VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SPEECH ANALYSIS FOR PROCESSING OF MUSICAL SIGNALS

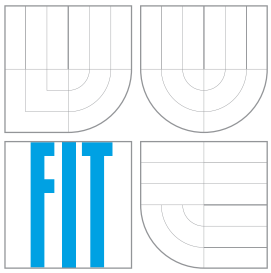
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ MÉSZÁROS

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ANALÝZA ŘEČI PRO ZPRACOVÁNÍ ZVUKOVÝCH SIGNÁLŮ

SPEECH ANALYSIS FOR PROCESSING OF MUSICAL SIGNALS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ MÉSZÁROS
AUTHOR

VEDOUCÍ PRÁCE — doc. Dr. Ing. JAN ČERNOCKÝ
SUPERVISOR

BRNO 2015

Abstrakt

Hlavním cílem této práce je obohatit hudební signály charakteristikami lidské řeči. Práce zahrnuje tvorbu audioefektu inspirovaného efektem *talk-box*: analýzu hlasového ústrojí vhodným algoritmem jako je lineární predikce, a aplikaci odhadnutého filtru na hudební audio-signál. Důraz je kladen na dokonalou kvalitu výstupu, malou latenci a nízkou výpočetní náročnost pro použití v reálném čase. Výstupem práce je softwarový plugin využitelný v profesionálních aplikacích pro úpravu audia a při využití vhodné hardwarové platformy také pro živé hraní. Plugin emuluje reálné zařízení typu talk-box a poskytuje podobnou kvalitu výstupu s unikátním zvukem.

Abstract

The primary goal of the thesis is to enhance musical signals with signs of human speech. This involves the creation of an audio effect inspired by the *talk-box*, by analyzing the vocal tract with a suitable algorithm like linear prediction and applying the calculated filter to the musical audio signal. An emphasis is given to excellent output audio quality, low latency and small processing overhead for real-time use. The outcome is a usable software plug-in targeted to professional audio editing applications and for live performance as well using a suitable hardware platform. It will emulate the real talk-box equipment or provides similar audio quality with a unique sound.

Klíčová slova

Lineární predikce, Audio syntéza, Kódování řeči, Modelování zvukových efektů, VST pluginy, LADSPA, DAW, Zpracování signálů

Keywords

Linear prediction, Audio synthesis, Voice coding, Sound effects modeling, VST plugins, LADSPA, DAW, Signal processing

Citace

Tomáš Mészáros: Speech Analysis for Processing of Musical Signals, diplomová práce, Brno, FIT VUT v Brně, 2015

Speech Analysis for Processing of Musical Signals

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Dr. Ing. Jana Černockého. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Mészáros
May 27, 2015

Poděkování

Děkuji svému vedoucímu diplomové práce panu doc. Dr. Ing. Janu Černockému, který podporoval nápad této práce a poskytl užitečné rady a odbornou pomoc při řešení. Poděkování patří i lidem, kteří pomohli s testováním.

Contents

List of Figures

Chapter 1

Introduction

The techniques of speech analysis are heavily challenged in today's industry whether speaking about communication technology or mobile and embedded systems. This work is intended to examine the most common algorithms of speech processing from a musical perspective, and modify them appropriately to provide artistic features, excellent output quality and sufficiently short processing time for real-time use.

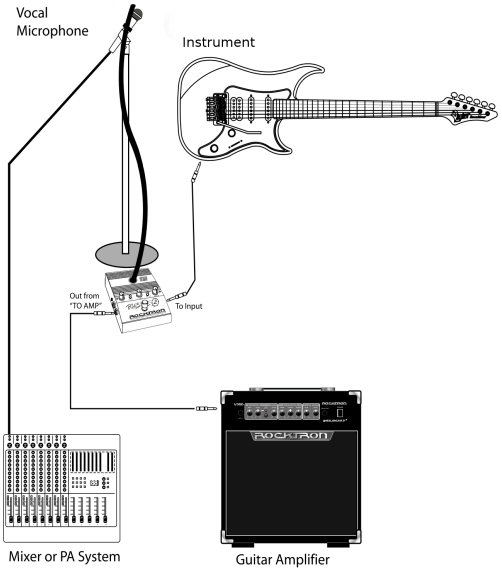
The use-cases of making an instrument sound as if it was talking are very diverse and leaves a vast space for creativity. The two most common uses are to make the human sound robotic and to make the instrument's own sound more human-like [1]. An *instrument* in this context can include effect modules and modulation devices as well. An early technology for achieving this functionality was the so-called *talk-box* effect emerging from the late 1940s and used heavily by Peter Frampton. He used the *Hail talkbox* construction where the instrument's signal is fed into a compression driver[1] which is connected to a long plastic tube (Figure 1.1). The other end of the tube has to be placed into the musicians mouth to articulate and shape the sound traveling through the tube. The resulting signal is recorded with a microphone and sent to an output speaker after amplification. Although this technology is relatively old, a lot of musicians (Ritchie Zambora, Slash, Joe Perry,...) still use it to get the exact sound of the legendary effect.

Digital alternatives had to catch-up in quality with the original talk-box and a lot of projects attempted this issue with different approaches. The *analogue music vocoder* is one that became wide-spread and dominant throughout the late 70's and early 80's electronic music. This is a fairly complex piece of electronics that uses bandpass filters to track the spectral envelope in different regions of the speech spectrum and applies the formant structure onto a carrier signal provided by a keyboard for example. Speech coding algorithms like linear prediction were also used occasionally, but their popularity fell behind the vocoder. An example of linear predictive coding can be heard on the song *Notjustmoreidlechatter*[2] composed by *Paul Lansky* from the album *More Than Idle Chatter* released in 1994. These techniques were designed to explore the artistic potential of human voice and computer synthesis and not primarily to simulate the talk-box as a physical system. In fact, it is very hard or nearly impossible to find a suitable emulation of the talk-box. All such attempts resulted in a different sound texture and some of them gained popularity for their own features. Examining the potential of speech coding algorithms for talk-box emulation is taken as a side quest of this work. This could be very hard to achieve given

[1] A small specialized diaphragm loudspeaker.

[2] Available from http://www.amazon.com/More-Than-Idle-Chatter-Lansky/dp/B000003GJ8

the fact that all sorts of complex physical and acoustic phenomena are making this effect so authentic. Yet it will serve as the main inspiration along the development.



(a) A typical talk-box configuration.

(b) Joe Satriani performing on a talk-box setup.

Figure 1.1: Concept of the talk-box effect.

Today's studio trends are causing massive migration of analogue effects into their software equivalents. Digital Audio Workstations (DAW) use plugins to provide the desired effect in studio quality. *The result of this thesis is a plugin module as an emulation of the talk-box effect.* A two channel audio card with a microphone is sufficient to use the plugin which extracts the formants from the speech and synthesizes them with the signal from the instrument. The plugin has to deal with requirements like real-time use, excellent audio quality, musical texture and response, and a relatively unique sound to stand out from the average vocoder-like effects accessible on the market or be at least comparable to them in quality. In addition, my goal was to produce a publicly available (reference) implementation of a voice-driven effect with an interesting sound. Linear prediction and voice coding algorithms will be in the focus throughout the research phase to find an acceptable solution as these – for some reason – gained seemingly less attention compared to other voice synthesis techniques in the music industry. Any findings regarding sound quality and processing performance optimizations would represent a contribution to the communication and multimedia industry as well.

