Developing TheStringPhone

David Stubbe Teglbjærg and Jesper Skovgaard Andersen

June 1, 2015

Abstract

This thesis describes the development of **TheStringPhone**, a physical modeling based polyphonic digital musical instrument that uses the human voice as excitation. The core parts of the instrument has been implemented by the aurthors from scratch in C++ and is thoroughly explained in the thesis. This includes a Biquad Filter, FIFO delay, circular buffer delay, the Karplus-Strong algorithm, Peak and RMS Compressor and a FDN Reverb. The focus has primarily been on the sound generating part of the instrument and **TheStringPhone** is only evaluated from an engineering perspective.

Contents

0.1 Introduction

> "there are no theoretical limitations to the performance of the computer as a source of musical sounds, in contrast to the performance of ordinary instruments." [49]
>
> - Max Matthews (1963)

The decoupling of input device and sound engine in Digital Music Instruments (DMI) present an expansive potential that empowers musicians with novel sonic resources, beyond all the limits of acoustical instruments [65]. However, by giving developers the power of designing the causal relationship between input and output, a lot of the feeling and natural elements - that are the strength of the acoustic instrument - are easily missed [65][56][33]. If one only focuses on a specific part of the interaction, say the control organ, one might loose aspects crucial for the overall usage. This is a general problem when constructing DMIs. For many years the prevailing research focused on improving the sonic capabilities of the synthesis algorithms [19] while interfacing has had a minor emphasis. In the 21st century times changed and motivated a wide spectrum of works on New Interfaces for Musical Expression (NIME). The design of musical interfaces gained a significant importance and concepts from a broad range of fields - human-computer interaction (HCI), electrical engineering, computer vision, digital signal processing, machine learning and artificial intelligence - where integrated [33]. However, with this new trend a segregation has occurred and as Jorda puts it:

> "NIME people focus on the input-playing side while acousticians and synthesists concentrate on the output-sounding one" [...] "the fact is that very few attempts are being made at studying the design of new musical instruments -tools for playing and making music- as a conceptual whole." [33]

Furthermore Jorda points out that the term *"musical instruments"* is often being employed instead of either input devices or sound generators. If one utilizes a non-holistic approach to instrument development, one must accept a high risk that the outcome might be too generic or non-specific ending up being either too simple, mimetic or technology based. On the other hand, a holistic approach can quickly become too cumbersome or complex which also leads to inappropriate design choices.

The focus of our previous projects have been on designing the input-playing side of DMIs. The approach have been to take a given sound engine and then figure out how to design a more expressive way of controlling it. Now, in order to get a better understanding of the design of DMIs we feel a need for a better understanding of the design process from the perspective of

the acousticians and synthesists. In this way we will be in a better position to approach the development of DMIs from a holistic point of view. Based on this motivational ground we have spent our master thesis on developing **TheStringPhone**, a physical modeling based polyphonic digital musical instrument that uses the human voice as excitation. The core parts of the instrument has been implemented from scratch in C++ and the focus has primarily been on the sound generating part of the instrument.

0.2 Thesis outline

This thesis will explain the research and development of **TheStringPhone** and is organized in 3 main chapters, followed by a conclusion and future perspectives. A project folder containing a version of **TheStringPhone** together with the source code and other relevant resources has been created and can be downloaded from:

https://thestringphone.wordpress.com/resources

Throughout the thesis we will refer to audio examples that can all be found in the project folder.

- **Chapter 1** starts out by presenting the scientific background within DMI's. Since **TheStringPhone** uses the voice as excitation, a survey of existing musical systems based on voice driven synthesis and control will be presented.
- **Chapter 2** will explain the design choices made and give an overview of the **TheStringPhone**.
- **Chapter 3** explains the implementation of each individual part of **TheStringPhone** in detail. Each part will include both the theoretical background as well as detailed descriptions of how each part has been implemented in C++.

Chapter 1

Scientific Background

1.1 Digital musical instruments

An acoustic instrument consist of an excitation source and a resonating system, that couples the vibrations from the excitator to the surrounding atmosphere. In acoustic instruments excitation source and resonator are chained together and a musician playing the instrument must therefore keep supplying energy for both controlling and producing the sound. There are however few exceptions, such as organs, where the control organ and the sound generator are separated.

In contrast to acoustic instruments, DMIs can always be divided into an input device and a sound generator thereby splitting the instrument chain [15]. Figure 1.1 shows Wanderley and Depalle's [82] representation of a digital musical instrument. This model provides a good insight into the various components of a DMI and how they interact.

Gestures refer to the movement of the player or user. The *gestural controller* refers to the gestural acquisition system and can be everything from a simple slider to a motion capturing system. The gestural controller typically defines the possible gestures, creating constraints and thereby guiding the user in how to use the instrument. The gestural controller gives feedback depending on the action taken by the user. The *sound production* system refers the sound engine or synthesis model. Mapping defines how parameters of the gestural controller drives parameters of the sound engine. This could be a slider controlling the amplitude of the sound engine or a rate of a waving motion controlling pitch etc. Secondary feedback refers to the change in sound caused by the gestural action of the player [82][37][33].

The division of input device and sound generator, frees constructors from the natural constraints found in physical instruments. For example some instruments require more strength to play than others e.g. bass and baritone saxophones. Certain acoustic instruments can be hard to play softly, while others are difficult to play loudly. Extraordinary effort may be needed in

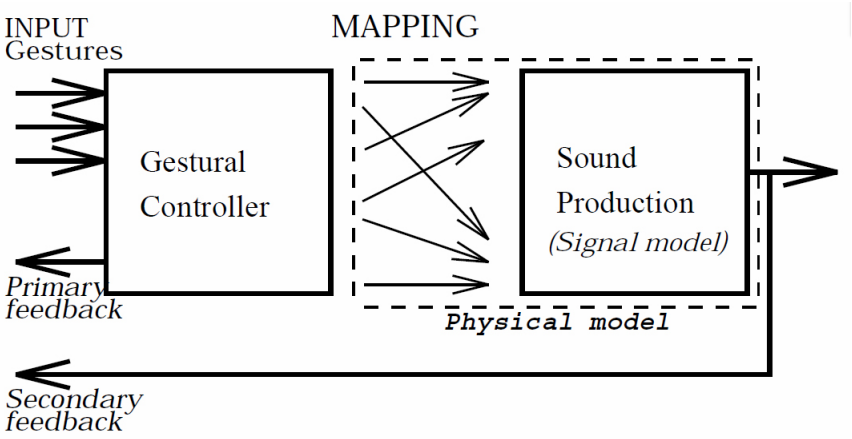

Figure 1.1: A representation of a digital instrument. (Illustration taken from [82])

some instruments to play extremely high or low notes and tuning the instrument may be tedious. An instruments timbre is also predetermined by its physical construction. These limitations can all be overcome by DMIs thereby creating musical flexibility. With DMIs a single controller can suddenly create low bass sounds as easily as high pitched sounds, all with minimum effort. Tuning can be done with the push of a button and timbre has no physical limitations. Being able to transcend the physical can allow for adding new expressiveness to already familiar timbre and is inviting from a compositional standpoint.

This however all comes with a price, namely the reduction of the feel associated with producing a certain kind of sound [65]. Magnusson [46] argues that the decoupling of sound engine and energy input makes the mappings between them arbitrary and thereby symbolic. This therefore defines the digital instrument as a medium for hermeneutic relations whereas the acoustic instrument is to be seen as a channel for embodiment relation since we use our bodies to understand and learn acoustic instruments. Digital instruments, on the other hand, also require an understanding of the symbolic mapping between input and output. Although a digital instrument can be designed to be controlled by physical input, the mapping of sensors to sonic output is still symbolic and one must therefore not only focus on learning the motor skills, but also the symbolic mapping between input and output. As Magnusson explains:

> "To work with symbolic tools means that one has to continually switch modes from focusing on the world to focusing on the tool with regular intervals and to a more pronounced degree than in acoustic instruments. This detachment from the terminus of our activities could be paraphrased as a disruption in flow and is

present in the majority of existing digital music systems." [46, p. 173]

The gestural controller and the sound engine are two separate independent systems and their relation relies on *mapping* of gestural parameters to sound production parameters. However, the choice of mapping depends very much on the choice of gestural controller and the choice of sound engine influences what kind of gestures work most natural. This interdependence between the components make the design of a digital musical instrument a very complex process, and although a holistic approach may be best, it can also be cumbersome. On the contrary, as mentioned in the motivation, developing gestural controller and sound engine in conjunction, will most likely yield the best results [37].

1.1.1 Gestures and the gestural controller

Gestural control of computer generated music can be seen as a specialized branch of human-computer interaction and is a field that involves the combined study of simultaneous control of multiple parameters, timing, rhythm and training [81]. In the context of musical devices a gesture refers to performer actions produced by the instrumentalist and a gestural controller is the input part of a digital musical instrument where physical interaction with the player takes place. The most well known gestural controller must be the MIDI keyboard, which only provides users with two parameters i.e. velocity and pitch. Additionally most keyboards have modulation and or pitch bend wheels (some also aftertouch), but they all imply making separate actions from the traditional keyboard gesture. Controlling all sounds (percussive as well as non percussive) with a keyboard controller is obviously not optimum and faces inherent limitations [31]. Pressing [61], Roads [65] and Paradiso [58] all explore the myriad of gestural controllers, important parameters for controlling different types of instruments and their effect on music playing. The field is in a state of rapid progress and the interested reader is referred to the recent NIME proceedings [1] for a complete list of new developments in the field.

In the study of gestures there are different levels of detail that can be analyzed. Some gestures are directly related to the production of sound (functional gestures) while others may not be clearly connected to sound production (physiological gestures) e.g. gestures present during professional performance. When designing gestural acquisition systems for sound production it is obviously the functional gestural characteristics that needs to be identified.

Gestures need to be captured by an acquisition system for further use in the system. Typically acquisition is performed in three different ways:

[1] http://www.nime.org

- **Direct acquisition**, where sensors are used to monitor performer actions. Usually each physical variable of the gesture is captured by a separate sensor and the signals present isolated physical features, such as pressure, linear or angular displacement, acceleration, etc.

- **Indirect acquisition** concentrates on gestures isolated from structural properties of the produced sound. By analyzing the fundamental frequency or spectral envelope of the sound performer actions can be derived. Such methods is usually constituted by signal processing.

- **Physiological signal acquisition** is the analysis of physiological signals, such as muscle signals (EMG) or brainwave signals (EEG). Several systems have implemented this type of acquisition [75][76], but such techniques are usually hard to master since it may be difficult to separate meaningful parts of the signal.

In general, direct acquisition has the advantage of simplicity compared to indirect acquisition, however direct acquisition may sometimes underestimate the interdependency of variables.

When looking at sensor characteristics sensitivity, stability and repeatability are often considered to be the most important [81]. In a musical context the choice of sensor obviously needs to match the task at hand. A sensor output that is precise, but not very accurate may be satisfactory when mapped to loudness, but not if it is used to control pitch. In this case the inaccuracy will probably be more noticeable.

When using sensors for gesture acquisition the signals obtained are usually analog in the form of voltage signals. Such signals need to be converted in order to use them as input for the computer. This is done by sampling the voltage signal and converting each sample into a suitable format, usually MIDI.

When several sensors have been assembled as part of a device **a gestural controller** has been born. When classifying such controllers there is typically distinguished between:

1. instrument-like controllers.

2. extended controllers.

3. alternate controllers.

A huge part of commercial controllers is MIDI controller versions of traditional instruments and thereby fit the first category. An advantage of using traditional instruments as a model is adaptability. A musician who is already familiar with the instrument mimicked can immediately apply highly developed performance skills. One might fear that fewer virtuoso musicians will come out of new devices, since there seems to be a tendency

towards a tradition-bound music education [65]. On the other hand traditional instruments might very well be limited when it comes to taking full advantage of a specific synthesis method and the control capabilities are mostly reduced when compared to their acoustic ancestors [33]. As a counterpart instrument-like controllers offer an expanded sound palette.

The second category includes traditional instruments with extra sensors as add-ons that afford additional control possibilities.

Alternate controllers include everything that is not based on or is an extension of an instrument. Where adaptability is the advantage of instrument-like controllers this category has the advantage of being tailored to fit a specific synthesis method, thereby exploiting its unique capabilities. As Collins puts it:

> "the possibilities inherent in digital synthesis, processing, and playback call for new modes of control that require special input devices and interaction styles."[10]

The feedback given by the gestural input is another important aspect and is traditionally divided into **primary feedback** and **secondary feedback**.

As explained, primary feedback refers to the feedback given by the gestural controller and includes visual, auditory, and tactile-kinesthetic feedback. This could for example be the tactile feedback given when hitting the string on a guitar or the visual feedback given when choosing oscillator on a synthesizer. Most instruments include tactile-kinesthetic feedback of various types and instruments without, such as the Theremin, can be extremely hard to master. Primary feedback usually serves the purpose of guiding the player when playing the instrument and can also be used to ease the process of learning the instrument. For example, it takes a shorter time to reach a moderate level of playing a guitar compared to a violin, partly because of the tactile (and visual) feedback given by the frets on the guitar neck.

Secondary feedback focuses on the actual change in sound. The crucial aspect of instrument design is to create a natural relationship between gesture and output. This leads us to the area of mapping.

1.1.2 Mapping

Hunt and Wanderley define mapping as:

> "[...] the act of taking real-time performance data from an input device and using it to control the parameters of a synthesis engine." [27]

In other words, as Jorda writes in [33], mapping is

> "[...] the connection between gestural parameters (input) and sound control parameters or audible results (output)." [33]

So, mapping is simply the bridge between input device and synthesis engine. As soon as you decouple input device and synthesis engine, the mapping scheme becomes a crucial factor in determining the behaviour and nature of the instrument [27][28][47][33].

There are overall two ways of implementing mapping:

- By use of generative mechanisms, such as neural networks to perform mapping.
- By use of explicitly defined mapping strategies.

The first provides a mapping strategy by means of internal adaptations of the system and the selection of most important features among the set of signals, whereas the later is developed explicitly by the instrument designer. Generative mapping use machine learning techniques to derive mapping strategies and can be applied to user-adapted systems, where the system is able to adapt to user behaviour based on user input examples.

Explicit mapping defines a fixed relationship between input parameters and synthesis parameters and is usually divided into four categories:

- **One-to-one**, where one input parameter is controlling one synthesis parameter.
- **One-to-many**, where one input parameter may influence several synthesis parameters at the same.
- **Many-to-one**, where one synthesis parameter is driven by two or more input parameters.
- **Many-to-many**, where many input parameters may influence several synthesis parameters.

In [27] Hunt conducted a series of experiments focusing on how various types of mapping influence the accuracy and quality of a player's performance. He focused not only on the difference between the various types of mapping, but also on the difference between the user having to continuously supplying energy in order to change parameters. This could for example be mapping the rate of change of a slider to the pitch of the synthesis model. In one of his experiments, he created interfaces implementing three mapping strategies: (1) simple one-to-one connections between input and output, (2) one-to-one requiring the user's energy as an input and (3) many-to-many connections from input to output also requiring the user's energy as input. In regards to (3) he concludes that:

> "at first it seemed counter-intuitive to most users, but they rapidly warmed to the fact that they could use complex gestural motions to control several simultaneous parameters without

> having to 'de-code' them into individual streams. Many users remarked how 'like an instrument' it was, or 'how expressive' they felt they could be with it." [27]

This observation correlates well with Serafin and Gelineck's proposed guideline in [25] about "making the user work". This is due to the fact that mapping in acoustical instruments are often slightly non-linear and seldom one-to-one mappings [33] and mimicking this summons the same feeling as when playing a traditional instrument.

Many thoughts has been presented on how to design for and evaluate digital musical instruments and controllers of all kinds [7][25][24][11][12][56][33][65].

1.2 Interaction models

Interaction with DMIs has in many cases been described as a traditional HCI communication loop, which regards the user as someone acting on a system, the system then senses what the user is doing and produces feedback, which is perceived by the user, who then acts accordingly [24]. In addition to the traditional communication loop Paine [57] includes the audience in his interaction model. He focus on issues related to the development of authentic performance interfaces that provide sufficiently convincing gestural control during performance.

Johnston et al. [32] developed an interaction strategy which uses on-screen objects that respond to user actions thereby identifying three modes of interaction i.e. *instrumental*, *ornamental* and *conversational*. The biggest factor differentiating the three modes of interaction is the issue of control. In the instrument mode the musician is aiming for complete control, in the ornamental mode the musician surrenders control and in the conversational mode sharing of control is involved - seen as a continuous shift in the balance of power between the instrument and the musician. Their study conclude that the conversational mode is the most interesting, but also the most difficult mode to design for. In order to support this mode the musician needs to have full instrument controllability while simultaneously being able to introduce new, and occasionally surprising, musical material.

Likewise Chadabe [9] regards the interactive relationship between a musician and his instrument as conversational. He sees an interactive instrument as a dynamic system that has its own identity e.g. make its own decisions and produce unpredictable and distinctive output. For Chadabe Interactive means mutually influential.

Drummond [18] argues that interactive systems have the potential for variation and unpredictability and can be seen as a composition or structured improvisation of an instrument. Such systems blur the lines of the traditional distinction between composing, instrument building, systems design and performance. This means that the performer can influence, affect and

alter the underlying compositional structure and the instrument can take on performer like qualities, thereby making the evolution of the instrument into the composition. During the development of such interactive works the composer may become instrument designer, programmer, performer, etc. all at the same with the result that:

> "the instrument is the music and the composer is the performer."
> [8]

It is a process in which the computer influences the performer as much as the performer influences the computer.

1.3 The voice as a gestural controller

Most of the music controllers available today is based on tactile, haptic or gestural interaction. This sets a physical limit that restricts the control of processing parameters. Multiple interfaces would not solve the problem since the musician is still limited in bandwidth as we only have two hands. The human voice is however a spare bandwidth that is usually not exploited in order to gain additional synthesis control, even though the voice is our most proficient and prolific means of communication [19]. Why not use it as an additional control palette?

1.3.1 A survey of voice driven DMIs

During the years there has been proposed several musical controllers driven by the human voice. The probably most well known is the *vocoder* invented in 1928 (under the name *voder*). Originally the vocoder were used in telecommunications where it was used as an analysis/synthesis system for speech coding, but in the 1960's people started using it for musical applications. Basically the *vocoder* generate synthetic sound, which is driven by the human voice. A source of musical sounds is used as the *carrier* and the examined speech signal is then used as a *modulator*. During the analysis stage the speech signal is split into a number of frequency bands and it is the spectral envelope of these bands that are used as a modulation source. An extension to *The Vocoder* is the *Phase Vocoder* [20], which is based on the Short-Time Fourier Transform (STFT) and allows for many manipulations in the spectral domain [17], such as high fidelity time-scaling and pitch transposition. One application of the phase vocoder is cross synthesis, which is a technique that is based on the fact that we can multiply the magnitude spectrum of two analyzed sounds point by point [31] resulting in a source sound (e.g. the voice) controlling another sound (e.g. a car).

Another application is voice-to-MIDI interfaces, such as digitalEar [2] and

[2] http://www.digital-ear.com/digital-ear/index.asp

MIDI-voicer[3], but they seem really restricted when it comes to accessing interesting spectral features of the signal input. In the following we will therefore introduce some generic and more extended work that has been done. Most of the work presented originates from the academic world. Also, this overview is not intended to be an exhaustive walk-through of all existing voice controlled musical applications, but more a selection of diverse applications that illustrate different perspectives on how to integrate the human voice as a musical controller.

The first work to present a more extended idea of mapping was *The Singing Tree* [53]. It was developed at MIT as a part their *Brain Opera* installation and is a novel interactive musical interface that can respond to vocal input with real time aural and visual input feedback. The Singing Voice present a more extended mapping scheme where 10 different dynamic parameters are extracted and mapped to an ensemble of different MIDI instruments, each using a different sound source for re-synthesis. Thereby each contribute to the overall character of the tree. Features used in this project includes pitch, loudness, formants, cepstra and is for the most part mapped to multiple parameters using probability and randomness.

Auracle [63] is an interactive environment, which is based on multiple users simultaneous participation. It is a group based instrument that analyzes vocal input in order to control a synthesizer. The system works as a distributed music making platform over the internet. The goal of *Auricle* is to engage a broad public (i.e. non-specialized audience) in 'playing' with sound.

In 2006 Yamaha released their *easy trumpet* (EZ-TP) system, which allows the user to synthesize sound either by singing or blowing into the trumpet. The easy trumpet might also be used as a MIDI controller. Since this product is commercial, it has not been possible to find any information about the audio processing algorithms used.

The *Larynxophone* [43] use extracted voice features to drive a real-time cross-synthesis engine (consisting of a wind instrument database and the input voice signal). Pitch onset is used to query samples from the database, while excitation gain is mapped to velocity, slope is mapped to modulation and depth is mapped to aftertouch.

In [42] Loscos and Aussenac present *The Wahwactor*, which is a wah-wah effect controlled by the users voice. A wah-wah pedal consist of a resonant bandpass filter with a variable center frequency f_c that is changed by moving a pedal back and forth with the foot. Instead of using a foot *The Wahwactor* is instead controlled by the users voice, using [wa-wa] utterances. It is explained that the wah-wah effect resembles a [wa-wa] utterance because the voice spectra characteristics of the phonemes [u] and [a] produce a modulated sound that is perceived as the effect of a resonant filter moving upwards in

[3]http://expressor-midi-voicer.download-382-34020.datapicks.com

frequency [42]. They present and evaluate five different voice descriptors as a candidate for *The Wahwactor* and concludes that LPC and the Low-band Spectral Weighted Area are the best choices for the control parameter since they are better linked to the phonetic evolution. However the area descriptor is chosen since its found to be more robust.

The effectivesness of vocal imitation of sound concepts has been the focus of several researchers lately [6][5][40][41]. In [6] Cartwright and Pardo present a huge data set centered around crowd-sourced vocal imitation of audio concepts, which they argue is a natural way of communicating an audio concept. Their work is centered around collecting thousands of crowd-sourced vocal imitations along with data on the ability to correctly label the imitations. Being able to approximate an audio concept in this way would be analogous to approximate a visual concept by a visual sketch. Cartwright and Pardo's driving motivation for collecting such data is to enable users to communicate with software (e.g. programming a synthesizer) in a natural way by having a computer that can understand vocal imitations. The authors have also created *SynthAssist* [5], which is an audio production tool that allows user to interact with synthesizers in a more natural way. Instead of programming a synthesizer through knobs and sliders users can communicate sound ideas using samples like existing recordings or vocal imitations. *SynthAssist* take such sample as input and comes up with suggestions, which the user rate during an interactive search process. When the target sound is found the user can play and edit it using a traditional interface if desired. They argue that this approach is more similar to how one might communicate sound ideas to another human.

