values were chosen to define a "medium" modulation and "high" modulation. These configurations also corresponded to the effects used for the two-alternative forced choice test (2AFC) as discussed in section 6.2. In the Individual VST Evaluation in section 6.3, participants found five types of modulation (light, medium, high, ultra-high and extreme) for the reverb configuration (extreme modulation is near to convert the reverb effect into a chorus). For the sake of avoiding an overcrowded evaluation filled with all modulations graphs, only no modulation, medium and high were decided to be analyzed. Gains of the delay lines of the FDNs were tuned to have the highest possible value before the system lose its stability.

All responses are gathered in Appendix B.

7.1.3 FDN4: No modulation, medium modulation, and high modulation

The FDN order 4 presented this IR (see figure B.1), this WGN response (refer to figure B.2), and these T60 times (see figure B.3).

7.1.4 FDN8: No modulation, medium modulation, and high modulation

The FDN order 8 presented this IR (see figure B.4), this WGN response (refer to figure B.5), and these T60 times (see figure B.6).

7.1.5 FDN16: No modulation, medium modulation, and high modulation

The FDN order 16 presented this IR (see figure B.8), this WGN response (refer to figure B.9), and these T60 times (see figure B.10).

7.1.6 Arrangement of Effects

TVC, as said beforehand, is a digital audio effect comprised of sustain and reverberation effects working together. One can come up with the different arrangements of these two effects:

- Series: Sustain + Reverberation
- Series: Reverberation + Sustain
- Parallel: Reverberation // Sustain

These arrangements were also evaluated thereupon all with an FDN8 with high modulation configuration.

Series: Sustain + Reverberation

Configuring the effects in series, sustain followed by reverb showed this response (see figure 7.3).

Series: Reverberation + Sustain

Configuring the effects in series, reverb followed by sustain, the system showed this response (see figure 7.4).

Parallel: Reverberation // Sustain

In this configuration, reverb is in parallel with sustain, presenting the latter response showed in figure 7.5).

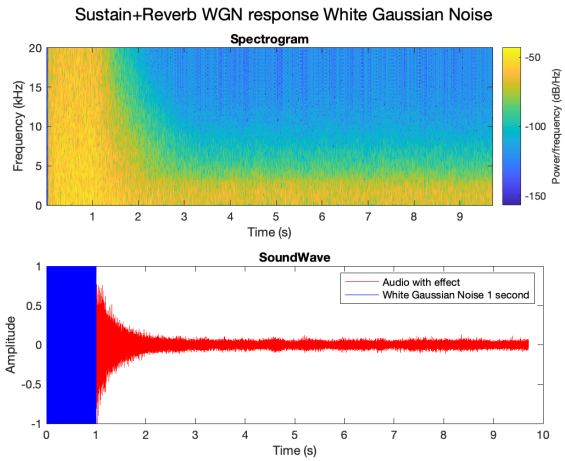


Figure 7.3: Sustain+Reverb WGN response.

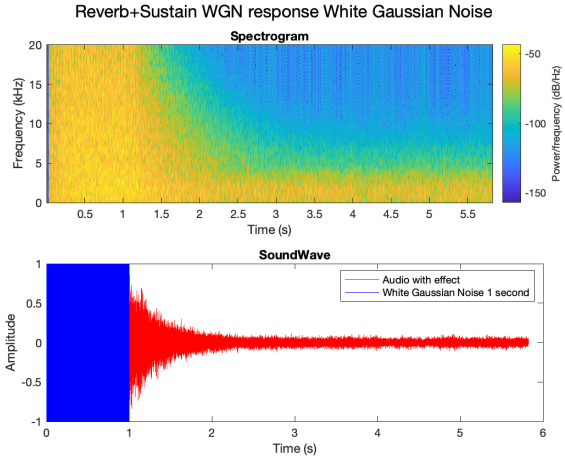


Figure 7.4: Reverb+Sustain WGN response.

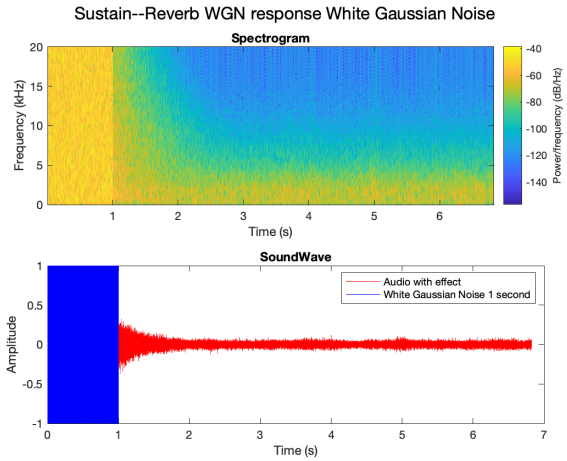


Figure 7.5: Reverb//Sustain WGN response.

7.2 Two-alternative forced choice audio test Results

In total, 20 people (7 females and 13 males) participated in the survey. Participants were recruited using social media (Facebook) and personal acquaintances of the author. This test was carried out using only a web browser, no system requirements were needed. Participants were only allowed to participate in only one test from the two presented in this project (not to bias results). Ages of participants were ranging from 23 to 42 years old, with a mean of 24.9524 year old, showing the following results (see figure 7.6. For each cell, the leftest numbers indicate how many people preferred the row sound, versus the rightest number which indicates how many people preferred the column sound. 7 participants played at least on

People Results	FDN4_NOmod	FDN4_Medium	FDN4_High	FDN8_NOmod	FDN8_Medi	FDN8_High	FDN16_NOmod	FDN16_Medium
FDN4_NOmod	X	X	X	X	X	X	X	X
FDN4_Medium	12#8	X	X	X	X	X	X	X
FDN4_High	7#13	10#10	X	X	X	X	X	X
FDN8_NOmod	10#10	11#9	9#11	X	X	X	X	X
FDN8_Medium	12#8	9#11	12#8	6#14	X	X	X	X
FDN8_High	7#13	10#10	10#10	9#11	12#8	X	X	X
FDN16_NOmod	8#12	9#11	10#10	13#7	11#9	10#10	X	X
FDN16_Medium	7#13	9#11	7#13	9#11	5#15	7#13	13#7	X
FDN16_High	6#14	5#15	11#9	7#13	7#13	6#14	6#14	8#12

Figure 7.6: Results of the 2AFC audio test.

musical instrument. Their musical experience ranged from 2 years til 5 or more years, with a mean of 4 years musical practice. Filtering the test including only people with music expertise shows these results (see figure 7.7).

From results of figure 7.6, people preferences were not really inclined by any sound type in general. These maybe ambiguous results are further discussed in

8.2.

Musical People	FDN4_NOmod	FDN4_Medium	FDN4_High	FDN8_NOmod	FDN8_Medi	FDN8_High	FDN16_NOmod	FDN16_Medium	FDN16_High
FDN4_NOmod	X	X	X	X	X	X	X	X	X
FDN4_Medium	6#1	X	X	X	X	X	X	X	X
FDN4_High	3#4	2#5	X	X	X	X	X	X	X
FDN8_NOmod	3#4	4#3	5#2	X	X	X	X	X	X
FDN8_Medium	5#2	2#5	5#2	3#4	X	X	X	X	X
FDN8_High	3#4	2#5	2#5	3#4	5#2	X	X	X	X
FDN16_NOmod	1#6	3#4	4#3	5#2	4#3	1#6	X	X	X
FDN16_Medium	2#5	3#4	3#4	5#2	1#6	6#1	6#1	X	X
FDN16_High	1#6	2#5	4#3	1#6	4#3	3#4	2#5	4#3	X

Figure 7.7: Results of the 2AFC audio test (only people with musical experience). When all but one agree on a decision, cell is highlighted.

For results reading easiness, "sub-matrices" of 7.7 will be represented in figures 7.8, 7.9 and 7.10.

	FDN4_NOmod	FDN4_Medium	FDN4_High
FDN8_NOmod	3#4	4#3	5#2
FDN8_Medium	5#2	2#5	5#2
FDN8_High	3#4	2#5	2#5

Figure 7.8: Participants preferred sound: FDN4 VS FDN8.

From figure 7.8 it can be observed that there was not an absolute preference for a FDN order or modulation (31 people have liked more order 8 in comparison of 30 of order 4), however it can be emphasized that when it comes to high modulation, a preference for a no modulated or less modulated sound is desired. Participants comments found this "high" modulation it less defined and a bit annoying. When "high" modulation was selected, a lower order is preferred.

	FDN4_NOmod	FDN4_Medium	FDN4_High
FDN16_NOmod	1#6	3#4	4#3
FDN16_Medium	2#5	3#4	3#4
FDN16_High	1#6	2#5	4#3

Figure 7.9: Participants preferred sound: FDN16 VS FDN4

From figure 7.9 it can be observed that FDN order 4 with no modulation is preferred over any FDN order 16 possibility. Comments were that order 16 sound is too wide, too reverberating and bad-defined. In this case, the preference over another modulation over "high" modulation was not so clear defined.

From figure 7.10 it can be observed that FDN16 was preferred over FDN8 with no modulation (excluding "high" order 16 modulation). Although, FDN8 "high" modulation was preferred over FDN16 with no modulation. Although when "high" modulation comes to FDN16, FDN8 with no modulation was preferred. There was not a clear preference between FDN8 with "medium" modulation and FDN16 with no modulation, however, if both were with not modulation,FDN16 is liked more.

	FDN8_NOmod	FDN8_Medi	FDN8_High
FDN16_NOmod	5#2	4#3	1#6
FDN16_Medium	5#2	1#6	6#1
FDN16_High	1#6	4#3	3#4

Figure 7.10: Participants preferred sound: FDN16 VS FDN8.

7.3 VST Evaluation Results

In total, 15 people (1 females and 14 males) participated in the survey. Participants were recruited using social media (Facebook) and personal acquaintances of the author with some already known music and DAW expertise. All participants in this test required to have some experience working with DAWs and VSTs. 12 participants used MacOS and 3 Windows. 10 participants used Ableton Live, 2 for Reason, 1 for Reaper, 1 for VCV Rack, and 1 Logic Pro X and as their test DAW. Participants have been using DAW ranging from 1 year till 20, with a mean of 9.6333 years. Answers were received and studied using the EUSurvey platform (refer to figure C.1 in Appendix B).