The list of requirements does not stop at audio aspects. The result has to be available on all major platforms and for the widest range of DAW softwares. With a well designed framework, the algorithm can be packaged into multiple plugin formats to support as many workstations as possible. The final product has to be usable "as is" without an engineering degree and all configurations of controlling parameters have to be meaningful and usable in an appropriate situation. A uniquely designed user interface for the plugin is also part of the vision.

1.1 Organization of the Thesis

The thesis is divided into six major parts, following a progression from the most general topics to more and more specific issues regarding the development. In the first part, it will continue to raise as much interest as possible in the subject of speech synthesis and music creation practices with studio editing softwares and audio effects in general. The goal is to point out the exact segment in which the final product could be deployed. The second part involving *Chapter 2* will be dedicated to more specific theoretical issues and formal definitions, that may be indispensable to describe the thesis product itself. This description will reside in the next two parts with *Chapter 3* being a design section, where the tonality remains theoretical, but switches over to a focused plan for assembling the final software product. *Chapter 4* is intended to describe the very specific problems and unpredictable (or hardly identifiable) implementation issues in the design phase. *Chapter 5* gives place to evaluation and user testing. Finally, the last chapter provides a general discussion on the results and lays out the future directions of improvements and distribution.

1.2 A brief history of digital singing

Attempts to create artificial speech has begun way before the digital era. Some experiments were already made back in the 18th century lending voices to statues (and similar avatars) via speaking tubes and mysterious mechanical machines to impress the public. One of the earliest successful attempts at speech synthesis occurred in Russia in 1779 when Kratzenstein constructed a mechanical model of the human vocal tract that was capable of reproducing a few steady state vowels. The first recorded success in synthesizing connected speech was achieved by Kempelen Farkas (Wolfgang von Kempelen) in 1791 when he completed the construction of an ingenious pneumatic synthesizer (Figure 1.2) that was driven by a bellows with the air being forced past a whistle and an adjustable leather "vocal tract" [2].

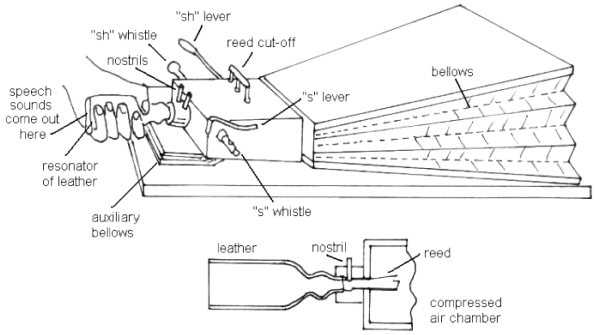


Figure 1.2: Pneumatic speech synthesizer developed by von Kempelen in 1791. [3]

In 1961, the first singing computer appeared on the scene. The *IBM 7094* sang the song *Daisy Bell* with the vocals programmed by John Kelly and Carol Lockbaum and the accompaniment by Max Mathews. This was an inspiration for Arthur C' Clark as the song made its way to the legendary movie *2001 Space Odyssey* released in 1968. The earliest computer music project at Bell labs in the late 1950s yielded a number of speech synthesis systems capable of singing, one being an early acoustic tube model of Kelly and Lockbaum [4]. This was considered computationally and economically too expensive at the

time to be used for music composition.

An early legacy of voice synthesis is the *VODER* device patented in 1939 by Homer Dudley at Bell Labs. The technology was first demonstrated publicly at the *1939 New York World's Fair*. It allowed speech generation using a controlling interface which was fairly complicated. The solution consisted of a parallel array of ten electronic resonators arranged as contiguous band-pass filters spanning the important frequencies of the speech spectrum (such a system is sometimes referred to as a spectrum synthesiser). The device was controlled via a keyboard (i.e., played like a piano). Ten finger keys controlled the output gain of each of the filters, a wrist bar controlled the selection of aperiodic hiss or periodic buzz, whilst a foot pedal controlled the pitch of the buzz. Three additional keys supplied appropriate stop-like transient excitation [2]. Werner Meyer-Eppler[3] recognized the capability of the Voder to be used in electronic music. The device required a skilled operator to produce intelligible speech and making singing voice was even more challenging.

Under the acronym VOCODER (VOice CODER) was hiding a device capable of coding speech efficiently for further transmission and communication purposes. Research was already in progress from the late 1920s at Bell Labs yielding the pair of devices known as the VOCODER for analysis and the VODER for speech synthesis which became more and more interesting for the scientific world. It was finally shown that intelligible speech can be produced artificially. Actually, the basic structure and idea of VODER is very similar to present systems which are based on the source-filter model of speech [5].

Figure 1.3: A picture of the VODER in use. [3]

The *Phase Vocoder* debuting in 1966 (Flanagan and Golden, Bell Labs), implemented using discrete Fourier transform, has found extensive usage in the music industry. Despite that it was not primarily developed for speech coding, it can be considered a vocoding effect as it allowed the transformation and recreation of speech with different properties like the pitch or playback speed. Perhaps the most notable implementation was produced by Mark Dolson in 1983 which took advantage of the increasing computing power at the time. Today's *auto-tune* effects are based on the phase vocoder principle.

Linear prediction was a breakthrough in speech processing and had a noticeable impact on digital music as well (*Lansky*, mentioned in chapter 1). The success can be related to

[3]Werner Meyer-Eppler (30 April 1913 – 8 July 1960), was a Belgian-born German physicist, experimental acoustician, phoneticist and information theorist. Source: http://en.wikipedia.org/wiki/Werner_Meyer-Eppler

the mathematical similarity with the source-filter abstraction of the vocal tract [4]. This algorithm will be discussed in the following chapters in detail.

1.3 Digital audio effects

There are a few misconceptions about certain phrases used in the field of signal processing when it comes to sound waves in general. Terms like *sound effects*, *sound transformation*, *sound processing* or *audio effects* are used many times for the same subject, despite they may refer to slightly different disciplines. Even the word "effect" can be confusing as it represents the perception of certain cause or phenomenon in the mind of a person, which is inevitably subjective and can be hard to define formally. With this in mind, one could separate the two most frequently used terms *sound effects* and *audio effects* by an analogy of the object being made and the tool used for creation [6]. The shift of the meaning of "effect" prevails in the semantics of these two terms: a *sound effect* is the change or modification being perceived itself and the tool used to make that change is an *audio effect*. A *sound effect* can provide natural or processed sounds either by synthesis or recording to produce specific effects on perception used to simulate actions, interaction or emotions in various contexts. On the other hand, an *audio effect* is the subject of research and falls into the field of signal processing. The fact that these terms are used interchangeably is not accidental. The most usual audio effects are known to modify the sound at its surface level by means of the retrievable amount of information from the digitalized waveform. Filters, delay lines, frequency domain transformations can modify the sound quite moderately (thinking of distortion effects, especially...) but the used methods are still touching the surface of the information contained in the samples with little intelligence at all. An example of deeper modification could be the *phase vocoder* which extracts the pitch and envelope and resynthesizes the sound using only the information from the samples and not the samples themselves. Today's audio effects are reaching a maturity where these "levels" are melting together and it is not a coincidence that the terminology melts with it at the same time. The definition of "audio effects" provided by the *DAFX Book* [6] and explained above will be used throughout the rest of the thesis for the sake of consistency.