Janer [30] presents an example of the singing voice used to drive the synthesis of a plucked bass guitar. He experiments with two different synthesis techniques; one is based on physical modelling and the other is based on spectral morphing. Janer argues that the mapping encompasses two layers. The first layer concerns the interface while the second is related to the synthesis parameters. In the case of physical modelling based synthesis the string excitation is triggered by the voice energy envelope onset detection. Janer concludes that the goal of real-time performance was achieved, but the musical control was limited, and therefore further development is needed. In his Ph.D thesis *Singing-Driven Interfaces for Sound Synthesizers* Janer address the use of a singing voice as a controller for synthesizers. The system proposed is based on imitating the sound of the instrument, mapping voice pitch and loudness to corresponding instrument features. The outcome is a real-time score and a continuous value parameter, which is derived from the first two formants of the analyzed signal and is used for timbre modulation. He concludes that by controlling a singing voice synthesizer, voice input to voice synthesis mappings can be direct.

Deacon [16] proposes to use the voice as an additional control layer, which without any hardware dependencies can be seen as an unused resource. He

demonstrates that vocal control can be extended to integrating words or speech recognition to control non-musical parameters of a digital musical instrument. His system has been developed using MaxMSP.

Fasciani [19] focus on developing vocal driven control for DMIs, which is based on a generative and adaptive mapping scheme. Aiming at creating a generic and adaptive method that allow for extraction and mapping of gestural characteristics to an arbitrary number of parameters. The mapping is algorithmically generated and do therefore not depend on prior decisions or knowledge nor any user specification. From those aims he developed the *Voice-Controlled Interface for Digital Musical Instruments* (VCI4DMI), a vocal interface that minimize user involvement in the setup stage, but, maximizes the breadth of explorable sonic space [19]. He concludes that his interface is a significant advancement of the state of the art in automated gesture mapping and vocal control of musical instruments.

1.3.2 The human voice apparatus

The act of speaking is a result of an incredible precise and complex mechanical process that can almost be seen as a hybrid of a wind instrument and a string instrument. The voice apparatus (see Figure 1.2) includes a *source* (wind pushed from the lungs by the diaphragm resulting in excitation pulses), a component that *vibrate* (the vocal cords) and a series of *resonant chambers* (the pharynx, the mouth and nasal cavity). The pitch of the outputted sound is then determined by the frequency of the excitation pulse.

Usually a distinction is made between *voiced* and *unvoiced* speech. In phonetics voiced speech refer to speech sounds that is associated with vocal chord vibration, which is the case during normal speaking/singing. The latter refer to situations where the vocal folds do not vibrate as they are kept open, but very close together, which result in an irregular and turbulent airflow. Let's first have a look at voiced speech.

When air stream passes through the vocal folds they start to vibrate. If we denote the interval of the air pulses T_{pulses} then the frequency of these vibrations corresponds to the frequency of a certain perceived tone in the following way

$$freq = \frac{1}{T_{pulses}} \tag{1.1}$$

with harmonics at its multiple. The vibration of the vocal folds is then passed through the vocal tract, which acts as a resonant system. Here the pulses are filtered thereby shaping the spectrum of the voiced output. This shaping is what we usually refer to as *articulation* and the resonances of the vocal tract are called *formants*. In most cases the first four or five formants

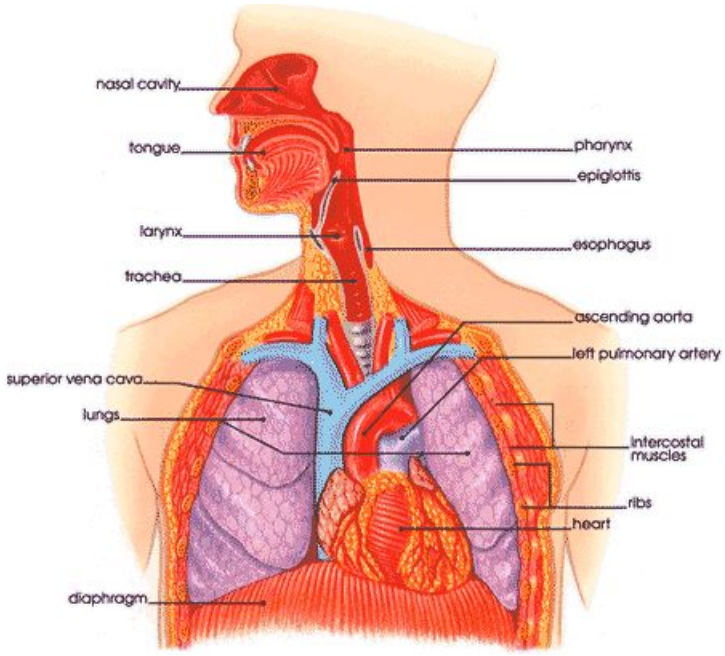

Figure 1.2: A representation of the human voice apparatus. (Illustration taken from: http://www.jennysawer.com/Thesis/images/figure1.jpg)

are considered the most relevant [19]. The length and shape of the vocal tract has an impact on the formant frequencies generated and when we speak or sing the length of the vocal tract is varied continuously. The frequency range together with the level of the harmonics is affected by age, gender and overall physiological condition, therefore it varies across individuals. When we speak the range typically lie between 100Hz and 400Hz, while singing range from 35Hz to 1500Hz [19]. In a single individual it is usually limited to a maximum of two octaves.

Figure 1.3 compare spectrograms of the words hot, hat, hit and head spoken with both a high pitched and a low pitched voice. The frequencies with the most energy concentrated indicate the regularly spaced harmonics (i.e. the red stribes), which originate from the pulses created by the vocal chords as explained above. The regions that carry the most energy are the formants.

During unvoiced speech (like whispering) the vocal folds do not generate periodic pulses, but instead aperiodic sounds with a broad spectrum, thereby resulting in the noisy perturbation that is the base of most unvoiced sounds.

Traditionally the speech system is simplified and modeled as a source/filter system where an excitation is filtered by the resonances of the vocal tract [31]. Because of this, human speech is usually approached as a subtractive synthesis mechanism where the vocal cords generate a buzzy excitation and

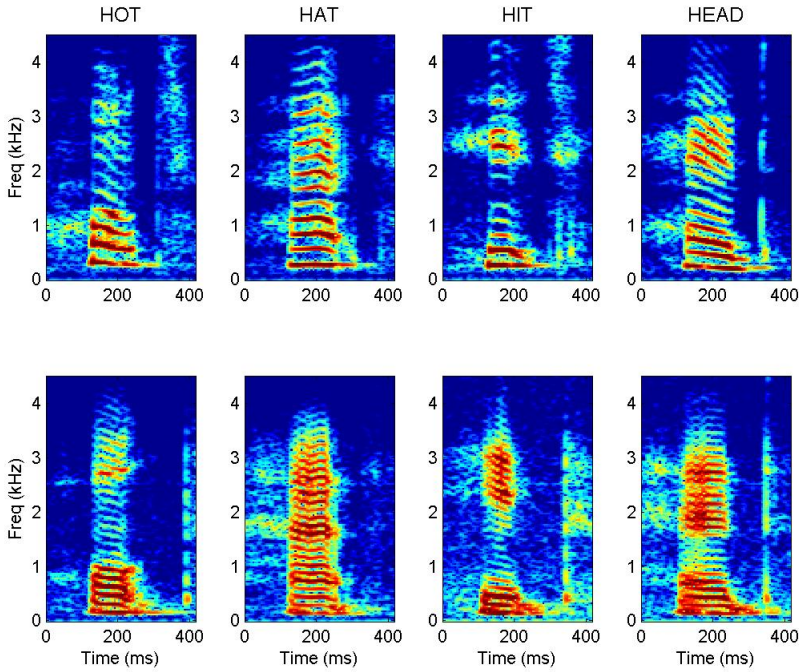

Figure 1.3: Spectrograms of the words hot, hat, hit and head spoken with a high pitched voice (top) and with a lower pitched voice (bottom). (Illustration taken from: https://auditoryneuroscience.com/sites/default/files/Fig1-16color.jpg)

the vocal tract act as a filter that create resonances.

1.3.3 Signal processing methods for voice signal analysis

When doing voice signal analysis the signal is typically divided into overlapping blocks of samples where the size of a block is referred to as *the window* or *frame size* and the overlap is the windows *step size*. Window size and step size vary across signal processing applications, but in the domain of automatic speech recognition (ASR) the signal is typically chunked into overlapping windows, with a length in the range of 10-40 ms and a step size of 10-20 ms. At this level the signal can be considered quasi-stationary [19]. The processing typically result in a vector of low-level features that can be seen as a compact representation compared to the original sequence. In the following we will look into some of the different feature extraction algorithms that all aim to capture either the state of the vocal apparatus or characteristics relevant to the auditory system.

Root-mean square or short time energy

Describes the instantaneous power of the signal and can be used as an expression for *voice activity*. The Root-Mean Square (RMS) is defined as the square root of the mean of the squares of a sample

$$RMS = \frac{1}{N}\sum_{n=0}^{N-1}[s(n)]^2 \tag{1.2}$$

Autocorrelation

Denotes the correlation of a function with itself and can be used to detect deterministic signals that are masked by a random background signal. A pure sine wave (deterministic signal) persists over all time displacements, whereas random noise tends towards zero for large displacements. Autocorrelation can be viewed as a signal convolved with a time-reversed version of itself [13] and is usually denoted with the greek letter phi

$$\phi_n = corr(f,f)(n) = f \star f = \frac{1}{N+1-n}\sum_{m=0}^{N-n} f(m)f(n+m) \tag{1.3}$$

where the term 1/(N+1-n) is a normalization factor.
Autocorrelation computes a time lag domain function that expresses the similarity of a signal to lagged versions of itself and can be used to find a signals *periodicity*.

Zero-crossing rate

Is the rate of sign changes in a signal and can be used as a rough estimation of the signal *brightness*

$$ZCR = \frac{1}{N-1} \sum_{n=0}^{N-2} \mathbb{I}\{s(n)s(n+1) < 0\} \tag{1.4}$$

where s is a signal of length T and the indicator function $\mathbb{I}\{A\}$ is 1 if its argument A is true and 0 otherwise.

Linear predictive coding

Linear predictive coding (LPC) is one of the most powerful speech analysis techniques and is typically used for encoding speech signals at low bit rates. LPC can also be used to estimate speech parameters as it automatically extract the gross spectral features of an input signal, designs a filter to match those, and result in a source that can be used to drive the filter. Since **TheStringPhone** makes use of LPC a more thorough explanation is presented in section 3.2.

Short-Time Fourier Transform (STFT)

Is used to determine the frequency and phase content of local (i.e. small) sections of a signal as it changes over time. It is based on the Fourier Transform, which models the input sound as a sum of harmonically related sinusoids and can be seen either as a sequence of windowed Fourier transforms or as a bank of filters [45].

Feature extraction

In the spectral domain a number of functions exist for extracting spectral features. These can be used to find 'patterns' in the signal that cannot be achieved in the time domain [31]. In [60] Peeters enumerate a whole set of spectral features that can be calculated from the STFT. To mention a few:

- **Spectral centroid** is a measure of the spectrums center of mass.
- **Spectral spread** is the spread of the spectrum around its mean value.
- **Spectral skewness** gives a measure of the asymmetry of a distribution around its mean value.
- **Spectral slope** is a representation of the amount of decrease in spectral amplitude.

The above mentioned techniques are just some of all the signal processing methods used for voice signal analysis and the interested reader is referred to [19], [31] and [60] for a more detailed list.

Chapter 2

Designing TheStringPhone

Since the process of designing a DMI is complex there are many interdependent choices one needs to make. What input device and which synthesis engine should one use, and how can these two be related in a meaningful way? Taking a holistic approach can be cumbersome resulting in the fact that many take either an approach to design only the input device, while others focus on the synthesis engine. As stated in the motivation our aim is to move towards the holistic approach. However, before being able to do so, we need a better understanding of the design process from the point of view of the synthesists. Our focus will therefore be on designing and evaluating **TheStringPhone** from an engineering standpoint, while lesser energy will be used on the mapping and interaction design. Despite this, we will still take advantage of the fact that we are developing **TheStringPhone** from a low level perspective. This means taking advantage of the fact that we are able to better tailor the sound engine so it fits the input device and vice versa.

2.1 Choosing a sound engine

In the holistic approach the choice of where to start (i.e. with the gestural controller or the sound engine) is like the chicken and the egg and one obviously needs to start somewhere. There are several synthesis techniques to choose from, but in our case we find Physical Modeling synthesis (PhM) an interesting alternative to common techniques, such as additive synthesis, subtractive synthesis, FM synthesis and wavetable synthesis. This is because PhM imitates the properties of sound - e.g. type of excitation and resonant body [15] - in contrast to waveform and spectrum properties [79]. In comparison to the commonly used wavetable synthesis technique, PhM can be a far more powerful approach: physical models lead to a wider range of sounds and expressiveness, whereas each timbre or performance expression must be prerecorded when samples are used [7]. The parameters used in

PhM are directly related to physical properties of the real world. This means that a user can capitalize on his/her fundamental understanding of physical causality [25], thereby making the conceptual understanding of PhM more intuitive. A fundamental principle of PhM synthesis is the interaction between exciter and resonator [15]. An excitation causes vibration and the resonance is the response of the body of an instrument to the excitation vibration. Since PhM synthesis can model this interaction it tends to communicate a sense of the gesture behind the emission of sound [65] in contrast to abstract methods that are controlled by mathematical formulas with no direct relation to gestural control.

PhM provides an acoustical sounding output, which is pleasing and we find this to be a good choice of sound engine for **TheStringPhone**.

2.2 Choosing an input device

The Karplus-Strong algorithm needs an excitation source. Usually white noise is used [36], but we find it interesting to experiment with the human voice as an excitation source instead. We find the human voice to be one of the most interesting forms of expression due to it's versatility. The voice can exhibit a broad range of extremely expressive variations beyond those of pitch and loudness e.g. Mongolian throat singing, Inuit vocal games and human beatboxing and the vocal apparatus is a rich source of information. Voice-based musical interfaces has the potential to offer a huge level of expressiveness [74] that can be both intuitive and fulfilling for a performer. Using a microphone as source input for **TheStringPhone** not only allows the user to use his/her voice as excitation, but it also opens up to a wide range of possibilities where one can use everyday sounds of the real world to excite the instrument.

This is an interesting way of coupling the input device to the synthesis engine, because it allows for a very coupled relationship between the input energy and output energy. Whereas many other DMIs have only a symbolic mapping, the input from the microphone is directly related to the synthesis engine serving as a part of the sound engine. Furthermore, as explained in section 1.1.2, the fact that one has to continuously supply energy to the system, makes the instrument feel more natural mimicking a real instrument [25].

Due to its simplicity the Karplus-Strong algorithm is not very musical interesting in itself, but by using the voice as excitation we extend the algorithm to produce a wider sound palette and at the same time allow for more musical expressiveness.

We have chosen a traditional MIDI-keyboard as input device for determining the produced pitch of the Karplus-Strong. Although it faces inherent limitations [31], there are several reasons for doing so. Firstly, because of

our focus on developing the sound production part of a DMI we have to limit ourselves in experimenting with input gestures. Secondly, using traditional instruments as a gestural controller has an advantage of adaptability. Using a keyboard and a microphone as input devices, will make the use of the instrument very intuitive for the average musician. A musician who is already familiar with a keyboard, would immediately be able to use the instrument and apply highly developed performance skills [65].

By choosing a high decay time on the Karplus-Strong, **TheStringPhone** allows for a conversational mode of interaction (as explained in section 1.2), where the musician can make short vocal noises and then "play" the resulting sound by using the MIDI-keyboard.

2.3 Extending the timbre control

The most common way of performing simple timbre control in synthesizers is probably by using an ADSR envelope generator. It mimics the way that loudness and spectral of acoustic instruments change over time and has a very influential role in a sounds timbre. We will therefore add an ADSR envelope generator to **TheStringPhone**.

Since the voice is already used as an excitation source and the users hands are used for manipulating the MIDI-keyboard we want to experiment with using the voice as additional timbre control. Unlike the MIDI-keyboard, the voice is a source of many parameters, which can be analyzed and used for sound synthesis and control (as explained in section 1.3.1). Referencing section 1.1.1, this would also extend the instrument to implement a form of indirect acquisition, instead of just using the voice as an exciter and the MIDI-keyboard, which are methods of direct acquisition. Along with amplitude envelopes, filtering is a very common way of controlling a sounds timbre. Since LPC analysis results in a set of filter coefficients mimicking the vocal tract, we can use this to filter the Karplus-Strong. In this a more expressive instrument could arise. On the other hand, both the Karplus-Strong and the vocal tract resembles comb filters, so using the LPC coefficients as filter for the Karplus-Strong could turn out to have too little influence on the resulting sound. This is however uncertain due to the fact that the LPC filter would continuously change according to the vocal input and might therefore result in interesting and unexpected timbre variations that corresponds well with the idea of interactive surprisal found in the conversational mode.

2.4 Adding compression and reverb

Dynamic range compression is often used in the process of sound recording and production. It is used to reduce the dynamic range of sounds which contains various parts of unequal amplitude. By applying compression to

TheStringPhone we want to have control over the sudden transients that might arise during usage. Compression can also add a certain warmth and character to a signal and make it sound fuller.

The Karplus-Strong algorithm sounds very dry on its own. A common way to add life and warmth to a sound is to add reverberation. Adding a reverb to the signal chain would therefore enhance the quality of the sound of the instrument and make it sound more pleasing to listen to. Implementing reverberation will also allow us to gain experience in designing audio effects.

2.5 TheStringPhone - a system overview

Figure 2.1 shows a system diagram of **TheStringPhone** instrument as a first prototype and is the version that has been developed during this project.

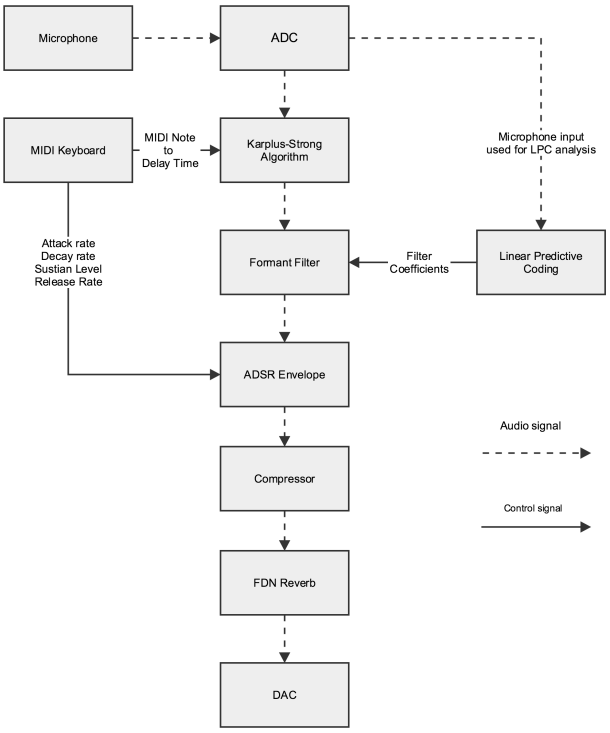

Figure 2.1: System diagram of **TheStringPhone**.

At its heart **TheStringPhone** has a Karplus-Strong plucked string algorithm as a synthesis engine. It uses the human voice as an excitation source and the instruments pitch is controlled via MIDI-keyboard. The output signal is passed through a formant filter that is extracted from the users input voice signal. In order to smooth the sound and place it in a space **TheStringPhone** has a reverb built into it. The dotted lines Figure 2.1

represent the audio signal path and the solid lines represent control signals (MIDI messages or other forms of communication).

Chapter 3

Implementing TheStringPhone

This chapter describes the implementation of the **TheStringPhone** in depth. For each part of the instrument we will describe the theory and mathematics used, how the part was coded and what purpose it serves. We will also describe how each part effects the signal and what considerations that has been made when going from theory to practice. The audio examples used throughout this chapter can be found in the accompanying resource folder under 'audio examples'. They are arranged so each section of this chapter has a corresponding audio folder. It is assumed that the reader has a working knowledge of digital filter theory. If this is not the case we refer to Appendix .1 for a thorough introduction.

3.1 The Karplus-Strong algorithm

In order to have a better foundation for explaining the Karplus-Strong algorithm, we will start by introducing the case of a simple vibrating string as well as the concept of standing waves.

3.1.1 The ideal vibrating string

In air sound propagates as traveling waves. The rigid boundaries at the edges of a string cause most energy to be reflected back into the string, thereby preventing it from radiating away [44]. If we excite a string at a regular interval we would get forward and backwards propagating waves, traveling at the same time and causing interference (Figure 3.1). The result of two traveling waves propagating in opposite directions on the same string is that the traveling waves do not actually 'travel' anymore, but stand still vibrating vertically instead (this is called transverse motion and an example can be seen in Figure 3.2).