7.3.1 Regarding installation

Nearly any participants did not have much trouble with installation: "Fairly easy. Drag, drop and open is as simple as it gets.". 2 Mac users ended up using the Audio Unit version of the plugin if VST did not work. 2 Windows users were having problem with DAW recognizing the VST, which was solved trying it in another different DAW (Reaper). Some users needed a 32 bits windows version, which was not available and could not perform the survey

7.3.2 Regarding GUI

Most participants agreed that "It looks nice. Limited options of control makes it look relatively easy to go to. ", "It is cute and unique", "I liked the simple design and the reduced amount of controls, I really liked the fact that it looks like a real physical pedal.". Some more picky, complained about parameters not having technical names, in contrast, they " slowly realized what each knob was doing while playing around" or " It's not obvious what some parameters are actually doing at first sight, but it made me also interested in trying the plugin." Suggestions about improving the GUI were about "reorder the knobs to reflect the signal path (e.g. dry/wet should be last)", "Decide to use either sliders or knobs", "would be cool with some "special" knobs", "It would be nice to have both effect integrated within the same GUI".

7.3.3 Respecting performance

Participants stated mostly a positive feedback, with statements such as "didn't ever "explode" or go into unstable states despite trying really hard. My ears are safe!!", "How easy it was to get a pretty deep reverb effect and a sustaining droning sound. While playing on a clean electric guitar, I definitely got some Brian Eno-type sounds.", "Reverberation to seem so psychedelic (in a good way).", "The FDN sounds super nice!", " Maybe the resolution of the slider was too small for sounds low in volume. ", "It was nice to first experience the plugin with a really wet and reverberated default setting.", "you can play a chord in order to let it play to infinity and then play melodies over it".

7.3.4 Pros and Cons

As bad performance, participants stated that "convert my input signal to mono", "had some clicking ", "Mistakes can happen easily while recording with such an effect, so it would be a bit annoying in this case. ", "threshold ready was a difficult parameter to understand". As in contrast, other users declared: "sustain is very smooth and has nice micro-variations when you just let it ring", "The FDN. It sounds really nice!! ", "modulation behavior adds a nice touch ", "more interesting results compared to usual reverbs", "design", ""extreme" preset of the reverb", "The option to dry/wet mix the signal with the sustained signal is nice! Sustain Threshold is useful, and heaven/sky switch is a quick and easy switch between a more or less "wet".

7.3.5 Regarding enhancements/suggestions

Participants affirmed some features they would like to find in a plugin like this, such as "ability to modulate the tail of the sustain, and a built-in side-chain", "visual feedback for setting the threshold of the sustain", "Automation/kill switch over when the sustainer sustains.", "Blushing cheeks animation on the cloud, the cloud could be animated depending on the parameters or analysis of the plugin output", "some visual cues on the sustain plug", "A reset / panic button. ", "reversed reverb" option", "possibility to choose the waveforms of the modulation in FDN reverb".

7.3.6 Regarding Full Control version

"Everything clear regarding the parameters and the control of parameters was very intuitive.", "I found it difficult to get good reverb results, but on the other hand I

liked the weird sounds that it can produce.", "It's definitely crazy, but in a fun way. I made my MIDI track sound sort of hellish.", "too big GUI size".

7.3.7 About price

People thought (as mean) TVC is worth price of 40 dollars.

7.3.8 Regarding preferred configuration

8 people sent his/her preferred configuration with the results figure 7.11 shows. Of

REVERB	NO_mod	Light	Medium	High	UltraHigh	Extreme	Heaven	Sky	
	0	0	2	3	2	1	4	4	
									MEANS
Wet	1	0.71	0.63	0.39	0.61	0.58	0.66	0.75	0.6663
Output	0.44	0.14	0.12	0.93	0.45	0.1	0.39	0.28	0.3563
SUSTAIN									
Threshold	0.16	0.38	0.28	0	0.24	0.38	0.65	0.63	0.34
ReadyThreshold	0.21	0.25	0.33	0.89	0.19	0.49	0.33	0.67	0.42
Wet	0.3	0.69	0.17	0.72	0.56	0.27	0.68	0.84	0.5288
Output	0.2	0.26	0.75	0.04	0.25	1	0.21	0.26	0.3712
VelvetTouch	630	767	736	329	900	184	400	135	510
Arrangement of the system									
Sustain + Reverb	4								
Reverb + Sustain	1								
Reverb//Sustain	0								
	TOTAL = 5								

Figure 7.11: Participants preferred configuration of TVC.

this 8 people, only 5 sent a screenshot where the signal flow could be noticed and verify if the arrangement of the system they chose was Reverb + Sustain, vice-versa or parallel.

7.3.9 Extras

Although not asked in the survey, some participants like TVC so much that sent the author music files using the effect. These can be found in the GitHub of the project [25].

Chapter 8

Discussion

8.1 Technical Evaluation

As we can observe from section 7.1, there were many things to consider from the several obtained graphs which will be considered separately:

8.1.1 Regarding Sustain Effect

The outputted silence when using an impulse (refer to figure 7.1), made sense as sustain algorithm (when firstly initialized) takes whatever audio input comes as the audio snippet to be sustained. From sustain implementation in chapter 4, we can see that in order to create the illusion of the audio being sustained, of an audio snippet, some samples (determined by velvet noise) of it are summed each loop. If audio snippet is nearly fully comprised of 0s', and just an extremely few number of joint samples are 1's and afterwards is low-passed, then the expected output is composed also by 0's, and therefore, silence.

Also, regarding the WGN response, this behaviour (low pass and amplitude descent) was the expected one from the implementation described in chapter 4. Noise was sustained smoothly over time.

8.1.2 Regarding Reverb Effects

The following figures referred in this section 8.1.2 are to be found in Appendix C.

FDN4

As reader can notice from figure B.3, not great T60 times were achieved by this order. The use of modulation tends to disperse the energy of the sound wave making the amplitude of the reverberated tail smaller (see figures B.1 and B.2).

Also, T60 times were relative attenuated when higher modulated. Probably, due to energy being dispersed quicker than in medium modulation. From the hearing experience of the noise, little more can be said. However, modulation caused some more pleasant sensations in the hearing as will be showed in the 2AFC test section 6.2.

FDN8

As we can see from figure B.6, T60 times have increased and modulation again tended to disseminate the energy of the reverberation in the low frequencies. Some harmonic appeared in the reverberation tail as we can see from figures B.4 and B.5 when no modulation was used. Similarly to the behaviour of FDN4, the use of modulation contributed to the scattering of the frequencies energy and the highest it was, the faster this decay was.

As a side note, as commented previously, the FDNs were tuned to have the highest possible value before the system losses its stability. If these gains were a higher value, the following behaviour would arise; disturbing harmonics will appear in the spectrograms continuously in time. An interesting result, was that, if modulation was used, this disturbing harmonics disappeared (gets randomized) and higher T60 times could be achieved as we can see form figure B.7.

FDN16

From figure B.10 we can see that T60 have increased, but not significantly. The harmonic behaviour was even more diffused in this higher order as we can see from figures B.8 and B.9. The scattering of the frequencies in the reverberation tail was more evenly distributed and not only present mostly in low frequencies as FDN4 or FDN8 showed in figures B.2 and B.5. Also, the "turquoise columns" in the spectra of B.5 (and the ones that also appear vaguely in B.2 at the early moment where WGN stops) were not visible anymore.

A low frequency tail appeared in the spectrogram B.9 in contrast to the other FDN orders. This is possibly thought to be happening due to the chosen output gains, which were not far from causing instability in the system.

8.1.3 Regarding different Arrangements

Reverb plus Sustain response in figure 7.4 was pretty similar to Sustain plus Reverb in figure 7.3 although not equal, as none of them constitute a linear time invariant system. Although, from subjective listening, Sustain plus Reverb in figure 7.1.6 was slightly more pleasant to the ear when evaluating a guitar jamming rather than noise.

As for Reverb in parallel with Sustain, as we can see from figure 7.5 and the

reader can notice, the sudden "stop" of the noise is not desirable nor pleasant at all. This was due to the channel where only sustain was ON; when the noise is stopped, the sustained noise was kept, but there was not a "smooth" transition due to the fact that the audio effect chain in that track was not ended by a reverb, compressor, or any other audio effect of dynamic volume control.

Taking all into consideration, best selected sounding arrangement was: Series: Sustain + Reverberation 7.1.6. This was the reasoning for choosing this configuration for the two-alternative forced choice audio test in evaluation in section 6.2 to analyze its subjective performance.

8.2 Two alternative forced choice audio test

The question asked in this test after listening to 2 audio files was: "*Which sound do you prefer/like the most?*", as we can confirm also from figure 6.1.

From comments received from participants in the survey, their preferences were explained. There were comments of type: "*I prefer this because is brighter, reverberating and wider*", versus comments of type: "*The other sound is really bad-defined and too blurry*", see figure 8.1 as an example of this participant preferences.

my_question3:1	my_question3_2:1
1	**More amplitude in sound**
2	1st one was too airy, might cause some problems later in the mix
2	Too much sustain on first
2	Better defined
2	More quiet
1	**It was dirtier**
1	**Brighter, more space**

Figure 8.1: Participants liking a defined sound and participants prefering a more dirty/reverberated sound (in bold).

From this, the following conclusion could be extracted:

Some participants preferred/liked more a clear, well-defined sound, in contrast with participants preferring a wider, more dirty and reverberated sound.

As there were not a clear statement about which criteria to follow and an open/-more subjective question was made, results were maybe biased and not confirming clearly which sound sounds with more reverb, but rather which was more pleasant. It was a subjective test, and people may have different opinions in their ratings of how a sound should sound to be a better sound.

Moreover, audio quality files in .mp3 format (due to some platform issues .wav were not used) may have made difficult participants evaluation apart from some of them using really cheap no-brand earphones. This may have led to some inconclusive results from the 6.2 test for general people, as we can see from figure 7.6.

When selecting only people with music experience, results became clearer as we

can see from figure 7.7. Maybe music people criteria is more unified in comparison with general people.