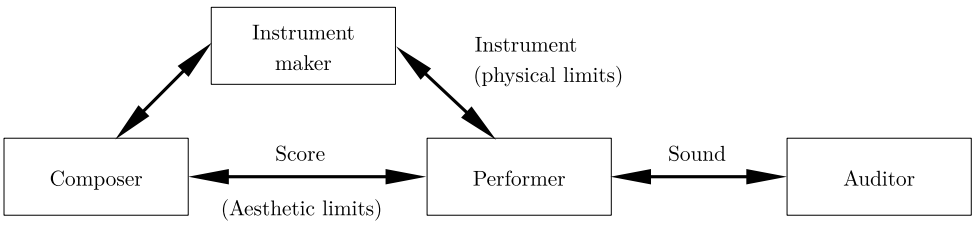


Figure 1.4: Communication flow in sound effect evaluation. [6]

The result of this work has to be an audio effect targeted to digital workstations. It could be appropriate to make a small overview of currently available effects and their categorization, only to point out where this "guitar speech" effect could fit into the picture of studio effects, which music production softwares are the potential hosts and what features

they provide. The usual communication flow between the actors of music creation (evaluators of audio effects) is shown on Figure 1.4 where each actor could be a different entity or even the same in every case depending on the situation. The categorization can be based on the requirements of any from these actors and would yield fairly diverse schemes as all of them have a bit different motivation.

The *instrument maker's* perspective is perhaps the most relevant from a technical standpoint reflecting the implementation concepts. The following type of components are known to exist in this class: [6]:

- Filters and delays (re-sampling)
- Modulators and demodulators
- Non-linear processing
- Spatial effects
- Time-segment processing
- Time-frequency processing
- Source-filter processing
- Adaptive effects processing
- Spectral processing
- Time and frequency warping
- Virtual analog effects
- Automatic mixing
- Source separation.

Perceptual properties are relevant to the end-users, namely the producers, musicians or composers. Nevertheless, it may also be a foundation for developing user tests and interfaces [6]:

Loudness The perceived intensity of sound. Relates also to dynamics, and phrasing (playing styles), levels in musical terminology like *pianissimo* (pp) or *fortissimo* (ff). A couple of effects is based on this quality such as compressors, limiters, tremolo (amplitude modulation), noise gates, and so on...

Time and Rhythm related to duration, tempo, and rhythmic modifications (*accelerando*, *deccelerando*)

Pitch Denotes the manipulation of note heights, intonation, transposition or harmony changes. (e.g., pitch shifter)

Spatial Hearing Environment acoustics, motion effects (Doppler, rotary speaker) as well as source localization (distance, azimuth, elevation)

Timbre Captures the texture, or essence of the sound. More specifically it relates to spectral attributes, like formants, short term time features as transients and attacks. Generally, the timbre is what makes two sounds with the same pitch different. Example effects include choruses, distortions, equalizers, or even vibrato.

1.4 Audio workstations and pluggable effect modules

Starting from the 1970s, the tools of music production have slowly but inevitably transformed from multi-channel tape recorders to integrated software and hardware solutions. Quality issues from the AD/DA conversion could not hold back the economic and practical benefits of computer based recording system and nowadays, the change-over has almost finished. Although analogue recording devices are still part of the chain, the center or "brain" of the recording system is a *digital audio workstation* running on a PC or Mac depending on the particular software. Since a computer has no tapes but a hard drive, the available space allows a non-destructive workflow whereas the old technology had to delete a previous recording. A DAW can have several *"tracks"* like a multi-channel tape recorder with many additional features, like containing multiple layers on each track or dynamically allocate or remove unused tracks. A recorded piece of audio can be processed by a unique signal chain assembled for a particular track with a separate time-line of parameter values (sometimes referred to as the *automation curve*). Additionally, a signal chain can utilize *inserts* and *sends* to route the audio data flow by preference.

Figure 1.5: Preview of *Ardour3* with several plugin modules in use.

Another major benefit is the opportunity for individual audio effect developers to distribute their work in a form of external plugin modules (shortly *plugins*). These can be loaded into a signal chain corresponding to one of the workstation's tracks and used real-time in the monitored output as well. From a technical viewpoint and especially as a programming concern, these modules are compiled shared or dynamic libraries[4] as far as the computing logic goes [7]. Static information and meta-data can reside in separate descriptor files in a chosen data definition language (XML or similar). Plugin creation can be done with a specific API or framework that defines the interface of the shared library. Frameworks do exist for different platforms with various approaches but their interfaces can be unified with additional effort.

[4]Standard OS dependent binary file, .dll in Windows, .so in Linux...

1.4.1 Overview of DAW applications

In order to build a rough picture of current DAW solutions and to choose a reference instance for developing the plugin, an investigation has been made concerning common and well known DAW applications:

Steinberg Cubase/Nuendo One of the oldest representatives. It offers an impressive set of internal effects and support mainly the VST plugin type as it was developed primarily for this product.

Apple Logic The DAW supported by Apple and available only for their platforms. A major advantage is the excellent real-time support of a Mac system. *Audio Unit* plugins are supported natively and others can be utilized through special wrapper plugins.

Avid ProTools Simply speaking, the industry standard in digital audio production. Instead of providing a stand-alone software, it covers hardware interfaces ("tools") for I/O and acceleration. It owns a unique plugin format as well which was discussed above.

Ardour3 Although this DAW is significantly less widespread than the previous "big players", it is an important and only living representative from the open-source world of professional audio editors (primarily but not exclusively for Linux) and is amongst the few existing open-source professional music production softwares at all. It is a suitable candidate to be a reference DAW for the development phase as well.

The provided list includes only a few of the most significant DAWs, without mentioning a lot of high quality but similar applications. Plugin modules are unable to run individually without a hosting environment. In technical terms a host can be any piece of program able to load the library at run-time and pass through audio and control data. DAW applications are also hosts and each of them supports slightly different features. The mode of operation (real-time use, GUI support, HW acceleration, etc.) may also differ from each other. However, some features are assumed implicitly, like the ability to display a plugin's own UI, or in case of no special graphics, the host has to generate the control widgets based on the available metadata.

1.4.2 Plugin modules

A general scheme of a plugin is shown in Figure 1.6. A reasonable API should provide a straightforward documentation on writing plugin instances. The framework will almost certainly define a function with the appropriate linkage as an entry point. Moreover, virtually all of them specify a function in which the processing will take place (usually `process`, `run` or a similar name). Further details about this topic can be found in [8] for example.

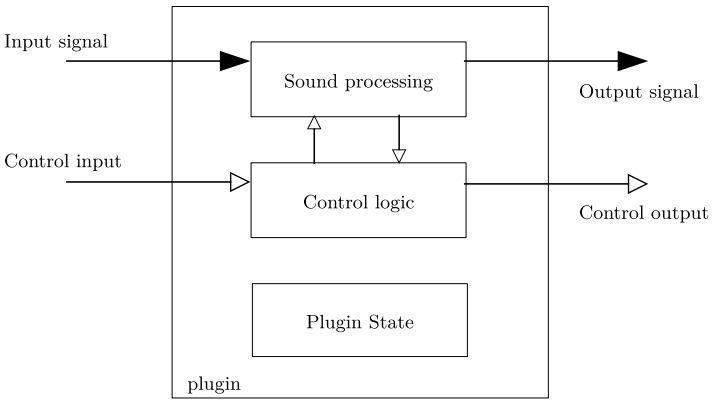


Figure 1.6: Anatomy of a software plugin. [8]

A non-exhaustive overview of the common plugin APIs is provided below:

Steinberg's VST A proprietary plugin framework from the company Steinberg. It appeals with the widest range of support amongst audio applications. Moreover, it supports unique user interface creation and MIDI functionality.