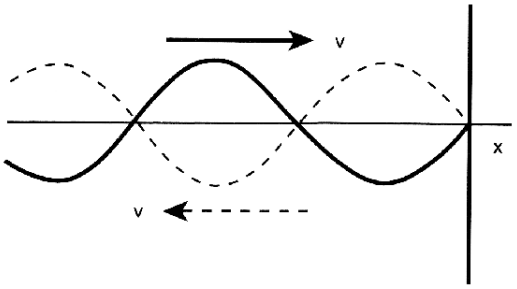

Figure 3.1: Illustration of left and right travelling waves. (Illustration taken from: http://demoweb.physics.ucla.edu)

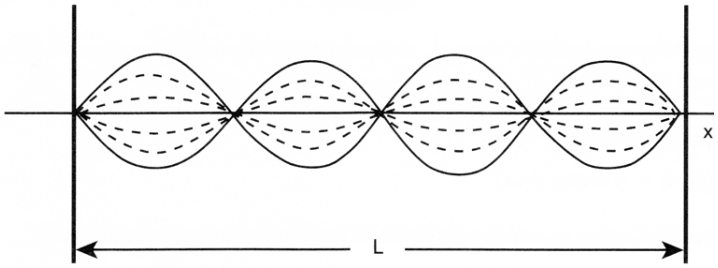

Figure 3.2: Illustration of standing waves. (Illustration taken from: http://demoweb.physics.ucla.edu)

In order to describe the motion of a vibrating string we use the following equation

$$y(x,t) = y_l(t + \frac{x}{c}) + y_r(t - \frac{x}{c}) \tag{3.1}$$

which says that any vibration of the string can be expressed as a combination of two separate traveling waves, one going in the left direction (y_l) and one going in the right direction (y_r) and the waves travel at the rate of c (x is displacement). The speed of the traveling wave can be characterized by the string's tension and linear mass density ρ:

$$c = \sqrt{\frac{K}{\rho}} \tag{3.2}$$

It turns out that a string tied at both ends of length $L/2$ causes natural resonances to occur at $k \cdot 2\pi/L$ radians/sec [59]. In a vibrating string

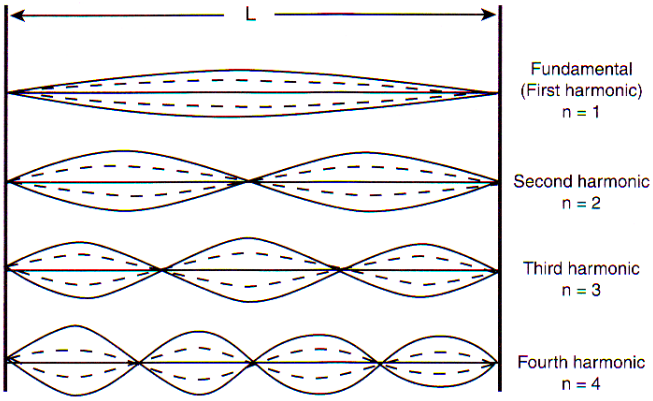

Figure 3.3: The different modes of a plucked string. (Illustration taken from: http://demoweb.physics.ucla.edu)

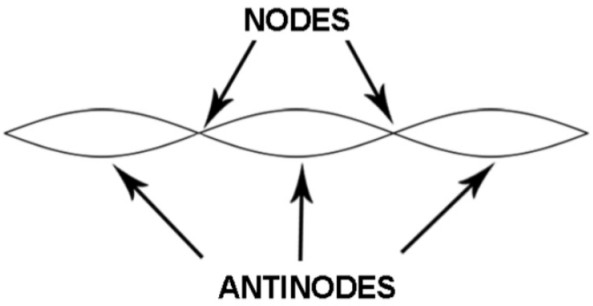

Figure 3.4: Nodes and antinodes of a vibrating string. (Illustration taken from: http://demoweb.physics.ucla.edu)

situation multiple vibrational modes occur (Figure 3.3) as well as corresponding nodes and antinodes (Figure 3.4). The first mode is equivalent to the fundamental frequency, the second mode vibrates twice as fast, which is equivalent to the octave (i.e. second harmonic) and so on. In stringed instruments modes and nodes can be thought of as harmonics which heavily influence the instrument timbre. For an ideal string system where there is no energy loss at all, the standing waves would never decay. For real world instruments this is obviously not the case as energy dissipates in the form of acoustic energy as well as thermal and friction-based energy due to the bridge and nut part of the instrument [44]. Difference equations (and thereby digital filters) can be used to approximate and represent standing waves and is exactly what digital waveguide synthesis is all about.

3.1.2 Digital Waveguides - modelling a plucked string

The Karplus-Strong algorithm was proposed in 1983 by Kevin Karplus and Alex Strong [36], and later recognized as a specific case of the *waveguide model* invented by Julius Smith as presented in [68]. At their heart waveguide models consist of delay lines and filters making them ideal for implementing using digital signal processing. The idea is to digitally represent the left and right traveling waves as seen in Figure 3.1 as two delay lines as seen in Figure 3.5. The inverting multipliers between the two delay lines model the strings termination points (e.g. the bridge and nut of a guitar).

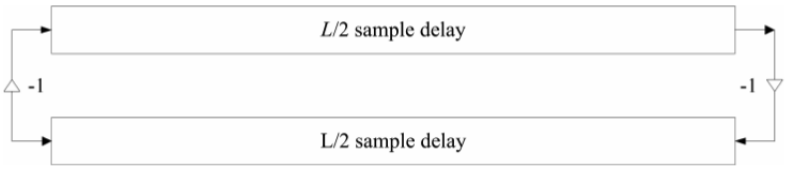

Figure 3.5: A simplified digital waveguide that models the left and right traveling waves of a string. (Illustration taken from: [59])

As we are dealing with a linear time invariant system we can collapse the two delay lines of length $L/2$ into a single delay line of length L. The inverting multipliers of our termination points can be collapsed to a single variable that takes care of all energy losses. The collapsed delay line can be seen in Figure 3.6 where q does not only represent energy loss due to termination points (as we have in our simplified model in Figure 3.5), but also account for the energy loss due to internal friction within the string as well as drag with the surrounding air [68].

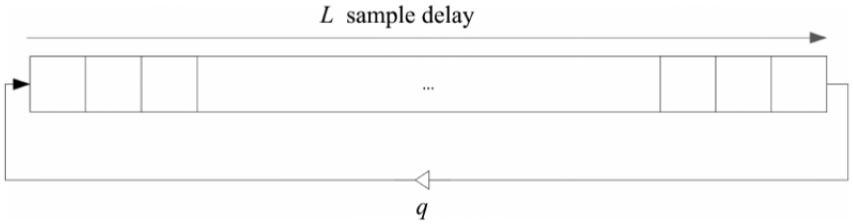

Figure 3.6: A consolidated waveguide of length L with feedback component q that mimic energy loss. (Illustration taken from: [59])

What we have at this point is a comb filter where our feedback component q needs to be strictly below 1 for stability. However there is one problem with this representation; all the vibrational modes are dampened equally resulting in all the harmonics dying away with the same rate q. In the real world the higher modes of a string die away faster than the lower

modes leaving the fundamental mode to die away last. In order to model this high frequency decay Kevin Karplus and Alex Strong came up with the idea of inserting a low-pass filter in the feedback chain. Since the low-pass filter is a part of the feedback loop the high frequency content is repeatedly filtered away as time passes, thereby modelling the decay rates of the vibrational modes in a natural way. The tone produced will be perceived by the ear as a complex pitched tone with an exponential decay, suggestive of a plucked string [45]. The Karplus-Strong model is depicted in Figure 3.7, where $x[n]$ is the input signal, $y[n]$ the output signal, z^{-L} is a delay line of length L and LPF is a low-pass filter.

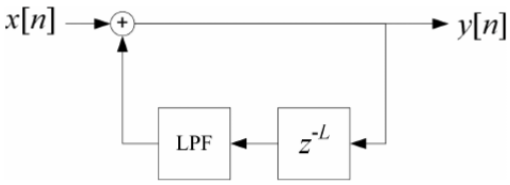

Figure 3.7: The Karplus-Strong model

The frequency of the tone produced by the model corresponds to:

$$f = \frac{f_s}{L} \tag{3.3}$$

where L is the length of the delay line and f_s the sampling frequency. One of the major problems with the Karplus-Strong model is that the fundamental frequency cannot be accurately controlled [79], due to the fact that the model uses a non-fractional delay line. Furthermore, the model does not allow one to specify a pluck position on the string. Extensions and improvements have been done [29] since Karplus and Strong proposed their basic model in 1983. By adding DSP blocks to the model it is not only possible to model the pluck position and fine tune the instrument, but also to model the instruments body and allow for decay time alterations, etc.

3.1.3 Implementation

As mentioned above the heart of the Karplus-Strong is a delay line. They exist as basic components of many time domain based signal processes and their basic function is to delay a sample by an amount of time, which can be set in seconds, milliseconds or samples [39]. Delay lines with very short delay can be used to implement filters. A first-in first-out (FIFO) structure as shown in figure 3.8 is typically used in cases where a delay of only a few samples is needed. However, since the samples are continuously copied from one memory location to the next, FIFO is not an efficient way when larger

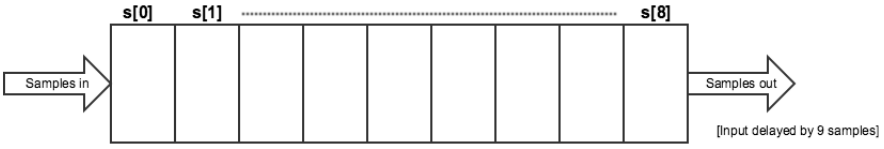

Figure 3.8: First-in first-out delay line.

delay times are needed. In this case the most common data structure used is a circular buffer as shown in figure 3.9. In this structure two pointers are used. One to read the output sample and another to write the current sample to the delay line. The distance between these two determine the delay time. For example, if the write pointer is 10 samples in front of the read pointer, the result is a delay of 10 samples.

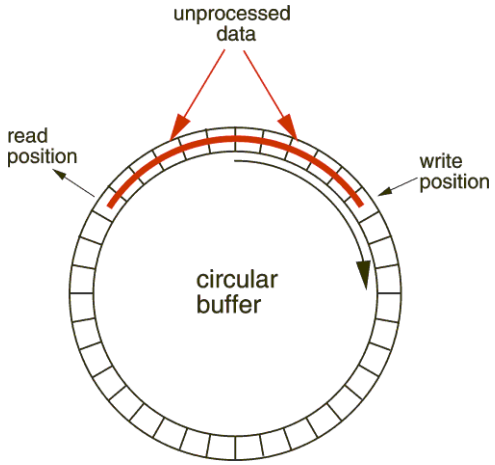

Figure 3.9: A circular buffer. (Illustration taken from: http://luaview.esi-cit.com/datalog˙manual.html)

In the implementation of **TheStringPhone** a FIFO delay is used to implement the lowpass filter and a circular fixed delay is used to simulate the traveling waves of the wave equation. The process function of the Karplus-Strong is shown in Figure 3.10. The function first reads the delayed sample from the delay line (line 63) which is returned as output. The sample is hereafter filtered by a one zero lowpass filter (line 64). The filtered output is then scaled and summed with the input (coming from the microphone) and hereafter written to the delay line (line 65). Lastly, the pointers of the delay are updated (line 66).

The circular buffer delay implementation used in the Karplus-Strong

```
62      void process (float input, float& output) {
63          output = delay.read();
64          oneZero.process(output, z1);
65          delay.write(input+z1*feedback);
66          delay.updatePointers();
67      }
```

Figure 3.10: Karplus-Strong process function.

process function can be seen in Figure 3.11. The essentials of the delay are read(), write() and updatePointers() functions and **rp** and **wp** represent the read and write pointer respectively. In line 55 the write pointer is advanced (decremented) and in line 59 the read pointer is set as equal to the write pointer plus the delay, which is specified by the variable called "theDelay". The rest of the code makes sure the pointers warp around instead of going out of bounds.

```
42      // write data into line
43      void write(double data) {
44          dly[wp] = data;
45      }
46
47      // read data from line
48      double read() {
49          return dly[rp];
50      }
51
52      // advance read, write pointers
53      void updatePointers()
54      {
55          wp--;
56          if(wp < 0)
57              wp = kMaxDelay-1;
58
59          rp = wp+theDelay;
60
61          if(rp > kMaxDelay-1)
62              rp -= kMaxDelay;
63
64          if(rp > kMaxDelay-1)
65              rp = kMaxDelay-1;
66
67          if(rp < 0)
68              rp = 0;
69      }
```

Figure 3.11: Circular buffer delay implementation.

The body of the one zero lowpass filter process function is shown in

Figure 3.12

```
34      void process (float input, float& output)
35      {
36          //Transposed Direct II Form
37          output = z1 + input * a0;
38          z1 = input * a1 - output * b1;
39      }
```

Figure 3.12: One Zero lowpass filter process function.

This mode of implementation is called the *Transposed Direct II Form* [70]. $z1$ is the delayed sample and $b0$, $b1$ and $a1$ are the filter coefficients. It is basically another way of implementing a FIFO delay line with a delay length of one sample. Depending on the filter coefficients, this implementation can be used to implement a wide variety of filters. Figure 3.13 shows the MATLAB code we have used in designing the one-zero low-pass filter.

```
1 -    a = [0.5, 0];
2 -    b = [1, 0.5];
3
4 -    [h,w] = freqz(b,a);
5 -    scale = 1/max(abs(h));
6 -    b = b*scale;
```

Figure 3.13: MATLAB code for designing the one-zero low-pass filter used in the Karplus-Strong loop.

The last two lines normalizes the gain of the filter by dividing the output coefficients with the maximum output of the transfer function. Figure 3.14 shows the frequency response of the filter.

Before being summed with the input and written to the delay, the output of the filter is scaled by a feedback factor. This controls how strongly the output is fed back into the delay and therefore how quickly the signal attenuates. If the feedback factor is > 1 then the signal will continue to rise, if the signal is < 1, then the signal will attenuate. Figure 3.15 shows the impulse response of **TheString Phone**'s Karplus-Strong implementation with three different feedback coefficients. Please refer to the resource folder for the sound files for each of the impulse responses.

With a feedback of 1 the full output signal is fed back into the delay. The attenuation, seen in the left most plot in Figure 3.15, is therefore solely caused by the lowpass filter in the loop. A longer decay means that one has to be less active in exciting **TheStringPhone** with the voice. If the signal is only attenuated by the lowpass filter a lot of low frequency energy can quickly build up if the user keeps giving input to the microphone. causing the signal to distort. This was in fact the case when testing **TheStringPhone** using a feedback factor of 1. As can be seen in the right most plot of Figure

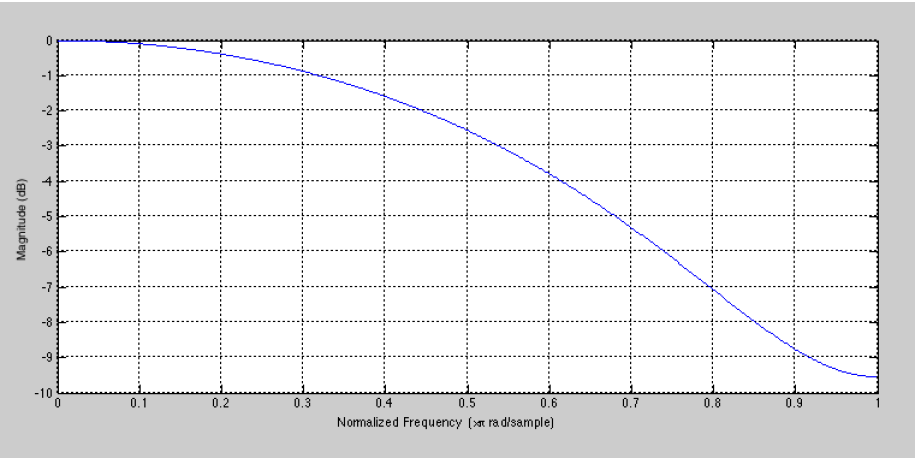

Figure 3.14: Frequency response of the one-zero filter used in the Karplus-Strong.

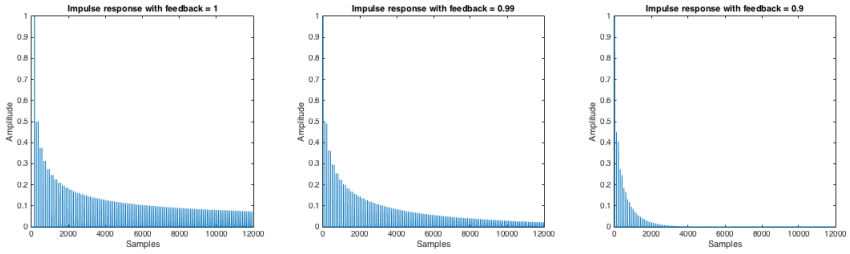

Figure 3.15: Karplus Strong impulse response with feedback 1 (left), feedback 0.99 (center) and feedback 0.90 (right).

3.15 a feedback of 0.9 results in a much quicker decay thereby solving the problem. However, while it is important that the user can scream into **TheStringPhone** without it clipping, it is also important that he/she must not have to use excessive energy to get a sound out of the instrument. One therefore have to balance this relationship. The middle plot of Figure 3.15 shows the impulse response with feedback set to 0.99, which was found to balance the relationship between feedback factor and vocal input activity without causing distortion. This is of cause a subjective thing depending on how the instrument is used. Sound files of the different Karplus-Strong attenuations where input is given from the microphone can be found in the resource folder.

An easy functional addition could be to allow the user to control how fast the Karplus-Strong attenuates.

As explained by Equation 3.3 the frequency of the Karplus-Strong is determined by the sample rate f_s and the delay line length L. Solving for the delay line length L allows us to specify a frequency and know the

delay length needed in order for the Karplus-Strong algorithm to output the desired frequency:

$$L = \frac{f_s}{f} \tag{3.4}$$

In **TheStringPhone**, the pitch of the Karplus-Strong is controlled by a MIDI keyboard. This is done by converting the MIDI note to frequency by Equation 3.5 and then inserting it into Equation 3.4 to get the length of the delay line.

$$f = 2^{(MIDI-69)/12)} * 440 \tag{3.5}$$

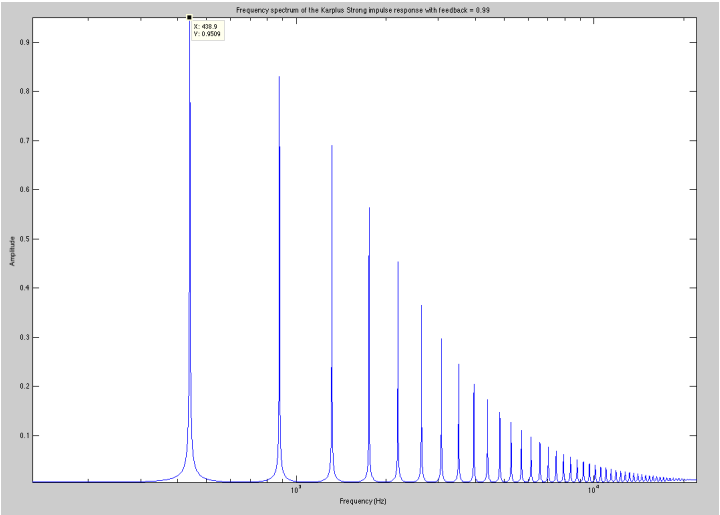

Figure 3.16: Frequency spectrum of the Karplus-Strong impulse response with feedback = 0.99.

In Figure 3.15 the Karplus-Strong is tuned to 440 Hz (middle A) or MIDI note 69 using Equation 3.5. Figure 3.16 shows the frequency spectrum of the impulse response from Figure 3.15. Here one can see a clear fundamental around $440Hz$ and an overtone series giving the sound the characteristics of a plucked string. Due to the lack of non-fractional delay lines as mentioned in Section 3.1.2 the fundamental is actually at $438,8$, which means the instrument is slightly out of tune in relation to the general pitch standard. Inserting 44100 as f_s and $440Hz$ as f in equation 3.4 give us a delay length of 100.23. Since the delay length must be an integer value the delay length is rounded off to 100 therefore causing the instrument to be slightly out of tune. This is a problem especially at high frequencies. For long delay lengths (low pitches) a slight difference in delay length does not cause a big change

in pitch. However, higher pitches sits much closer together frequency-wise and a slight difference in delay length therefore has a much bigger impact.

Introducing a filter that can contribute a small delay without altering the loop gain into the loop is one solution to this [29]. This is important to implement if the **TheStringPhone** is to be used in a setting with other instruments, but not a necessity in this prototype.

3.2 Linear predictive coding and the formant filter

TheStringPhone uses linear predictive coding (LPC) to create a formant filter which models the vocal tract of the user. It does so by decomposing the input signal from the microphone into an excitation signal, commonly referred to as *the residual* (the unfiltered pulses generated by the vocal chords) and a set of filter coefficients that models the formant effect of the vocal tract (formant filter) [62].

Originally LPC was applied in telecommunications as a means of data reduction i.e. voice compression. The fundamental interest was to reproduce speech at a distance using the representation requiring the least bandwidth to transmit [45]. For musical applications however there is generally no point in re-synthesizing the original signal. Here the usefulness lies in the ability to modify the sound in ways that are either difficult or impossible to do otherwise. Obviously LPC can be used to model speaking and singing, but also to model woodwind instruments, birds, whales and in theory any resonant sound source can be modeled. The timing and pace of the analyzed speech signal can also be altered or the spectral content can be modified. The formant filter of one sound can also be cross-synthesized with another sound to form a vocoder.

In general LPC works by first analyzing the speech signal by estimating the formants. When a formant filter has been estimated the formants from the speech signal is removed by inverting the formant filter (this process is referred to as *inverse filtering*) in order to arrive at the original excitation signal. This excitation function looks like an impulse train for vowels and the sound is probably best characterized as a 'buzz'. For consonants the source sounds like broadband noise [45]. The original signal can then be re-synthesized by driving the residual signal through the estimated formant filter.

3.2.1 Linear prediction

A prediction algorithm tries to find samples at positions outside a region where one already has samples [65] and can therefore be seen as an extrapolation of a set of samples. Obviously prediction includes the possibility of being wrong, therefore such algorithms also include an error estimation.

The process starts out with a linear predictor (a difference equation) that express each new sample as a linear combination of previous samples. The coefficients of our linear predictor is referred to as *prediction coefficients* and an estimation is found by solving a set of linear equations that result in a filter that matches the frequency response of our input (the vocal tract). If we can predict a signals behavior in the time domain we can characterize its behavior in the frequency domain [45]. The difference equation for a linear predictor is

$$y(n) = \hat{x}(n+1) = \sum_{i=0}^{m} a_i x(n-i) \tag{3.6}$$

where m is the number of past samples taken into consideration. This is a standard FIR filter form, but the output is an estimate of the next sample in time. At this point our basic problem is to determine the set of predictor coefficients a_i directly from the speech signal so that the spectral properties of our desired digital filter will match those of the speech waveform within our analysis frame [62]. The spectral characteristics of speech vary over time [77] therefore the predictor coefficients at a given time must be estimated from short segments of the incoming signal. This is done by finding a set of predictor coefficients that minimize the mean-squared prediction error (MSE) for each frame

$$MSE = \frac{1}{N} \sum_{i=0}^{N-1} [\hat{x}(n+1) - x(n+1)]^2 \tag{3.7}$$

When a set of prediction coefficients has been computed it is easy to compute the error signal by subtracting the real input from our estimated value

$$\epsilon(n+1) = \hat{x}(n+1) - x(n+1) \tag{3.8}$$

There exist several methods for arriving at the set of predictor coefficients which yield a minimum MSE [13]. One method is the autocorrelation method, which **TheStringPhone** make use of and will therefore be the only one presented here.