Hence, from the results we can conclude that FDN4 was preferred over any type of FDN16; participants declared that FDN16 was"too" wide, while FDN4 was more well-defined. It was also remarkable that FDN8 "medium" modulation was clearly preferred over FDN16 "medium" modulation. FDN order 16 was creating a "too" wide reverb which participants were not finding too pleasant. Although, FDN16 with any modulation type was preferred over FDN order 8 without modulation, which could lead to concluding that modulation makes reverb more pleasant. Conversely, this did not hold comparing FDN16 to FDN4, however, participant reasoning changed; FDN16 seemed to be a "too crowded" reverb and more definition was desired even for a reverb, and therefore, the preference for choosing FDN4. This participants statement was supported as when "high" modulation came to FDN16, FDN8 even with no modulation was preferred.

This suggest that FDN16 could be substituted by a lower order and people will still find it pleasant and not much different.

If we compare FDN4 versus FDN8, in general, a modulated sound was preferred over no modulation at all, which could indicate that modulation adds a more "pleasant" sound to reverberation. This did not clearly holds when comparing with FDN16, but as mentioned beforehand, FDN16 was less liked in general compared to other orders due to its "too much" reverberation.

8.3 VST Evaluation

From the extracted results of the survey (refer to Appendix C.1), several points could be noticed:

- Installation was mostly easy to perform
- GUI looked nice, simple, suggestive and unique, although not really technical.
- Easy to get a deep/psychedelic reverb and sustain droning
- Brian Eno-type sounds, usable in ambient music
- Some controls could be prettified. An unified version and stereo version was desirable
- Modulation added a nice touch, more interesting than usual reverb
- It was crazy, and it was fun

Also, from results showed by figure 7.11, it can be agreed that modulation is preferred over no modulation and chosen by everyone, which confirms the comments participants stated about modulation. "Heaven" and "Sky" modes were equally preferred and low values of *SustainThreshold* were chosen. Little higher values (but close to *SustainThreshold* value) of *ThresholdReady* were elected, which may mean this parameter was not much clear enough by its only name, or people liked it to be near *SustainThreshold*, which can may also make sense depending on instrument you are using. No data of the instrument participants were testing TVC on was recorded, which might be a good parameter to take into account in the future.

The outcomes of this questionnaire did really much match with what stated in the Project Goals section 1.5: An attractive, simple GUI, achieving a deep-/psychedelic and really wet reverb, which recreates an unreal spaces and over which you can play melodies thanks to the infinity automatic sustain.

Aside from the good results from this evaluation, several enhancements and suggestions were acknowledged by participants such as making "stereo" the effect, putting sustain killer or trigger by pressing a button, and possibilities of reversing the reverb or modifying the wave-forms of the modulation.

In reference to the GUI, some missing characteristics were found to be quite relevant. Such as placing some visual cues when sustain is actually sustaining, prettifying controls, integrating both effects in just one GUI, using only knobs or sliders and augmenting the definition of the background, will help to enhance the visual richness of the plugin and make it even more attractive.

Chapter 9

Conclusion and Future Work

In this master thesis, a real-time plugin of sustain and reverberation was presented. After reviewing the state of the art in sustain and reverb audio effects and investigating the different plugins the market offer for sustain and reverb effect, a technique (as a starting point) was developed for each effect: Convolution with velvet noise for sustain and Velvet Feedback Delay Networks for reverberation (that is why both effects as one system is called "The Velvet Cloud"). As for reverberation, a new modality not present in state of the art was proposed in this project: Modulated Feedback Delay Networks. The aforementioned has been implemented as two different VSTs which were found to have a deep, psychedelic and more interesting sound than usual reverb with an unique and suggestive GUI. A technical evaluation was performed over the VST possibilities as well as a Two Alternative Forced Choice test to evaluate how its configuration affects sound subjectively. All in all, project goals presented in section 1.5 can be regarded as completed.

A more profound analysis of how modulation configured the output of the system and how it influences people rating reverberated sound is left for future work. Additionally, further possibilities of modulation are to be explored, such as how different modulation wave-forms affect reverb in a technical and a subjective way. Also, many suggestions by participants were to be implemented although not feasible due to lack of time, such as making an all-in-one VST, stereo output, solving the "crackling" when changing VST parameters, a button to trigger sustain, and visual animations of the GUI. Besides, although working properly, searching new methods for saving computational speed in different ways could be an enhancement to the plugin which may allow it to be feasible for a purchase and a release to the market. An implementation of the VST in JUCE could be a step towards it.

Bibliography

[1] "A new approach to digital reverberation using closed waveguide networks". In: *Proc. Int. Computer Music Conf, Vancouver, 1985*. 1985, pp. 47–53.

[2] Adil Alpkocak and Malik Kemal Sis. "Computing impulse response of room acoustics using the ray-tracing method in time domain". In: *Archives of Acoustics* 35.4 (2010), pp. 505–519.

[3] AudioThing. *Fog Convolver - Convolution Reverb VST/AU/AAX Plugin - AudioThing*. https://www.audiothing.net/effects/fog-convolver/. 2020. (accessed: December 2020).

[4] Sean Costello. *The Philosophy of ValhallaSupermassive - Valhalla DSP*. https://valhalladsp.com/2020/05/06/the-philosophy-of-valhallasupermassive/. 2020. (accessed: December 2020).

[5] Sean Costello. *Valhalla Super Massive - Valhalla DSP*. https://valhalladsp.com/shop/reverb/valhalla-supermassive/. 2020. (accessed: December 2020).

[6] Audio Damage. *AD034 Eos 2*. https://www.audiodamage.com/products/ad034-eos-2. (accessed: December 2020).

[7] Stefano D'Angelo. *Atlante pedal samples*. https://soundcloud.com/user-884388289-540474067/sets/atlante-pedal-samples. 2020. (accessed: December 2020).

[8] Stefano D'Angelo and Leonardo Gabrielli. "Efficient signal extrapolation by granulation and convolution with velvet noise". In: *Proc. 21st Int. Conf. Digital Audio Effects (DAFx-18), Aveiro, Portugal*. 2018, pp. 107–112.

[9] Electro-Harmonix. *EHX.com | Oceans 12 - Dual Stereo Reverb | Electro-Harmonix*. https://www.ehx.com/products/oceans-12. (accessed: December 2020).

[10] Jon Fagerström, Benoit Alary, Sebastian J. Schlecht, and Vesa Välimäki. *Multi Track Audio Player*. http://research.spa.aalto.fi/publications/papers/dafx20-vfdn/. 2020. (accessed: December 2020).

[11] Jon Fagerström, Benoit Alary, Sebastian J. Schlecht, and Vesa Välimäki. "Velvet-Noise Feedback Delay Network". In: *International Conference on Digital Audio Effects*. 2020.

[12] Christopher D Green. "Introduction to Elemente der Psychophysik Gustav Theodor Fechner". In: *Classics in Psychology, 1855-1914: Historical Essays*. 1860.

[13] Minimal System Group. *Dreamscape, Dreamscape plugin, buy Dreamscape, download Dreamscape*. `https://www.pluginboutique.com/products/589-Dreamscape`. (accessed: December 2020).

[14] Bo Holm-Rasmussena, Heidi-Maria Lehtonenb, and Vesa Välimäkib. "A new reverberator based on variable sparsity convolution". In: *Proc. of the 16th Int. Conference on Digital Audio Effects (DAFx-13)*. Vol. 5. 6. 2013, pp. 7–8.

[15] HooRNet. *HoRNet HCS1, vintage guitar compressor sustainer pedal plugin*. `https://www.hornetplugins.com/plugins/hornet-hcs1/`. 2019. (accessed: December 2020).

[16] iZotope. *Neoverb Features*. `https://www.izotope.com/en/products/neoverb/features.html`. 2020. (accessed: December 2020).

[17] Hanna Järveläinen and Matti Karjalainen. "Reverberation modeling using velvet noise". In: *Audio Engineering Society Conference: 30th International Conference: Intelligent Audio Environments*. Audio Engineering Society. 2007.

[18] Ronald H Jones Jr and Bruce D Jobse. *Real-time digital audio reverberation system*. US Patent 5,530,762. 1996.

[19] Jean-Marc Jot. "Efficient models for reverberation and distance rendering in computer music and virtual audio reality". In: *ICMC*. 1997.

[20] Jean-Marc Jot and Antoine Chaigne. "Digital delay networks for designing artificial reverberators". In: *Audio Engineering Society Convention 90*. Audio Engineering Society. 1991.

[21] JOYOaudio. *JGE-01 Infinite sustain JOYO*. `https://www.joyoaudio.com/product/196.html`. (accessed: December 2020).

[22] Marc Lingk. *TimeFreezer - Audio and Music Instruments*. `http://www.timefreezer.net/`. (accessed: December 2020).

[23] Marc Lingk. *TimeFreezer - Audio Plugin - VST AU | TimeFreezer*. `https://timefreezer.bandcamp.com/releases`. 2019. (accessed: December 2020).

[24] Marc Lingk. *TimeFreezer Instrument - wwwtimefreezer*. `https://www.youtube.com/watch?v=lAZqNe-lhq8`. 2009. (accessed: December 2020).

[25] Carmen Muñoz Lázaro. *TheVelvetCloud: Sustain Reverb for reaching the velvet clouds*. `https://github.com/chachipirulin/TheVelvetCloud`. 2020. (accessed: December 2020).

[26] David S McGrath. "Huron-a digital audio convolution workstation". In: *Audio Engineering Society Convention 5r*. Audio Engineering Society. 1995.

[27] Fritz Menzer and Christof Faller. "Unitary matrix design for diffuse jot reverberators". In: *Audio Engineering Society Convention 128*. Audio Engineering Society. 2010.

[28] James A Moorer. "About this reverberation business". In: *Computer music journal* (1979), pp. 13–28.

[29] Daniel Müllensiefen, Bruno Gingras, Lauren Stewart, and Jason Jií Musil. "Goldsmiths Musical Sophistication Index (Gold-MSI) v1. 0: Technical Report and Documentation Revision 0.3". In: *London: Goldsmiths, University of London.* (2013).

[30] Koka Nikoladze. *Koka's Rotary Magnetic Bow on Vimeo*. `https://vimeo.com/160780036`. 2016. (accessed: December 2020).

[31] Jyri Pakarinen, Vesa Välimäki, Federico Fontana, Victor Lazzarini, and Jonathan S Abel. "Recent advances in real-time musical effects, synthesis, and virtual analog models". In: *EURASIP Journal on Advances in Signal Processing* 2011 (2011), pp. 1–15.

