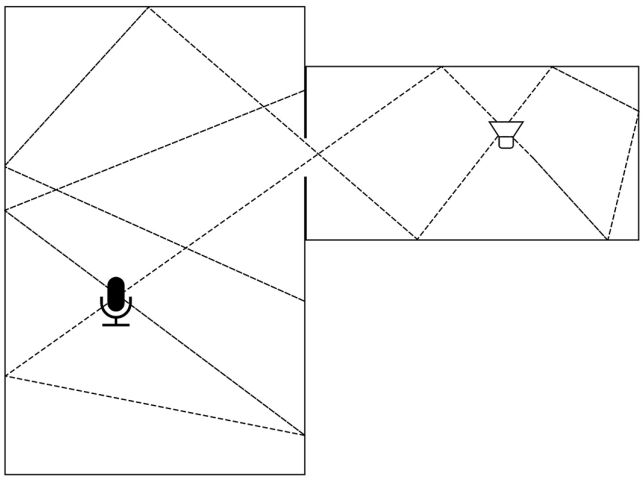
REAL-TIME IMPLEMENTATION OF COUPLED-ROOM FEEDBACK DELAY NETWORKS MODIFIABLE THROUGH OPEN SOUND CONTROL

BY
ODDUR INGI KRISTJÁNSSON



AALBORG UNIVERSITY COPENHAGEN

SOUND & MUSIC COMPUTING MASTER THESIS

DECEMBER 2021

Aalborg University Copenhagen

Aalborg University Copenhagen
A.C. Meyers Vænge 15, 2450
Copenhagen SV, Denmark

Secretary: Judi Stærk Poulsen
Phone: 9940 2468
judi@create.aau.dk

Semester: 9th Semester

Title: Real-time Implementation of Coupled-room Feedback Delay Networks Modifiable Through Open Sound Control

Project period: Fall 2021

Semester theme: Master Thesis

Supervisor(s):
Stefania Serafin
Sebastian J. Schlecht
Karolina Prawda

Members:
Oddur Ingi Kristjánsson

Oddur Kristjánsson

Abstract:

Digital methods for creating artificial reverberation have been investigated for more than 50 years, starting with digital all-pass filters. Later development led to the Feedback Delay Network approach which is considered as a state-of-the-art artificial reverberation. However, research has mostly focused on simulating a single room's impulse response while coupled room acoustics have lacked investigation. This project focuses on designing a real-time implemented plugin for simulating coupled room acoustics through Feedback Delay Networks. Furthermore, the plugin allows for remote control of parameters with the Open Sound Control protocol. Different evaluation methods for subjective and objective evaluation are proposed for the plugin. Although technical limitations resulted in the implementation of only single Feedback Delay Network, the plugin can be used for research purposes by comparing offline rendered reverberation to its real-time implemented counterpart.

Contents

0.1 Abbreviations

- Finite Impulse Response - **FIR**
- Infinite Impulse Response - **IIR**
- Digital Signal Processing - **DSP**
- Low-Pass Filter - **LPF**
- High-Pass Filter - **HPF**
- Band-Pass Filter - **BPF**
- All Pass Filter - **APF**
- Low Shelving Filter - **LSF**
- High Shelving Filter - **HSF**
- Feedback Comb Filter - **FBCF**
- Digital Waveguide Wetworks - **DWN**
- Feedback Delay Network - **FDN**
- Early Reflection - **ER**
- Impulse Response - **IR**
- Room Impulse Response - **RIR**
- Decibel - **dB**
- User Datagram Protocol - **UDP**
- Open Sound Control - **OSC**
- New Interfaces for Musical Expression - **NIME**

List of Figures

Chapter 1

Introduction

Along with the development of music recording and broadcasting in the early 1900s, the need for artificially created reverberation arose. Initially, specifically designed echo chambers were made where a loudspeaker located inside the chamber played a dry audio signal which was then summed with the response of the space [32]. Later development included electromechanical devices such as spring- and plate reverberation [17, 47]. In the 1960s, digital methods for artificial reverberation were proposed by Schroeder and Logan with the use of digital allpass filters [40]. Further development added comb filters to give a psychoacoustically appropriate fluctuation in the frequency response [41]. Later, the Feedback Delay Network (FDN) approach was suggested by Gerzon [11] and in 1991, Jot and Chaigne's [14, 15] FDN methodology allowed for adjustable reverberation time of individual frequency bands [47]. The FDN can be seen as a vectorised feedback comb filter and is considered as a state-of-the-art [47] artificial reverberation.

While the focus has mostly been on designing artificial reverberation to match a single room's impulse response (IR), there was a lack of investigation on modelling coupled room acoustics. Successful modelling of the perceptual experience of transitioning between two rooms has been investigated [18, 19], where IRs were recorded at regular intervals with a spherical microphone. Although highly accurate, the approach suffers from its computational load. In [7], a grouped FDN (GFDN) method was proposed as an efficient way to model coupled room's acoustic with controllable aperture, where each individual room is modelled with an FDN. This method has also been successfully implemented as a real-time application[1].

Converting offline modelling into a real-time application can be a tedious process and adjusting individual parameters may require reprogramming the source code. This project investigates how a real-time application built on the FDN approach for coupled room acoustics with remotely modifiable parameters at run-time can be designed and implemented.

[1] https://github.com/orchidas/GFDN

Chapter 2

Analysis

The following chapter describes reverberation, what it is, how it exists in the real world, as well as the different techniques used to simulate reverberation in the abstract world.

2.1 Reverberation

Reverberation as a term refers to sound being prolonged by the environment's reflective surfaces. When a sound is emitted in a room, it travels around the room, reflects off the different surfaces and travels back to the listener, carrying an 'imprint' of the space [47]. Once a sound wave reaches the ears of the listener, it has, in most cases, been reflected by one or more surfaces or objects in the room. Furthermore, a number of reflections are often needed before the sound waves become inaudible [10]. A room's acoustics can be measured as an impulse response (IR) which, along with reverberation, can be broken down into three stages, the *Direct Path*, the *Early Reflections*, and the *Late Reverberation*, as shown in figure 2.1.

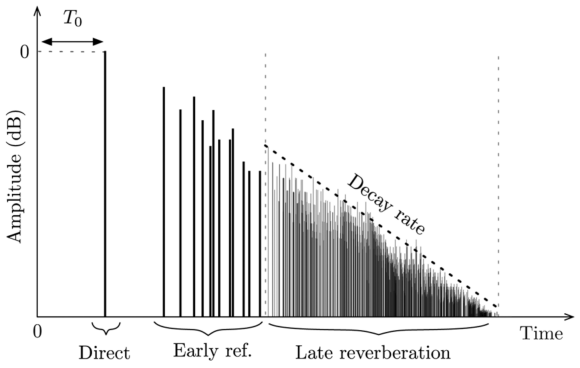


Figure 2.1: An example of the different stages of reverberation. Figure from [47].

2.1.1 Direct Path

The direct path, sometimes called the *direct sound*, is the first sound heard by the listener. This sound propagates straight from the sound source to the listener's position. As shown in figure 2.1, the direct sound arrives at the listener after a delay of T_0 which depends on the distance between the sound source and the listener [10]. As this sound

has not reflected off of objects in the space, the direct sound carries the information about sound source location [47].

2.1.2 Early Reflections

The early reflections reach the listener shortly after the direct sound and they consist of sound waves that have reflected once or more off of objects or surfaces in the room. Due to the spherical distance attenuation and absorption properties of different surface materials, the energy of early reflections is reduced compared to the direct sound [10]. Furthermore, the early reflections give a strong sense of the space, especially of its geometry and surface materials [47], as well as the size of the space as they can still be recognised as separate reflections [33].

2.1.3 Late Reverberation

Late reflections, or late reverberation, refer to the most diffuse part of the IR where individual reflections are no longer distinguishable from one another. These reflections are usually referred to as the reverberation of a space [10]. The time needed for all sound to be attenuated by 60dB is called the reverberation time and is often denoted T_{60} [47]. In the 1890s, W.C. Sabine developed a formula to calculate the reverberation time which still to this day is named the *Sabine Reverberation Formula* (equation 2.1) [3]. Here, V is the room's volume (m^3), and $S\bar{a}$ is total area absorption (sabins) [10].

$$T_{60} = \frac{0.161V}{S\bar{a}} \tag{2.1}$$

2.2 Artificial Reverberation

With the development of music recording and broadcasting, the need for artificially created reverberation arose. Without any reverberation, the sound produced by the studio environment resulted in a 'dry' sound which was especially present when close micing an audio source as the microphone failed to pick up any room acoustic [47]. This need for reverberation led to the introduction of artificial reverberation, as early as the 1920s [47]. The following sections describe different techniques used to create artificial reverberation with analog and digital methods.

2.2.1 Analog Methods

In order to create an artificial reverberation, a number of devices and methods were designed. In the beginning, echo chambers, specifically designed enclosures with reflective walls, were built. The dry recorded signal was played through a loudspeaker positioned inside the room and the response of the space was then recorded with a microphone placed in an appropriate position to minimise the direct sound [32]. The reverberated signal would then be summed with the original dry signal at a desired

proportion. These chambers allowed for damping materials to be added to better control the reverberation time [4, 30]. While providing a high-quality reverberation, the echo chambers lacked flexibility [47].

Additionally, electromechanical devices such as spring reverberations and reverberation plates were developed [47]. In the late 1920s, Hammond [17] invented the spring reverberator, an electromechanical transducer which, by applying an electrical signal, excited vibrations in helical springs where each spring element produced a decaying series of echoes [47]. The simplest spring reverberators used two electromagnetic coils and two springs held under tension. In spring reverberators, the higher frequencies are of much lower amplitude than lower frequencies [25]. In plate reverberators, a thin steel[1] plate is excited with an electrodynamic actuator, placed around the plate's centre while a contact microphone is used at the plate's end to pick up its vibrations [2]. Damping material could then be used alongside the plate to better control the reverberation time. As the plate response is quickly dense and the higher frequencies travel faster than the lower frequencies, plate reverberation has a characteristic sound. [47].

2.2.2 Digital Methods

Approximately 40 years after the invention of the echo chambers (section 2.2.1), the first digital algorithms for reverberation were proposed [47]. The following sections describe some of the most common digital methods for creating artificial reverberation.