3.2.2 Autocorrelation method for LPC coeficcients

As shown in section 1.3.3 the autocorrelation function holds information about a signals self similarity at delayed times. We can therefore use it to arrive at our linear prediction coefficients by forming an autocorrelation matrix denoted R [13]

$$R = \begin{pmatrix} x \star x(0) & x \star x(1) & x \star x(2) & \cdots & x \star x(m) \\ x \star x(1) & x \star x(0) & x \star x(1) & \cdots & x \star x(m-1) \\ x \star x(2) & x \star x(1) & x \star x(0) & \cdots & x \star x(m-2) \\ x \star x(3) & x \star x(2) & x \star x(1) & \cdots & x \star x(m-3) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x \star x(m) & x \star x(m-1) & x \star x(m-2) & \cdots & x \star x(0) \end{pmatrix} \tag{3.9}$$

This m x m matrix of autocorrelation values is symmetric with all diagonal elements equal (called a *Toeplitz matrix*) and can be solved through several procedures [62]. We can get our least squares predictor coefficients by forming

$$A = [a_0, a_1, a_2, a_3 ... a_m] = PR^{-1} \tag{3.10}$$

where P is a vector of prediction correlation coefficients

$$P = [x \star x(1), x \star x(2), x \star x(3), ..., x \star x(m+1)] \tag{3.11}$$

3.2.3 Implementation

The LPC main processing loop can be seen in Figure 3.17. This code is written by Ge Wang for his **rt_lpc** tool [83], but has been adapted and used in **TheStringPhone**. Ge's code is heavily based on code written by Perry Cook.

```
228     // find the autocorrelation of the signal, with pitch
229     *pitch = autocorrelate( x, len, lpc->corr );
230
231     // construct the R matrix
232     for( i = 1; i <= order; i++ )
233         for( j = 1; j <= order; j++ )
234             (*(lpc->R))[i-1][j-1] = lpc->corr[abs((int)(i-j))];
235
236     // invert R
237     lpc->R->invert( *(lpc->res) );
238
239     // find the coefficients A = P*R^(-1)
240     for( i = 0; i < order; i++ )
241     {
242         coefs[i] = 0.0f;
243         for( j = 0; j < order; j++ )
244             coefs[i] += (*(lpc->R))[i][j] * lpc->corr[1+j];
245     }
```

Figure 3.17: Code implementing the LPC analysis

The signal is first autocorrelated in line 229 and the correlation matrix is hereafter constructed in the double for-loop. In line 237 the matrix is

then inverted and the matrix equation is solved for A in the following for-loop. The array *coefs* now hold the filter coefficients. The last step of predicting the residue is not done because only the coefficients are used in **TheStringPhone**. The coefficients are used to design a formant filter that filters the output of the Karplus-Strong. A Nth order LPC will result in N coefficients.

```
40      void process(float &output) {
41          for(int j = 0; j < order; j++ )
42              output += Zss[j] * coefs[j];
43
44          for(int j = order - 1; j > 0; j-- )
45              Zss[j] = Zss[j-1];
46
47          Zss[0] = output;
48      }
```

Figure 3.18: Formant filter process implementation

The process function of the formant filter is shown in figure 3.18. The first for loop adds the delayed samples weighted by the filter coefficients and the second for loop shifts every sample by one in the delay line. This way of implementing filtering is not as computationally efficient as the *Transposed Direct II Form* used in the one-zero lowpass filter for the Karplus-Strong (see Figure 3.12). However, using this form allowed for a fast and easy way of changing the filter order thereby making it easy for us to test different LPC orders. Figures 3.19, 3.20 and 3.21 show the frequency responses of the formant filter with an 8th order, 16th order and 30th order LPC respectively. Here the input to the microphone is the author giving the vowel "Ihh" as input.

High frequency formants have less energy and is less prominent in the filtering. Higher order LPC results in more formants and therefore in better quality and accuracy, but it also requires more processing power. In a musical application such as the **TheStringPhone**, where efficiency is more important than accuracy, a lower order makes sense. However, when comparing the frequency response of the 30th order formant filter with the 8th and 16th order formant filters it is clear that these are not predicting the formants very well. The 8th order filter is lacking a pole around 250Hz and the 8th and 16th order filters have poles that are more prominent at 700Hz and above compared to the 30th order. These errors stem from the fact that the frequency response of the vocal tract is too complex to represent with just a few poles. Even though you only need 4 formants (8 poles) to make speech intelligible, more poles are needed to estimate the vocal tract from a given speech signal. However, the term intelligibility does not make much sense in relation to the **TheStringPhone**. Here it is much more important that the filter responds effectively to changes in the vocal input, soy the user easily can understand the causality of what input leads to what output. De-

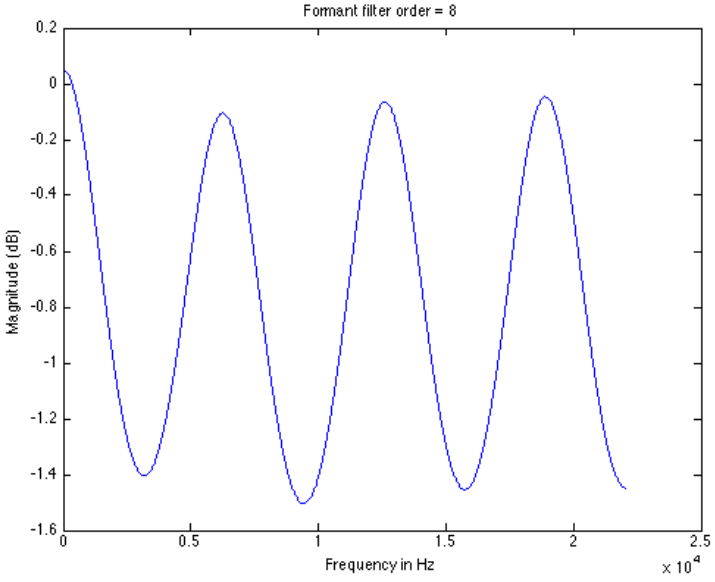

Figure 3.19: Formant filter from a 8th order LPC.

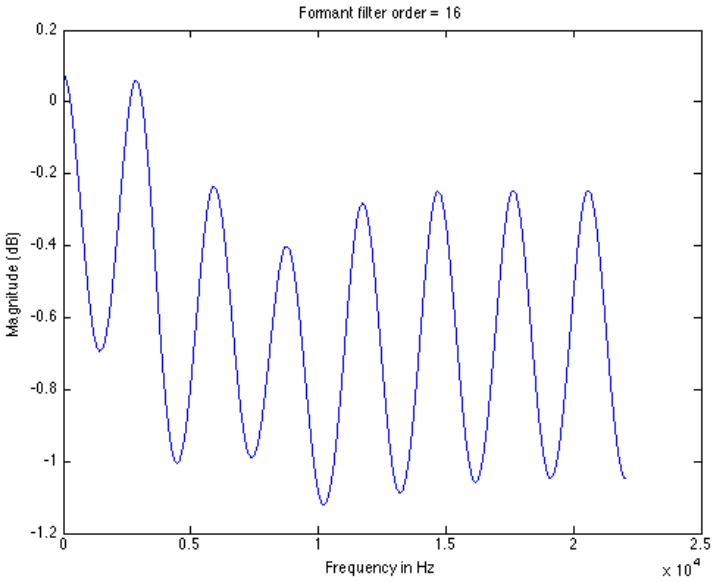

Figure 3.20: Formant filter from a 16th order LPC.

spite this, an order of 30 has been used as the difference in latency of using a 8th order versus a 30th order LPC is very little.

Due to the similarity of the formant filter's all pole model and the

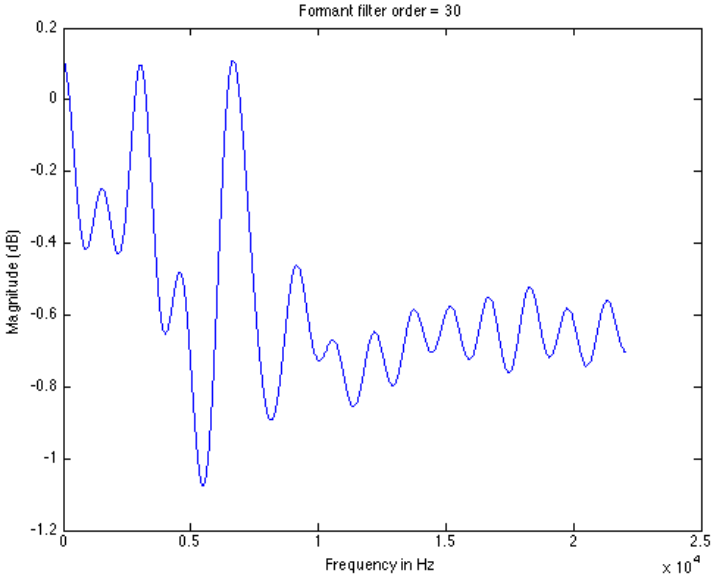

Figure 3.21: Formant filter from a 30th order LPC .

Karplus-Strong's comb filter resemblance, we had initial concerns about whether filtering the Karplus-Strong with a formant filter would give an effect prominent enough to be heard in the final output. While this does not appear to be an issue, mostly due to the dynamic nature of the LPC formant filter, other problems arose. Firstly, an absence of vocal input to the microphone creates a situation where the LPC is trying to predict a filter from little or no input. An example of a filter created from LPC coefficients predicted from such an input can be seen in Figure 3.22.

With peaks above $30dB$ this filter does not look good. Figure 3.23 shows the pole zero plot of the filter and indeed we find poles outside the unit circle confirming that the filter is not stable. This filter is going to create click and pops in the output signal. This can be heard in the 'No_voice_input.wav' found in the resource folder for this section.

The reason why re-synthesis normally works with LPC is that the output signal is multiplied by the power of the input estimated by the LPC. This means that the volume envelope of the signal is matched to the vocal input making sure that the LPC only analyses meaningful input. This is however not a suitable solution for the **TheStringPhone** because we do not want the input energy to exactly match the output energy. In order to design for *the conversational mode*, as explained in section 1.2, a longer decay is more appropriate, because this allows the user to input energy and then listen to the result and adjust his performance accordingly. A solution could be to make sure that the filter coefficients of the formant filter are only changed

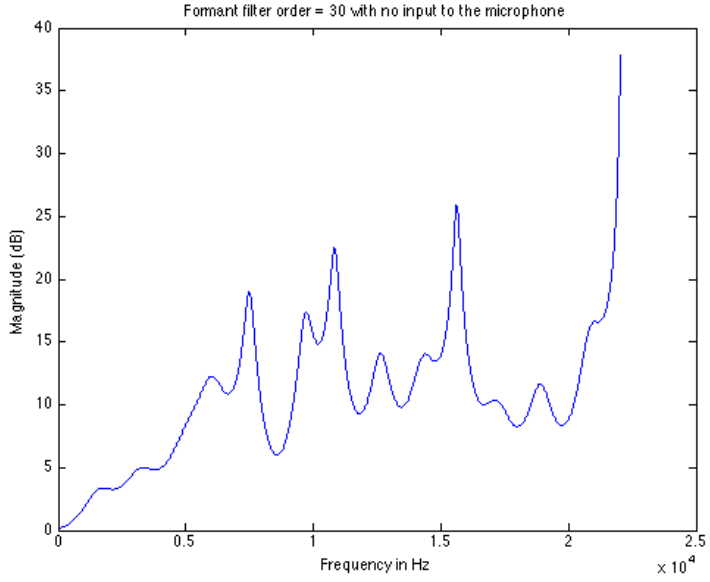

Figure 3.22: Formant filter from 30th order LPC with no microphone input.

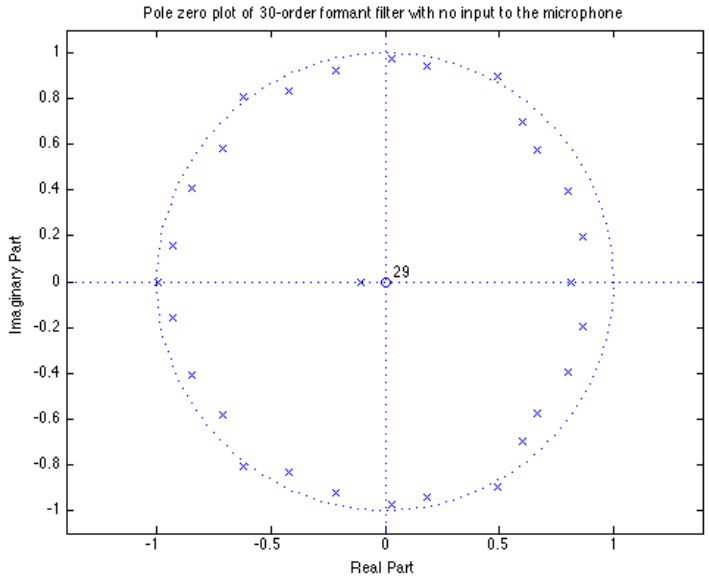

Figure 3.23: Pole zero plot of formant filter from 30th order LPC with no microphone input.

when there is a vocal input signal for the LPC to analyze. A first step in this direction is to detect whether there is *any* input to the microphone at

all. This could easily be done by calculating the RMS of the signal or use the power prediction of the LPC as an estimate input level. This means that when the signal goes below a certain threshold the filter becomes static and the coefficients are not changed. None of these completely solved the problem. It is difficult to find a suitable threshold as this is dependent on the input gain as well as the type of microphone used. Furthermore, if the estimated coefficients are considerably different from the "saved" coefficients the filter is going to jump, creating discontinuities which results in click and pops in the output signal.

Because of the above mentioned problem we tried various other audio chains to see if anything interesting popped up. One was to do a LPC re-synthesis with white noise (audio example found in project folder: 'white_noise_resynthesis.wav') and then use this as excitation for Karplus-Strong. This basically created a more noisy Karplus-Strong, which did not suite the direction we were going in (audio example found in project folder: 'white_noise_resynthesis_karplus.wav'). Another approach was to mix the re-synthesis signal with the microphone input. This made the output more intelligible, but not more musically interesting (audio example found in project folder: 'resynthesis_mixed_with_karplus.wav'). The most musically interesting, from our point of view, is actually omitting the formant filter and then experimenting with various inputs to the microphone. This can both be vocal and non-vocal.

3.3 The ADSR-envelope

After the microphone input has been processed by the Karplus-Strong and filtered by the formant filter it is sent through an ADSR envelope generator. An ADSR is a four-stage envelope generator comprised of an *attack* period, a *decay* period, a *sustain* portion and a *release phase* [39]. Figure 3.24 shows a graphical representation of the four stages. The envelope generator works the following way: when a key is pressed a gate ON signal is sent, starting the attack stage. The attack stage goes from 0 to 1 in the amount of time defined by the attack rate. When the envelope generator reaches 1 it switches to the decay stage. Here the amplitude goes from 1 to the user-defined sustain level (which can be anything from 0 to 1) in the amount of time defined by the decay rate. The signal then stagnates at the sustain level until the key is released. When this happens a gate OFF signal is sent starting the release stage and the envelope generator begins to move towards zero at the rate defined by the release control.

The curves of an envelope generator can be either linear or exponential. The exponential generators are often found in hardware devices caused by the charging and discharging of capacitors with current limited by resistors [64]. Such exponential segments work well for amplitude envelopes, since

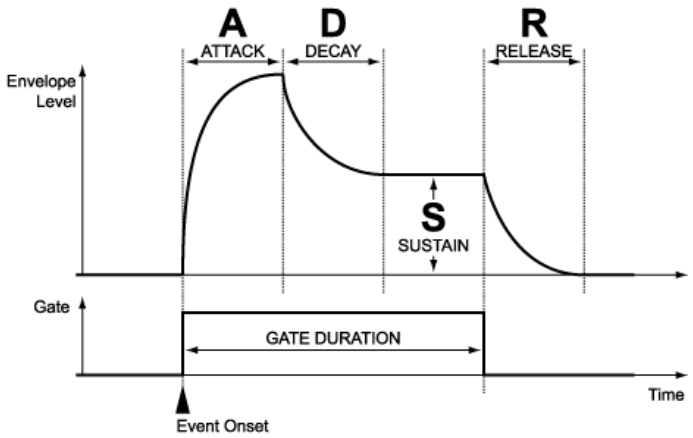

Figure 3.24: ADSR amplitude envelope. (Illustration taken from: http://www.audiomulch.com/images/blog/southpole-expedition-part-3-pattern-sequenced-adsr-envelopes-adsr-timing.png)

we hear on a log scale [45], thereby making the volume changes sound linear. Notice that the envelope generator depicted in Figure 3.24 make use of exponential segments. Exponential segments can be implemented using a one-pole filter, since these yield perfect exponential decay and work by simply feeding back a percentage of the previous output, where the percentage value is controlling the rate of the resulting exponential curve. Such a filter is also called a leaky integrator and is widely used in DSP applications to create smooth transitions [64].

3.3.1 Implementation

Figure 3.25 shows the code for handling incoming MIDI note messages.

```
221     // Handle midi note on and off
222     if(msg.velocity > 0) {
223         voicer->getVoice(msg.pitch)->adsr.keyOn();
224
225     }
226     else {
227         voicer->getVoice(msg.pitch)->adsr.keyOff();
228     }
```

Figure 3.25: Set frequency and trigger ADSR on key press.

First a check is made for whether the velocity of the message is above 0. If this is true it means that the user has pressed a key and the ADSR of the corresponding voice is gated ON. If the velocity on the other hand is 0, the ADSR is gated OFF. The signal is then multiplied by the ADSR value as can be seen in line 373 of Figure 3.26. The ADSR used is from the

```
369         for(Voice* v : voicer->voices) {
370             if(v->adsr.getState() != v->adsr.IDLE) {
371                 v->process(inp0, voiceOut);
372                 formantFilter->process(voiceOut);
373                 out += voiceOut * v->adsr.tick();
374             }
375         }
```

Figure 3.26: Main processing loop of **TheStringPhone**.

Synthesis Tool Kit (STK) by Perry Cook [1]. The code for the tick function is shown in figure 3.27.

```
115 inline StkFloat ADSR :: tick( void )
116 {
117   switch ( state_ ) {
118
119   case ATTACK:
120     value_ += attackRate_;
121     if ( value_ >= target_ ) {
122       value_ = target_;
123       target_ = sustainLevel_;
124         state_ = DECAY;
125     }
126     lastFrame_[0] = value_;
127     break;
128
129   case DECAY:
130     if ( value_ > sustainLevel_ ) {
131       value_ -= decayRate_;
132       if ( value_ <= sustainLevel_ ) {
133         value_ = sustainLevel_;
134         state_ = SUSTAIN;
135       }
136     }
137     else {
138       value_ += decayRate_; // attack target < sustain level
139       if ( value_ >= sustainLevel_ ) {
140         value_ = sustainLevel_;
141         state_ = SUSTAIN;
142       }
143     }
144     lastFrame_[0] = value_;
145     break;
146
147   case RELEASE:
148     value_ -= releaseRate_;
149     if ( value_ <= 0.0 ) {
150       value_ = 0.0;
151       state_ = IDLE;
152     }
153     lastFrame_[0] = value_;
154
155   }
156
157   return value_;
158 }
```

Figure 3.27: tick() function from the stk::adsr class.

[1] https://ccrma.stanford.edu/software/stk/

The function is composed of switch-case statements with a case for each of the four ADSR stages. One can see that this is a linear ADSR because of the way the current value is incremented or decremented in the attack, decay and release stages. This can be seen in line 120, 131 and 148 where the rate is continuously added as a constant resulting in a linear increase/decrease. The amount of increase/decrease is controlled by the *attackRate*, *decayRate* and *releaseRate* variables, which are set by the user using sliders on the MIDI-keyboard. Sound examples of two different ADSR settings can be found in the resource folder ('long_attack.wav') and ('short_attack.wav')

3.4 The peak compressor

The output of the ADSR is sent to a compressor to limit the signals dynamic range in order to make it sound fuller and louder.

3.4.1 Dynamic range control

Compression is a part of the family of dynamic range techniques, which transform the amplitude of signals. An audio signals dynamic range refers to the difference between the softest and loudest parts of the signal. In a compressor the amount of amplification is controlled by its signal input and when the signal input rises above a specified threshold the compressor attenuates it. We can graph the transform function of a compressor in order to show the relationship between its input and output levels - once the input signal exceeds a specified threshold it is attenuated according to a chosen compression ratio. An example can be seen in Figure 3.28 and show different input/output compression ratios, which determine the amount of compression applied.

Instantaneous compressor response is usually not sought because it introduces distortion to the signal [38]. Because of this a compressor usually applies the gain reduction smoothly instead of instantaneously making it a nonlinear time-dependent system [26]. Controls are provided for the time constants; *attack* controls the time it takes the gain reduction to kick in and *release* defines the time it takes to bring the gain back up to the normal level. A compressor is usually also equipped with a *make up gain* control at the compressor output, which allow for matching the input and output levels. Without a make up gain the output signal will always sound quieter than the uncompressed signal, because the compressor only attenuates a signal and never boosts it. The threshold-determined point where the compression kicks in is referred to as the *the knee*. If the transition is sharp we refer to it as a *hard knee* and the result is a noticeable compression. A softer transition where the ratio gradually grows is called a *soft knee* and result in a less perceptible compression effect.

Simplified speaking a compressor consist of three parts:

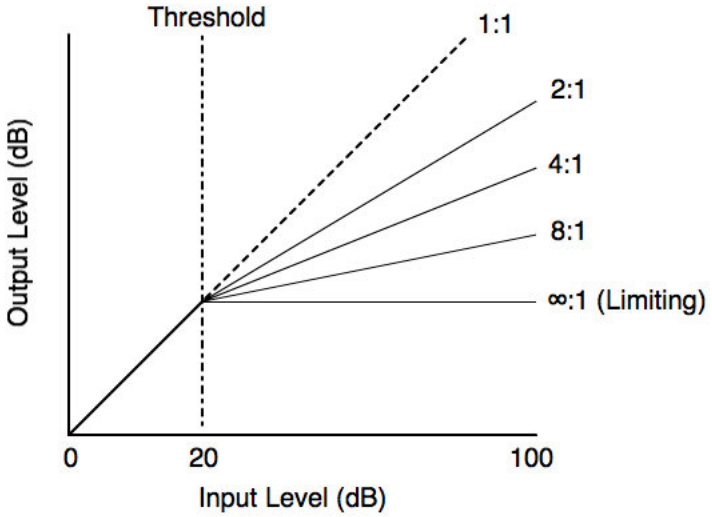

Figure 3.28: A compressor transfer function. (Illustration taken from: http://www.practical-music-production.com/images/audio-compressor.jpg)

- a *level detector* that monitors the amplitude of the input signal and reacts when the amplitude exceed the threshold point.
- a *gain computer* that determines the amount of gain reduction applied to the incoming signal.
- a *gain stage* that is responsible for attenuating the input signal by a varied amount of decibels, once it exceeds the threshold point.

When a signal enters the compressor it is split in two copies; one is sent to the gain stage and the other to a side-chain where the gain computer applies the required gain reduction to the gain stage.

There exist two different architectures for gain computing; feed-back and feed-forward compression.