[32] João Paulo Caetano Pereira, Gilberto Bernardes, and Rui Penha. "Musikverb: A harmonically adaptive audio reverberation". In: *Proc. of the 21st Int. Conference on Digital Audio Effects (DAFx-18), Aveiro, Portugal*. 2018.

[33] Joan Prat Rigol and Mihailo Kolundzija. "Virtual sustain pedalling for guitar by sinusoidal modeling". In: Universitat Politècnica de Catalunya, 2012.

[34] Karolina Prawda, Vesa Välimäki, and Sebastian J Schlecht. "Improved reverberation time control for Feedback Delay Networks". In: *Proc. 22th Int. Conf. Digital Audio Effects (DAFx-19)*. 2019.

[35] Karolina Prawda, Silvin Willemsen, Stefania Serafin, and Vesa Välimäki. "Flexible Real-Time Reverberation Synthesis with Accurate Parameter Control". In: *23rd International Conference on Digital Audio Effects*. 2020.

[36] Mark Rau and Orchisama Das. "An "Infinite" Sustain Effect Designed for Live Guitar Performance". In: *Audio Engineering Society Convention 143*. Audio Engineering Society. 2017.

[37] Loïc Reboursière, Christian Frisson, Otso Lähdeoja, John A Mills, Cécile Picard-Limpens, and Todor Todoroff. "Multimodal Guitar: A Toolbox For Augmented Guitar Performances." In: *NIME*. 2010, pp. 415–418.

[38] Andrew Reilly and David McGrath. "Convolution processing for realistic reverberation". In: *Audio Engineering Society Convention 98*. Audio Engineering Society. 1995.

[39] Davide Rocchesso. “Maximally diffusive yet efficient feedback delay networks for artificial reverberation”. In: *IEEE Signal Processing Letters* 4.9 (1997), pp. 252–255.

[40] Davide Rocchesso and Julius O Smith. “Circulant and elliptic feedback delay networks for artificial reverberation”. In: *IEEE Transactions on Speech and Audio Processing* 5.1 (1997), pp. 51–63.

[41] Sebastian J Schlecht and Emanuël AP Habets. “Feedback delay networks: Echo density and mixing time”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.2 (2016), pp. 374–383.

[42] Sebastian J Schlecht and Emanuël AP Habets. “Modal decomposition of feedback delay networks”. In: *IEEE Transactions on Signal Processing* 67.20 (2019), pp. 5340–5351.

[43] Sebastian J Schlecht and Emanuël AP Habets. “Time-varying feedback matrices in feedback delay networks and their application in artificial reverberation”. In: *The Journal of the Acoustical Society of America* 138.3 (2015), pp. 1389–1398.

[44] Manfred R Schroeder and Benjamin F Logan. “"Colorless" artificial reverberation”. In: *IRE Transactions on Audio* 6 (1961), pp. 209–214.

[45] Julius Orion Smith. *Physical audio signal processing: For virtual musical instruments and audio effects*. W3K publishing, 2010.

[46] Julius Orion Smith. “Time Varying Reverberators”. In: *Physical Audio Signal Processing*. online book, 2010 edition. `http://ccrma.stanford.edu/~jos/pasp/`, pp. 47–53. (accessed: December 2020).

[47] Jan-Jakob Sonke and Diemer de Vries. “Generation of diffuse reverberation by plane wave synthesis”. In: *Audio Engineering Society Convention 102*. Audio Engineering Society. 1997.

[48] John Stautner and Miller Puckette. “Designing multi-channel reverberators”. In: *Computer Music Journal* 6.1 (1982), pp. 52–65.

[49] Gijsbert Stoet. “PsyToolkit: A novel web-based method for running online questionnaires and reaction-time experiments”. In: *Teaching of Psychology* 44.1 (2017), pp. 24–31.

[50] Gijsbert Stoet. “PsyToolkit: A software package for programming psychological experiments using Linux”. In: *Behavior research methods* 42.4 (2010), pp. 1096–1104.

[51] Strymon. *BigSky - Multidimensional Reverberator - Reverb Pedal - Strymon*. `https://www.strymon.net/product/bigsky/`. 2020. (accessed: December 2020).

[52] Strymon. *Strymon BigSky - Cloud Reverb machine audio demo - YouTube*. `https://www.youtube.com/watch?v=wvH8Or_rkFY`. (accessed: December 2020).

[53] Eric Tarr. *Hack Audio is creating tutorial videos on computer programming and audio engineering | Patreon*. https://www.patreon.com/hackaudio?fbclid=IwAR0Lp-XZlMDWbrSwnQhR-EExootMMaUN3kGsczljcNjVpK7Otg00cJmKHMg. 2020. (accessed: December 2020).

[54] *The Amazing EBow :: Home*. https://ebow.com/. 2020. (accessed: December 2020).

[55] Anders Torger and Angelo Farina. "Real-time partitioned convolution for Ambiophonics surround sound". In: *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No. 01TH8575)*. IEEE. 2001, pp. 195–198.

[56] European Union. *EUSurvey - Welcome*. https://ec.europa.eu/eusurvey/. 2020.

[57] Vesa Välimäki, Bo Holm-Rasmussen, Benoit Alary, and Heidi-Maria Lehtonen. "Late reverberation synthesis using filtered velvet noise". In: *Applied Sciences* 7.5 (2017), p. 483.

[58] Vesa Välimäki, Heidi-Maria Lehtonen, and Marko Takanen. "A perceptual study on velvet noise and its variants at different pulse densities". In: *IEEE transactions on audio, speech, and language processing* 21.7 (2013), pp. 1481–1488.

[59] Vesa Valimaki, Julian D Parker, Lauri Savioja, Julius O Smith, and Jonathan S Abel. "Fifty years of artificial reverberation". In: *IEEE Transactions on Audio, Speech, and Language Processing* 20.5 (2012), pp. 1421–1448.

[60] Daniel Werner. *ExperimentalScene > Software > ES SpatialVerb 5.7.2*. https://www.experimentalscene.com/software/spatialverb/. 2020. (accessed: December 2020).

[61] Silvin Willemsen, Stefania Serafin, and Jesper R Jensen. "Virtual analog simulation and extensions of plate reverberation". In: *14th Sound and Music Computing Conference*. 2017.

[62] The XX. *The XX Intro HQ - YouTube*. https://www.youtube.com/watch?v=lFGnsdV-sR4. 2017. (accessed: December 2020).

[63] Udo Zölzer. *DAFX: digital audio effects*. John Wiley & Sons, 2011.

Appendix A

FullControlFigures

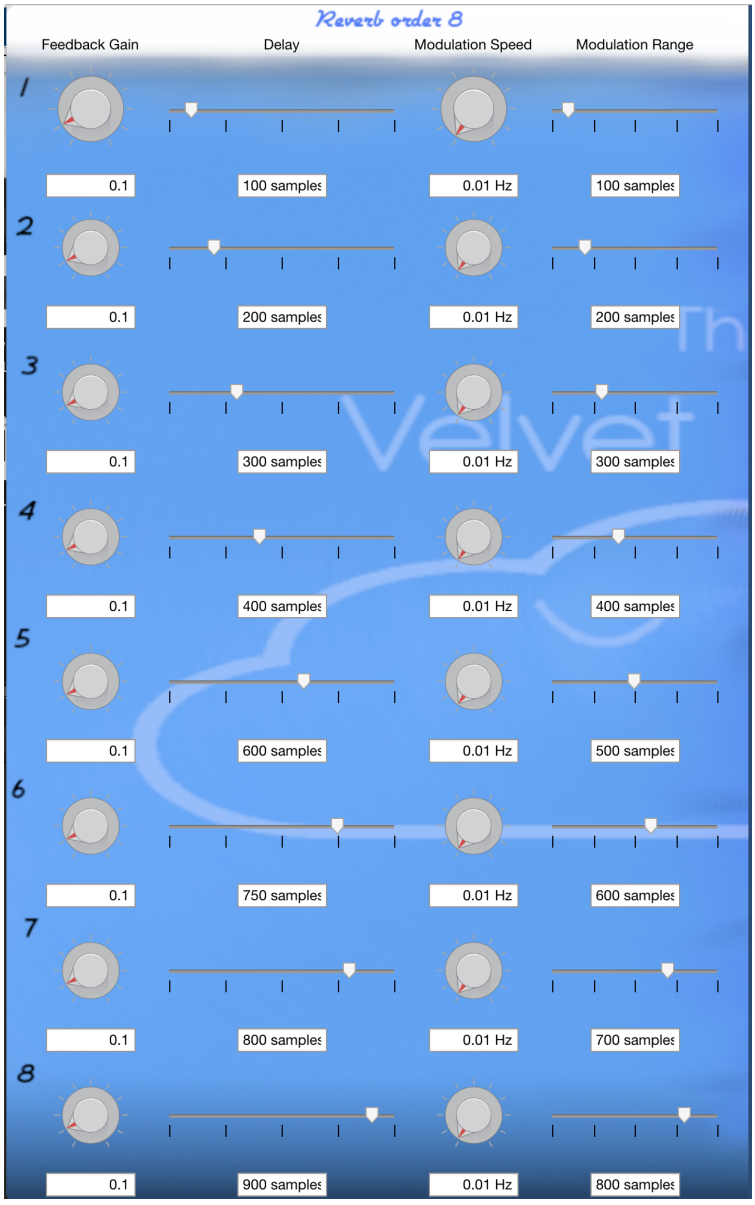


Figure A.1: GUI of reverb order 8 of Full Control version.