2.2.2.1 Schroeder and Moorer Reverb

In the 1960s, Schroeder and Ben Logan proposed the digital allpass filter (APF) [40]. By feeding an audio signal into an APF, a series of 'colourless' decaying echoes was produced. This is due to the APF's nature of passing all frequencies with equal gain [47]. In order to increase the echo density, Schroeder and Logan proposed at least five APFs connected in series [40]. Schroeder and Logan further refined the reverberation and proposed the approach of adding comb filters to give a psychoacoustically appropriate fluctuation in the reverberator's frequency response [41]. Here, four comb filters connected in parallel were proposed along with two APFs in series for a natural-sounding artificial reverberation, allowing for a wide choice of decay, reverberated sound and mixing ratios [38]. This approach is known as the *Schroeder Reverb* [45].

Schroeder's reverb, however, produced 'metallic' sounding reverberation. Further improvements to the algorithm led to inserting a finite impulse response (FIR, section 2.3.1) filter at the start of the signal chain to simulate the early reflections [21, 39]. Furthermore, Moorer stated the importance of making all the delay lengths mutually prime in order to reduce peaks building upon the same sample [21]. This approach of adding the early reflections at the start of the signal chain, before being processed by

[1] Among other materials, such as gold foil

parallel feedback comb filters (FBCF) and APFs in series, is often referred to as the *Moorer Reverb* [45]. Figure 2.2 displays a block diagram of both the classical Schroeder and Moorer reverb.

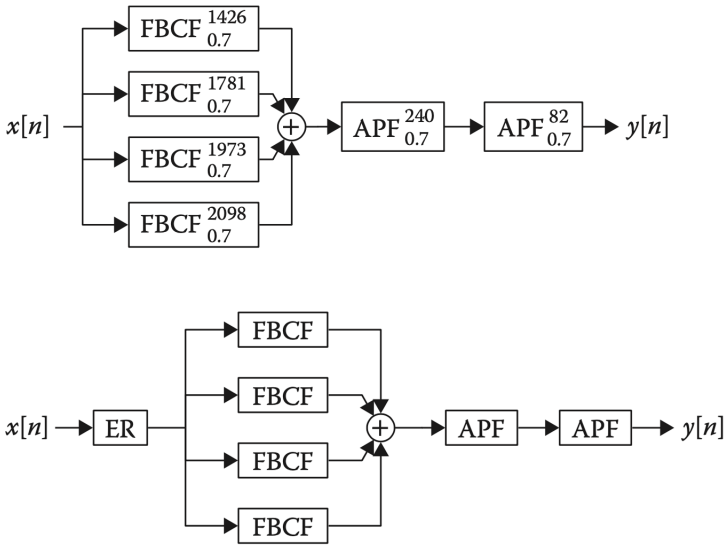


Figure 2.2: Above: Schroeder Reverb. Below: Moorer Reverb. Figures from [45].

2.2.2.2 Convolution

Convolution reverbs were introduced in the late 1990s and to this date are quite popular in music production [47]. The recorded IR of a specific is convolved with the input audio signal as shown in equation 2.2, where $h(t)$ is the IR.

$$y(t) = x(t) * h(t) \tag{2.2}$$

The advantage of the convolution is that with a high-quality room IR (RIR), the convolution will yield a very accurate representation of the room. The drawback, however, is that it is very computationally demanding, as can be seen by equation 2.3. For each sample index (n) increase, all overlapping samples at time n are multiplied, added together, and set as output. Here, k is the length of the IR. As mentioned in [47], a 1 second RIR at the sample rate of 48,000 Hz will require 48,000 multiplications and additions per output sample. These calculations grow proportionally to the number of channels.

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k] \cdot h[n-k] \tag{2.3}$$

Another drawback of the convolution method is that the room reverberation is only accurate for the specific place in the room the RIR was recorded. Therefore, for a realistic simulation of any receiver position in a room, multiple RIRs and interpolation procedures may be needed. Additionally, the introduction of more RIRs will result in more disk space usage.

2.2.2.3 Feedback Delay Networks

Building on the Moorer FBCF approach (section 2.2.2.1), an algorithm called Feedback Delay Network (FDN) was introduced and can be regarded as a vector of FBCFs [43, 47]. Here, each delay block is fed back into itself creating repetitions in the signal [45]. As can be seen in figure 2.3, instead of individual gains on each delay line, a feedback matrix is used instead.

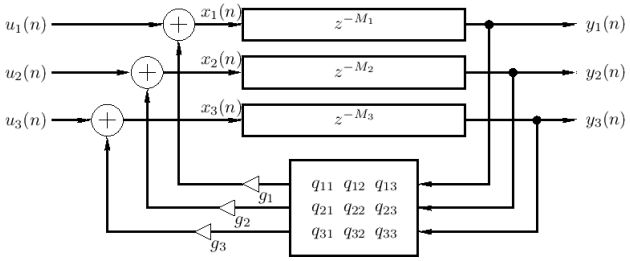


Figure 2.3: A 3rd order FDN. Figure from [42].

The FDN method was first introduced by Gerzon [11] where he proposed the use of an orthogonal matrix for the feedback matrix reverberation unit. A square matrix is orthogonal if its inverse is identical to its transpose [50] as shown in equation 2.4.

$$A^{-1} = A^T \tag{2.4}$$

By cross-coupling several of the FBCF, Gerzon believed the artificial reverberation to sound good compared to individual FBCF and that the orthogonal feedback matrix would obtain rich cross-coupling [42]. Later, Stautner and Puckette [44] proposed a specific gain matrix (shown below), where g controls the reverberation time [14], to be used for FDNs which was believed to give good stability [41], that is successive powers of the gain matrix become smaller instead of larger [44]. This matrix can be considered a one-row sign inversion of a Hadamard Matrix [42]. A Hadamard matrix is a square matrix where half the adjacent cells in the next rows/cells have the same sign (+/−) and half have an inverted sign [49].

$$A = g\frac{1}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

At the start of the 1990s, Jot and Chaigne [14, 15] developed an FDN methodology which allowed for adjusting the reverberation time of individual frequency bands and this approach is the current level of application [42, 47]. Most notable additions to the earlier shown FDN (Figure 2.3) are the input gains b, output gains c, the low-order filter called 'tone-corrector' by Jot [14] $E(z)$ at the end of the signal chain, as well as a dedicated dry path d. This is shown in Figure 2.4. In order to simulate how higher frequencies attenuate faster than lower frequencies [21], frequency-dependent per-sample low pass filtering can be inserted after each delay line instead of the g gains [31]. This will be further explained in section 2.3.3.

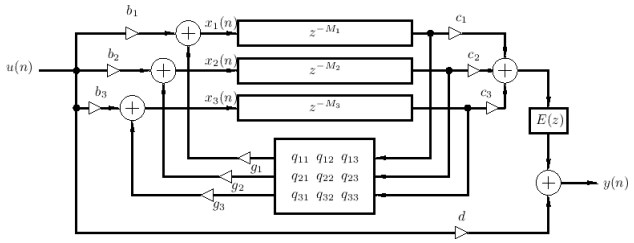


Figure 2.4: Proposed FDN by Jot. Figure from [42].

FDNs are still considered the most efficient method to create artificial reverberation [47]. Through the relatively low-level approach, the FDNs can simulate how audio behaves in the real world since apart from each delay block being fed back into itself, the output from all the other delay blocks are fed into it as well, similar to how audio reflects from one object to another instead of reflecting in isolation [45]. Additionally, FDNs allow for modification of multiple different aspects such as the in- and output gains, the feedback matrix [34] and feedback delays [12], as well as the early reflections [35]. However, a great challenge when designing FDNs is the trade-off between *computational complexity*, *mode density*, and *echo density*, as reduced modal density can lead to metallic sounding artefacts, while reduced echo density can cause rough rattling sounds. With a higher number of delays, both modal and echo density is increased, but so is the computational complexity [35].

The feedback matrix is a crucial aspect for high-quality artificial reverb as it has a strong impact on modal distribution as well as time-domain evolution of the reverberator. Modal distribution is how resonant modes of the filter are spread with respect to frequency. In digital reverberation, the modes are dependent on the length of delay lines [46]. As an example, the FBCF used in Schroeder's reverb (section 2.2.2.1), has a nearly harmonic set of resonances as seen in figure 2.5, which is why the APF was added to flatten the magnitude response. In FDNs, on the other hand, the modal frequencies are distributed uniformly across the frequency spectrum [13, 37]. In order to increase the perceived modal density of the FDN, time-varying some elements is a known strategy, be it the length of the delay lines or the filter coefficients [46]. Schlecht et al. [36] showed that an FDN reverberation with a modulated feedback matrix was perceived to be of higher quality than a non-modulated FDN.

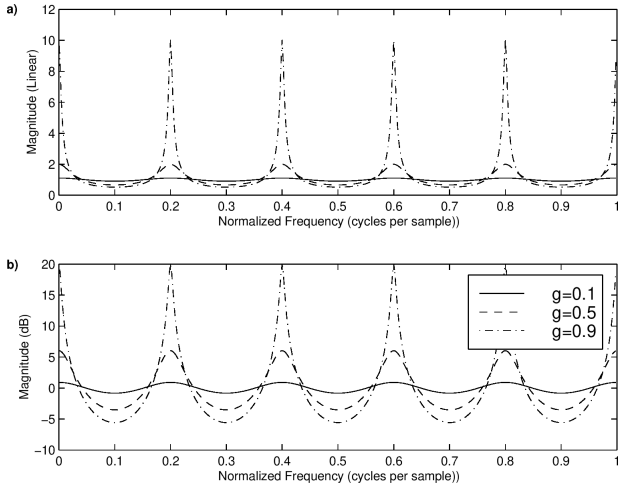


Figure 2.5: The harmonic resonances of a FBCF, where g = filter's feedback gain. Figure from [41].

2.2.3 Coupled Rooms Modelling

Coupled room acoustics refers to two rooms/volumes, usually differing in size and absorption, that are linked together with an opening [6]. As FDNs have mostly been designed to match a single measured IR, investigation on how they can be used to model coupled rooms has lacked [7]. In [7], the focus was on simulating reverberation of two coupled rooms with FDNs. This method, called Grouped Feedback Delay Network (GFDN), consists of two FDNs, one for each modelled room. In a GFDN, delay lines with the same target T60 are grouped together unlike with FDNs where all delay lines share the same decay characteristics [7].

In coupled rooms acoustics, a sound source in one of the rooms will travel to the other room and spill back into its original room. This can be better understood by looking at figure 2.6, where the sound source is located in the smaller room (R1). The audio will then decay at a different rate in R2 due to its different acoustics.