The feed-back architecture (Figure 3.29) derives a control signal from the compressor output signal. Since the gain computer is fed with an already compressed signal the gain computer only needs to be accurate over a small range, however such a system cannot achieve true limiting, since this would need infinite negative amplification to calculate the control voltage. Feed-back compressors are desirable for their fast attack and release times [2] and are often used for analog implementation.

The feed-forward system (Figure 3.30) derives its control signal from the input signal and therefore the gain computer has to be accurate over the whole dynamic range in contrast to a feed-back compressor. With this architecture true limiting can be achieved.

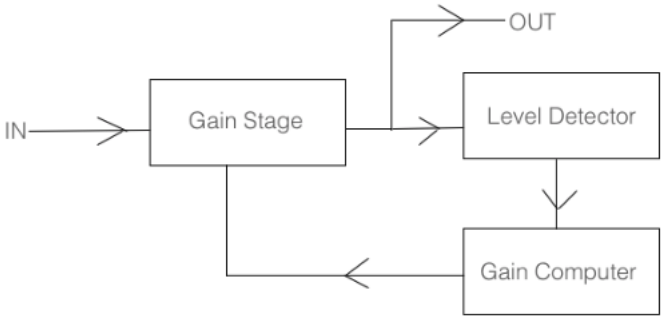

Figure 3.29: Feed-back compression architecture.

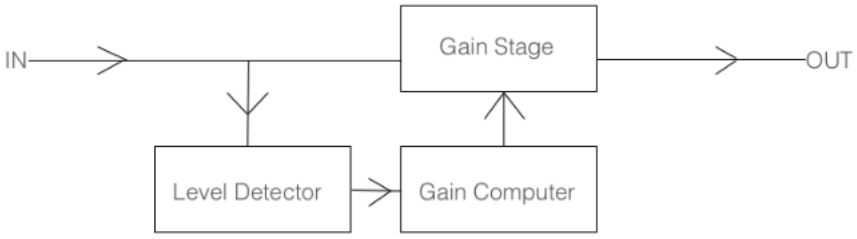

Figure 3.30: Feed-forward compression architecture.

TheStringPhone uses feed-forward compression, as can be seen when examining the code in Figure 3.31. Here the input signal is sent to the level detector and not the output of the gain stage as in the feed-back architecture.

```
44 void Compressor::process(float input, float &output)
45 {
46     float level_estimate, log_level;
47
48     //////////  LEVEL DETECTOR  ////////////////////
49     peak_detector.process(input, level_estimate);
50
51     ////////////  GAIN COMPUTER  ///////////////////
52     log_level = dB(level_estimate);
53
54     gainval = 1.0f;
55     if(log_level > logthresh)
56     {
57         dbgainval = logthresh - log_level + (log_level - logthresh) / comp_ratio;
58         gainval = dB2lin(dbgainval);
59     }
60
61     ////////////// GAIN STAGE ///////////////////////
62     output = input*gainval;
63 }
```

Figure 3.31: Feed-forward peak compressor code.

In the following we will have a look at the different stages involved in the compressor and show how each of them has been implemented. We will hereafter show how it effects the signal.

3.4.2 Level detector

The detector determines the compressors 'dynamics' [21] i.e. the way in which the gain reduction follows the signal envelope. The attack and release times are usually introduced through a smoothing filter [26], which can be simulated as a digital one-pole filter

$$y(n) = \alpha y(n-1) + (1-\alpha)x(n) \tag{3.12}$$

where α is the filter coefficient, x(n) the input, and y(n) the output. Note that this is the same digital one-pole filter mentioned in section 3.3 about exponential attack, decay and release curves of envelope generators. The reason why it is typically used, is because it can be seen as an iterative solution to an exponential function. This can be seen if one examines the filter's step response:

$$y(n) = 1 - \alpha^n \tag{3.13}$$

In the case of the compressor, the time constant α is defined as the time it takes the system to reach $1 - \frac{1}{e}$ of its final value (approx. 63%). Thus we have

$$\alpha = e^{-1/(\tau f_s)} \tag{3.14}$$

The decay is usually defined as the time it takes the compressor to return to $1 - (1 - \frac{1}{e})$ of its uncompressed state. Figure 3.32 shows the code which calculates the attack and release time of the compressor.

```
26
27      void setTauRelease(float tauRelease, float fs) {
28          a1_r = exp( -1.0 / ( tauRelease * fs ) );
29          b0_r = 1 - a1_r;
30      }
31
32      void setTauAttack(float tauAttack, float fs) {
33          a1_a = exp( -1.0 / ( tauAttack * fs ) );
34          b0_a = 1 - a1_a;
35      }
36
```

Figure 3.32: Attack and release functions of the peak compressor.

The signal level may be determined by either peak sensing (the signals absolute value) or based on a measurement of the RMS (RMS-sensing). The compressor implemented in **TheStringPhone** makes use of peak sensing.

In analog peak compressors the attack and release rate are coupled [26]. This means that the release rate is influenced by the attack rate. In a digital design we can eliminate this problem by implementing separate smooth filters for attack and release:

$$y(n) = \begin{cases} \alpha_A y[n-1] + (1-\alpha_A)x_[n] & x[n] > y[n-1] \\ \alpha_R y[n-1] + (1-\alpha_R)x_[n] & x[n] \leq y[n-1] \end{cases} \tag{3.15}$$

This can be seen in Figure 3.33. Line 44 and 46 are more compact ways of writing equation 3.15. for attack and release respectively. Line 43 check whether we are attacking or releasing. If the signal is larger than the previous level estimate, then we are in the attack stage and if it is smaller we are in the release stage.

```
42      void process (float input, float& output) {
43          if ( fabs( input ) > levelEstimate )
44              levelEstimate += b0_a * ( fabs( input ) - levelEstimate );
45          else
46              levelEstimate += b0_r * ( fabs( input ) - levelEstimate );
47          output = levelEstimate;
48      }
49  };
```

Figure 3.33: Main processing loop for the level detector of the compressor.

If one wants to implement RMS compression, the following difference equation can be used:

$$y_L^2(n) = \alpha y_L^2(n-1) + (1-\alpha)x_L^2(n) \tag{3.16}$$

which is basically the same as equation 3.12, but with each sample squared. The RMS detector is more related to the perceived signal loudness[48] and is useful when we are interested in a smoothed average of the incoming signal.

3.4.3 Gain computer

This is the stage that generates a control signal to determine the gain reduction that will be applied to the signal. Figure 3.34 shows the static compression curve which relates input level to output level.

When the signal is below the threshold of compression the input signal equals the output signal:

$$l_O = l_I \tag{3.17}$$

When the input, on the other hand, exceeds the threshold of compression the output signal is related to the input signal in the following way:

$$l_O = l_T + \frac{l_I - l_T}{\rho} \tag{3.18}$$

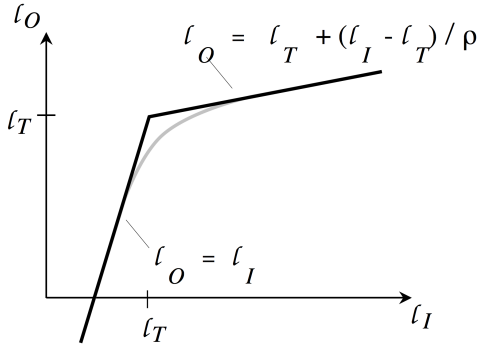

Figure 3.34: Static compression curve of the compressor.

where ρ is the compression ratio. Note here that the values of l are in dB and therefore in log space. Next we find the amount of gain we need to reduce l_O with:

$$g(l_I) = l_O - l_I \tag{3.19}$$
$$= l_T + \frac{l_I - l_T}{\rho} - l_I \tag{3.20}$$
$$= l_T - l_I + \frac{l_I - l_T}{p} \tag{3.21}$$
$$= \frac{l_I - l_T}{1/\rho - 1} \tag{3.22}$$

Converting equation 3.22 to linear space yield the following equation for a gain computer for a feed-forward compressor:

$$\Phi_F = \left(\frac{\lambda_I}{\lambda_T}\right)^{\frac{1}{\rho-1}} \tag{3.23}$$

where λ_I is the estimated input signal and λ_T is the threshold value and ρ is the compression ratio. Figure 3.35 shows the code for the gain computer implemented in **TheStringPhone**. Recognize line 57 as equation 3.21. The reason log space has been used is because it requires simpler arithmetic operations, which makes it more efficient for implementation.

Figure 3.31 shows the main process function of the peak compressor.

If one wanted to implement the feedback architecture a slightly different approach must be taken:

$$l_I = l_T + (l_O - l_T)p \tag{3.24}$$
$$g(l_I) = l_O - l_I \tag{3.25}$$
$$= l_O - l_T - (l_O - l_T)p \tag{3.26}$$
$$= (l_O - l_T)(1 - \rho) \tag{3.27}$$

```
51      //////////////  GAIN COMPUTER  ///////////////////
52      log_level = dB(level_estimate);
53
54      gainval = 1.0f;
55      if(log_level > logthresh)
56      {
57          dbgainval = logthresh - log_level + (log_level - logthresh) / comp_ratio;
58          gainval = dB2lin(dbgainval);
59      }
```

Figure 3.35: Code for the compressor's gain computer.

Converting equation 3.27 to linear space yields the following equation for a gain computer for a feedback architecture:

$$\Phi_B = \left(\frac{\lambda_O}{\lambda_T}\right)^{1-\rho} \tag{3.28}$$

where λ_I from the feed-forward architecture has been replaced by λ_O, which is the estimated output signal.

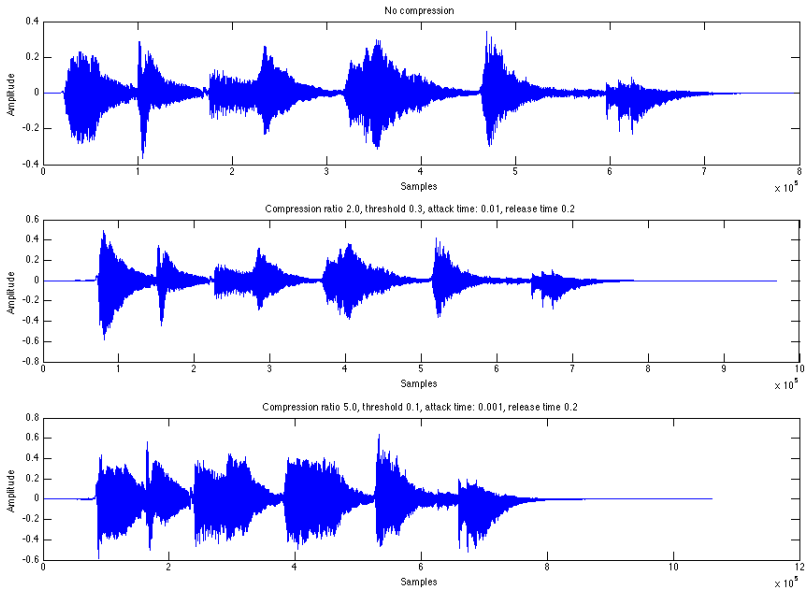

Figure 3.36: A vocal snippet sent through **TheStringPhone** with no compression (top), with compression ratio 2.0, threshold 0.3, attack time 0.01 and release time 0.2 (center) and with compression ratio 5.0, threshold 0.1, attack time 0.001 and release time 0.2 (bottom).

Figure 3.36 shows the same vocal snippet sent through the **TheStringPhone** with its compressor having different settings. In the top one no compression is done. The center shows the vocal snippet with a small amount of

compression and the bottom shows a more heavy compression. The sound files for each compression setting can be found in the resource folder. Due to its faster attack time, the compressor in the bottom figure, kicks in faster after each peak, making the dynamic range smaller. When applying a makeup gain, it keeps the peaks the same level, while enhancing the lower parts. This makes **TheStringPhone** sound louder and fuller which was the original intend.

3.5 FDN Reverb

After the signal has been compressed it is processed by a feedback delay network (FDN) reverberator. This section will explain the theory of digital reverberation and filter delay networks.

3.5.1 Room acoustics and reverberation

Reverberation is a naturally occurring acoustical effect that becomes evident when listening to sound in spaces with ceilings and reflective surfaces. Such spaces reinforce the sound emitted by thousands of closely spaced echoes bouncing off the surfaces. Therefore sound arrives at the listener in stages. A sound source will have a *direct path* where it propagates in a straight line from source to listener [35]. The direct path will be followed by *early reflections* from nearby objects and surfaces (such as first and second order reflections). The direct path reveals the direction of the source, whereas the early reflections convey a sense of the geometry and materials of the space [78]. The time delay of each reflection is proportional to the time it takes the impulse to travel from the sound source to the walls and then to the microphone [44]. The amplitude of the reflection is inversely proportional to the distance traveled and directly proportional to the size of the reflecting surface, and inversely proportional to the material the surface is made of. Over time, there are so many reflections that the sound ends up being composed of plane waves distributed with uniform randomness in all directions. This stage is called the *late reverberation* and forms the tail of the reverb impulse response which is arguably indistinguishable from Gaussian noise with an evolving color and level[3]. Late reverberation tells us something about the size of the reverberant space together with the absorbing power of the materials present [65]. The decay time of the late reverberation is called the *reverberation time* (denoted t_{60}), and refers to the time it takes the response to exponentially reach a 60-dB decay level from its peak amplitude (1/1000 of its peak energy). The sound quality of a specific architecture depends on the details of its reverberation impulse response [55]. The *impulse response* of a room is its output when presented with a brief input signal e.g. the unit impulse $\delta(n)$. The impulse response is also used to measure the reverberation of a room and Figure 3.37 shows an example of a reverberant

impulse response with an indication of the direct signal, the early reflections and the random distributed late reverberation. It is these three stages that artificial reverberation tries to simulate.

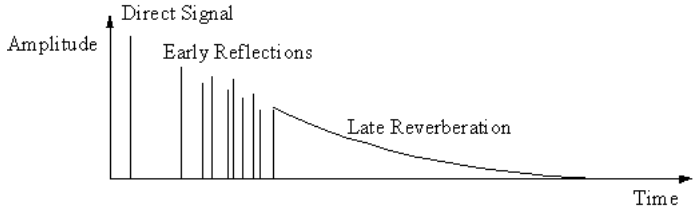

Figure 3.37: Schematic example of a room impulse response. (Illustration taken from: [22])

3.5.2 Artificial Reverberation

There are overall two approaches to artificial reverberation; a physical approach (convolution based algorithms and physical room models) and a perceptual approach (delay networks) [78]. The first tries to recreate artificially the exact reverberation of a room. This approach has the advantage of offering a direct relation between physical specifications of the room and the resulting reverberation [23]. The physical approach is usually achieved by convolving the impulse response of a room with a dry source signal. The impulse response can be a recording from a real room or simulated from a geometric model of a virtual one. This approach is typically very computational expensive and is not well suited for real time computation. However, with the increasing computational power real time convolution is possible as can be seen in a project such as the recreation of the acoustics of Hagia Sophia in the Bing Concert Hall at Stanford[1]. Several convolution based commercial reverberation units have also emerged the last years e.g. the *convolution reverb*[2] effect of Ableton Live and *IR-L convolution reverb*[3] developed by waves. Despite this, convolution is still computationally expensive and rather inflexible [23] as it does not allow for an easy way to achieve real-time parametric control of the resulting reverberation's perceptual characteristics. Therefore most artificial reverberation algorithms take the approach which attempts to model real room reverberation by reproducing only the salient characteristics of the room [22]. The idea here is that each perceptual attribute can be associated with a physical feature of the impulse response and what we want is to construct a filter that reproduces these attributes. With this approach the algorithm will provide real-time control of the perceptually relevant parameters. **TheStringPhone** uses

[2]https://www.ableton.com/en/packs/max-live-essentials/
[3]http://www.waves.com/plugins/ir-l-convolution-reverb

a Feedback Delay Network reverberator, which belongs to the category of perceptual oriented reverberators and we will therefore only focus on this approach in the following.

The hardest problem in creating inexpensive artificial reverberation is to simulate the desired qualities of the late reverberation, which is a smooth (but not too smooth) decay and a smooth (but not too regular) frequency response. Acquiring a exponential decay is no problem. The hard issue lies in making it smooth and free of "flutter", "beating" or other unnatural irregularities, which happens when the echo density is too low. Echo density is a measure of echoes per second and, in general, a smooth decay results when the echo density is sufficiently high. Focus of much of the research in artificial reverberation has been to achieve a high enough echo density. Feedback delay networks are presently considered to be among the best choices for high-quality artificial reverberation [71] and has therefore been used in **TheStringPhone**. The subject of artificial reverberation was initiated by Manfred Schroeder in the 1960s and in the following we will briefly describe Schroeders original algorithms based on recursive comb and allpass filters before we dive into feedback delay networks.

Schroeder's unit reverberators

During the 20th century a number of electro-mechanical reverberation devices were developed. These included tape delays, spring reverberators and reverberation plates. These typically had a comb filter like response which introduced an undesired color to the reverberation. Then in 1961 Schroeder introduced the first idea of artificial reverberation based on digital comb filters and allpass filters [67]. He introduced allpass filters in order to solve the problem of the comb filters' frequency response not being flat. He proposed various designs using both these filters which he referred to as *unit reverberators*. Figure 3.38 shows the comb filter used, which has the following transfer function

$$H(z) = \frac{z^{-\tau}}{1 - gz^{-\tau}} \tag{3.29}$$

The filter contains a feedback loop in which an input signal is delayed by τ samples and scaled by a gain g, and then routed back to be added to the latest input signal. A small delay time (typically below 10 ms) results in spectral effects where the delayed version is not heard as a separate delay. This is due to the integration time of the ear. However, when the delay time is larger than 10 ms the filter creates a series of exponentially decaying echoes instead. This filter introduces coloration to the output of the reverberator due to the fact that the amplitude-frequency response is not flat [67]. In order to solve this, Schroeder added $-g$ times the input into the output of

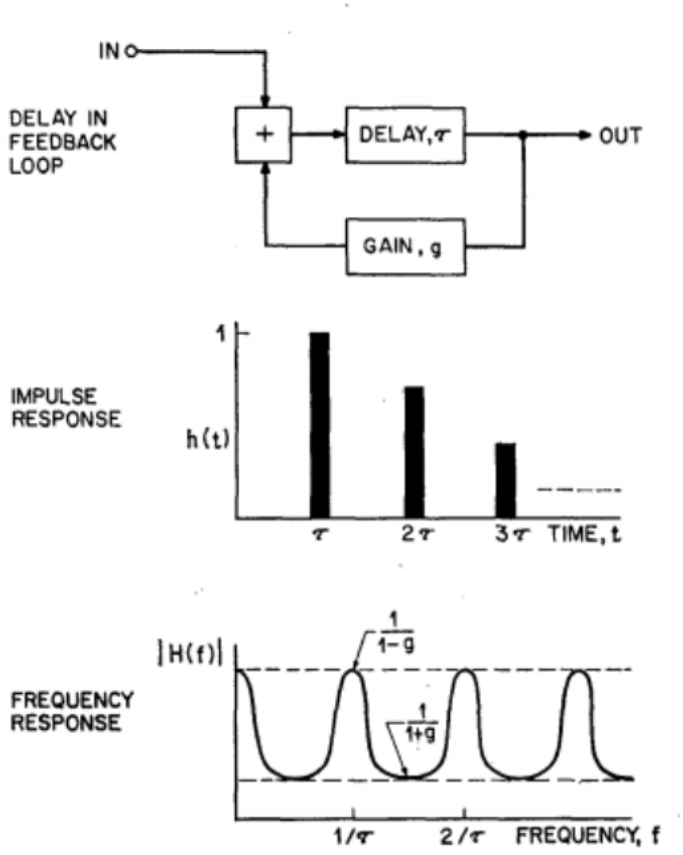

Figure 3.38: Shroeder's comb filter. (Illustration taken from: [67])

the delay and changed the comb filter into an allpass filter [4] obtaining the following transfer function:

$$H(z) = \frac{z^{-\tau} - g}{1 - gz^{-\tau}} \tag{3.30}$$

The spectral result can be seen in Figure 3.39.

Allpass filters transmit all frequencies equally well, but sharp transient signals are colored by the filter because it introduces frequency-dependent delays [50]. With a delay time between 5 and 100 ms the allpass filter results in a series of exponentially decaying echo pulses just like a comb filter [65]. Since both the comb filter and the allpass filter are able to generate a series of decaying echoes they can be used in seriel or parallel to create echo densities. When connected in parallel their echoes will add together, but when connected in series each echo generated in one unit triggers a series of echoes in the next (with the total number of echoes being the product of the number of echoes of each unit), which is just what we need for creating

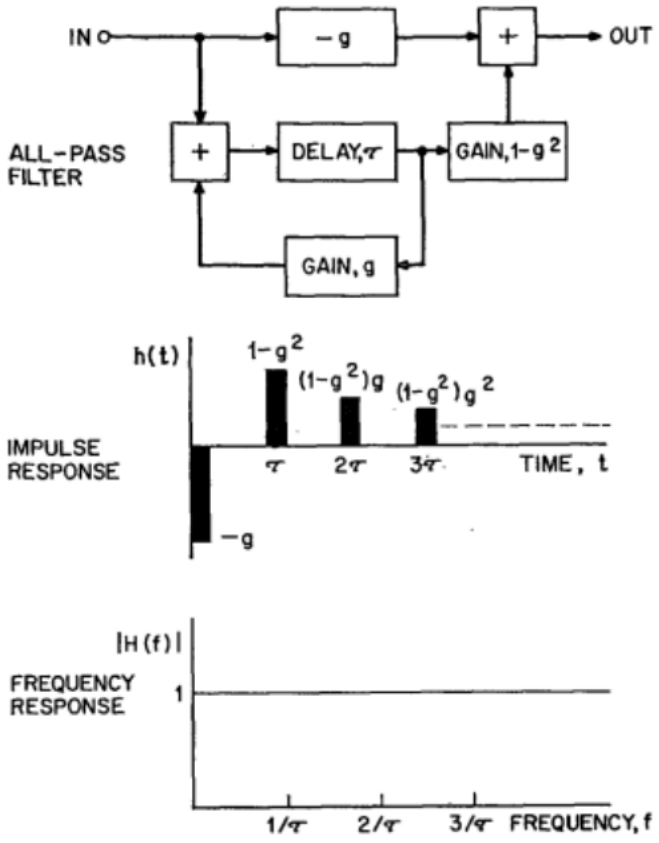

Figure 3.39: Schroeder's allpass filter. (Illustration taken from: [67])

echo density.