Audio Units The plugin standard for Apple's Mac platform. It covers system-wide audio effects as well as studio applications with all the essential features like GUI support, real-time processing, etc.

Avid Audio eXtension Primary format for the *ProTools* family of applications. Its unique feature amongst the competition is the ability to use hardware acceleration using the proprietary accessories bundled with a ProTools package.

LADSPA A slightly outdated format that has been considered as the system-wide standard for GNU/Linux based operating systems. It shines with the most simple and yet very well designed API which consists of a single header file. Open-source portability and unbeatable simplicity has a drawback of lacking a dedicated MIDI support or custom user interface creation.

LV2 A fully fledged modern plugin API replacement for the emerging Linux audio infrastructure intended to be the successor of the previous LADSPA format. It supports GUI creation with existing widget toolkits, MIDI messages, data description with static metafiles, and many other features.

JUCE audio framework This is a unique member of the list being a wide-ranging C++ class library for building cross-platform applications and plugins (as stated in their official website[5]). If an audio plugin is made with the JUCE library, it can be compiled to support many of the previous formats simultaneously. It offers high level signal processing classes as well, and is not limited to plugin development, but aims to be an overall audio software toolkit.

[5] http://www.juce.com/

Type	Maker	API License	features	Notes
VST	Steinberg	proprietary	MIDI, GUI	widest popularity
Audio Units	Apple	proprietary	MIDI, GUI	only for Apple systems
Audio eXtension	Avid	proprietary	MIDI, GUI	HW accel.
LADSPA	ladspa.org	open-source	–	Very simple API
LV2	lv2plug.in	open-source	GUI, MIDI	GUI with any toolkit
JUCE	ROLI ltd.	open-source	GUI, MIDI	supports multiple formats

Table 1.1: Summary of plugin architectures

Some plugin technologies can be utilized in more lightweight audio editors (e.g., *Audacity* or *Garageband*) with offline and destructive editing approach. Furthermore, they can be used in system wide audio daemon services as *Pulseaudio* can use LADSPA plugins or the *Coreaudio* framework with *Audio Units* plugins.

1.5 Similar voice-driven audio plugins

This section is dedicated to investigate similar products on the market as the developed plugin is intended to be. Vocoder plugins are fairly easy to obtain even for free of charge as part of several plugin bundles (e.g., the open source *CALF* plugins, the free *MDA* VST bundle).

Figure 1.7: Preview of the *AIR Boxing Talk* plugin. Source: protoolsproduction.com

More complex commercial formant extraction plugins are also available, although not quite common and tend to lean towards interesting voice synthesis effects instead of really simulating a talk-box unit. Some realizations advertise themselves as "digital talkbox" plugins, like the free *ARTICULATOR Evo* from the company *Antares* or the *Boxing Talk* from the *AIR plugins* bundle, but they sound just like a precise vocoding effect in almost all cases. Even a broader web search for a dedicated talk-box simulation yields no reasonable result if an exact talkbox plugin is desired. This phenomenon sets a suspicion that the mentioned physical simulation is far more complex than it would be cost-effective, or simply impossible yet with the current speech analysis techniques. Nevertheless, even the free alternatives can provide a strong inspiration of what a synthesis plugin is capable of and how the UI layout is arranged. A lot of extra modes, and modulations can be found in these effects, seemingly trying to fill the gap between the digital, robotic sounding vocoders and the acoustic, organically sounding talk-box solutions.

Chapter 2

Theoretical Overview of Voice Synthesis

Analyzing the performer's speech is the most fundamental part of this work and requires some formal basis. As the title suggests, this chapter provides an introduction to the field of speech processing to support any deliberation occurring in the design phase. The goal is not to provide a comprehensive description of speech coding but to cover the thesis subject itself without major theoretical gaps.

2.1 Spectral properties of the human vocal tract

A short introduction of the biological aspects is appropriate before describing the spectral model on its own.

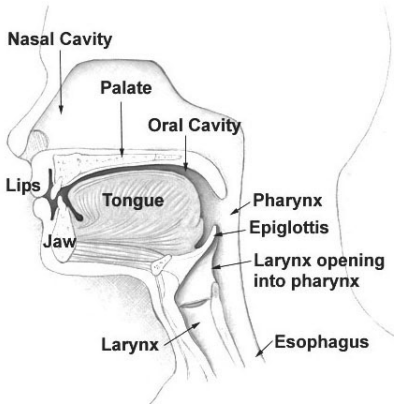


Figure 2.1: Biological speech system. Source: Wikipedia[1]

Without going into further details, the three main stages of speech production are as follows [9]:

1. Air flow from the lungs provides the acoustic power needed to make any sound at all.
2. The flow is modified by the periodically opening and closing larynx that results in a sound source or excitation.

[1] en.wikipedia.org/wiki/Motor_theory_of_speech_perception

3. The vocal tract articulates a distinct vowel from the incoming signal.

A wide-spread representation of the speech system is the *source-filter* model, as the source is coming from the vocal cords and the vocal tract acts as an acoustic filter [1]. It is a system that has its own impulse response $h(t)$ and spectral transfer function $H(f)$ with each configuration.

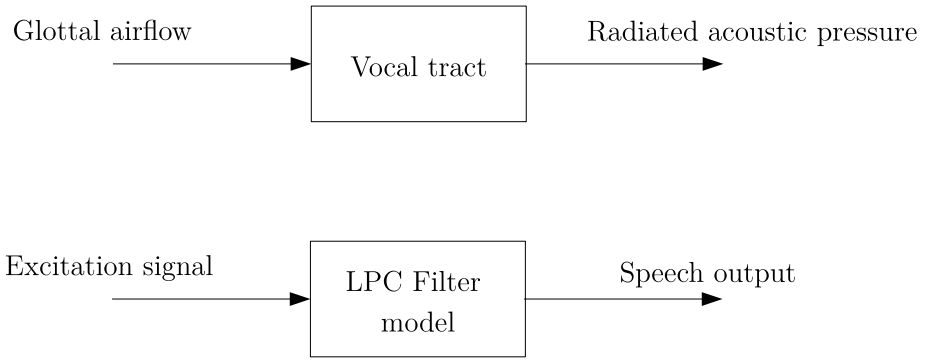


Figure 2.2: Analogy between the biological speech system and the source-filter model. [1]

The transfer function is obviously not constant as the mouth shape is changing continuously. The implication is that the filter modeling the vocal tract is time varying. A sound source has to excite this filter in order to produce speech. If this excitation signal is periodic, the result will be a *voiced* sound. With a stochastic or noisy excitation, the speech will be *unvoiced*.