Here, a mixing matrix controls the amount coupling between the two rooms. As mentioned in [7] and implemented in [35],[2], the mixing matrix can be written as equation 2.5, where M are orthogonal matrices, θ are the mixing angles of the independent rooms and ϕ is the coupling angle [7].

$$M(\theta_1, \theta_2, \phi) = \begin{bmatrix} M(\theta_1)cos\phi & M(\frac{\theta_1}{2})M(\frac{\theta_2}{2})sin\phi \\ -M(\frac{\theta_2}{2})M(\frac{\theta_1}{2})sin\phi & M(\theta_2)cos\phi \end{bmatrix} \tag{2.5}$$

[2] example_coupledRooms.m

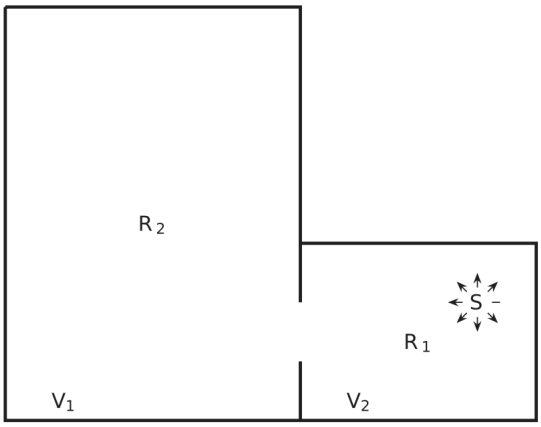


Figure 2.6: Coupled Rooms. Figure from [7]

These mixing matrices can be scaled by a Householder matrix.

$$\begin{aligned} H &= I - 2uu^T \\ u &= \begin{bmatrix} cos\phi & -sin\phi \end{bmatrix} \\ H &= \begin{bmatrix} -cos2\phi & sin2\phi \\ sin2\phi & cos2\phi \end{bmatrix} \end{aligned} \tag{2.6}$$

When $\phi = 0$, there is minimum coupling between the rooms while $\phi = \frac{\pi}{4}$ gives maximum coupling [7]. For an easier understanding of the term *coupling*, it can be imagined as a door between the two rooms. The more the door is open, the more coupling there is.

Source placing can then be controlled by the b and c gains of the GFDN (see figure 2.7), where the b gains can be considered the sound source and c gains can be considered the receiver. Similar to the FDN shown in figure 2.4, these gains control the input and output gains of each FDN. If $b_1, c_1 = 0$, the sound source and receiver are positioned in the same room. Furthermore, if $\phi = 0$, the rooms are decoupled (imagine a fully closed door), resulting in no output from the GFDN [7].

Das and Abel [7] proposed first-order low shelf filters for absorption with T60 values controlled by DC and Nyquist gains, and transition frequency. These filters were implemented as bi-quad filters. Each room was simulated as an FDN of 16 delay lines, resulting in a 32 delay line GFDN with controllable T60, parameterized at DC and Nyquist and f_c for each room.

This was then further expanded by Das, where a real-time implementation in JUCE[3] was created[4]. As can be seen in figure 2.8, the user has the option to control multiple

[3] https://juce.com
[4] https://github.com/orchidas/GFDN

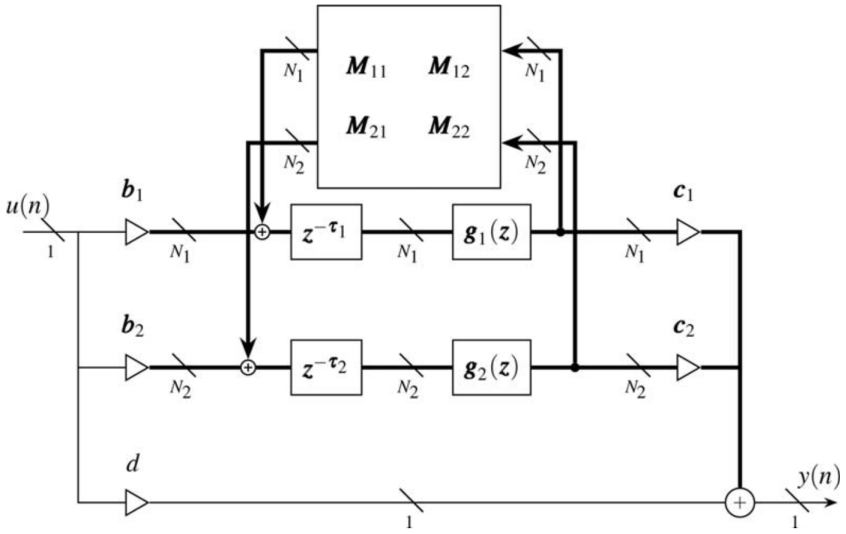


Figure 2.7: Block diagram of a GFDN. Figure from [7].

parameters such as in which room the source and listener is positioned, how much coupling there is, etc.

Figure 2.8: Real-time plugin of a GFDN. Figure from Github[3]

The coupled room GFDN method by Das and Abel, is a clever way of modelling coupled room acoustics. Additionally, the approach can be expanded into N number of rooms

[7]. The real-time implementation, furthermore, is an efficient product that allows it to be used in a wide variety of scenarios, be it music production, or for Virtual Reality (VR) applications.

2.3 Absorption Filters

In order to simulate decaying sound with digital signal processing (DSP), an algorithm that reduces energy over time is needed. As higher frequencies decay faster than lower frequencies, frequency-dependent filtering will provide the most realistic frequency decay over time.

There is a wide variety of filters used for equalisation, low/high-pass filters (LPF/HPF), band-pass filters (BPF), comb filters, shelving filters, and more [45]. Important terms used to describe filters are listed below [45]:

- **Pass Band:** The frequency range where a signal passes through unchanged
- **Stop Band:** The frequency range where a signal is attenuated by the system
- **Transition Band:** The frequency range between the pass- and stop band
- **Cutoff Frequency (f_c):** The point in the spectrum when a signal is attenuated by 3dB
- **Filter Order:** The number of samples of delay. First-order filters e.g., use only a single sample of delay

These aforementioned filters are generally designed in two ways, as an FIR filter or as an infinite impulse response (IIR) filter. The following section describes the differences between the two methods.

2.3.1 FIR and IIR Filters

FIR filters are feed-forward implementation of filters. The delay lines are not fed back into the system but added with the unprocessed signal (figure 2.9) and are, therefore, always stable (section 2.3.2). However, they require a much higher filter order compared to IIR for the same level of filtering and their delay is often way longer.

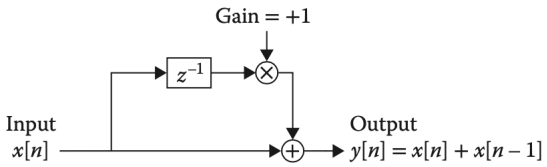


Figure 2.9: An FIR LPF block diagram. Figure from [45].

IIR filters' structure is recursive, that is, the output samples are fed back into the filter for further processing [8]. In order for the filter to be considered stable, the denominator of the transfer function has to be < 1.

The filter coefficients of the IIR filter are described using $[b_0, b_1, ..., b_M]$ and $[a_0, a_1, ..., a_M]$, where b_0 is the gain for the signal without a delay, b_1 is the gain for the one-sample delay, etc., and M describes the number of samples. Furthermore, the b_m gains relate to the feedforward paths while the a_m gains relate to the feedback paths [45]. Note that these b_m filter coefficient values are unrelated to those used when describing FDNs (section 2.2.2.3).

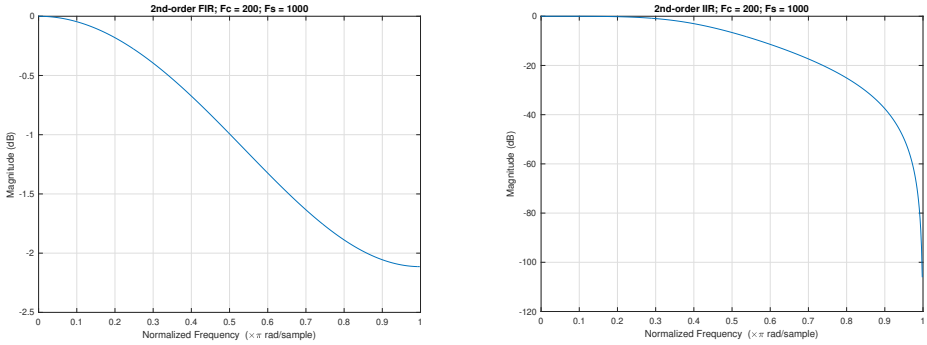


Figure 2.10: Left: 2nd-order FIR LPF. Right: 2nd-order IIR LPF.

2.3.2 Pole-Zero Analysis

An IIR filter can become unstable if not carefully designed. That is, the output gain will increase in each processing. The instability is depicted in figure 2.11. Here, a simple 1-sample delay IIR filter (equation 2.7) is visualised with an $\alpha = 1.5$, where α is the gain in the feedback path. Thus, the output gain increases exponentially.

$$y[n] = x[n] + \alpha y[n-1] \tag{2.7}$$

A good way to visualise whether a system is stable or not is by pole-zero analysis, and here Euler's formula (equation 2.8) can be used.

$$e^{j\theta} = cos(\theta) + jsin(\theta) \tag{2.8}$$

By knowing the system's differential equation, the frequency response can be computed by the use of Euler's formula as is shown in equation 2.9 and more specifically, its magnitude is calculated by $|H(e^{j\theta})|$.

$$H(e^{j\theta}) = \frac{\sum_{m=0}^{L} b_m e^{-j\theta m}}{1 - \sum_{m=1}^{N} a_m e^{-j\theta m}} \tag{2.9}$$

On a pole-zero plot, the poles (the denominator/a coefficients) have to be < 1 for the filter to be stable while the zeros (numerator/b coefficients) can be either within or outside of the unit circle [8].

In short, the zeros and poles are visualised in relation to a unit circle when plotted with a pole-zero plot. Through a frequency sweep, if $e^{j\theta}$ is 'closer' to a zero, the system will attenuate the signal and when 'closer' to a pole, the system will amplify the signal. The frequency sweep starts at 0 and travels counter-clockwise on the unit circle until π [24].