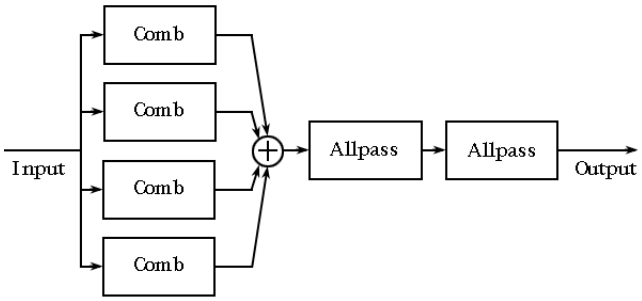

Figure 3.40: Schroeder's original reverberator, which consist of parallel comb filters fed into two allpass filters.

Schroeder connected comb filters in parallel in order to avoid spectral anomalies. Comb filters are not good for connecting in series because the only frequencies that will pass are those that correspond to peaks in both

comb filter responses. The parallel comb filter architecture initiate a train of echoes that are summed and fed to a series of two allpass filters in order to form the reverberated output signal (as can be seen in Figure 3.40). The serial connection of allpass filters is due to the phase distortion they introduce [65]. Nonuniform amplitude response due to phase cancellation may occur by connecting them in parallel.

The output is obviously dependent on the delay times used, which determine the spacing of the echoes and the amplitude factors, which determine the decay rate. Note that it is important to choose delay times that are relatively prime to one another for natural sounding reverberation [51]. This is because delay line lengths that are divisible by any number N will have coinciding echoes at multiples of N causing a sensation of discrete echoes in the decay. The idea of using comb filters and allpass filters for reverberation is efficient in creating reasonable simulation of global reverberation, but it lacks the detailed acoustic properties of an actual space.

3.5.3 Feedback delay networks

In [72] J. Stautner and M. Puckette introduced the idea of *feedback delay networks* (FDN) as structures for artificial reverberation, which can be seen as a generalization of Schroeder's parallel comb filter design[23]. The strongest limitation of a parallel comb filter architecture lies in the difficulty of obtaining sufficient echo density [23] with a reasonable number of unit filters [34][22]. Feedback delay networks on the other hand are able to generate much higher echo densities, given a sufficient number of non-zero feedback coefficients and pure delay lengths [14].

The structure the FDN is based on a set of N delay lines connected in a feedback loop through a *feedback matrix* **A** (Figure 3.41).

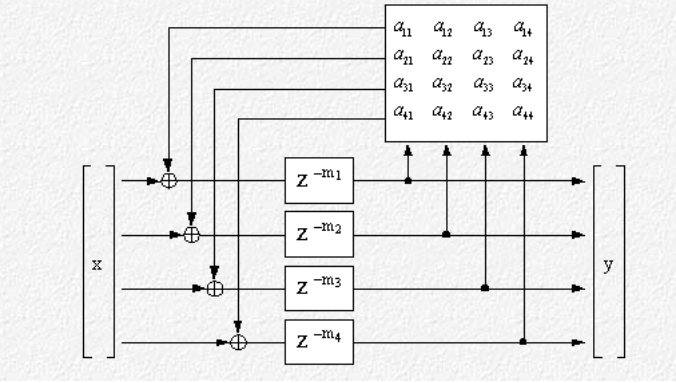

Figure 3.41: Stautner and Puckettes four channel feedback delay network. (Illustration taken from: [72])

Each coefficient a_{mn} corresponds to the amount of signal coming out of

delay line n sent to the input of delay line m. Some important points they make about the system is:

- Stability is guaranteed if the feedback matrix $\mathbf{A}$ is chosen to be the product of a unitary matrix and a gain coefficient g, where $g < 1$. The matrix they suggest is

$$A = g\frac{1}{\sqrt{2}}\begin{pmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \tag{3.31}$$

  where g control the reverberation time.

- Absorptive losses can be simulated by placing a lowpass filter in series with each delay line.

- It is possible to customize the early reverberant response by injecting the input signal appropriately into the interior of the delay lines.

They note that tonal coloration is present in the late decay and suggest randomly varying the delay lengths to reduce the coloration. In [34] Jot and Chaigne further generalize Stautner and Puckettes system to the one seen in Figure 3.42. Their structure has two important properties

- The reverberator can be designed with arbitrary time and frequency density while simultaneously guaranteeing absence of tonal coloration in the late decay [23].

- The reverberator can be specified in terms of the desired reverberation time $T_r(\omega)$ and frequency response envelope $G(\omega)$.

3.5.4 Implementation

When an input is given to the Jots General Delay Network, it is split in N copies. Each of these copies are sent to a delay line and an absorption filter. The output is hereby mixed by a mixing matrix and fed back into the input of the filter delay network. The following sections will describe the implementation of each step in depth.

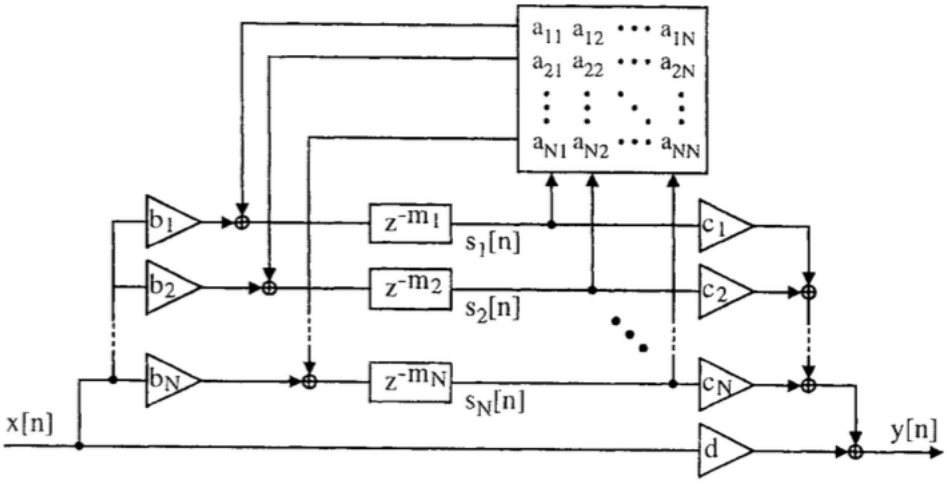

Figure 3.42: Jots general feedback delay network. (Illustration taken from: [34])

```
float dlens[kNumDelays]={2023,2153,2291,2438,2595,2761,2939,3127,3328,3542,3769,4011};
```

Figure 3.43: Array of the delay lengths of each delay line of the FDN Reverb.

Delay lines

The implementation of the delay line is the same as the one used in the implementation of the Karplus-Strong algorithm. Please refer to section 3.1.3 for any further description. The delay lines are stored in an array and when the FDN reverb is initialized the delay length of each delay is set as defined by an array of delay lengths as shown in Figure 3.43.

The delay lengths are typically mutually prime [67], as this reduces the effect of many peaks piling up on the same frequency, leading to a more dense and uniform decay [52]. The delay lengths should be chosen to ensure a sufficiently high modal density in all frequency bands [71]. This makes sure that no specific frequencies are accentuated which would cause "ringing tones", "flutter" or uneven amplitude modulation in the late reverberation impulse response.

Absorption filter

Absorptive filters are associated with each delay in the system in order to effect a frequency dependent reverberation time [34].

The filters serve the purpose of simulating the absorption caused by walls and other absorbent objects found in rooms. Because shelf filters allows separate gain control for both high and low frequencies, using these

instead of simple lowpass filters is an easy way of giving extra control over the reverb [71]. An analog shelf filter can be constructed by combining the transfer function of an analog lowpass filter:

$$h(s) = g\frac{1}{s/\rho + 1} \tag{3.32}$$

and an analog highpass filter:

$$h(s) = g\frac{s}{s + \rho} \tag{3.33}$$

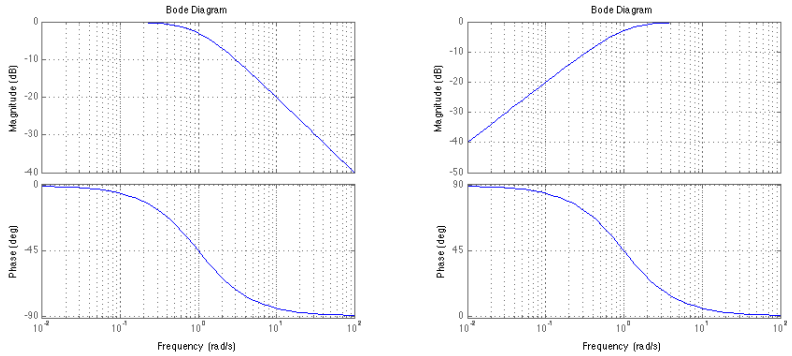

Figure 3.44: Bode plot of an analog lowpass filter with gain $g = 1$ and cutoff frequency $\rho = 1rad/s$ (left) and an analog highpass filter with gain $g = 1$ and cutoff frequency $\rho = 1rad/s$ (right).

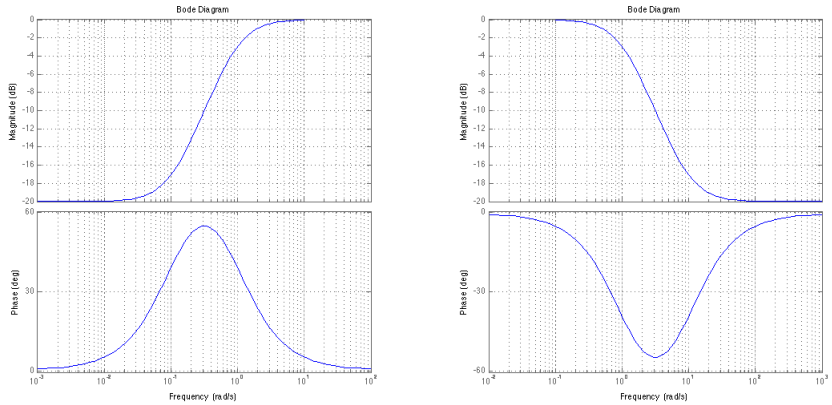

Figure 3.45: Bode plot for a shelf filter with $g_l = 0.1$, $g_h = 1$ and $\rho = 1rad/s$ (left) and a shelf filter with $g_l = 1$, $g_h = 0.1$ and $\rho = 1rad/s$ (right).

where ρ is the cutoff radian frequency and g is the gain. Bode plots of these two filters can be seen in Figure 3.44. By combining the two filters one can generate a shelf filter. This is done by simply adding a lowpass filter of gain g_l and transition frequency ρ and a highpass filter of gain g_h:

$$h(s) = \frac{g_h s/\rho + g_l}{s/\rho + 1} \tag{3.34}$$

Figure 3.45 shows a bode plot for a shelf filter with $g_l = 0.1$, $g_h = 1$ and $\rho = 1rad/s$ (left) and a shelf filter with $g_l = 1$, $g_h = 0.1$ and $\rho = 1rad/s$ (right).

In order to go from the analog to the digital domain the transfer function of the shelf filter needs to be transformed from a continuous-time system representation to a discrete-time representation. In other words the transfer function of the shelf filter needs to be converted from the s-domain to the z-domain. This can be done using the bilinear transform, where one replaces s in Equation 3.34 by:

$$s = \frac{2}{T_d}\frac{1 - z^{-1}}{1 + z^{-1}} \tag{3.35}$$

where T is the sample period $1/fs$. So given a continuous-time transfer function $H(s)$ we apply the bilinear transform as:

$$H(z) = H_c\left(\frac{2}{T_d}\frac{1 - z^{-1}}{1 + z^{-1}}\right) \tag{3.36}$$

where $H(z)$ denotes the discrete time-transfer function and H_c denotes the continuous-time transfer function. This maps the entire $j\omega$-axis in the s-plane onto one revolution of the unit circle in the z-plane. One must note that this results in a non-linear mapping from $-\infty \leq \Omega \leq \infty$ to $-\pi \leq \omega \leq \pi$, which means that the frequency axis is going to suffer from a nonlinear compression [54] as shown in Figure 3.46.

In order to compensate for this it is common to do pre-warping which ensures that the cutoff frequency is correct after the transformation. This can be done using the following equation:

$$\omega = 2arctan(\Omega T/2) \tag{3.37}$$

where ω is the frequency in the digital domain and Ω is the frequency in the analog domain. Figure 3.47 shows the code for implementing the bilinear transform.

The variables $g0$, $g1$ and $transition$ in line 314, 318 and 319, are the low frequency gain (g_l), high frequency gain (g_h) and the transition frequency (ρ). $g0$ and $g1$ is calculated based on the desired decay times for low frequencies (T_{60}^0) and high frequencies (T_{60}^∞), which is specified by the user. In

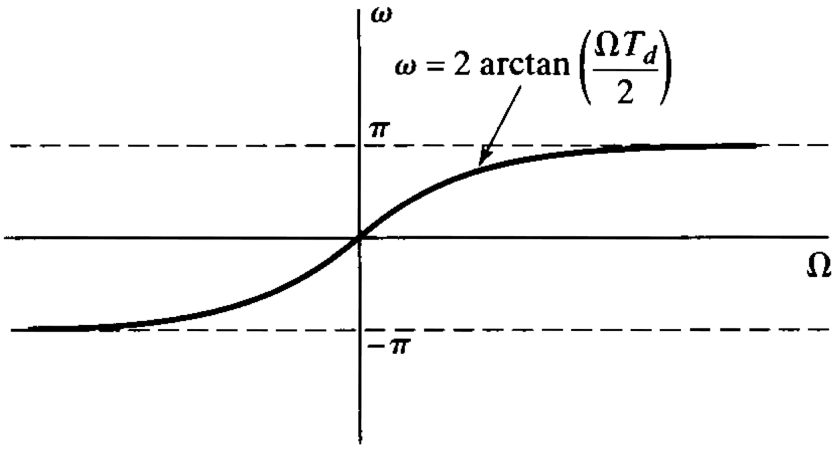

Figure 3.46: Mapping of the continuous-time frequency axis onto the discrete-time frequency axis by bilinear transformation. [54]

```
312     // Pre-warp center frequency
313     double T = 1.0/fs;
314     double fc = (2.0/T)*tanf(transition*T/2.0);
315     double wc = fc*2*pi;
316
317     // ####### Define coefs
318     b0 = g0;
319     b1 = g1*1.0/wc;
320     a1 = 1.0/wc;
321
322     // ####### Bilinear transform #########
323     double K = 2.0/T;
324
325     norm = a1*K+1;
326     b0z= (b1*K+b0)/norm;
327     b1z= (b0-b1*K)/norm;
328     a1z= (1-a1*K)/norm;
329
330     // ######## Return coefs #########
331     *(pcoefs) = b0z;
332     *(pcoefs+1) = b1z;
333     *(pcoefs+2) = a1z;
```

Figure 3.47: Code implementing the bilinear transformation.

order to be able to specify a desired decay time for low and high frequencies we need an expression for g_l and g_h as a function of T_{60}^0 and T_{60}^∞.

In general the decay time can be written as $e^{-t/\tau}$. Since T_{60} is defined as the time needed for the signal to reach -60dB, i.e 0.001 in amplitude we have:

$$e^{-T_{60}/\tau} = 0.001 \rightarrow T_{60} = -\tau log(0.001) \tag{3.38}$$

Now, we want to take into account the whole feedback loop which in-

cludes both a delay line of length T and a shelf filter h. For each loop the signal is passed through the filter. This means that for a wave of frequency ω, it will have a gain of $h(\omega)$ every T seconds. So after nT seconds the gain is $h(\omega)^n$. The decay time t can then be extrapolated as being $h(\omega)^{-t/T} = e^{tlog(h(\omega))/T}$, which in turn leads to

$$\tau = -T/log(h(\omega)) \quad \text{or} \quad h(w) = e^{-T/\tau} \tag{3.39}$$

If we take the values at $\omega = 0$ and $\omega \to \infty$ we get

$$g_l = e^{log0.001T/T_{60}^0} \tag{3.40}$$

$$g_h = e^{log0.001T/T_{60}^\infty} \tag{3.41}$$

Figure 3.48 shows the code for calculating g_l and g_h.

```
// low frequency asymptote (determined by t60low)
g0 = exp(roundTrip*log(0.001)/t60low);
// high frequency asymptote (determined by t60high)
g1 = exp(roundTrip*log(0.001)/t60high);
```

Figure 3.48: Code for calculating the gain of the shelf filters.

where $roundTrip$ is $delaylength/sampleRate$ in seconds. These are then used to define the coefficients of the shelf filter as can be seen in Figure 3.49

```
169     // filter coefficients
170     b0 = g0;
171     a1 = 1.0f/rho;
172     b1 = a1*g1;
```

Figure 3.49: Code for defining the coefficients of the shelf filter.

where rho is the transition frequency, $g0$ is the low frequency gain and $g1$ is the high frequency gain. The shelf filters are designed each time the T_{60}^0, T_{60}^∞ or the transition frequency is changed as can be seen in Figure 3.50

```
for (i=0; i<kNumDelays ; i++){
    designShelf(pcoefs,dlens[i], TransitionValue, T60LowValue, T60HighValue);
    fbfilt[i].setCoefs(coefs);
}
```

Figure 3.50: For loop which designs the shelf filters.

Where *fbfilt[i]* is an array containing all the filters.

Mixing matrix

The mixing matrix serves the purpose of simulating the mixing of all the reflections found in a room. The choice of matrix will influence the echo

density and general sound of the reverb. In general we want a matrix which creates a rapidly increasing echo density. Mixing matrices with the most off-diagonal energy will do this. In order to see why we can inspect Figure 3.42. We see that each delay line is sent through the mixing matrix. The first column in the matrix determines the amount of the first delay line that is fed back to each of the delay lines. So a_{11} is fed back to the first delay line, a_{21} is fed to the second delay line and so on. Having a matrix with a lot of off diagonal energy ensures that energy from one delay line is sent to the other delay lines. Having a mixing matrix with only diagonal entries (like the identity matrix) will on the other hand result in no mixing. This is because each delay line will only be fed back to itself resulting in a parallel allpass structure.

Evaluating the reverb

In order to evaluate the reverb from a engineering point of view, it is useful to measure the echo density of the reverberation with a given matrix and delay lengths. As explained, echo density, is commonly expressed in terms of the number of reflections per second. This measurement is however not always meaningful. It is unclear how to best define a reflection, and sampling rate becomes a limiting factor in counting reflections. This has been as addressed by Abel and Huang in [3] and they propose another measure, which uses the fact that once a reverberator is sufficiently mixed the impulse response taps takes on Gaussian distribution. Using a sliding window over the impulse response of the reverberator, one can therefore count the number of taps outside the standard deviation for the window. This count is normalized by the count expected for Gaussian noise and the result is referred to as the *echo density profile*:

$$\eta(t) = \frac{1/\text{erfc}(1/\sqrt{2})}{2\delta + 1} \sum_{\tau=t-\delta}^{t+\delta} 1\{|h(\tau)| > \sigma \tag{3.42}$$

where $h(t)$ is the impulse response of the reverberator, $2\delta + 1$ is the window length in samples, σ is the standard deviation:

$$\sigma = \left[\frac{1}{2\delta + 1} \sum_{\tau=t-\delta}^{t+\delta} h^2(\tau) \right]^{1/2} \tag{3.43}$$

$1\{\cdot\}$ is a function returning 1 when its argument is true and 0 otherwise, and $\text{erfc}(1/\sqrt{2}) = 0.3173$ is the expected fraction of samples lying outside a standard deviation from the mean for a Gaussian distribution [3].

Figure 3.51 show MATLAB code for calculating the normalized echo density profile. Line 11 calculates the standard deviation and corresponds to Equation 3.43 and line 12 corresponds to Equation 3.42.

```
1  function n = echoDensityProfile(h, window_length, fs)
2      ws = (window_length/1000)*fs; % Window length in samples
3      N = floor(ws/2-1); % Number of windows
4
5      window = hann(2*N+1)/sum(hann(2*N+1)); % Normalized hanning window
6
7      n = zeros(length(h),1); % Output vector
8
9      for t = N+1:length(h)-N;
10         index = t-N:t+N; % Create index
11         stddev = sqrt(mean(h(index).^2)); % Standard deviation
12         n(t) = window'*(abs(h(index)) > stddev) / 0.3173; %Apply window and calc NED
13     end
14 end
```

Figure 3.51: MATLAB code for calculating the normalized echo density profile.

Figure 3.52 shows the echo density profile of our FDN reverberator using four different mixing matrices. These are a hadamard mixing matrix, 2 randomly generated skew-hermitian matrices (one with more diagonal energy than the other) and for reference the identity matrix. For all four plots the T_{60}^{0} is set to 2 seconds and T_{60}^{∞} is set to 0.5 seconds. A window length of 20 milliseconds has been used.

We want to choose a feedback matrix for **TheStringPhone** with a lot of off-diagonal energy, since this results in a smooth decay. We see that the skew-Hermitian matrix with a decay of 0.5 (the one with most off diagonal energy) and the Hadamard matrix have the fastest growing echo densities. It relates to the fact that the off diagonal energy in their respective mixing matrices are more energetic, encouraging the mixing of the FDN channels and the randomization of the echoes. The skew-Hermitian matrix with a decay of 1 has more diagonal energy, which result in a slower mixing. The identity matrix does not mix the channels at all thereby resulting in no mixing. When inspecting the normalized echo density function of the identity matrix there is still a slow increase and is due to the smearing of the echoes rather than mixing.

If we were to base our choice of mixing matrix on the echo density profile we want to choose either the skew-Hermitian matrix with a decay of 0.5 or the Hadamard matrix. However, when listening to their impulse response we hear that the Hadamard matrix produces a late reverberation, which contains a beating that is not desirable. This beating is not present when using the skew-Hermitian matrix because it produces a more randomized mixing of the echoes. Audio examples of the impulse response of the reverb with the different mixing matrices can be found in the resource folder. Audio examples of **TheStringPhone** with and without the FDN reverb is given

as well.

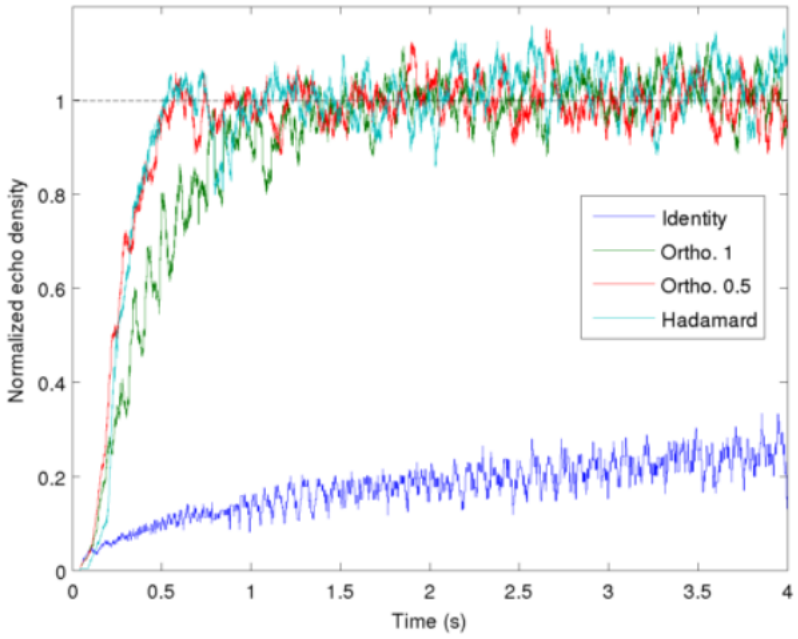

Figure 3.52: Normalized echo density functions of an identity mixing matrix (indentity), a randomized skew-Hermitian matrix with decay 0.5 (ortho 0.5), a randomized skew-Hermitian matrix with decay 1 (ortho 1) and a hadamard matrix (hadamard).