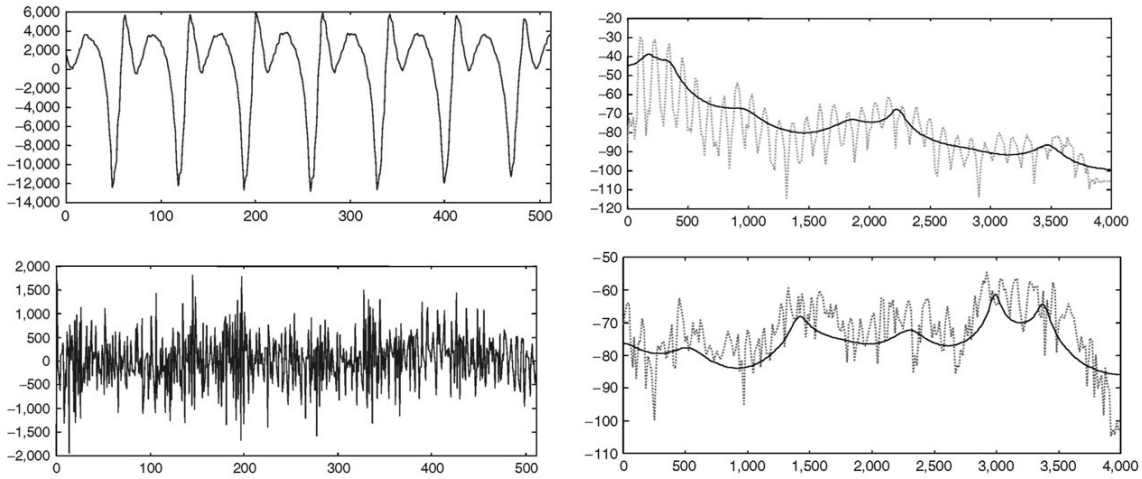


Figure 2.3: Time (left) and spectral domain (right) samples of voiced (top) and unvoiced (bottom) speech. Source: what-when-how.com

A real-time speech processing application has to continually track the state of the vocal tract in such short intervals (*frames*) where it becomes much or less stationary. The spectral shape of voiced fragments depends on two major factors:

1. The fundamental frequency and the accompanying harmonics F_0 inducted by vocal cord vibrations.

2. The spectral envelope with several distinctive local maxima created by the vocal tract. These peaks are referred to as *formants*.

The human hearing system acts like a filter bank and is most sensitive to the 200 - 5600 Hz frequency range in terms of perception [10]. This is where the first three formants as the most important spectral features occur in speech, characterizing any human-like manifestation. Reproducing the spectral envelope of the speech is the key to create a reasonable talk-box simulation.

2.2 Source-filter separation

The previous section has led to a conclusion that any manipulation with the speech has to rely on the accuracy of detecting and extracting the formant structure *separated* from the excitation source. Referring to chapter 1.3, this can be considered as a deeper analysis of sound samples. Before going into further details of how the separation could be done, the meaning of the term *spectral envelope* has to be defined accurately to know what is going to be separated exactly. Taking a purely harmonic signal, the envelope could be imagined as the curve which passes through the points denoting the harmonics in a frequency domain representation. The question remains open concerning what interpolation has to be used to retrieve parts of the curve in-between the harmonics. This definition stays no longer valid, if additional noise or non-harmonic components are present in the signal. In this case, the nature of the spectral envelope becomes dependent on what exactly makes up the excitation and what is the resonance. With all these concerns in mind, the (non-formal) definition found in [6] is borrowed, stating that *"a spectral envelope is a smoothing of a spectrum, which tends to leave aside the spectral line structure while preserving the general form of the spectrum."*

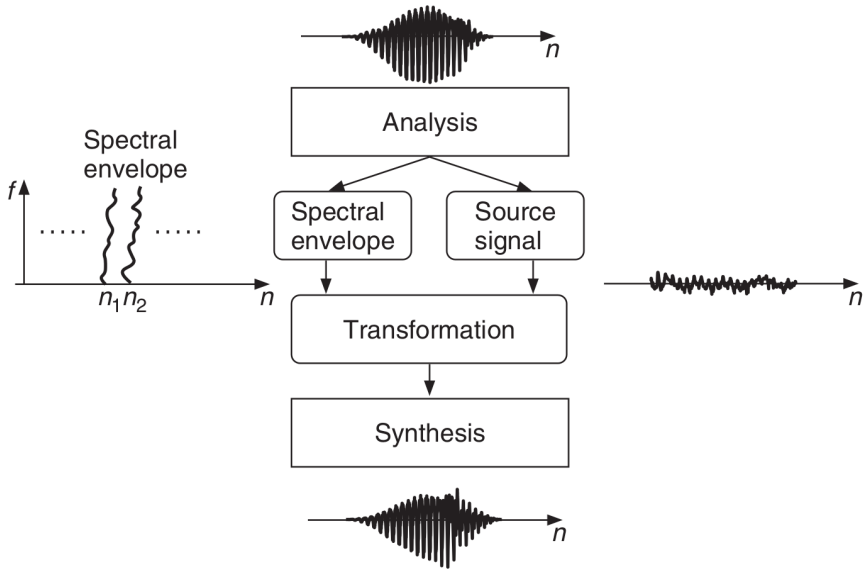


Figure 2.4: Source-filter analysis work-flow. [6]

Three reference techniques are wide-spread (with many variations) to perform a source-filter separation:

The channel vocoder with several frequency bands to estimate the signal energy inside each band and assemble the approximation of the spectral envelope.

Linear prediction yielding an all-pole filter corresponding to the envelope estimate.

Cepstral analysis techniques perform a conversion of spectral domain information with multiplicative filter components into a logarithmic scale with additive (and thus separable) source-filter components.

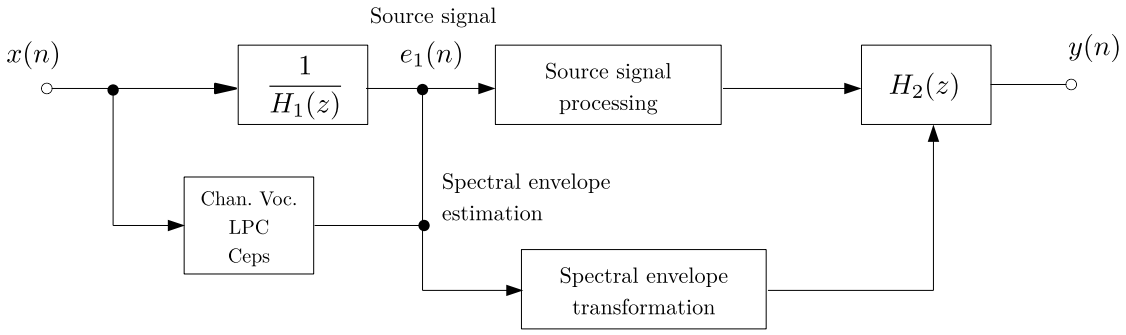


Figure 2.5: Source-filter estimation scheme. [6]

Figure 2.5 depicts a typical manipulation scheme where the spectral envelope of the signal $x(n)$ is removed by filtering with the inverse $H_1(z)$ filter and replaced by a different envelope denoted as $H_2(z)$.

2.3 Linear prediction algorithms

Linear prediction is a widely used mathematical apparatus to estimate the behavior of discrete-time signals from their history. Applications are not restricted to sound waves, usages can be found in many engineering areas, but the subject covered in this thesis is focused on how an acoustic filter could be retrieved from a short speech signal segment. This topic is essential regarding the design phase. Despite the fact, that detailed descriptions can often be found in literature, a formal section based on [6] is dedicated to the subject to be a well of backward references through the development. This section assumes that the reader is familiar with basic digital filtering concepts like FIR and IIR filters or the $\mathcal{Z}$-transform.