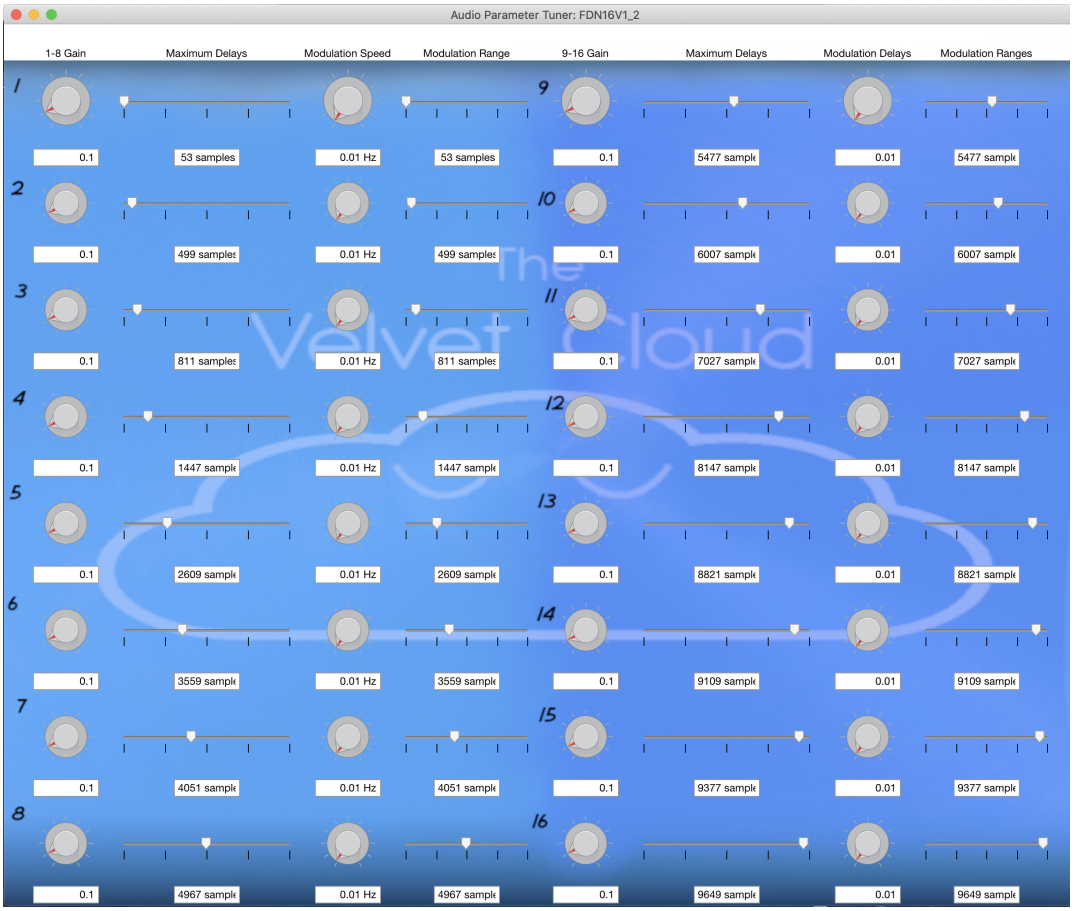


Figure A.2: GUI of reverb order 16 of Full Control version.

Appendix B

Technical Evaluation Graphs/Figures

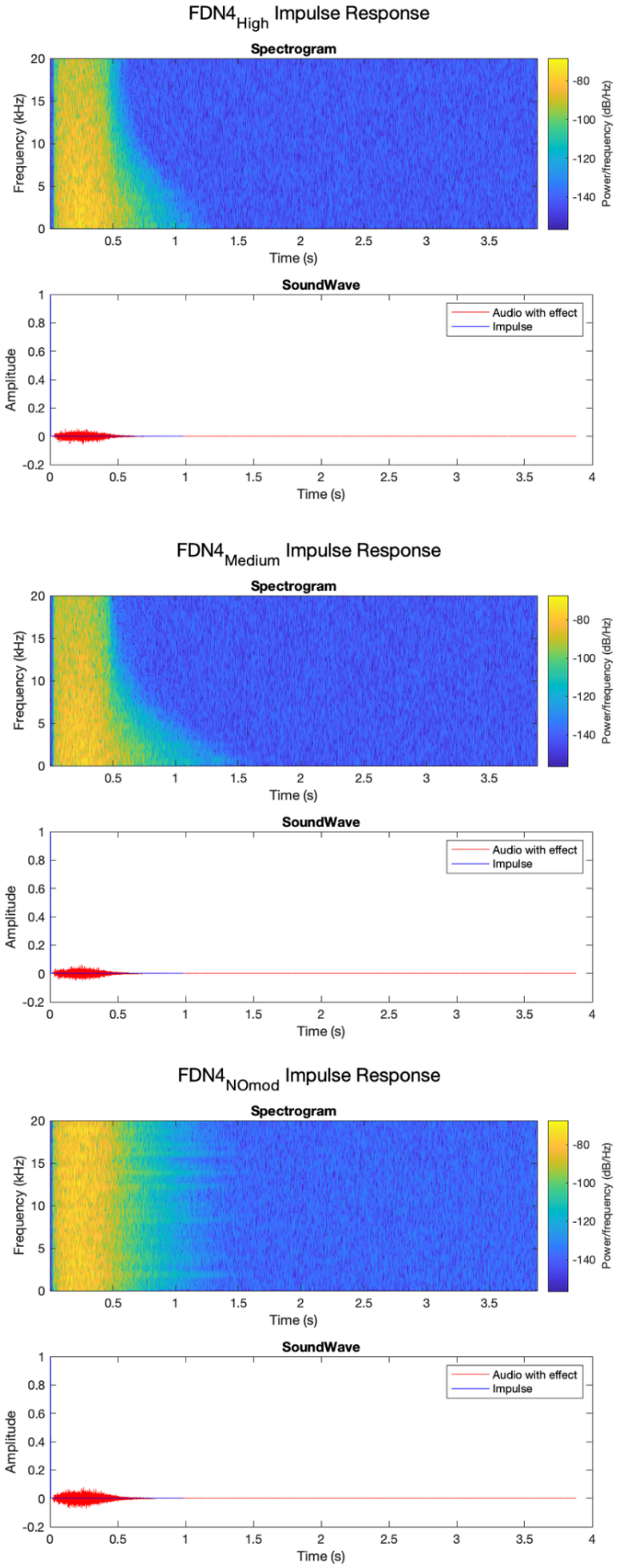


Figure B.1: FDN4 Impulse Response, for High modulation, Medium modulation and NO modulation.

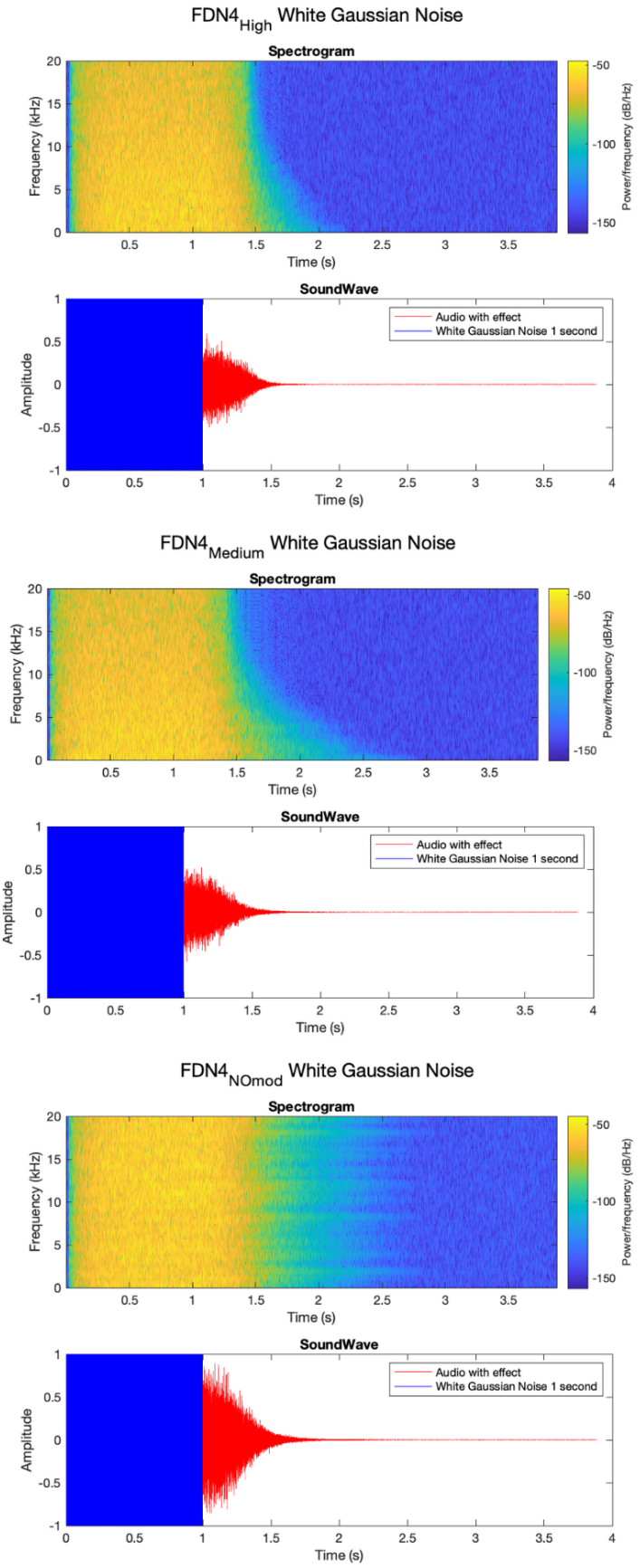


Figure B.2: FDN4 WGN response, for High modulation, Medium modulation and NO modulation.