The aforementioned filter is depicted on a pole-zero plot in figure 2.12, and as can be seen, the pole is located outside of the unit circle as it is an unstable system. Also shown is the filters magnitude response where the lower frequencies increase in amplitude while the higher frequencies are attenuated.

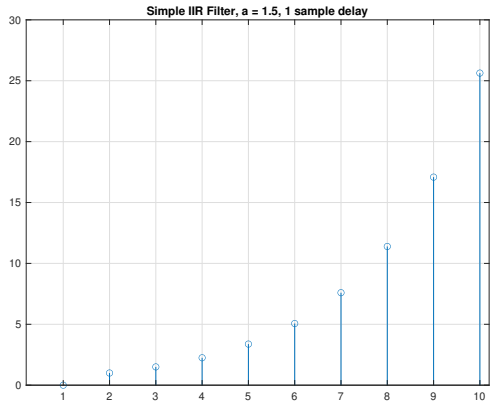


Figure 2.11: Unstable IIR filter.

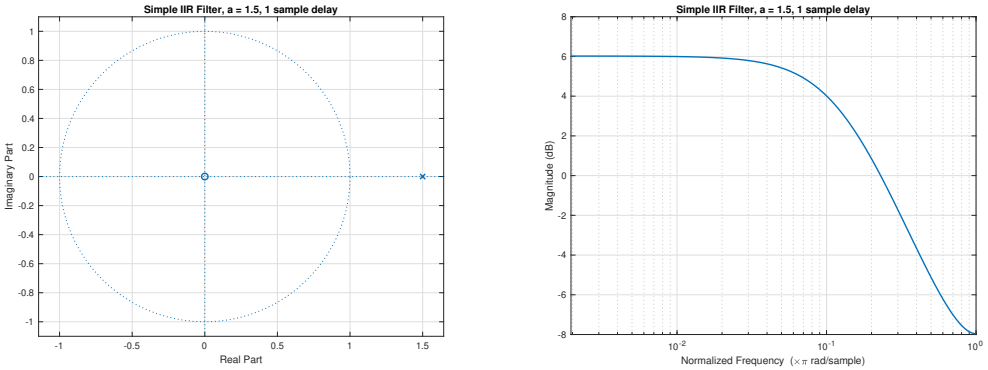


Figure 2.12: A pole-zero plot and magnitude response of an unstable IIR filter.

2.3.3 Shelving Filters

A very flexible and powerful type of equalisation is the parametric equaliser. A special type of filter often used for the high and lower frequencies is the shelving filter [48]. For a smooth transition between the affected and unaffected frequency regions, a first-order shelving filter can be designed. Equation 2.10 shows the transfer function of a first-order low shelving filter (LSF).

$$H(z) = \frac{G\ \tan(\omega_c/2) + \sqrt{G} + [G\ \tan(\omega_c/2) - \sqrt{G}]z^{-1}}{\tan(\omega_c/2) + \sqrt{G} + [\tan(\omega_c/2) - \sqrt{G}]z^{-1}} \tag{2.10}$$

Here, a gain value G is applied to all frequencies below or above a defined f_c [48]. This gain value can be defined as $\sqrt{G}$ in order to retain the magnitude response when varying the gain. Here, $\omega_c = 2\pi f_c/f_s$ is the cutoff frequency in radians and f_s is the sampling frequency.

The simplified transfer function of the first-order LSF can then have the form of equation 2.11, where p represents the poles, q represents the zeros, and g acts as a scaling factor [48].

$$H_\rho(z) = g\frac{z-q}{z-p} \tag{2.11}$$

This approach of shelving filter design allows for modifying the transfer function (equation 2.10) into a high-frequency shelving filter (HSF). By changing G into $1/G$, and multiplying the numerator by G, the magnitude response is shifted vertically [48]. Multiplying both numerator and denominator by $\sqrt{G}$ cancels divisions by G resulting in equation 2.12. This, furthermore, means that to convert an LSF to an HSF, the pole and zero of the filter can be interchanged [48]. For a more detailed description, a step-by-step approach to this filter is explained in [48].

$$H(z) = \frac{\sqrt{G}\ \tan(\omega_c/2) + G + [\sqrt{G}\ \tan(\omega_c/2) - G]z^{-1}}{\sqrt{G}\ \tan(\omega_c/2) + 1 + [\sqrt{G}\ \tan(\omega_c/2) - 1]z^{-1}} \tag{2.12}$$

If we look at equation 2.12 and compare it with a general transfer function (equation 2.13), we see that a_1 consists of the term inside the square brackets, while a_0 is the rest of the numerator. The same goes for the b_0 and b_1 coefficients.

$$H(z) = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1}} \tag{2.13}$$

For a more clear overview, the coefficients of a LSF is shown in equation 2.14 and for an HSF in equation 2.15.

$$
\begin{aligned}
b_0 &= G\,\tan(\omega_c/2) + \sqrt{G} \\
b_1 &= G\,\tan(\omega_c/2) - \sqrt{G} \\
a_0 &= \tan(\omega_c/2) + \sqrt{G} \\
a_1 &= \tan(\omega_c/2) - \sqrt{G}
\end{aligned}
\tag{2.14}
$$

$$
\begin{aligned}
b_0 &= \sqrt{G}\,\tan(\omega_c/2) + G \\
b_1 &= \sqrt{G}\,\tan(\omega_c/2) - G \\
a_0 &= \sqrt{G}\,\tan(\omega_c/2) + 1 \\
a_1 &= \sqrt{G}\,\tan(\omega_c/2) - 1
\end{aligned}
\tag{2.15}
$$

The coefficients are, furthermore, normalised, resulting in $a_0 = 1$ [22], and therefore not present in the differential equation below [45].

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] - a_1 \cdot y[n-1] \tag{2.16}$$

As mentioned in section 2.1.3, reverberation time is defined by the time it takes the energy to decay by 60 dB. In order to get the desired per-sample filter gain value from the T60 value, the following equations can be used [1].

$$g_{dB} = -60\frac{1}{T_{60} f_s} \tag{2.17}$$

$$g_{lin} = 10^{\frac{g_{dB}}{20}} \tag{2.18}$$

$$g_i = (g_{lin})^{m_i} \tag{2.19}$$

In equation 2.17, a target T60 is used to get the needed attenuation value in the decibel scale. In order to convert it from the decibel's logarithmic scale into a linear scale, equation 2.18 can be used. Lastly, for the gain value to become dependent on the delay line length m_i, equation 2.19 is used.

In figure 2.13, two HSFs and LSFs are shown. Here, f_c is set to 2kHz, with a gain of $\pm$5dB at a f_s of 44,100 Hz. The gain value is calculated from equation 2.18 before the coefficients are calculated.

The pole-zero plot of the HSFs and LSFs furthermore reveals the first-order of the filters, as well as their stability. Looking at figure 2.14, we see that the HSF only has one pole and one zero, where the pole is within the unit circle. As mentioned above, the pole-zero plot for the HSF with -5dB gain is identical to the LSF with a 5dB gain, and vice versa.

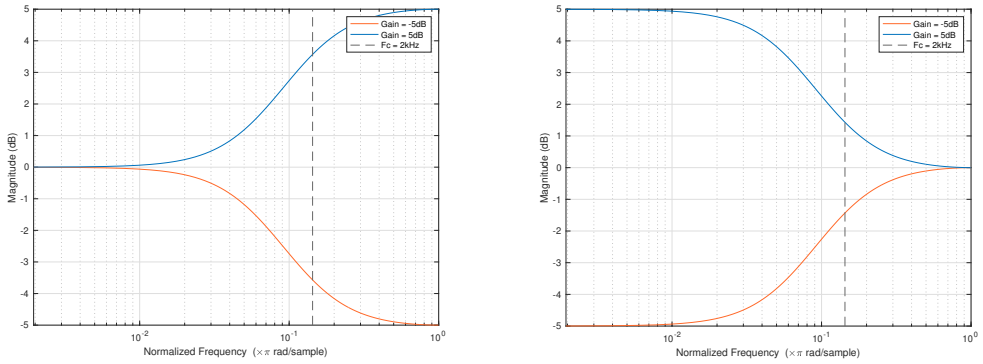


Figure 2.13: First-order HSF (left) and LSF (right) with gain ±5dB and f_c at 2kHz.

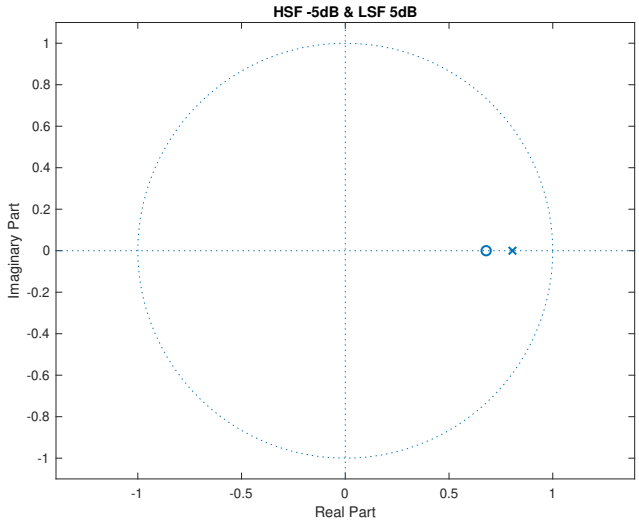


Figure 2.14: Pole-zero plot of an HSF with -5dB gain and an LSF with 5dB gain.

2.4 Problem Statement

It has been shown that artificial reverberation can be achieved with different methods, where each use case is different. The design can either be a real-time implementation[5] or offline rendering for more computationally demanding methods such as convolution [47].

The drawback of both real-time implemented and offline rendered products can, however, be that in order to change or re-program one thing, the code must be modified and the project has to be rendered again. This is especially evident in real-time virtual

[5] https://github.com/orchidas/GFDN

studio technology (VST) plugins, where in most cases, the user has no control of the code. If the plugin can be modified, it is also necessary that the user has access to the code and can understand it before building the plugin again. This approach can be tedious and time-consuming when only small adjustments are needed.

This project is done in collaboration with professor Sebastian J. Schlecht[6] and doctoral candidate Karolina Prawda[7] from Aalto University in Helsinki, Finland. The initial goal for this project was to make a real-time implementation of modelled coupled-room acoustics. In [18] and [19], the perceptual experience of transitioning between two rooms was investigated. There, four different transitions were measured with a fourth-order spherical microphone array in 5cm intervals, resulting in 101 IRs for each transition. Through time-varying convolution methods, the auralization was evaluated in VR and was shown to be perceived as highly natural. The computational load of this approach is, however, very high. By only looking at the Wav format IRs[8], they consist of 25 audio channels, 2 seconds in length, at a sample rate of 48kHz. By looking back at the example in section 2.2.2.2, this would result in $48,000 * 25 * 2 = 2,400,000$ calculations per output sample.