The basic model works with a discrete time input signal $x(n)$ and tries to predict the next sample as a *linear combination* of past samples. The prediction of $x(n)$ is computed using an FIR filter by:

$$\hat{x}(n) = \sum_{k=1}^{p} a_k x(n-k) \tag{2.1}$$

where p denotes the *prediction order* and a_k are the prediction coefficients. An *error or residual* signal can be calculated from the difference of the predicted samples and the original ones by the equation:

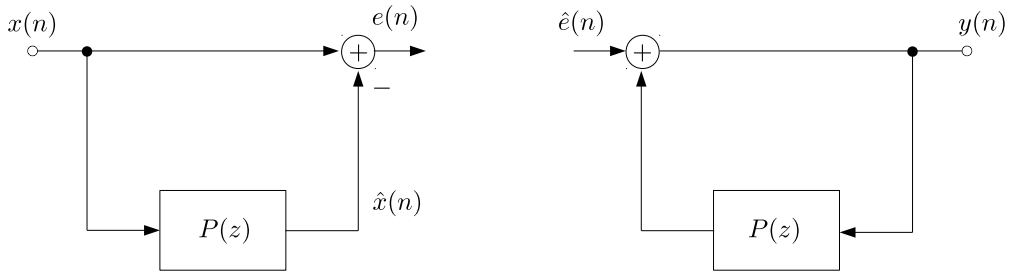
$$e(n) = x(n) - \hat{x}(n) = x(n) - \sum_{k=1}^{p} a_k x(n-k) \tag{2.2}$$

or working in the $\mathcal{Z}$ domain:

$$\text{the prediction filter: } P(z) = \sum_{k=1}^{p} a_k z^{-k} \tag{2.3}$$

$$\text{the prediction error: } E(z) = X(z) - \hat{X}(z) = X(z)[1 - P(z)] \tag{2.4}$$

Equation 2.4 forms the basis of analysis and synthesis and is referred to as the *feed-forward prediction* scheme (Figure 2.6) due to the direction of the calculation.



(a) analysis (b) synthesis

Figure 2.6: *Feed-forward prediction* block scheme.

Based on the previous derivation, the *prediction error filter* (simply *inverse filter*) is defined as:

$$A(z) = 1 - P(z) = 1 - \sum_{k=1}^{p} a_k x(n-k) \tag{2.5}$$

and a $\mathcal{Z}$ domain multiplication with the original signal will yield the *error* signal:

$$E(z) = X(z)A(z) \tag{2.6}$$

Using the approximated error (residual) signal as an input to the all pole filter $H(z)$ defined by equation 2.8 will produce an output signal $Y(z)$:

$$Y(z) = \hat{E}(z)H(z) \tag{2.7}$$

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - P(z)} \tag{2.8}$$

If $H(z)$ was the predicted envelope filter to the original signal $X(z)$ and $\hat{E}(z)$ is the estimated residual of $X(z)$, then $Y(z)$ should be a nearly matching version of the original input $X(z)$.

Knowing that $H(z)$ is practically a spectral model of the vocal tract (without a gain factor), it shall be called the *synthesis filter* or simply LPC filter. In case of desiring

compression, the low numerical values of $e(n)$ are opening the possibility of efficient quantization. This requires the minimization of the residual energy by obtaining an accurate LPC filter. The exact calculation of LPC coefficients resembles to a problem of finding the minimal energy for the error signal, $E = E\{e^2(n)\}$, thus calculating its first partial derivatives and setting them to zero. Solving this derivation would lead to $E\{x(n)x(n-i)\}$. The majority of literature suggests a solution using the autocorrelation values

$$r_{xx}(i) = \sum_{n=i}^{N-1} u(n)u(n-i) \tag{2.9}$$

together with the *Levinson-Durbin* recursion method for finding the coefficients of the inverse filter $A(z)$. In equation 2.9, $u(n) = w(n)x(n)$ denotes a windowed version of a block with N samples.

The best spectral fit can be found by minimizing the residual energy, but the gain factor of the calculated signal segment is still ignored. To accurately model the input signal $x(n)$, a modified synthesis filter is used corresponding to the following equation:

$$H_g(z) = G \times H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} \tag{2.10}$$

where G stands for the gain factor of the calculated segment. By using the autocorrelation coefficients according to equation 2.9, G can be obtained as:

$$G^2 = r_{xx}(0) - \sum_{k=1}^{p} a_k r_{xx}(k) \tag{2.11}$$

The last important term to define is the *prediction gain* that is essentially the ratio of the signal energy compared to the residual $e(n)$.

$$G_p = \frac{\sum_{n=0}^{N-1} x^2(n)}{\sum_{n=0}^{N-1} e^2(n)} \tag{2.12}$$

There are certain weaknesses of linear prediction when used for speech analysis mainly due to the assumption of linearity. The inverse filter $A(z)$ does not model the incidental nonlinearities, that may occur from various causes, like source-tract coupling, non-linear wall vibration losses, and aerodynamic effects resulting in deviations from the ideal source-filter model. The outcome of re-synthesis could sound "buzzy" if these factors reach an extreme [4].

2.4 Numerical representation of prediction coefficients

The coefficients a_k calculated from linear prediction are mostly a subject of further calculations or interpolation. Some of these operations require or suggest a different type of representation, to save computing resources, maintain filter stability, or aid miscellaneous post-processing algorithms. Needless to say, that all of these representations have to contain 100 % of the information from plain LP coefficients and provide sufficiently short time complexity regarding the conversion.

Line spectral frequencies It is generally known that LP coefficients have a wide numerical dynamic range which makes them ineffective for operations like quantization. On the other hand, *line spectral frequencies* have much better dynamic range. Also, LSF coefficients are ideal for interpolation between arbitrary speech segments. Another useful property is that LSF coefficients reside in the spectral domain and have a close relationship with the actual formant structure. A change in a particular coefficient will affect the power spectrum near the corresponding formant [9].

Reflection Coefficients A side-product of the *Levinson Durbin* recursion is a set of parameters known as the *reflection coefficients* of an acoustic tube model of the vocal tract.

Autocorrelation Function In certain circumstances, the coefficients calculated with equation 2.9 can be used without further calculation having attractive features for interpolation [9].

2.5 Analysis windows

The theory of speech analysis and synthesis works mostly with short time segments usually called *frames*. Determining the duration of these chunks is up to the perceptual and motoric limitations of our biology. More specifically, to ensure that the frame contains useful static information, it has to capture a duration as long as the inertial forces can keep the vocal tract in a stationary state. During slow speech, the vocal tract's shape and excitation can remain unchanged for even 200 ms. On the other hand, the average duration of a phoneme is considered to be 80 milliseconds [10]. Extraction of the formant structure needs a more or less stationary spectral footprint but it is still required to have enough samples to perform an accurate analysis. Additionally, if the signal is sliced into pieces, and frequency domain operations are applied to the segments, the continuity of spectral characteristics might be distorted if a simple rectangular "scissor" is used. This is a common signal processing issue and the solution is to use a *window* function for selecting each segment from the signal. The rectangular window would introduce several high frequencies due to the edges. Multiplying the input signal $s(n)$ with a window function $w(n)$ can reduce these negative effects as it eliminates the mentioned edges of the rectangle. The goal here is not to describe the theory of windowing, but to introduce, what kind of window functions may be considered in continuous real-time speech analysis for the best audio quality.