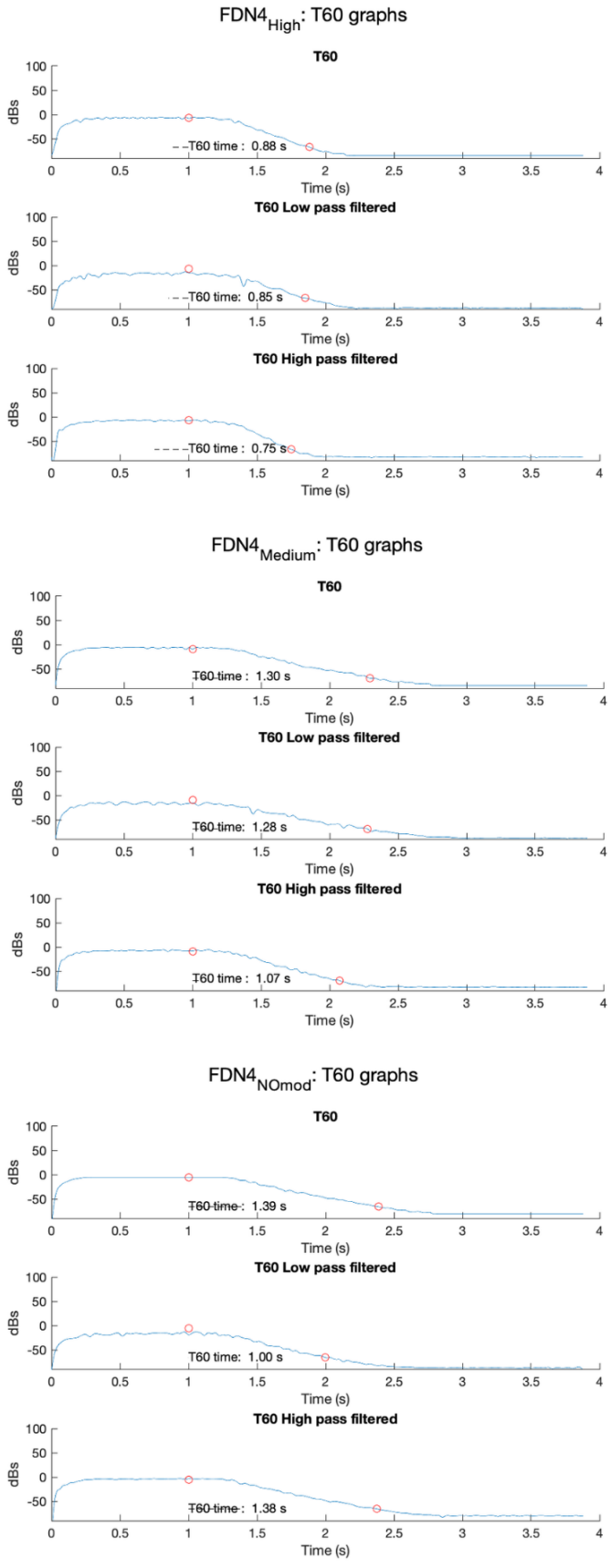


Figure B.3: FDN4 T60 times for High modulation, Medium modulation and NO modulation.

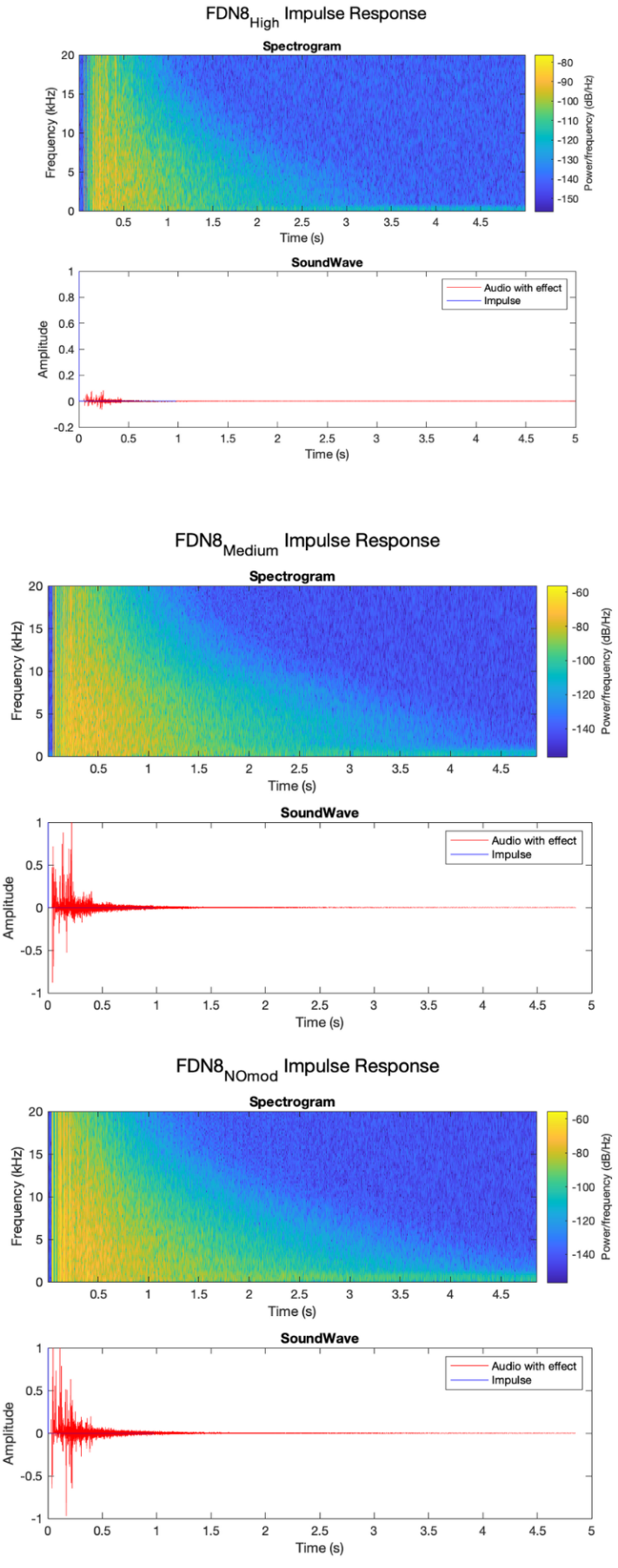


Figure B.4: FDN8 Impulse Response, for High modulation, Medium modulation and NO modulation.

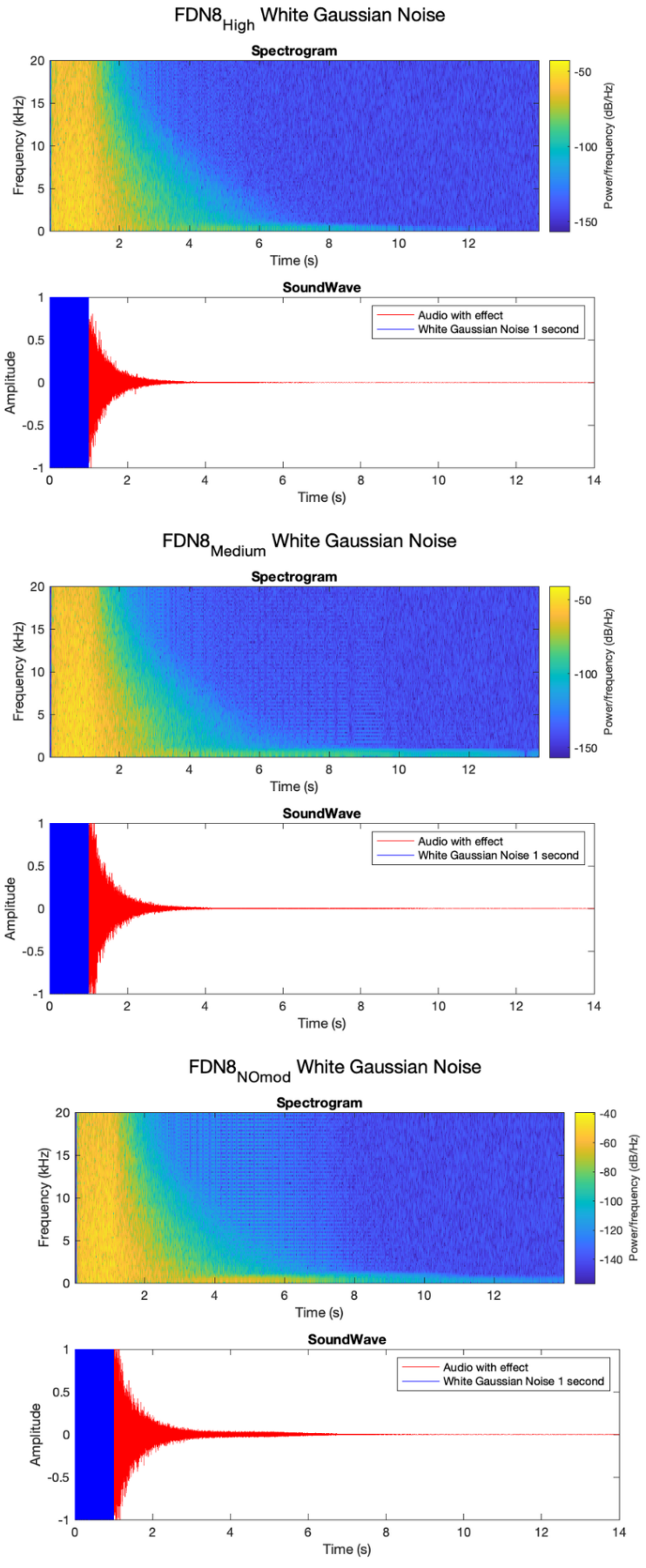


Figure B.5: FDN8 WGN response, for High modulation, Medium modulation and NO modulation.