As was presented in section 2.2.3, FDNs can be used for modelling coupled room acoustics. The initial goal for this project was to make a real-time implementation of the GFDN method proposed in [7] (section 2.2.3). However, only shortly after the project started, Das released their own implementation[9]. It is, therefore, the goal of this project to design and implement a reverberation plugin built on the FDN approach for coupled room acoustics where most parameters can be modified at run-time without the need to modify the source code. This can prove important in research scenarios where designers may want to compare offline rendered reverberators to real-time implementation. Another scenario could be in VR applications, where the reverberation could remotely be tweaked as needed without having to stop the application or build the VR project again.

The problem can therefore be described as

> Design a real-time implemented VST for simulating coupled room acoustics through FDNs which can be modified remotely at run-time.

[6] https://www.sebastianjiroschlecht.com
[7] https://www.aalto.fi/en/people/karolina-prawda
[8] http://doi.org/10.5281/zenodo.4095493
[9] https://github.com/orchidas/GFDN

Chapter 3

Design

This chapter describes the design process of the developed product. The requirements and functionality of the final product are presented as well.

3.1 Design Requirements

The product has to fulfil the following requirements:

1. Operate in real-time
2. Sufficient number of parameters should be modifiable
3. Has to be efficient enough to run without audible clicks
4. Stability
5. Portability
6. Visual feedback on the variables' values

3.2 Specifying Functionality

As this product should allow modification of the FDN algorithm at run-time without the need of adjusting the code, it is important that the user can receive visual feedback of the variables' values. This should ensure that the user is modifying the correct parameters, as well as getting visual confirmation of the change. Based on this, the initial design is depicted in figure 3.1. Here, each individual room has its own adjustable sliders and the user can change the feedback matrix which is displayed in the bottom left corner. In the bottom right corner the Open Sound Control (OSC) (section 4.3) status is displayed.

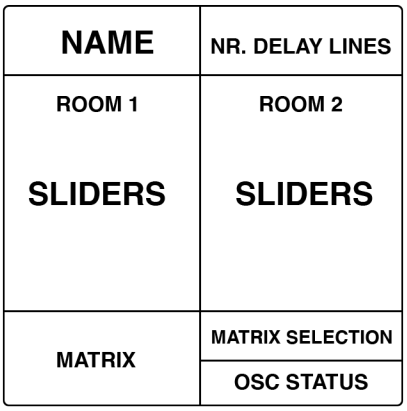


Figure 3.1: Initial design.

3.2.1 Real-Time Implementation

For this product to be successfully implemented, it will have to be able to run in real-time. That is, all parameters have to be modifiable and able to update without any rendering time. This can be achieved with frameworks such as JUCE, Max/MSP[1] or other software that can run and process samples at audio rate. Additionally, the product should be portable. That is, it should be able to run on different platforms, Windows/MacOS, mobile phones, VR devices, etc. In this work, JUCE was chosen as it allows for multiple plugin formats such as VST3, AU, AAX, standalone and Unity.

In real-time applications, audio disturbances such as clicks and other artefacts should be avoided. This can be achieved with interpolation of parameter values (section 3.2.1.1) as well as with efficient code. While some devices can handle computationally heavy tasks, others such as VR headsets and smaller devices have less processing power. One way the product can ensure efficiency between devices is by making the order of the FDN modifiable and allowing for turning delay line modulation either on or off [9].

3.2.1.1 Interpolation

Varying the FDN parameters, such as delay line lengths and matrix values, by changing them in one step leads to audible disturbances in produced sound [47]. Therefore, the change in variables needs to be implemented in smaller sub-steps. Here, the most computationally light method is *linear interpolation*, which estimates a value by linearly combining neighbouring samples (n and $n+1$) and can be expressed by equation 3.1, where f_{rac} is a fraction of a sample, $0 < f_{rac} < 1$ [45].

$$(1 - f_{rac}) \cdot x_n + (f_{rac}) \cdot x_{n+1} \tag{3.1}$$

[1] https://cycling74.com/products/max

This is shown in figure 3.2.

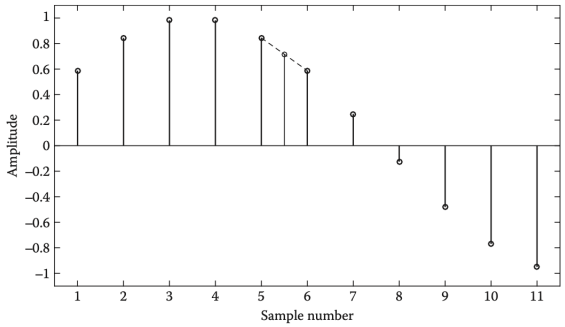


Figure 3.2: Example of an interpolated sample. Figure from [45].

There are, however, other more advanced interpolation methods with *Lagrange* interpolation specifically used in FDN reverberation [9].

In polynomial interpolation, normally referred to as Lagrange interpolation, the problem is to find the unique order sample polynomial which interpolates the samples [41]. An important property of Lagrange interpolation is that its magnitude does not exceed unity (see figure 3.3) which ensures the stability of the feedback network [9]. Lagrange interpolation also has a good fractional delay approximation at lower frequencies [16]. Moreover, an interpolation of order 4 or more can be used for an accurate low-frequency delay modulation [47]. Lagrange interpolation is given by equation 3.2, where $P(x)$ is the polynomial that passes through n samples [51].

$$P(x) = \sum_{j=1}^{n} P_j(x),$$
$$\text{where} \tag{3.2}$$
$$P_j(x) = y_j \prod_{k=1, k\neq j}^{n} \frac{x - x_k}{x_j - x_k}$$

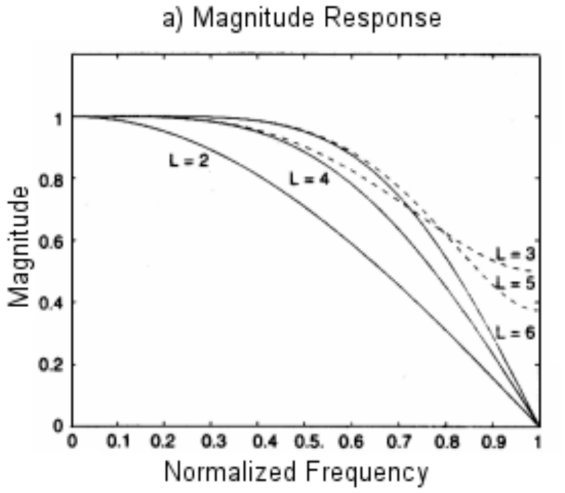


Figure 3.3: Lagrange interpolation where L = filter length. Figure from [9].

3.2.2 Modifiable Parameters

For the product to be usable as a sandbox product for implementation testing and evaluation, the following parameters are modifiable.

1. Number of delay lines
2. Delay line length
3. Delay line modulation
4. Feedback matrix values
5. Input gains
6. Output gains
7. Dry/Wet mix
8. Filter coefficients
9. Transition frequency

It is evident that, with the number of parameters needed to be adjusted and visually represented, the design idea in figure 3.1 is insufficient. Therefore, the design was expanded as seen in figure 3.4.

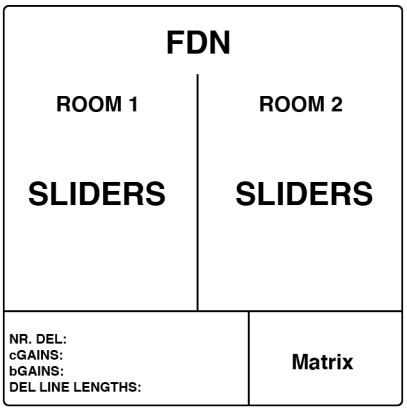


Figure 3.4: Modified Design.

3.3 Final Design

Based on the described requirements, the design was again updated. For more visual information to be visible, the user should be able to expand the information section where the input- and output gains are shown as well as the length of individual delay lines. The user should also have the option of changing the feedback matrix type and seeing its values. Furthermore, the order of the FDN should be modifiable. The matrix size grows proportionally to FDN order, that is N^2, where N is the order. Therefore, for higher-order FDNs, the displayed matrix would not fit in its space. By clicking a 'show matrix' button, the sliders are hidden and the matrix is shown instead. Moreover, this allows for more space for delay lengths, input- and output gains, as well as user data protocol (UDP) (section 4.3) information. Therefore, the design was revised as seen in figure 3.5.

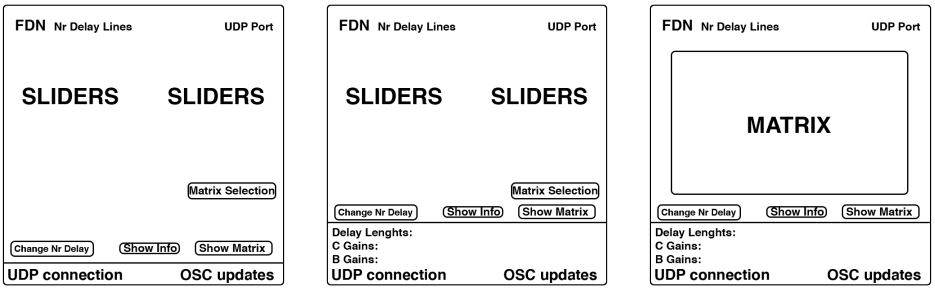


Figure 3.5: Final design. Left: Default view. Right: Information section expanded

Chapter 4

Implementation

In this chapter, the implementation of the plugin using the JUCE framework and a dedicated digital signal processing (DSP) class is presented. The following sections break down the implementation into smaller parts, where the most relevant implementation takes place in the FDN class. Lastly, an overview of a MATLAB[1] script is explained. The source code is available online at `https://github.com/Krummakot/FDN_Thesis`.

4.1 FDN Class

The FDN class is the core of the implementation. It performs the processing of the FDN algorithm and modifies the delay lines, the feedback matrix, and the input- and output gains.