Chapter 4

Conclusion and future perspectives

4.1 Conclusion

We have designed a new digital musical instrument called **TheStringPhone** and evaluated the different parts from an engineering perspective. In this process we successfully implemented a Biquad Filter, a FIFO delay, a circular buffer delay, the Karplus-Strong algorithm, a peak and RMS compressor and a FDN reverb from scratch in C++. We also used the coefficients from a linear predictive coding analysis of the vocal input to design a formant filter, which filters the output from the Karplus-Strong algorithm. This, however, did not work properly due to fact that the input from the microphone is not limited to vocal input. The instrument did however work well without the formant filter and we found this to be a very interesting combination of input device and sound engine. This was due to the fact that a microphone input allows for a wide range of instrument usages due to the complex and unpredictable nature of "input from the real world". However, the instrument was only evaluated from an engineering perspective and further testing within an human computer interaction framework is needed in order to conclude further on the usages of the instrument.

The purpose of the thesis was to extend our knowledge to a more holistic understanding of the process of designing digital musical instruments. We now have a good foundation for implementing DMIs from an interaction design perspective, but also from an engineering point of view. While this is good, the holistic approach *is* very complex and there are so many parameters to take into account when designing for these. Designing a DMI from scratch is a long process and each part has to be tested from several different perspectives. Focusing on the engineering part of the process let us to make assumptions about the interaction part of the instrument, which does not necessarily hold true. This is in fact the pitfall of not taking an

holistic approach to designing DMIs.

4.2 HCI and aesthetic perspectives

This thesis can be seen as the first iteration in developing **TheStringPhone**. In order to move towards a holistic approach, the next step would be to evaluate **TheStringPhone** from a HCI perspective as well as from an aesthetic perspective. The HCI perspective would indicate what could be improved in terms of gestural control and mapping. This means, through user testing, trying to answer questions like: is there a better way to control the pitch than using the MIDI-keyboard? Is the sonic palette of the instrument wide enough? Does the current parameter settings, such as decay on the Karplus-Strong, work for the majority or just for us? Or should one be able to adjust this parameter? Testing in terms of the aesthetic dimension would give insight into how well the instrument serves as a tool for artistic expression. This means answering question like: does the musician feel he can express himself through the instrument or does he feel any limitation? Is the instrument capable of moving an audience? And so forth. Each of these tests would point to corrections, which could be made.

4.3 Specific points of improvement

When using input from the microphone low frequencies can tend to influence the signal in a bad way. It is therefore always a good idea to high pass the input signal. In the next version of **TheStringPhone** a high pass filter will be added before the signal is sent to the Karplus-Strong.

As the the formant filter did not work properly extra timbre control is needed in order to give the instrument a wider sonic palette. Future studies could be made in what other vocal parameters could be interesting to analyze and what control parameter on the instrument they could control. This could for example be to map different phonemes from the voice to the cutoff frequency on a low pass filter.

The Karplus-Strong algorithm is the most simple case of a waveguide. A natural future step is to extend **TheStringPhone** to use more complex waveguides. This would allow for additional control and a wider sonic palette.

Bibliography

[1] Abel, J. S., e. a. Recreation of the acoustics of hagia sophia in stanford's bing concert hall for the concert performance and recording of cappella romana. *Presented at the International Symposium on Room Acoustics, Toronto, Canada* (2013).

[2] Abel, J. S., and Berners, D. P. On peak-detecting and rms feedback and feedforward compressors. In *Audio Engineering Society Convention 115* (Oct 2003).

[3] Abel, J. S., and Huang, P. A simple, robust measure of reverberation echo density. *Audio Engineering Society, Presenting at the 121st Convention* (2006).

[4] Alfred, R. *Digital Filters for Everyone*, 3 ed. Creative Arts and Sciences House, 2013.

[5] Cartwright, M., and Pardo, B. Synthassist: Querying an audio synthesizer by vocal imitation. In *Proceedings of the International Conference on New Interfaces for Musical Expression* (2014), pp. 363–366.

[6] Cartwright, M., and Pardo, B. Vocalsketch: Vocally imitating audio concepts. *CHI '15 Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (2015).

[7] Castagne, N., and Cadoz, C. 10 criteria for evaluating physical modelling schemes for music creation. *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFX-03)* (2003).

[8] Chadabe, J. *Electric Sound: The Past and Promise of Electronic Music. Upper Saddle River.* NJ: Prentice Hall, 1997.

[9] Chadabe, J. Electronic music and life. *Organised Sound 9* (2004), 3–6.

[10] Collins, N. Cargo cult instruments. *Contemporary Music Review 6* (1991), 73–84.

[11] Cook, P. Principles for designing computer music controllers. *Proceedings of the New Interfaces for Musical Expression (NIME)* (2001).

[12] Cook, P. Re-designing principles for computer music controllers: a case study of squeezevox maggie. *Proceedings of the New Interfaces for Musical Expression (NIME)* (2009).

[13] Cook, P. R. *Real Sound Synthesis for Interactive Applications*, 1 ed. A K Peters/CRC Press, 2002.

[14] de Lima, A. A., Netto, S. L., Biscainho, L. W., Freeland, F. P., Bispo, B. C., de Jesus, R. A., Schafer, R., Said, A., Lee, B., and Kalker, T. Quality evaluation of reverberation in audioband speech signals. In *e-Business and Telecommunications*. Springer, 2009, pp. 384–396.

[15] De Poli, G., and Rocchesso, D. Physically based sound modelling. *Organised Sound 3*, 1 (1998), 61–76.

[16] Deacon, J. The development of a software tool that employs vocals for the control of musical elements in a live performance. Master's thesis, University of Limerick, 2014.

[17] Dolson, M. The phase vocoder: A tutorial. *Computer Music Journal 10*, 4 (1986), 14–27.

[18] Drummond, J. Understanding interactive systems. *Organised Sound 14(2)* (2009), 124–133.

[19] Fasciani, S. *Voice Controlled interface for Digital Musical Instrument*. PhD thesis, National University of Singapore, 2014.

[20] Flanagan, J. L., and Golden, R. M. Phase vocoder. *Bell System Technical Journal 4* (1966), 1493–1509.

[21] Floru, F. Attack and release time constants in rms-based feedback compressors. *J. Audio Eng. Soc. 47* (1999), 788–804.

[22] Frenette, J. Reducing artificial reverberation requirements using time-variant feedback delay networks. Master's thesis, University of Miami, 2000.

[23] Gardner, W. *Applications of Digital Signal Processing to Audio and Acoustics*, 1 ed. Springer US, 1998.

[24] Gelineck, S. Exploratory and creative properties of physical-modeling-based musical instruments. *PhD. Thesis. Aalborg University Copenhagen* (2012).

[25] Gelineck, S., and Serafin, S. A practical approach towards an exploratory framework for physical modeling. *Computer Music Journal 32*, 2 (2010), 51–65.

[26] Giannoulis, D., Massberg, M., and Reiss, J. Digital dynamic range compressor design - a tutorial and analysis. *J. Audio Eng. Soc. 60*, 6 (2012).

[27] Hunt, A., and Wanderley, M. M. Mapping performer parameters to synthesis engines. *Organised Sound 7(2)* (2002), 97–108.

[28] Hunt, A., Wanderley, M. M., and Paradis, M. The importance of parameter mapping in electronic instrument design. *In procedings of the 2002 conference on New interfaces for musical expression (NIME), Dublin, Ireland* (2002).

[29] Jaffe, D., and Smith, J. O. Extensions of the karplus-strong plucked-string algorithm. *Computer Music Journal 7* (1983).

[30] Janer, J. Voice-controlled plucked bass guitar through two synthesis techniques. *In Proceedings of the 5th international conference on New Interfaces for Musical Expression* (2005), 132–135.

[31] Janer, J. *Singing-driven interfaces for sound synthesizers.* PhD thesis, Universitat Pompeu Fabra, Barcelona, 2008.

[32] Johnston, A., Candy, L., and Edmonds, E. Designing and evaluating virtual musical instruments: facilitating conversational user interaction. *Design Studies 29(6)* (2008), 556–571.

[33] Jorda, S. Instruments and players: Some thoughts on digital lutherie. *Journal of New Music Research* (2005).

[34] Jot, J., and Chaigne, A. Digital delay networks for designing artificial reverberators. In *Audio Engineering Society Convention 90* (Feb 1991).

[35] Kadis, J. *The Science of Sound Recording*, 1 ed. Focal Press, 2012.

[36] Karplus, K., and Strong, A. Digital synthesis of plucked-string and drum timbres. *Computer Music Journal 7*, 2 (1983), 43–55.

[37] Kvifte, T., and Jensenius, A. R. Towards a coherent terminology and model of instrument description and design. *In procedings of the 2006 conference on New interfaces for musical expression (NIME)* (2006), 220–225.

[38] Lachaise, B., and Daudet, L. Inverting dynamics compression with minimal side information. *Proc. of the 11th Int. Conference on Digital Audio Effects* (2008).

[39] Lazzarini, V., and Boulaner, R. *The Audio Programming Book*. The MIT Press, 2011.

[40] Lemaitre, G., and Rochesso, D. On the effectiveness of vocal imitations and verbal descriptions of sounds. *The Journal of the Acoustical Society of America 135, 2* (2014).

[41] Lemaitre, G. e. a. Vocal imitations and the identification of sound events. *Ecological Psychology 23 4* (2011).

[42] Loscos, A., and Aussenac, T. The wahwactor: a voice controlled wah-wah pedal. *In Proceedings of the 5th international conference on New Interfaces for Musical Expression* (2005), 172–175.

[43] Loscos, A., and Celma, O. Larynxophone: using voice as a wind controller. *In Proceedings of the 2005 International Computer Music Conference* (2005).

[44] Loy, G. *Musimathics: The Mathematical Foundations of Music (Volume 1)*. The MIT Press, 2007.

[45] Loy, G. *Musimathics: The Mathematical Foundations of Music (Volume 2)*. The MIT Press, 2007.

[46] Magnusson, T. Of epistemic tools: Musical instruments as cognitive extensions. *Organised Sound 14(2)* (2009), 168–176.

[47] Magnusson, T. Designing constraints: Composing and performing with digital musical systems. *Computer Music Journal, 34:4* (2010), 62–73.

[48] Massberg, M. Investigating dynamic range compression. Master's thesis, Queen Mary University of London, 2009.

[49] Mathews, M. The digital computer as a musical instrument. *Science, New Series, pp. 553-557 142*, 3592 (1963).

[50] Moore, F. *Elements of Computer Music*, 1 ed. Prentice Hall, 1990.

[51] Moorer, J. Signal processing aspects of computer music: A survey. *Proceedings of the IEEE 65*, 8 (1977).

[52] Moorer, J. A. About this reverberation business. *Computer Music Journal 3*, 2 (1979), 13–18.

[53] Oliver, W., Yu, J., and Metois, E. The singing tree: design of an interactive musical interface. *In Proceedings of the 2nd conference on Designing interactive systems: processes, practices, methods, and techniques* (1997).

[54] Oppenheim, A. V., and Schafer, W. R. *Discrete-Time Signal Processing*, 2 ed. Prentice Hall, 1999.

[55] Orfanidis, S. *Introduction to Signal Processing*, us ed. Prentice Hall, 1995.

[56] Overholt, D. The musical interface technology design space. *Organised Sound 14(2)* (2009), 217–226.

[57] Paine, G. Interactivity, where to from here? *Organised Sound 7(3)* (2002), 295–304.

[58] Paradiso, J. Electronic music: New ways to play. *IEEE Spectrum, 34(12):18–30* (1997).

[59] Park, T. H. *Introduction to Digital Signal Processing: Computer Musically Speaking*. World Scientific Publishing Company, 2009.

[60] Peeters, G. A large set of audio features for sound description (similarity and classification) in the cuidado project. Tech. rep., Ircam, Analysis/Synthesis Team, 2003.

[61] Pressing, J. *Synthesizer performance and real-time techniques*. Oxford Uninversity Press, 1992.

[62] Rabiner, L. R., and Juang, B. H. *Fundamentals of Speech Recognition*, 1 ed. Prentice Hall, 1993.

[63] Ramakrishnan, C., Freeman, J., and Varnik, K. The architecture of auracle: A real-time, distributed, collaborative instrument. *In Int. Conf. on New Interfaces for Musical Expression* (2004).

[64] Redmon, N. Envelope generators-adsr. http://www.earlevel.com/main/2013/06/01/envelope-generators/, 2013. Accessed: 2015-05-01.

[65] Roads, C. *The Computer Music Tutorial*, 1 ed. The MIT Press, 1996.

[66] Schilling, R. J., and H., S. L. *Fundamentals of Digital Signal Processing Using MATLAB*, 2 ed. Cengage Learning, 2011.

[67] Schroeder, M., and Logan, B. Colorless artificial reverberation. *Audio Engineering Society 9*, 3 (1961), 192–197.

[68] Smith, J. O. Physical modeling using digital waveguides. *The Computer Music Journal 16*, 4 (1992).

[69] Smith, J. O. Physical modeling synthesis update. *The Computer Music Journal 20*, 2 (1996), 44–56.

[70] Smith, J. O. Introduction to digital filters with audio applications, 2007.

[71] Smith, J. O. Physical audio signal processing: for virtual musical instruments and digital audio effects. *W3K Publishing* (2010).

[72] Stautner, J., and Puckette, M. Designing multi-channel reverberators. *Computer Music Journal 5*, 1 (1982), 52–65.

[73] Steiglitz, K. *A Digital Signal Processing Primer: With Applications to Digital Audio and Computer Music*, 1 ed. Prentice Hall, 1996.

[74] Stowell, D. *Making music through real-time voice timbre analysis: machine learning and timbral control.* PhD thesis, Queen Mary University of London, 2010.

[75] Tanaka, A. Musical technical issues in using interactive instrument technology with application to the biomuse. *Proc. of the 1993 International Computer Music Conference. San Francisco, Calif.: International Computer Music Association* (1993), 124–126.

[76] Tanaka, A. Trends in gestural control of music, chapter musical performance practice on sensor-based instruments. *Ircam - Centre Pompidou* (2000).

[77] Vaidyanathan, P. P. *The Theory of Linear Prediction - Synthesis Lectures on Signal Processing.* Morgan and Claypool publishers, 2007.

[78] Valimaki, V., and Smith, J. O. Fifty years of artificial reverberation. *IEEE Transactions on Audio, Speech and Language Processing 20*, 5 (2012).

[79] Valimaki, V., and Takala, T. Tutorial article: Virtual musical instruments - natural sound using physical models. *Organised Sound 1*, 2 (1996), 75–86.

[80] Vaseghi, S. V. *Multimedia Signal Processing - Theory and Applications in Speech, Music and Communications.* Wiley, 2007.

[81] Wanderley, M. M. Gestural control of music. Accessed: 2015-05-27.

[82] Wanderley, M. M., and Depalle, P. Gestural control of sound synthesis. *Organised Sound 14* (2009), 188–196.

[83] Wang, G. rt_lpc: real-time lpc analysis, synthesis, and visualization, 2003.

[84] Zölzer, U. Reverberation algorithms. In *DAFX: Digital Audio Effects*, U. Zölzer, Ed., 2 ed. Wiley, 2011.

Appendices

.1 Digital filter theory

Digital filters operate on numerical data and do what their physical namesakes do in coffee makers or water systems i.e. modifying the output - ideally in a beneficial way - with respect to its inputs. This is done by forming linear combinations of past input and output samples that are uniformly sampled in time [13]. Current and past inputs are usually denoted as

$$x(n), x(n-1), x(n-2), ...$$

where n is the current time, *n-1* is the time one sampling period before the current one and so on. Current and past output are usually denoted as

$$y(n), y(n-1), y(n-2), ...$$

Probably the most common and easy accessible way to characterize a filter is to plot its amplitude-versus-frequency response (commonly referred to as the filters **frequency response**). Each type of filter has its own characteristic frequency response curve. The most common filters can be seen on Figure 1 and are *lowpass, highpass, bandpass and bandstop (also called notch) filters.*

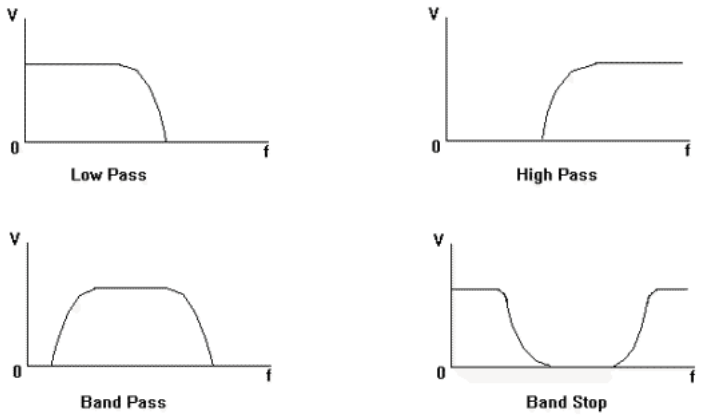

Figure 1: An illustration of the simple filters; *lowpass, highpass, bandpass and bandstop (also called notch)*

An important property of a filter is its **cutoff frequency** (f_c). This is defined as the point in the frequency range at which the filter reduces the signal to 0.707 of its maximum value [65]. This is so because the power of the signal at the cutoff frequency is proportional to the amplitude of the signal squared, since $0.707^2 = 0.5$. Therefore the cutoff frequency is

also called the *half-power point*. Another common term is the *3 dB point*, because 0.707 relative to 1 is close to $-3dB$. The area above the half-power point is referred to as the *passband* and the area below is referred to as the *stopband*. In addition the width of a bandpass and bandstop/notch filter is defined by the **bandwidth** (BW).

The **center frequency** (also referred to as f_0 *or* f_c) of a bandpass filter is the maximum point of amplitude and the center frequency of a band reject filter is the minimum point of amplitude (see Figure 2 for a simple bandpass filter).

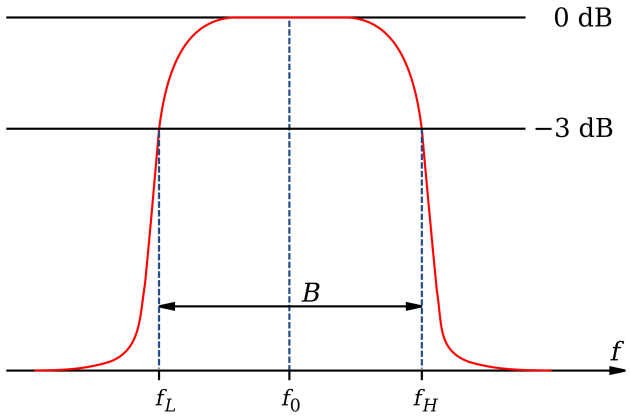

Figure 2: A simple bandpass filter with center frequency f_0 and bandwidth B together with low and high cutoff frequencies (f_L and f_H)

Ideal filters have a sharp transition which results in everything outside being maximally attenuated. These abrupt transitions are however not realizable and the area between the passband and the stopband is called the **transition band**. In non-ideal filters the slope is not linear leading up to the cutoff frequency. The steepness of a filter's slope is specified in terms of decibels of attenuation or boost per octave (usually referred to as *dB/octave*) [65].

Bandpass filters typically also has a **Q factor**, which represents the degree of resonance within a bandpass filter. Q can be found by dividing center frequency with the bandwidth

$$Q = f_c/BW$$

notice that when the center frequency is constant, adjusting Q is the same as adjusting the bandwidth.

Another property of a bandpass or band reject filter is its *gain* and refers to the amount of boost or cut of a frequency band. Say that the frequency response of a filter is $H(k)$, where k is frequency, then its gain is

$$H(k) = \frac{O(k)}{I(K)}$$

where *O(k)* and *I(k)* are the amplitudes at frequency k of the output and input signals, respectively [45].

.1.1 Finite and infinite filters

Digital filters can be either **Finite Impulse Response** (FIR) filters or **Infinite Impulse Response** (IIR) filters. The difference is on whether the filter only operates on current and past inputs **(FIR)** or if it also operates on past outputs **(IIR)**. A simple two-point moving average filter can be written as the simple difference equation

$$y(n) = 0.5(x(n) + x(n-1)) \tag{1}$$

Such a filter is a **FIR** filter, because it only operates on a finite number of delayed versions of its inputs. The number of delayed inputs is referred to as the filter 'order'. Filters of the form of Equation 1 are also called nonrecursive or all zero [13]. The difference equation for the general **FIR** is

$$\begin{aligned} y(n) &= \sum_{i=0}^{N} a_i x(n-i) \\ &= a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) + ... + a_N x(n-N) \end{aligned} \tag{2}$$

where N is the order, a is a set of N coefficients, and y is the output.

Now consider the simple **IIR** filter

$$\begin{aligned} y(n) &= \sum_{r=1}^{N} b_r y(n-r) \\ &= b_1 y(n-1) + b_2 y(n-2) + b_3 y(n-3) + ... + b_N y(n-N) \end{aligned} \tag{3}$$

Such a filter sum scaled delayed copies of the filter's output signal y and are therefore called an infinite impulse response filter. But in order to be useful a filter obviously must be able to receive a input from somewhere. How do we give such a filter an input? The solution is to couple an IIR filter to an FIR filter. By doing so we end up with a canonical filter that operates on both its inputs and outputs, with the general difference equation

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{s=1}^{M} b_s y(m-s) \tag{4}$$

In this case there is a feedback loop on the output. The output will be fed back and appear in scaled form again at the output in the next time step and so on. Since general filters have IIR components, such filters are also called IIR filters. Other terms used for this type of filter is *pole-zero filter* or *Aoto-Regressive moving Average* [13].