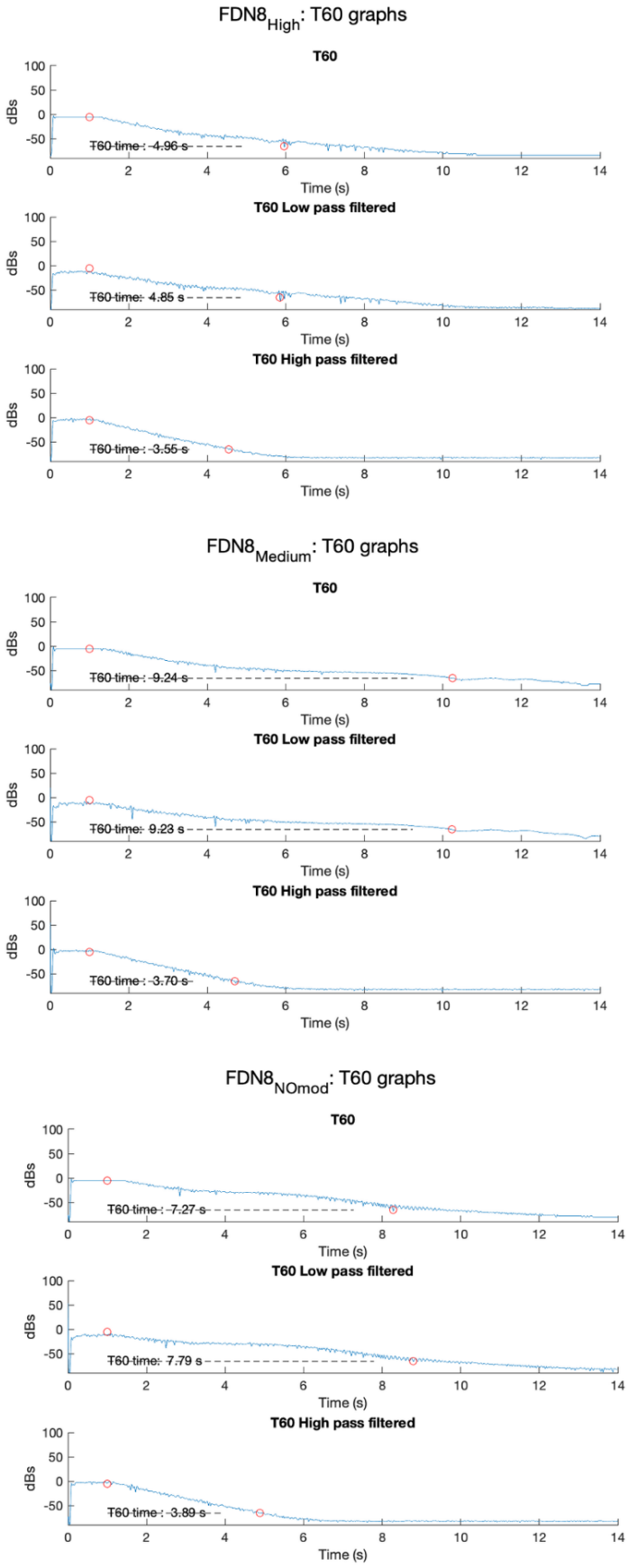


Figure B.6: FDN8 T60 times for High modulation, Medium modulation and NO modulation.

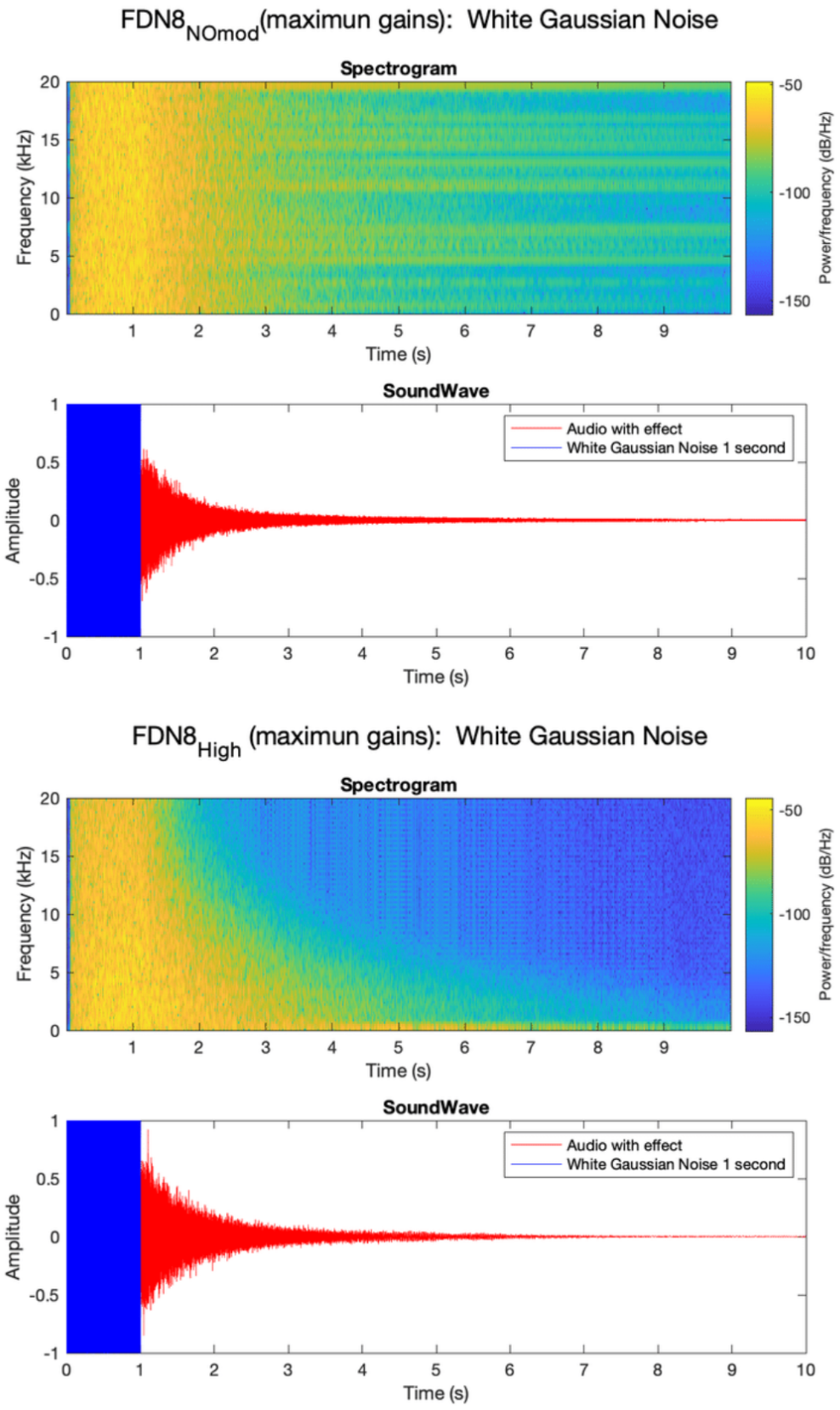


Figure B.7: FDN8with maximun feedback gains WGN response, with NO modulation and High modulation.

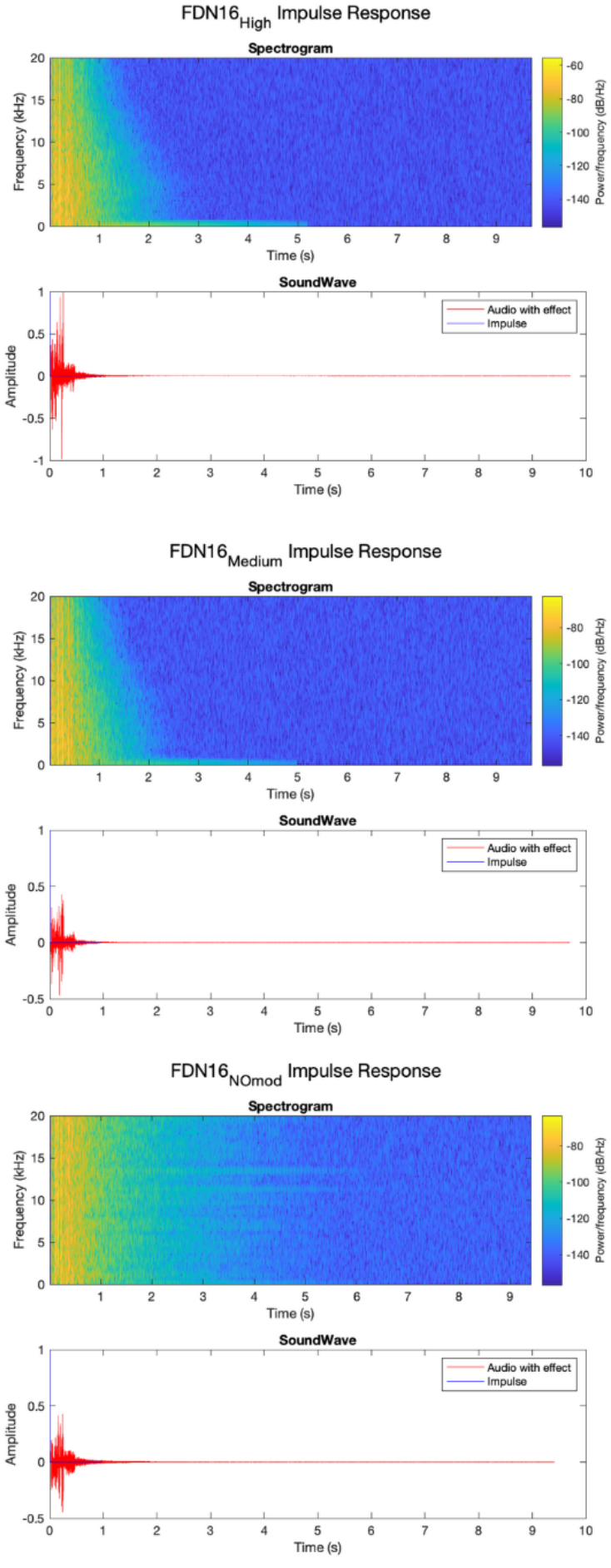


Figure B.8: FDN16 Impulse Response, for High modulation, Medium modulation and NO modula-