4.1.1 Delay Lines

Delay lines were implemented using JUCE's built-in DSP module, `dsp::DelayLine`. It handles the read- and write-pointer updating, and offers fractional delay calculation with *Thiran*, *Linear*, and *Lagrange3rd* as interpolation options[2]. Although the Lagrange interpolation should be of fourth-order or higher to achieve close to a flat frequency response [9], JUCE's delay interpolation algorithm only supports third order. However, as third-order Lagrange interpolation still gives good approximations for lower frequencies [16] and allows for real-time modulation of the delay[3], the offered third-order Lagrange interpolation has been chosen and considered sufficient in this project.

Since the order of the FDN and, thus, the number of the delay lines should be modifiable, the delay lines are initialised as an `std::vector` which allows for resizing as needed[4]. The interpolation method as well as the sample type of the delay line must be

[1] `https://www.mathworks.com/products/matlab.html`
[2] `https://docs.juce.com/master/namespacedsp_1_1DelayLineInterpolationTypes.html`
[3] `https://docs.juce.com/master/structdsp_1_1DelayLineInterpolationTypes_1_1Lagrange3rd.html`
[4] `https://en.cppreference.com/w/cpp/container/vector`

declared while initialising the vector. The delay lines' maximum delay[5] and the desired delay length in samples must as well be initialised at this moment.

As the delay lines are instances of the DSP class, they must also be prepared with a `dsp::ProcessSpec` structure which contains the program's sample rate, the maximum block size, and the number of channels the DSP algorithm should handle[6].

The delay line length implementation is heavily inspired by Das' implementation[7]. There, the delay lengths are initialised with a lower and upper range (in ms). Through an iterative approach, prime numbers within the range are found, stored in their own array and assigned to the *dspDelayLines* vector. The function used to find the prime numbers for the delay line length estimation, `findNPrime()`, is adapted from Das'[8] work and is left unchanged in this project's implementation[9].

Modulation can be done with a low-frequency oscillator (LFO), in this instance a sinus wave. The `dsp::Oscillator` class is used to create the LFO, where it is then initialised on each delay line. A modulation depth of 6 samples and a frequency rate f of $0.5\text{Hz} < f < 2\text{Hz}$ was chosen [9]. However, both the modulation depth and frequency may vary depending on the use case [36]. Thus, both variables have to be accessible and modifiable by the user. Below, a pseudo code of the implemented modulation is shown. The delay in samples for each delay line must be retrieved before being summed with the output of the LFO. The DSP delay line is then updated with the new value, which may be fractional. Since the delay lines are initialised with the Lagrange interpolation, the DSP delay line handles the processing.

```
delay = getDelayLength()
lfoOut = lfo.process()
newDelay = delay + lfoOut
DSPdelayLine.setDelay(newDelay)
```

4.1.2 Feedback Matrix

The JUCE DSP module includes a class, `dsp::Matrix`[10], to perform matrix operations. As the feedback matrix is square (section 2.2.2.3), it must be initialised at a size of N^2. The `Matrix` class can be constructed either with the number of rows, number of columns, as a data pointer to an array storing its values, or by using the public member functions *identity*, *toeplitz*, *hankel*, and *hadamard*. In the case of this project, the matrix is initialised in two different ways, as an *identity* matrix and as a matrix with a data pointer to an array of values. The reason for this will be further explained

[5] Its default size is 4 samples
[6] https://docs.juce.com/master/structdsp_1_1ProcessSpec.html
[7] https://github.com/orchidas/GFDN
[8] https://github.com/orchidas/GFDN/blob/Grouped_FDN/Source/FDN.cpp
[9] Apart from changing variable names
[10] https://docs.juce.com/master/classdsp_1_1Matrix.html

in section 4.3. The limitation of the `Matrix` class is that mathematical operations can only be done with other `Matrix` instances. Therefore, the delay line input and output vectors must be one-column instances of the `Matrix` class.

4.2 Absorption Filters

To simulate the frequency-dependent decay in a room, a filter is added to the signal chain. In this instance, two first-order shelf filters (section 2.3) are used at the end of each delay line. The filters are cascaded, which means that the signal is processed by an LSF first, before being passed to an HSF. The filter implementation is declared in its own class.

Below, a pseudo-code based on equations 2.15-2.19 can be seen. The filters realise frequency-dependent decay, which is also adjusted for the length of each delay line.

```
gDB = -60/(decayTime*sampleRate)
gLinear = pow(10,(gDB/20))

G = pow(gLinear,delayLength)

wc = cutoff/(sampleRate * 2 * pi)
tc = std::tan(wc/2)

b0 = sqrt(G)*tc + G
b1 = sqrt(G)*tc - G
a0 = sqrt(G)*tc + 1
a1 = sqrt(G)*tc - 1

a0inv = 1/a0
b0 *= a0inv
b1 *= a0inv
a1 *= a0inv
```

Furthermore, an HSF at a fixed frequency of 20.2kHz is used to equalise problematic high frequencies [28].

4.3 UDP - OSC

For real-time updating of variables, the OSC protocol, a real-time message communication among applications and hardware[11] can be used. Designed for low latency, flexible, and accurate communication for musical performance, it has been used extensively in the New Interfaces for Musical Expression (NIME) community. It supports a variety of data types (`int32`, `float32`, `String`, ...) and allows for multiple recipients of

[11] https://opensoundcontrol.stanford.edu/index.html

a single message[12]. The messages can then be transmitted over internet connections and/or between different programs on the same device through a UDP connection.

In relation to this project, it is not necessary to send OSC data but instead, the product needs to receive messages. A special module for receiving and handling OSC messages is a part of JUCE's framework.

4.3.1 Receiving OSC Messages

The OSC implementation in JUCE requires a conditional check before handling the received message. A pseudo-code of such an operation is shown below.

```
function oscMessageReceived(oscMessage) {
  if oscMessage is integer {
    get integer value
    assign integer value
  }
}
```

Each message can consist of multiple arguments of different data types. The *OSC Type Tag String* is used to gather information on the type and order of data being sent. As an example, when sending an OSC String, a float, integer and another OSC String, the *OSC Type Tag String* would state `"sfis"` (where s-string, f-float, i-integer). In order to identify which variables are being sent over OSC for this product, a String is added as the first argument to each message describing which modifications should be done. Figure 4.1 shows an example of how this check is performed.

Such checks are performed for all parameters that are modifiable with OSC. A complete list is shown in table 4.1.

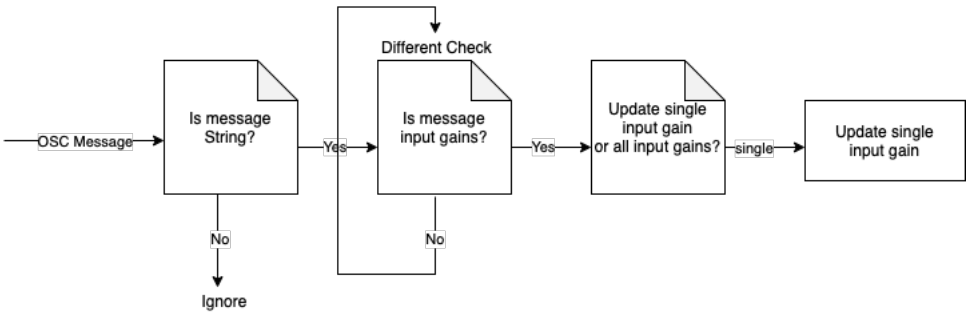


Figure 4.1: Flow of OSC check.

[12] https://opensoundcontrol.stanford.edu/spec-1_0.html

Parameter	Options	String Identifier	Additional
Input gain	Single/Whole	bGainSingle/bGainWhole	*For single: Index specified*
Output Gain	Single/Whole	cGainSingle/cGainWhole	*For single: Index specified*
Feedback Matrix	Single/Whole	matrixSingle/matrixWhole	*For single: Index specified*
Delay Line Length	Single/Whole	delaySingle/delayWhole	*For single: Index specified*
Dry/Wet Mix	Single	dryWet	
T60 LSF	Single	lowT60	
T60 HSF	Single	highT60	
Transitional Freq. LSF	Single	lowTransFreq	
Transitional Freq. HSF	Single	highTransFreq	
Modulation	On/Off	modulation	*Second string = "on"/"off"*
Modulation Depth	Single/Whole	modDepthSingle/modDepthWhole	*For single: Index specified*
Modulation Rate	Single/Whole	modRateSingle/modRateWhole	*For single: Index specified*

Table 4.1: OSC Values and identifiers.

4.3.2 Sending OSC Messages

The OSC messages can be sent to the JUCE plugin through every program that supports UDP and OSC. In this case, a MATLAB script was created for handling the updates with the `oscSend`[13] function. Firstly, a UDP connection has to be opened where an IP address and port number are defined. Here, MATLAB's built-in `udp('host', port` function is used before the connection is opened with the `fopen()` function. In this instance, the *OSC Address* is defined as `'/juce'`, which has to match the address defined in the JUCE plugin. Secondly, through the MATLAB script, it must be ensured that the defined number of delay lines matches the FDN order. New values are assigned to the variables before sending. An example of a feedback matrix initialisation and reshaping into a one-row vector before being sent as a whole is shown below. It is necessary to reshape the matrix as its values are stored as a single-row vector in the JUCE implementation. Additionally, an example of updating a single matrix index by a value of 0.3 is shown and lastly, receiving both of those matrix updates in JUCE.

[13] https://www.mathworks.com/matlabcentral/fileexchange/31400-send-open-sound-control-osc-messages

```
// ==== MATLAB ====
N = 16 % Nr of Delay Lines
% open up the connection
u = udp('127.0.0.1', 6448);
fopen(u);
path = '/juce';

matrix = randomOrthogonal(N); % create matrix
matrix = reshape(matrix,1,[]); % reshape matrix
idx = [0,0]; % specify index

% send matrix as whole
for i = 1:length(matrix)
   oscsend(u, path, 'sf', 'matrixWhole', matrix(i));
end

% send single matrix value
oscsend(u, path, 'siif', 'matrixSingle', idx(1), idx(2), 0.3);
// === JUCE ===
if (message[0].isString()) {
   String messageString = message[0].getString();
   if(messageString.compare("matrixWhole") == 0) {
      if (message[1].isFloat32()) {
         matrixCoefs[count] = message[1].getFloat32();
         count++;
         if (count >= nrDelayLines * nrDelayLines) {
            fdn.updateMatrixCoefficientsOSC(matrixCoefs, "whole");
            count = 0;
         }
      }
   }
   if(messageString.compare("matrixSingle") == 0) {
      if(message[1].isInt32() && message[2].isInt32() &&
         message[3].isFloat32()) {
         int row = message[1].getInt32();
         int col = message[2].getInt32();
         float val = message[3].getFloat32();
         matrixCoefs[row * nrDelayLines + col] = val;
         fdn.updateMatrixCoefficientsOSC(matrixCoefs, "single");
      }
   }
}
```

4.4 Smoothing Variables

An additional step is taken to avoid audible disturbances in the audio. Variables that can be updated in real-time, T_{60}, f_c, etc., are processed using JUCE's `SmoothedValue`[14] class. This class offers linear[15] smoothing between the old value and the target value at a specified rate (in seconds) at which the values are updated. This is used for e.g., the f_c of both the HSF and LSF, decay time, and delay length. When a respective slider value is changed, the target value is assigned to the `SmoothedValue` object and with a conditional statement, the value is incremented towards its target value as long as the current- and target value are not the same. This is shown in a pseudo-code below.