The Hamming window is considered as the most popular in case of FIR filtering. It weighs the samples near the center with the highest ratio and slowly decreases with a sinusoidal fashion close the edges of the set.

$$w(n) = 0.54 - 0.46\cos\left(2\pi\frac{n}{N}\right), \quad 0 \le n \le N \tag{2.13}$$

The Hanning window is very similar to the previous *Hamming* window with a difference that the *Hanning* window gets very close to zero at the edges.

$$w(n) = 0.5\left(1 - \cos\left(2\pi\frac{n}{N}\right)\right), \quad 0 \le n \le N \tag{2.14}$$

The exact Blackman window with its derivatives, the *Blackman* and *Blackman-Harris* window are believed to give the best out-of-band rejection [11].

$$y_i = x_i[0.42 - 0.5\cos(w) + 0.8\cos(2w)] \tag{2.15}$$

for $i = 0, 1, 2, \ldots, n-1$, where n is the window length.

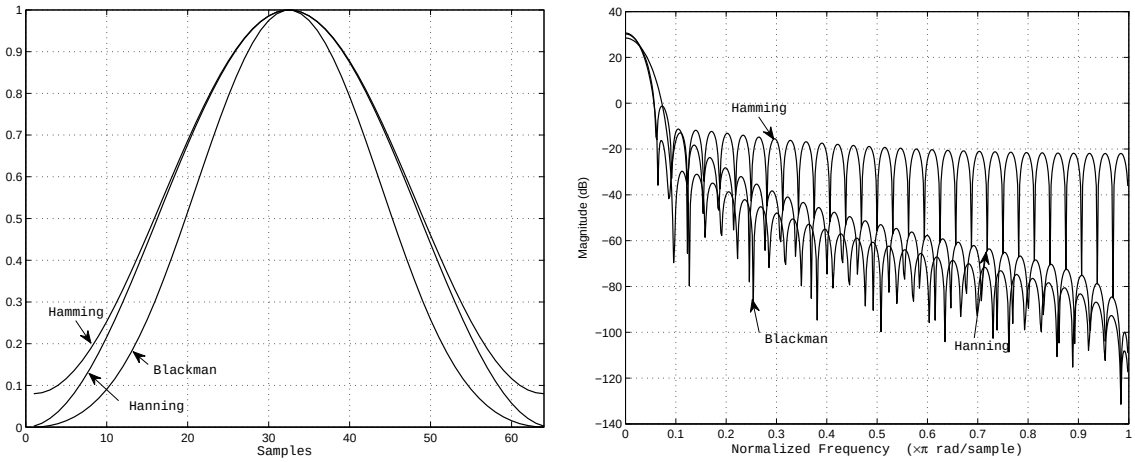


Figure 2.7: Comparison of the most common window functions.

Figure 2.7 shows a side-by-side comparison of the most common window functions. Many others are known to exist beside the mentioned window types, like the *triangular*, *Bartlett*, *Kaiser-Bessel*, *Poisson*, *Reimann*, just to mention a few. The question here is not only the type of window function, but the alignment and overlap ratio of each segment which is a specific design issue, and it will be discussed in Chapter 3.

2.6 High-Fidelity audio filtering

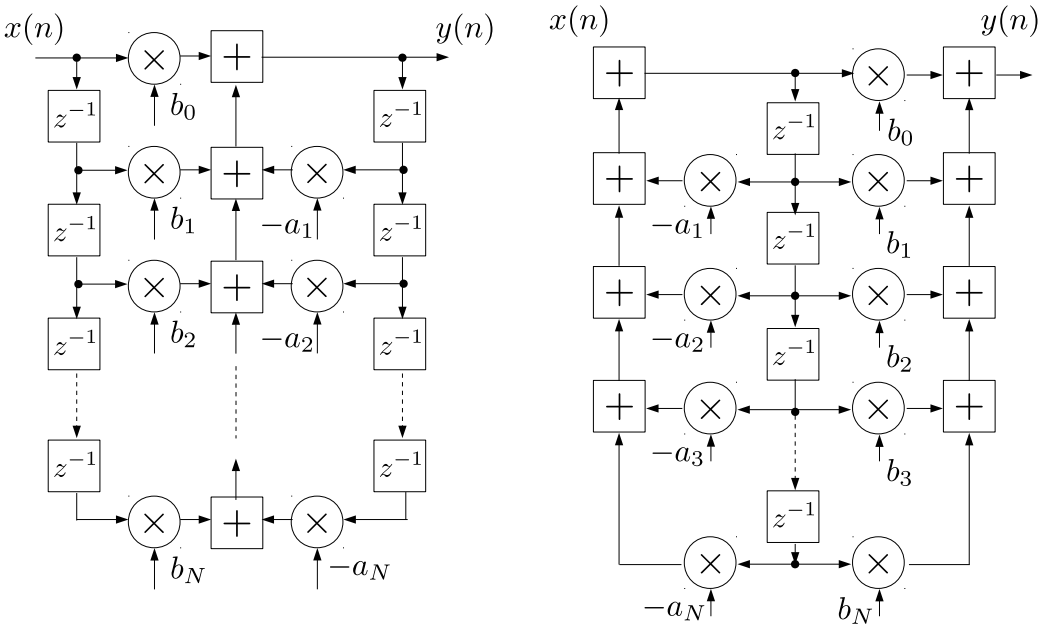
Filtering is the very essence of basic signal processing. Common description of how digital filters work can be found in many articles and a lot of tools are available for designing filters. Yet, when it comes to the implementation, a few issues just might pop up that have to be investigated carefully. It is not a secret, that the final product has to be ready at the time of writing these lines and the existence of this section is implied by specific problems which do require an additional theoretical basis regarding filter implementations in software.

When a transfer function of the form

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_i z^{-i}}{1 - a_1 z^{-1} - a_2 z^{-2} - \ldots - a_i z^{-i}} \tag{2.16}$$

is *realized* it means a conversion into a filter network or structure. There are generally more than one possible solutions each having unique qualities. The choice of structure can be influenced by factors such as the sensitivity to quantization, level of output noise due to arithmetic rounding or truncation, computational efficiency, number of systems used, and type of filter [12]. This section will briefly introduce the most significant topologies and their properties.

Direct forms Using the direct relationship between the filtering topology and the difference equation, the obtained structure is shown in Figure 2.8a. Coefficients of the transfer function serve directly as a multiplier in the corresponding structure which is known as the *Direct form 1 (DF1)*. As stated in [12], there are severe coefficient sensitivity problems, when the poles of the filter lie close to each other or near the unit circle in the $\mathcal{Z}$ plane. Input is processed first by the b_k coefficients, that is why this structure is sometimes referred to as a *zeros-before-poles* realization.



(a) Direct form 1

(b) Direct form 2

Figure 2.8: Direct form filter structures.

The structure in Figure 2.8b is an alternative realization known as the *Direct form 2* or *canonic form* due to the minimized number of utilized processing blocks. The properties regarding coefficient sensitivity are very similar to DF1, quantization and generation of high level roundoff noise is still present. These forms are popular for hardware realizations, because they allow for a high level of parallelism.

DF1 is known to be more resistant to overflow problems which is not guaranteed in DF2. According to [12]: *"For a filter with an input and output magnitude less than unity, the inputs to all multipliers can be expressed in fractional arithmetic... If modulo arithmetic is used, the partial sums of products may be allowed to overflow since it is known that the final output will be within the range of the number system used."*

Cascade form The cascade form (Figure 2.9) reduces the coefficient sensitivity and quantization problems known by the previous arrangements. Here, the transfer function is shaped into a cascade of first or second order sections $H_i(z)$:

$$H(z) = \prod_{i=1}^{K} H_i(z) \tag{2.17}$$

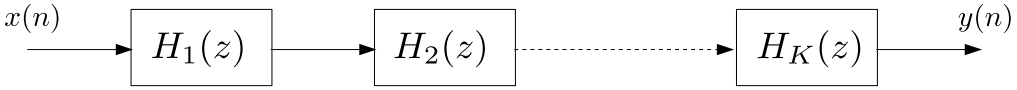


Figure 2.9: Cascade filter realization.