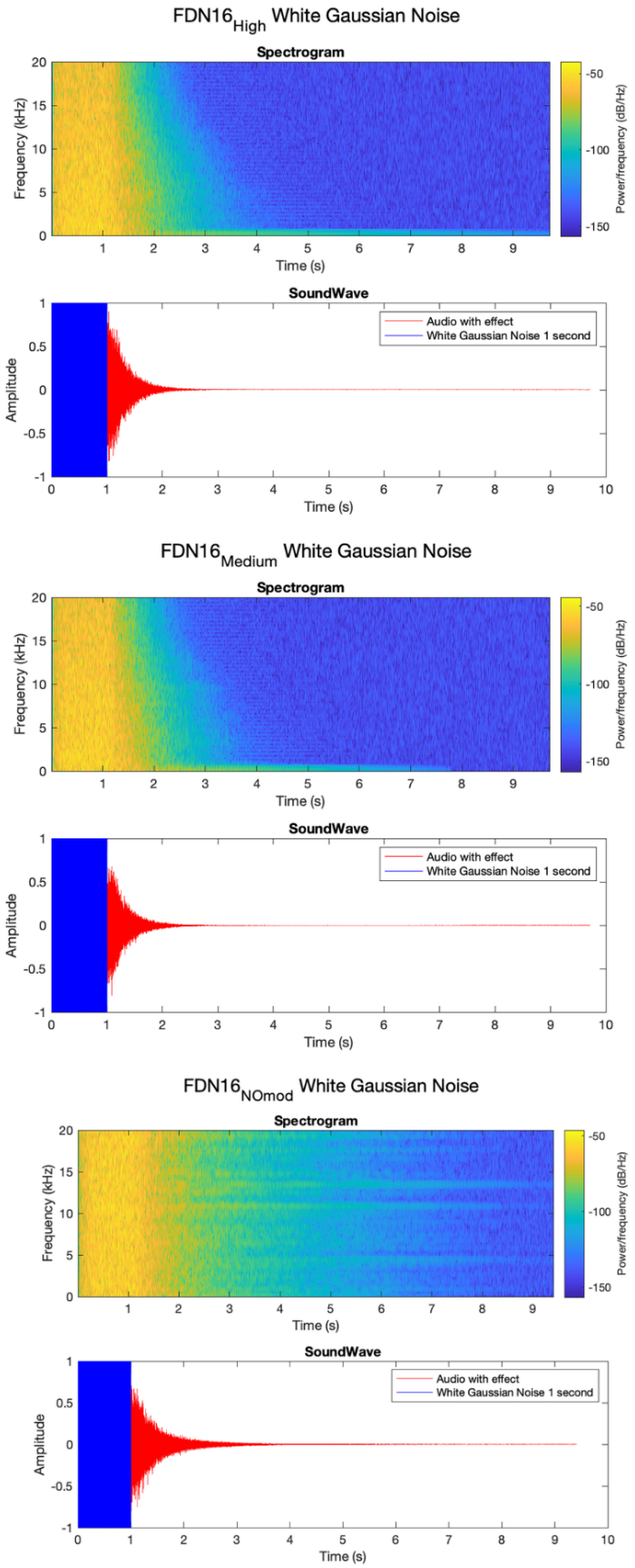


Figure B.9: FDN16 WGN response, for High modulation, Medium modulation and NO modulation.

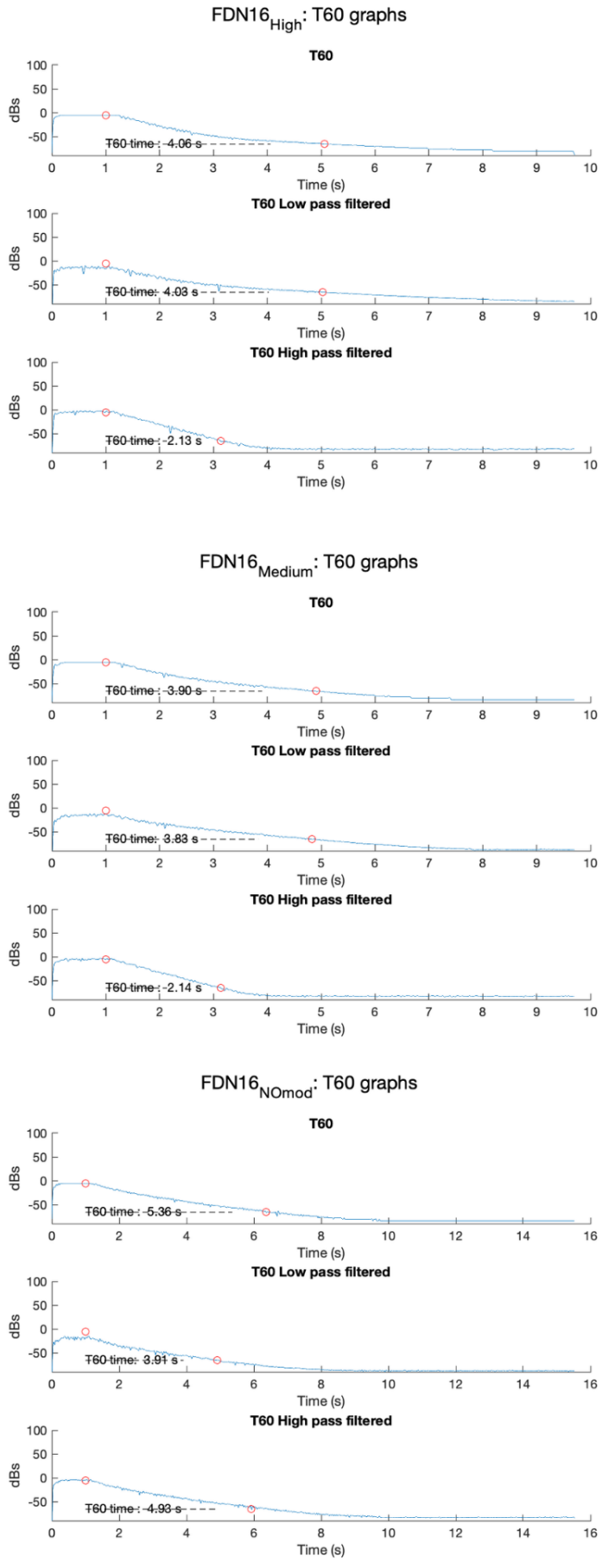


Figure B.10: FDN16 T60 times for High modulation, Medium modulation and NO modulation..

Appendix C

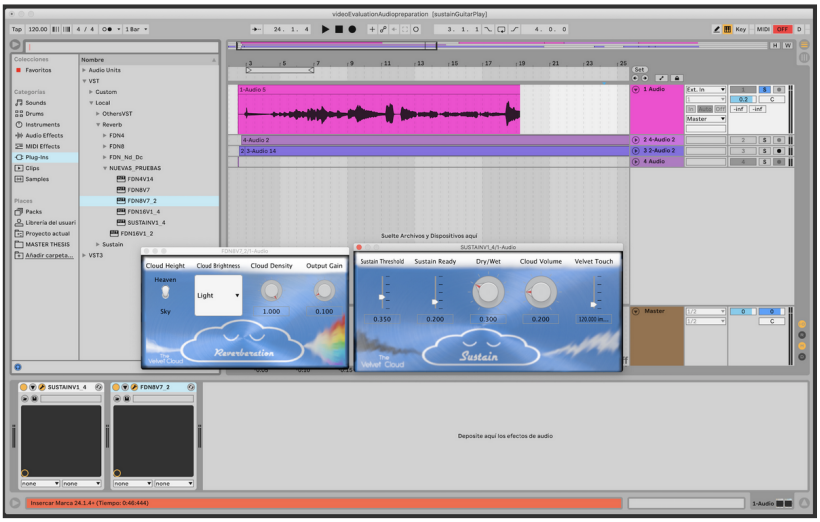
VST Evaluation Graphs/Figures

How was the experience of installing and running The Velvet Cloud?	What did you think The Velvet Cloud UI/design when you first saw it? Was anything hard to understand?	Anything in design that can be improved?	Play freely with the GUIs and set up the parameters to the configuration you like the most for the output sound. Make a screenshot of the VST configuration and upload it (less than 1MB, you can compress it or make a screenshot of the screenshot(reduced) to make the file smaller):	What surprised you the most when using The Velvet Cloud? (was there anything that worked differently than expected?)	What is your least favorite part about The Velvet Cloud?	What is your favorite part about The Velvet Cloud?	Was there anything about The Velvet Cloud that felt like a mistake or a bug?	If you could add any feature(s) what would it/they be?
Filter (3 chars minimum)	Filter (3 chars minimum)	Filter (3 chars minimum)		Filter (3 chars minimum)	Filter (3 chars minimum)	Filter (3 chars minimum)	Filter (3 chars minimum)	Filter (3 chars minimum)
Had trouble getting vst to work, ended up using AU	It looks nice. Limited options of control makes it look relatively easy to go to.	The background could be a little less dominating, it takes quite some attention, but apart from that it	Sk_rmbillede_2020-12-(	I had difficulties using and understand the sustain plugin. The FDN sounds super nice!	The sustain part, I had some clicking and I am not sure if it is because of my setting. It was	The FDN. It sounds really nice!	Some clicking in the sustain part	Maybe some visual feedback for setting the threshold of the sustain
No problem, except on the GitHub page it says "VST install (for Mac, Windows 64bits, Linux)" but it	(Sustain) The only knobs that I understand somewhat are Dry/Wet and maybe	Make design more polished and uniform. Decide to use either sliders or knobs.	Screen_Shot_2020-12-(	I had not used a sustainer before, so I was surprised to figure out that it sustained the sound indefinitely.	The sustainer does not feel very good to play live, maybe it's a bit slow to detect notes or something?	Sustaining a nice chord -- it can give the guitar an almost synthy quality.	The reverb has A LOT of modulation which results in audible detuning in some cases.	Automation/kill switch over when the sustainer sustains. It would be fun to play with a foot pedal that
It was alright. No hiccups	Appearance, I really like it. It is cute and unique. UI-wise, I was quite lost at first glance with what	Appearance-wise, maybe make the plugin interface higher resolution. At least, it looked pretty	Screenshot_2020-12-06	How easy it was to get a pretty deep reverb effect and a sustaining droning sound. While playing	It takes a bit of fiddling around to get the sustain behaviour that you want. While you let a note drone	The modulation behavior adds a nice touch to the reverb effect, giving it a pleasant timbre (in	Not playing with the guitar. But I did try to play it with a VST synth or instrument, and it definitely felt	Finer control of the modulation behavior. Let me control the rate, the depth of the modulation, whether
Fairly easy. Drag, drop and open is as simple as it gets.	Nothing was hard to understand. It all looked very straight forward.	I'm not sure if having the word 'Cloud' in front of the parameters is necessary. Also, I	SusRevTest.jpg	I wasn't expecting the Reverberation to seem so psychedelic (in a good way).	I don't like the Sustain much.	I do like the Reverberation plugin.	With the reverb plug-in, it was hard for me to tell the difference between 'Heaven' and 'Sky' without	No suggestions.

Figure C.1: EU Survey answers (not all answers).

Regarding preferred performance

Example configuration of VST in DAW:



For reference, here you can see a video of a guitarist playing with The Velvet Cloud.

Play your (electric) guitar or download this guitar.mp3 audio file and use the **Reverb and Sustain plugins** in your favourite DAW and see how it sounds. **Any configuration is valid** (ie: first sustain, after reverb, combining them in parallel...).

Play freely with the GUIs and set up the parameters to the configuration you like the most for the output sound.

- **Make a screenshot of the VST configuration and upload it**

(less than 1MB, you can compress it or make a screenshot of the screenshot(reduced) to make the file smaller)**:**

Select file to upload

Figure C.2: Screenshot of the EUSurvey.