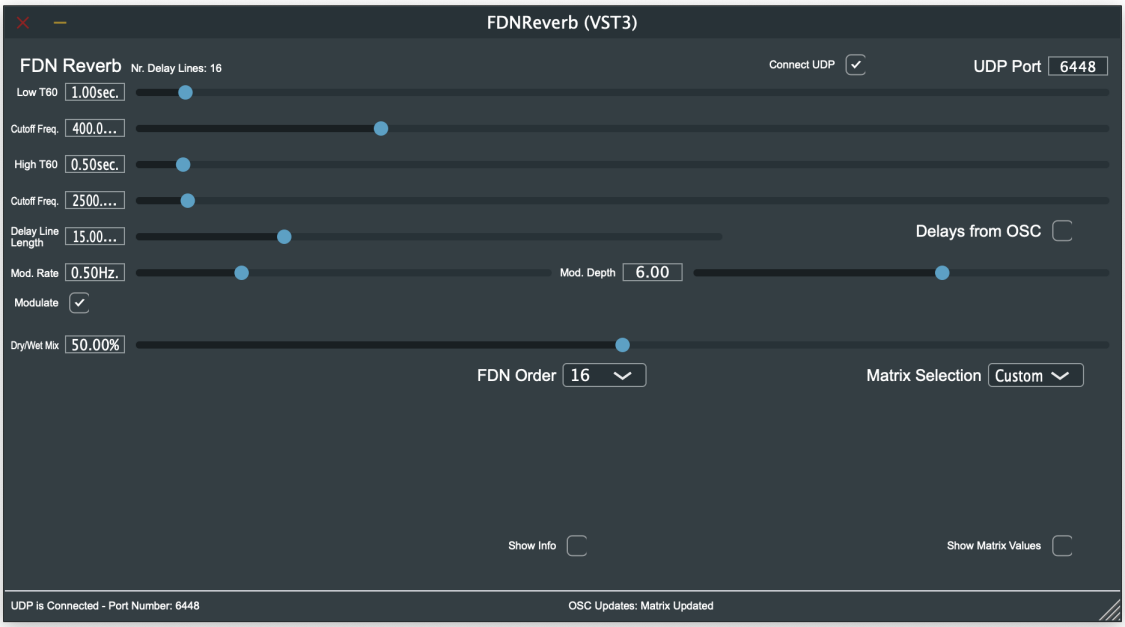
```
*slider value is changed*
smoother.setTargetValue()
while current value is not equal to target
   increment value
```

4.5 Limitations

Although JUCE's dedicated DSP module is a good tool for implementing audio applications and plugins, its limitations were experienced in this project, most notably with its `Matrix` class. Operations with other data types are limited, as an example with matrix multiplications. This may to some extent be countered by storing its data in an Array/vector data pointer with which operations are performed. The matrix also lacks the option of being resized. This can be problematic when modelling coupled room acoustics as the mixing matrix requires the Kronecker product of the reflection matrix [7]. Again, this may be countered with multiple instances of the `Matrix`, however, combined with resizeable FDN orders, this leads to multiple different matrices for each FDN, as well as for the GFDN.

Due to these limitations, the end product is a single FDN with modifiable parameters through OSC, shown in figure 4.2.

[14] https://docs.juce.com/master/classSmoothedValue.html
[15] https://docs.juce.com/master/structValueSmoothingTypes_1_1Linear.html



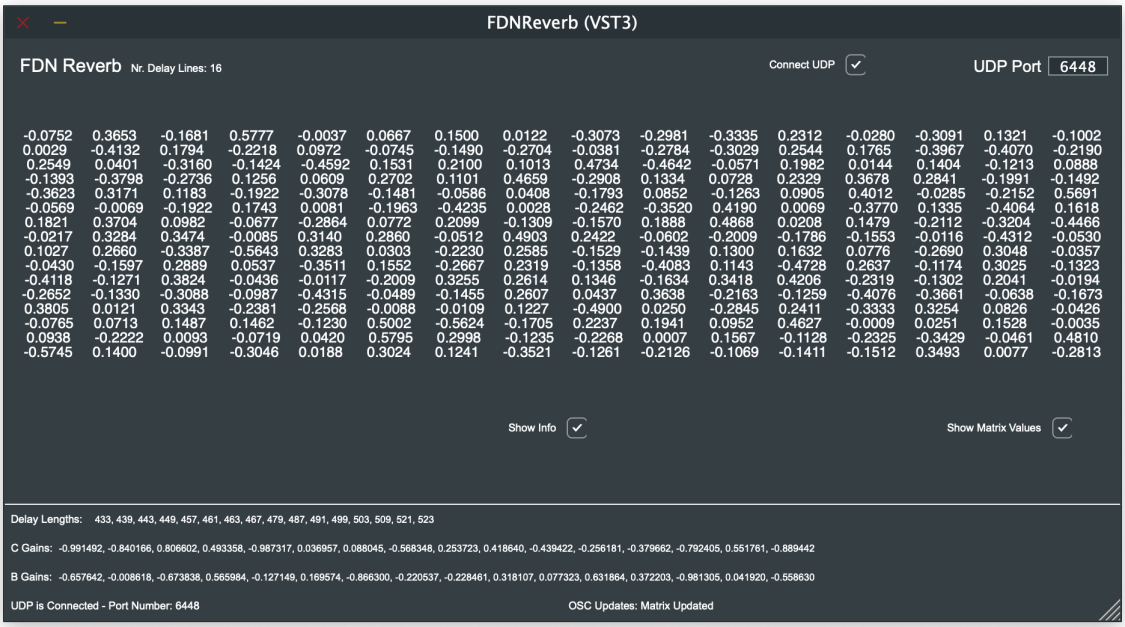


Figure 4.2: The final product. Above is the default state of the plugin. Below, expanded information section and matrix values are displayed.

Chapter 5

Evaluation Methods

The following chapter highlights the evaluation methods possible for this product. It has to be made clear that no evaluation was conducted. Rather, the sections below present how the product could be evaluated with subjective- and/or objective evaluation.

5.1 Objective Evaluation

The real-time accuracy of the FDN can be measured against an offline rendering. This approach can be taken to confirm that the designed FDN is giving the correct results. Here, an FDN can be designed in MATLAB with the help of the FDN Toolbox [35]. By providing the real-time implemented FDN with the same variable values, the two resulting outputs may be compared by looking at output values variable-to-variable. With this approach, filter coefficients, per-sample gain attenuation, or even IRs of the systems can be compared.

Depending on the use case of the product, the rate at which variables need to be updated can vary. If used in VR, even the smallest movements may require the variables to be updated. Humans usually do not perceive auditory latency until around 15-30 ms, although for trained musicians it can go as low as 5-10 ms [52]. Therefore, in an ideal situation, the latency of updating variables should be close to the lower limit.

This may be assessed by stress-testing the product by constantly updating variables to achieve the update rate that the algorithm cannot sustain. This test may be applied to both the rate of updates and the number of simultaneously updated variables. Related to this is the FDNs overall CPU usage. In [29], the CPU usage of a graphical FDN plugin was measured for different states, where increasing the FDN's order increased CPU usage exponentially. Additionally, fixing the IR and filter coefficient calculations greatly decreased CPU usage. Therefore, the CPU usage, both in relation to the FDN order, but also with and without modulation can be evaluated.

5.2 Subjective Evaluation

To evaluate different perceptual aspects of the implemented reverb, listening tests have become an essential tool to evaluate perceptual aspects such as sound quality [23]. Below, two examples are presented and proposed for subjective listening tests for this product.

Modulating the delay line length or the feedback matrix values can be seen to enhance the quality rating of an FDN. In [36], the MUSHRA method, where several sound treatments are presented to the participant and must be rated on a scale from 0 to 100 in comparison to a reference sound [20], was used. Three test conditions were compared through a listening test, an 8th-order FDN with modulation, an 8th-order FDN without modulation, and a 16th-order FDN without modulation. The participants were presented with four different test items, a solo piano, a snare drum roll, a french speaker, and a castagnette rhythm and then asked to rate each condition on a 100 point scale ranging from 'bad' to 'excellent' [36]. The results show that an 8th-order FDN with modulation was rated higher than an 8th-order FDN without modulation in all conditions. Furthermore, improved realistic liveliness of the reverberation was reported for the modulated FDN.

A different approach was taken in [9], where an 8th-order FDN with 4 modulated delay lines was compared with a 12th-order FDN with no modulation. The evaluation method was an ABX test, in which participants are asked to identify whether a randomly chosen audio sample 'X' is one of the known samples: 'A' or 'B' [5]. Firstly, the participants were presented to four different test items, an acoustic guitar, a solo male vocal, arpeggiating alto saxophone, and a full drum set playing a rock beat. They were then asked to state if they could hear a difference between the two algorithms, and those who were able were asked to state which they preferred. In three out of four cases, the participants preferred the FDN with modulated delay lines, and only for the drum recording was the non-modulated FDN preferred.

This type of ABX test of modulated vs. non-modulated FDN can, therefore, be used for the evaluation of this product.

In [27], a simulated auralization of the Notre Dame was compared to its measured counterpart. The participants were presented with three different arualization configurations and were asked to rate the similarity between the two played samples in each configuration. Below, some of the relevant terms used for rating are presented and described shortly [26].