.1.2 Impulse response of a filter

The effects of a filter can be viewed in either the time domain or the frequency domain and the effects of filtration can be shown by 'before' and 'after' images of the signal. But not all inputs reveal the effect in an optimum way. It is therefore desirable to find an ideal signal that will clearly show the characterization of a given filter. White noise can tell us how the filter responds in the frequency domain, but an equally important measure of a filter is how the filter responds to transients [65]. In order to do this we need a measure of the filters response in the time domain. For this we use the **unit impulse function** $\delta(n)$ which is defined as a single 1 located at $n = 0$, and all other indexed values are 0

$$\delta(n) = \left\{ \begin{array}{l} 1, \text{ n} = 0 \\ 0, \ n \neq 0 \end{array} \right\}$$

The output signal generated by a filter that is fed a unit impulse is called the **impulse response (IR)** of the filter and is the filter's signature in the time domain (see Figure 3) and corresponds exactly to the system's amplitude-versus-frequency response [65]. It is a characteristic of FIR filters that the impulse response will always be a replica of the filter coefficients [4]. Such impulse response will be finite in duration, just as there is a finite number of coefficients. The impulse response length of IIR filters is however not tied to the order of the filter [45]. For such filters the impulse response may die out, remain constant, or even grow over time.

.1.3 The transform function of a filter

Besides a difference equation a filter can also be represented in a mathematical form called the filter's **transfer function**. It is the filters spectral equivalent to the the impulse respons and for much filter analysis it turns out to be mathematically convenient [4]. By describing the filter in the frequency domain we can reveal valuable insight into the behavior and stability of a filter in response to different input frequencies. The transfer function

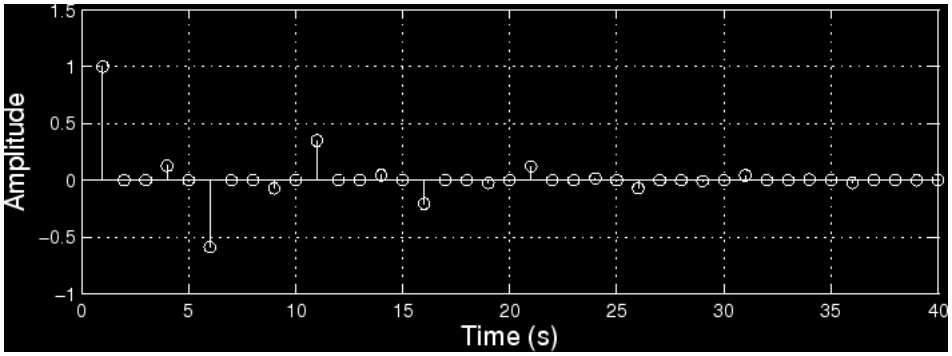

Figure 3: Impulse response of a comb filter

is a mathematical relationship between the input and output of a filter and is denoted *H(z)* and is written in what is called z-transform notation

$$H(z) = \frac{Y(z)}{X(z)} \tag{5}$$

A filters transfer function can be obtained by convolving the filters impulse response $h(n)$ with its input $x(n)$. By evaluating the filters transfer function at $z = e^{i\omega}$ we obtain the filters frequency response $H(\omega)$ [80]. The frequency response of a filter is typically defined as the characteristics of the filters output in terms of its magnitude and phase response when excited by an input signal [59]. These two characteristics are usually graphed as a *bode plot* and are plotted against a horizontal axis proportional to the logarithm of frequency. An example of a bode plot of a simple lowpass filter can be seen in figure 4. The magnitude plot shows how the filter amplifies or attenuates a signal as a function of frequency and the phase plot shows how phase changes as a function of frequency.

The magnitude response is commonly referred to as the filters gain $G(\omega)$ and can be found by taking the magnitude of the transfer function

$$G(\omega) = \|H(\omega)\|$$

A filters phase response $\Theta(\omega)$ indicates how much each frequency component gets advanced in phase by the system [66] and can be calculated by taking arctan2 to the real and imaginary parts of the signal

$$\Theta(\omega) = arctan\left(-\frac{Im[H(\omega)]}{Re[H(\omega)]}\right) = arctan2(Im[H(\omega)], Re[H(\omega)])$$

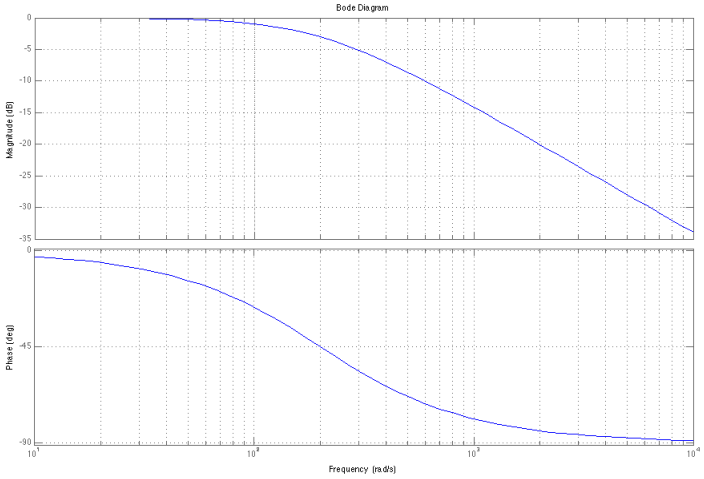

Figure 4: Bode plot of a simple lowpass filter

If a feedforward filter has coefficients that are symmetric about their center the phase response is proportional to ω, and that results in a fixed time delay [73].

.1.4 The z-transform

This transform is a analytical tool for digital filters, which maps the effects of sample delays into a two-dimensional image of the frequency domain called the complex z plane [65] and allows insight into the transient behaviour, the steady state behaviour, and the stability of a discrete-time system [80] (such as a digital filter). The two dimensional complex z-plane can be seen in figure 5 and consist of an imaginary (Im) and real (Re) axis. We can draw the unit circle by sweeping the digital frequency ω in a counterclockwise direction from 0 to π (DC to Nyqvist) with unity gain.

We must have a working knowledge of the z-transform in order to be able to understand the behavior of discrete-time filters (and systems in general). The z-transform represents a system in the complex z-plane in terms of the locations of the *poles* and *zeros* (i.e. the roots of the polynomial) of the filter transfer function. We can attain the poles and zeroes by setting the denominator and numerator of the transfer function equal to zero and then find the solutions with respect to z. Via the solution we can quickly determine the frequency response characteristics of the system as they directly reflect the resonant and dampening characteristics [59]. The poles reinforce and amplify the magnitude values at specific frequency locations (i.e. resonant locations) and the zeroes dampen or reduce the magnitude

values of a frequency response (dampening frequencies). By having one pole or one zero you cannot get complex response curves such as bandpass, peak and shelving filters. For this you need two poles and two zeroes (this type of filter is called a *biquad filter* and will be explained later). With a single point of change it is only possible to get lowpass and highpass responses.

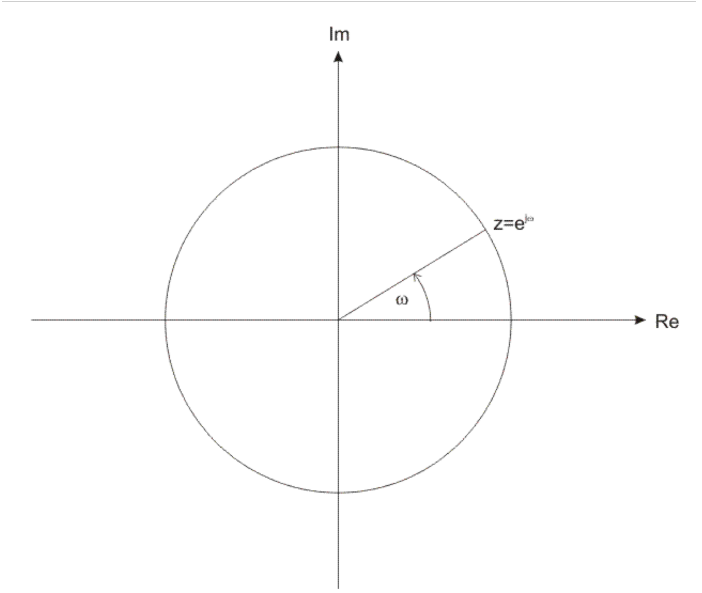

Figure 5: The complex z-plane

A pole corresponds to the root of the denominator, resulting in

$$H(z) = \frac{Y(z)}{0}$$

By dividing by zero $H(z)$ will go to infinity and will result in the magnitude response of the filter to resemble a tent proppep up at the pole location, thus creating a resonant location. Similar a zero take the form

$$H(z) = \frac{0}{X(z)}$$

thus pulling down the frequencies at the location of the zero.

To transform a filter using the Z-transform, simply capitalize all variables x and y, and replace all time indices *(n-a)* with the appropriate time delay operator Z^{-1} [13]. Thus the Z transformed version of Equation 1 would look like this

$$Y(z) = 0.5(X(z) + X(z)z^{-1})$$

and the z-transform of the general filter as described in Equation 4 looks like this

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_N z^{-N}}{1 + b_1 z^{-1} + b_2 z^{-2} + ... + b_M z^{-m}} \tag{6}$$

.1.5 The one-zero filter

With this added flexibility for manipulation and understanding let us consider the idea of the transfer function by looking at our averaging filter (Equation 1) with gain coefficients a_0 and a_1

$$y(n) = a_0 x(n) + a_1 x(n-1)$$

in z-transform this equation becomes

$$Y(z) = a_0 X(z) z^0 + a_1 X(z) z^{-1}$$

The transfer function of this filter then becomes

$$\begin{aligned} H(z) &= \frac{Y(z)}{X(z)} \\ &= \frac{a_0 X(z) z^0 + a_1 X(z) z^{-1}}{X(z)} \\ &= \frac{X(z)(a_0 z^0 + a_1 z^{-1})}{X(z)} \\ &= a_0 + a_1 z^{-1} \end{aligned}$$

We thus obtain the frequency response $H(\omega)$ by evaluating the z-transform on the unit circle i.e. by setting $z = e^{i\omega}$, so that $H(z) = a_0 + a_1 e^{-i\omega}$. Lets see how this filter responds to low frequencies by setting $\omega = 0$. If we set our gain coefficients to 1 the transfer function becomes

$$1 + 1 \cdot e^0 = 2$$

which means the filter amplifies energy a 0 Hz by a gain of factor 2. Conversely, if we set $\omega = \pi$ in order to see how the filter responds to high frequencies (and keep our gain coefficients at 1) the frequency response becomes

$$1 + 1 \cdot e^{-i\pi} = 1 + 1(-1) = 0$$

So when $\omega = \pi$ (half the sampling rate, which is the highest frequency representable without aliasing) the filter blocks the input. So clearly this is a lowpass filter. If we swept ω from 0 to π the values would gradually move from from 2 to 0.

But if we let our gain coefficient $a_1 < 0$ we create a highpass filter instead. This can be seen by setting $a_1 = -1$, and $a_0 = 1$. Now if we set $\omega = 0$ we get

$$1 + [(-1)e^0] = 0$$

and when $\omega = \pi$ we get

$$1 + [(-1)(-1)]] = 2$$

So depending upon the sign of a_1 this filter can act as a lowpass or a highpass. The first order one-zero filter has a single zero located at $z = -a$, and exhibits a maximum gain of $a_0 \cdot (1 + \|a\|)$ [13].

.1.6 The one-pole filter

As we just saw the one-zero filter forms the weighted average of two input samples to produce its output, but the one-pole filter forms its output by computing the weighted sum of the current input sample and the most recent output sample. Its difference equation is

$$y(n) = ax(n) + by(n-1)$$

Since this filter refers to its previous output it is *recursive*. What does this mean for the filter's response to an impulse? If we set $a = 1$ and $x = \delta$ the impulse response of the filter simply becomes

$$h(n) = b^n$$

This means the impulse response is infinite in length, asymptotically approaching zero without ever reaching it [45] i.e. an infinite impulse response (IIR) filter.

If we perform a z-transform of the impulse response in order to derive the frequency response of the one-pole filter we get an infinite z-transform. In order to circumvent this we keep b and z in the range -1 to 1, so the

values of later terms in the summation will become vanishingly small as n grows towards infinity.

The transfer function of our IIR filter in examination is

$$H(z) = \frac{a}{1 - bz^{-1}} = \frac{z}{z}\frac{a}{(1 - bz^{-1})} = \frac{za}{z - b}$$

by equating the denominator and numerator to zero we get the following two equations for the pole and zero respectively

$$\begin{aligned} z - b = 0 \rightarrow z &= b \\ z &= 0 \end{aligned}$$

since our zero is located at the origin we can neglect it as it affects all frequencies equally. We now have a filter with a single pole located at b. Since b must be a real number, the pole will always lie on the real axis of the z-plane. When $b > 0$ the pole will amplify frequencies near 0 Hz and when $b < 0$ the pole will amplify high frequencies. The closer the pole comes to the unit circle the greater the amplification (note that if b is not less than 1 we will get an unstable filter). If both a and b are set to 1, the first order one-pole filter becomes a *digital integrator*, which simply sums all prior input signals [13].

.1.7 The second order pole-zero filter (BiQuad)

If we want to implement a filter with more complex frequency response, such as something similar to a band-pass or band-stop filter, we need more than one pole or one zero. A BiQuad filter is a two-pole and two-zero IIR filter. In the time domain the BiQuad filter is

$$y(n) = x(n) + a_1x(n-1) + a_2x(n-2) - b_1y(n-1) - b_2y(n-2) \qquad (7)$$

and in the z-domain the BiQuad transfer function becomes

$$\frac{Y}{X} = \frac{1 + a_1z^{-1} + a_2z^{-2}}{1 + b_1z^{-1} + b_2z^{-2}} \qquad (8)$$

From the transfer function it is easy to see that the BiQuad filter is so named because its numerator and denominator consist of quadratic polynomials. It turns out that any polynomial can be factored into first and second order polynomials [13] therefore no matter the order the transfer function of the general filter (Equation 6) can always be decomposed into first and

second order filter building blocks. As seen in section .1.5 and .1.6 the second order two-pole blocks correspond to resonators and the second order two-zero components are anti-resonators, capable of placing a complex pair of zeros anywhere in the z-plane. The poles and zeroes can more or less be placed anywhere in the z-plane as long as coefficients a and b are complex. Equation 8 can also be written in the more common form by multiplying z^2 to both numerator and denominator

$$\frac{Y}{X} = \frac{z^2 + a_1 z^1 + a_2}{z^2 + b_1 z^1 + b_2} \tag{9}$$

Let's try to put this into a more compact form by finding the roots. The roots of the zeroes can be found by

$$z = \frac{-a_1 \pm \sqrt{a_1^2 - 4 \cdot a_2}}{2}$$

and similarly for the poles

$$z = \frac{-b_1 \pm \sqrt{b_1^2 - 4 \cdot b_2}}{2}$$

For complex roots the positions will always end up as a complex conjugate pair [59] of the form

$$z = Re \pm Im$$

or represented in polar form

$$z = R \cdot e^{\pm i\phi}$$

where R is the Euclidian distance defined by the real (Re) and imaginary (Im) components

$$R = \sqrt{Re^2 + Im^2}$$

and ϕ is found by

$$\phi = tan^{-1} \frac{Im}{Re}$$

By using the Euler identities we can represent the transfer function from Equation 9 as

$$H(z) = \frac{z^2 - 2 \cdot R_z \cdot \cos\phi_z \cdot + R_z^2}{z^2 - 2 \cdot R_p \cdot \cos\phi_p \cdot + R_p^2}$$

where the subscript z refers to the zeroes and p refer to the poles. If we multiply the numerator and denominator by z^{-2} we get

$$H(z) = \frac{1 - 2 \cdot R_z \cdot \cos\phi_z \cdot z^{-1} + R_z^2 \cdot z^{-2}}{1 - 2 \cdot R_p \cdot \cos\phi_p \cdot z^{-1} + R_p^2 \cdot z^{-2}}$$

from the above transfer function we can write the differential equation

$$\begin{aligned} y(n) =& x(n) - 2 \cdot R_z \cdot \cos\phi_z \cdot x(n-1) + R_z^2 \cdot x(n-2) \\ &+ 2 \cdot R_p \cdot \cos\phi_p \cdot y(n-1)\ R_p^2 \cdot y(n-2) \end{aligned} \tag{10}$$

Equation 10 describes the filter coefficients in terms of an exponential damping parameter (R_z for the zeroes and R_p for the poles) and a center frequency f_0 of resonance/anti-resonance [13], which is defined by the digital frequency ϕ

$$\phi = 2 \cdot \pi \cdot f \cdot T$$
$$f_0 = \frac{\phi_p}{2 \cdot \pi \cdot T}$$

Moving R_p towards 1 increases the gain at a particular resonant frequency f_0 (remember that R has to be strictly smaller than 1 in order for the filter to be stable). At the same time R_p acts as a dampening coefficient, and when $R_p = 0$, the filter becomes an FIR filter. Because of the zeros anti-resonance we can make a bandpass filter by placing the two zeros at DC and Nyquist as seen in Figure 6.

.1.8 The comb filter

This filter is very musical effective and has its name due to the shape of the filters magnitude response as seen in Figures 7 and 8. Comb filters are used in both digital reverberation processors and in the design of periodic waveform generators [55]. A comb filter can either be modeled via poles or zeroes i.e. as a FIR or IIR filter and the filter has two tuning parameters: amount of delay time (τ) and relative amplitude of the delayed signal to

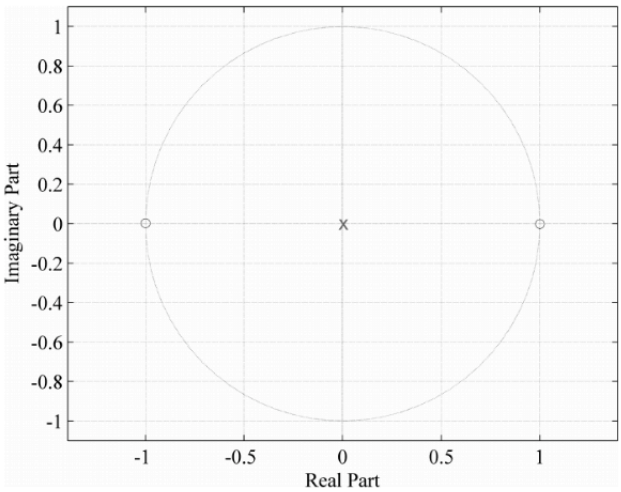

Figure 6: A pole-zero plot of a simple bandpass filter

that of the reference signal [84]. Comb filters have an effect in both the time and frequency domain. If we have a large value for τ the effect will be a distinct echo, but if τ is set below our perceptual threshold for time event segregation the comb filter will have a spectral effect instead. The sharpness of the valleys (zeroes of the FIR version) and peaks (poles for the IIR version) are a direct function of the filter coefficients [59] i.e. the closer to the unit circle the more pronounced effect. The difference equation of a FIR filter is given by

$$y(n) = x(n) + b_1 \cdot x(n - L)$$

where $L = \tau / f_s$. This filter amplifies all frequencies that are multiples of $1/\tau$ and attenuates alle frequencies in between (note that b_1 has to bee positive). If b_1 is negative we have the opposite situation.

The difference equation for the IIR comb filter is

$$y(n) = x(n) + a_1 \cdot y(n - L)$$

where y has to be strictly smaller than one as a stability criterion (as it is with all IIR filters). After each time delay the filter outputs a copy of the input signal with an amplitude of a_1^p, where p refers to the number of cycles that the original signal has gone through [84]. In effect the FIR and IIR versions are similar i.e. they affect the same frequencies, however the gain grows very high in the IIR filter resulting in narrower frequency peaks as $|a|$ comes closer to 1.

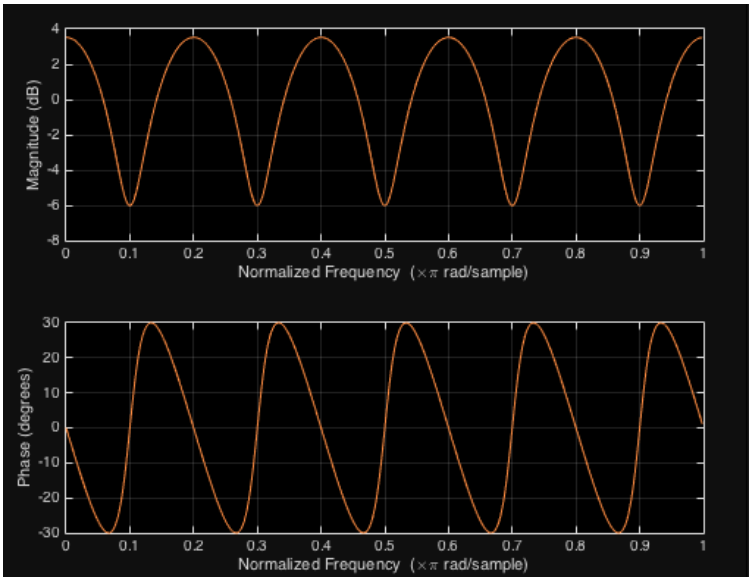

Figure 7: Bode plot of an FIR comb filter

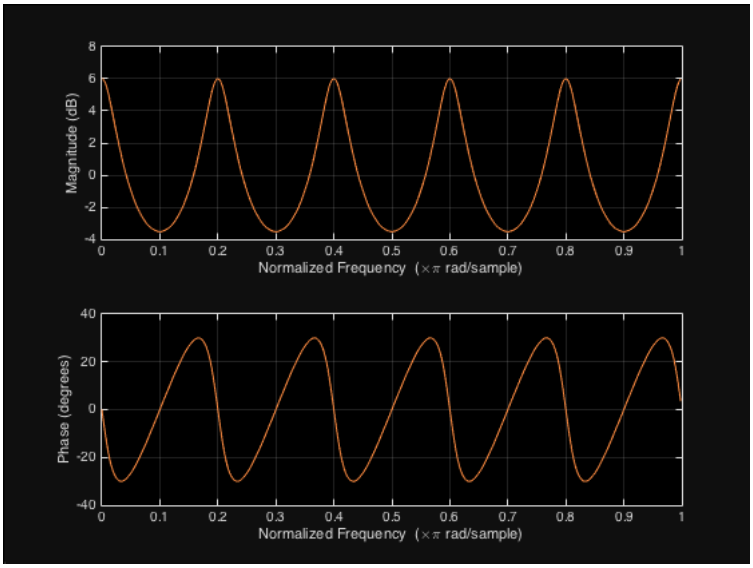

Figure 8: Bode plot of an IIR comb filter

So, unlike the FIR version the IIR version of a comb filter keeps an attenuated echo of the input in its feedback path, which gets accentuated every *Lth* sample and can result in a perceivable pitched output (as long as the output lies in the pitch range of course). Because of this it is possible to use a comb filter to filter signals with no pitch characteristics into pitched material.