- *Reverberance:* The perception of the sound decay.
- *Clarity:* To which degree discrete elements stand apart from each other.
- *Distance:* The perceived distance to the sound source.
- *Tonal Balance:* Changes in timbre or frequency balance. More tonal balance indicates more high-frequency content.
- *Coloration:* Modification in the sound's timbre from its original timbre. Less coloration indicates more natural sounding recording.
- *Plausibility:* How reasonable the recording sounds.

The participants were then presented with a 100 point graphic slider, with each extreme stating either "A is much more..." or "B is much more...", where the centre point (0) indicated no perceived difference [26].

By designing and matching the FDN with a recording of a real space, the FDN can be evaluated in relation to these terms.

Chapter 6

Conclusion

Based on research on artificial reverberation with FDN algorithms, a real-time implementation of an FDN plugin allowing for variable updates at run-time was developed with the JUCE framework. The goal of this project was to develop a product that can be used for evaluating and comparing offline FDN algorithms with their real-time counterpart. Through the UDP and OSC protocols, variables can be defined in a MATLAB (or other software) script and sent over to the plugin without the need to modify its source code. The initial problem statement was to simulate coupled room acoustics with a GFDN architecture, however, due to technical limitations, the end product implements a single FDN.

The product allows for controllable delay line lengths, FDN order, input and output gains, T_{60}, feedback matrix, as well as modulation of delay lines. Real-time requirements were considered to avoid audible disturbances when updating variables, such as Lagrange interpolation for the delay lines, linear interpolation for T_{60} modification, and multiplicative interpolation for f_c. Furthermore, to reduce the CPU usage, calculations for, e.g., filter coefficients, were only performed once the respective slider's value changed.

Chapter 7

Further Work

It is clear that the implementation of coupled room acoustics with the GFDN architecture was not completed. Although this is due to technical limitations, this should be achievable with JUCE's DSP module and native c++ code. The most noticeable limitation is present with `Matrix` class' lack of operations with other data types. Even though this may be countered with an `std::vector` data pointer, the DSP matrices are not resizable which makes Kronecker product of the reflection matrix (section 2.2.3) challenging. Although this may be countered with multiple instances of matrices depending on FDN order, a more efficient approach should be investigated.

Furthermore, the lack of evaluation is a serious limitation of this product. Although an FDN was successfully implemented, it can not be used as a research tool at this time as its accuracy has not yet been verified. Different subjective and objective evaluation methods are proposed and they may be adapted to the GFDN once implemented.

Acknowledgements

Special thanks go to Karolina Prawda for being a great help with implementation and for guidance with the thesis, Sebastian J. Schlecht for the project proposal and motivation, Silvin Willemsen for guidance and helpfulness, and Stefania Serafin for all the help with the project.

Chapter 8

Bibliography

[1] Benoit Alary, Archontis Politis, Sebastian Schlecht, and Vesa Välimäki. Directional feedback delay network. *Journal of the Audio Engineering Society*, 67(10):752–762, 2019.

[2] Kevin Arcas and Antoine Chaigne. On the quality of plate reverberation. *Applied Acoustics*, 71(2):147–156, 2010.

[3] Leo Leroy Beranek and Tim Mellow. *Acoustics: sound fields and transducers*. Academic Press, 2012.

[4] Barry A Blesser. An interdisciplinary synthesis of reverberation viewpoints. *Journal of the Audio Engineering Society*, 49(10):867–903, 2001.

[5] Jon Boley and Michael Lester. Statistical analysis of abx results using signal detection theory. In *Audio Engineering Society Convention 127*. Audio Engineering Society, 2009.

[6] David T Bradley and Lily M Wang. Room acoustics in coupled volume spaces. In *Building Integration Solutions*, pages 1–6. 2006.

[7] Orchisama Das and Jonathan S Abel. Grouped feedback delay networks for modeling of coupled spaces. *Journal of the Audio Engineering Society*, 69(7/8):486–496, 2021.

[8] Michael Francis. Infinite impulse response filter structures in xilinx fpgas. *Xilinx White Paper*, 2009.

[9] Jasmin Frenette. Reducing artificial reverberation algorithm requirements using time-variant feedback delay networks. *Florida: University fo MIAMI*, 2000.

[10] Anders Gade. Acoustics in halls for speech and music. In *Springer Handbook of Acoustics*, Springer Handbooks, pages 301–350. Springer New York, New York, NY.

[11] Michael A Gerzon. Synthetic stereo reverberation: Part one. *Studio Sound*, 13:632–635, 1971.

[12] David Griesinger. Improving room acoustics through time-variant synthetic reverberation. In *Audio Engineering Society Convention 90*. Audio Engineering Society, 1991.

[13] Janis Heldmann and Sebastian J Schlecht. The role of modal excitation in colorless reverberation. 2021.

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

[15] Jean-Marc Jot et al. *Etude et réalisation d'un spatialisateur de sons par modèles physiques et perceptifs*. PhD thesis, 1992.

[16] T.I Laakso, V Valimaki, M Karjalainen, and U.K Laine. Splitting the unit delay [fir/all pass filters design]. *IEEE signal processing magazine*, 13(1):30–60, 1996.

[17] Hammond Laurens. Electrical musical instrument, February 4 1941. US Patent 2,230,836.

[18] Thomas McKenzie, Sebastian J Schlecht, and Ville Pulkki. Acoustic analysis and dataset of transitions between coupled rooms. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 481–485. IEEE, 2021.

[19] Thomas McKenzie, Sebastian J Schlecht, and Ville Pulkki. Auralisation of the transition between coupled rooms. In *2021 Immersive and 3D Audio: from Architecture to Automotive (I3DA)*, pages 1–9. IEEE, 2021.

[20] Catarina Mendonça and Symeon Delikaris-Manias. Statistical tests with mushra data. In *Audio Engineering Society Convention 144*. Audio Engineering Society, 2018.

[21] James A Moorer. About this reverberation business. *Computer music journal*, pages 13–28, 1979.

[22] Sophocles J Orfanidis. *Introduction to signal processing*. Pearson Education, Inc, 2016.

[23] Etienne Parizet, Nacer Hamzaoui, and Guillaume Sabatie. Comparison of some listening test methods: a case study. *Acta Acustica united with Acustica*, 91(2):356–364, 2005.

[24] Tae Hong Park. *Introduction to digital signal processing: computer musically speaking*. World Scientific Publishing Co. Pte. Ltd, 2010.

[25] Julian Parker and Stefan Bilbao. Spring reverberation: A physical perspective. In *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx'09)*, pages 416–421, 2009.

[26] Barteld NJ Postma and Brian FG Katz. Perceptive and objective evaluation of calibrated room acoustic simulation auralizations. *The Journal of the Acoustical Society of America*, 140(6):4326–4337, 2016.

[27] Barteld NJ Postma, David Poirier-Quinot, Julie Meyer, and Brian FG Katz. Virtual reality performance auralization in a calibrated model of notre-dame cathedral. *EuroRegio2016, Porto, Portugal*, 2016.

[28] Karolina Prawda, Sebastian J Schlecht, and Vesa Välimäki. Improved reverberation time control for feedback delay networks. In *Proc. Int. Conf. Digit. Audio Effects*, pages 1–7, 2019.

[29] 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*, pages 16–23, 2020.

[30] Michael Rettinger. Reverberation chambers for broadcasting and recording studios. *Journal of the Audio Engineering Society*, 5(1):18–22, 1957.

[31] Davide Rocchesso and Julius O Smith. Circulant and elliptic feedback delay networks for artificial reverberation. *IEEE Transactions on Speech and Audio Processing*, 5(1):51–63, 1997.

[32] Henry Joseph Round and West Arthur Gilbert Dixon. Transmission and reproduction of sound. *U.S. Patent 1,853,286*, Apr 1932.

[33] Francis. Rumsey. Spatial audio, 2001.

[34] Sebastian Schlecht and Emanuël AP Habets. Reverberation enhancement systems with time-varying mixing matrices. In *Audio Engineering Society Conference: 59th International Conference: Sound Reinforcement Engineering and Technology*. Audio Engineering Society, 2015.

[35] Sebastian J Schlecht et al. Fdntb: The feedback delay network toolbox. In *Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx-20)*, 2020.

[36] Sebastian J Schlecht and Emanuël AP Habets. Time-varying feedback matrices in feedback delay networks and their application in artificial reverberation. *The Journal of the Acoustical Society of America*, 138(3):1389–1398, 2015.

[37] Sebastian J Schlecht and Emanuël AP Habets. Modal decomposition of feedback delay networks. *IEEE Transactions on Signal Processing*, 67(20):5340–5351, 2019.

[38] Manfred R Schroeder. Natural sounding artificial reverberation. In *Audio Engineering Society Convention 13*. Audio Engineering Society, 1961.

[39] Manfred R Schroeder. Digital simulation of sound transmission in reverberant spaces. *The Journal of the acoustical society of america*, 47(2A):424–431, 1970.

[40] Manfred R Schroeder and Benjamin F Logan. "colorless" artificial reverberation. *IRE Transactions on Audio*, (6):209–214, 1961.

[41] Julius O. Smith. History of fdns for artificial reverberation, 2010.

[42] Julius O. Smith. *Physical Audio Signal Processing*. http://ccrma.stanford.edu/j̃os/pasp/, 2021-11-24. Online book, 2010 edition.

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

[44] John Stautner and Miller Puckette. Designing multi-channel reverberators. *Computer Music Journal*, 6(1):52–65, 1982.

[45] Eric Tarr. *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB*. Routledge, 2018.

[46] V Valimaki, JD Parker, JO Smith, and JS Abel. More than fifty years of artificial reverberation. In *AES 60th Conference on Dereverberation and Reverberation of Audio, Music, and Speech*, pages 1–12, 2015.

[47] Vesa Valimaki, Julian D Parker, Lauri Savioja, Julius O Smith, and Jonathan S Abel. Fifty years of artificial reverberation. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(5):1421–1448, 2012.

[48] Vesa Välimäki and Joshua D. Reiss. All about audio equalization: Solutions and frontiers. *Applied Sciences*, 6(5), 2016.

[49] Eric W Weisstein. Hadamard matrix. *https://mathworld.wolfram.com/*, 2002.

[50] Eric W Weisstein. Orthogonal matrix. *https://mathworld.wolfram.com/*, 2002.

[51] Eric W Weisstein. Lagrange interpolating polynomial. *https://mathworld. wolfram. com/*, 2004.

[52] Steven Yantis and Richard A Abrams. *Sensation and perception*. Worth Publishers New York (NY), 2014.