This arrangement may also require a greater control over the ordering of sections and the pairing of the poles to achieve the best results.

Parallel form A sibling of the cascade arrangement where the first or second order function blocks are placed in parallel side-by-side to each other, as depicted on Figure 2.10. The parallel structure produces even lower levels of roundoff noise than the cascade arrangement [12].

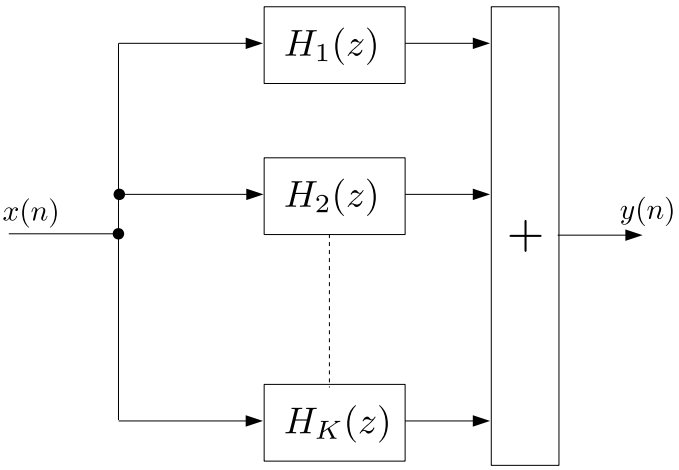


Figure 2.10: Parallel filter realization.

The provided list of arrangements is by no means complete and touches only the surface of how a digital filter can be implemented when a particular hardware architecture is under examination. The decision of which structure will be used, can be already predetermined by the purpose of filtering – an equalizer will surely benefit from a cascade implementation. Furthermore, a specific platform can introduce additional constraints. For instance, if integer arithmetic is unavoidable or parallelism is not an option. In fact, for a software plugin implementation, the fundamental sample data type is almost certainly specified by the API or framework used for plugin creation which is usually a single precision floating point type (`float` in C language). Parallelization of the filtering algorithm inside a plugin's source code can also be tricky as the host may be sensitive to thread-safeness of the called routines. Hardware acceleration is yet another issue which has to be thought over carefully. Ideally, acceleration should rely on the framework's features used for development (e.g., *Avid's Audio eXtension* plugins).

2.7 Zero State Response and Zero Input Response

An alternative way of filtering can be achieved through the transfer function's *zero state* and *zero input* response, abbreviated as ZSR and ZIR respectively. Suppose a transfer function $H(z)$ and a speech signal being defined as L dimensional vector [13]:

$$\mathbf{S} = [s(0), \ldots, s(L-1)], \quad s(n) = \sum_{i=0}^{\infty} h(i)u(n-i)$$

with $h(n)$ marking the infinite-length impulse response of $H(z)$ and $u(n)$ being an excitation signal. ZIR is nothing more than the response of a filter structure with arbitrary state to a zero input. This could be even infinitely long, if an IIR filter is used. The ZSR on the other hand, is the result of using a zero state filter for filtering an input signal. Considering a frame-based re-synthesis of a speech signal, where coefficients of $H(z)$ change with each frame of a usual $20 - 30\,ms$ length, there is a problem of handling the state of a constructed filter for $H(z)$ at the moment of a filter change taking place. A possible solution is to use the property of the transfer function being decomposable to a sum of the corresponding ZIR and ZSR in order to perform continuous filtering on a derived excitation signal $e(n)$. Formally (based on [13]):

$$\mathbf{S} = \mathbf{S_{ZIR}} + \mathbf{EH} \quad \mathbf{E} = [e(0), e(1), \ldots, e(L-1)]$$

with the matrix product $\mathbf{EH}$ being the zero state response of $H(z)$ using the L dimensional vector $\mathbf{E}$ as the input signal. In this case, $\mathbf{H}$ has to be a matrix containing samples of the impulse response in the following form:

$$\mathbf{H} = \begin{pmatrix} h(0) & h(1) & \cdots & h(L-1) \\ 0 & h(0) & \cdots & h(L-2) \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & h(0) \end{pmatrix} \tag{2.18}$$

A non-formal explanation is that the zero input solution is the response of the system to the initial conditions, with the input set to zero. The zero state solution is the response of the system to the input, with initial conditions set to zero. The complete response is simply the sum of the zero input and zero state response.

Chapter 3

Conceptual Design of the Talk-box Emulation

This chapter will outline a design plan of an audio effect using the previously described theoretical basis. The goal is to use illustrative design tools like block diagrams, to explain each part of the system and support technical decisions with proper reasoning. Additionally, outline some alternative and potentially viable solutions if found to be appropriate.

3.1 Technical description of the idea

The basic concept was roughly introduced in previous chapters. This section will guide the reader through a more specific description. The upcoming plugin is intended to be an extension effect module for home or studio recording setups, let it be called the *formantfilter* from now on. In practice, it requires minimalistic and usual equipment including a microphone and virtually any kind of musical instrument like a synthesizer or an electric guitar. A recording hardware interface is also implicitly assumed.

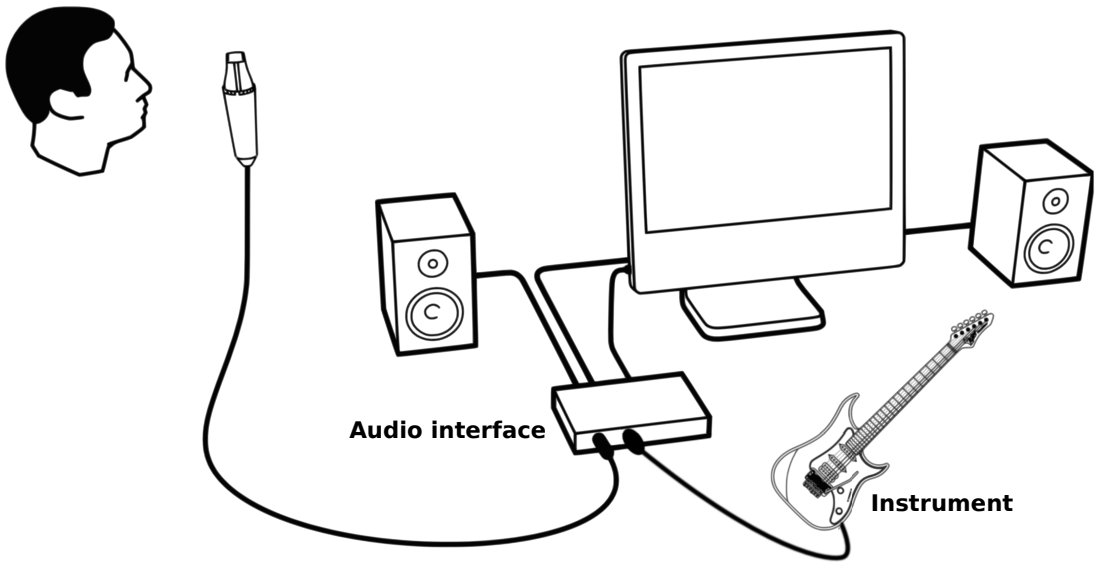
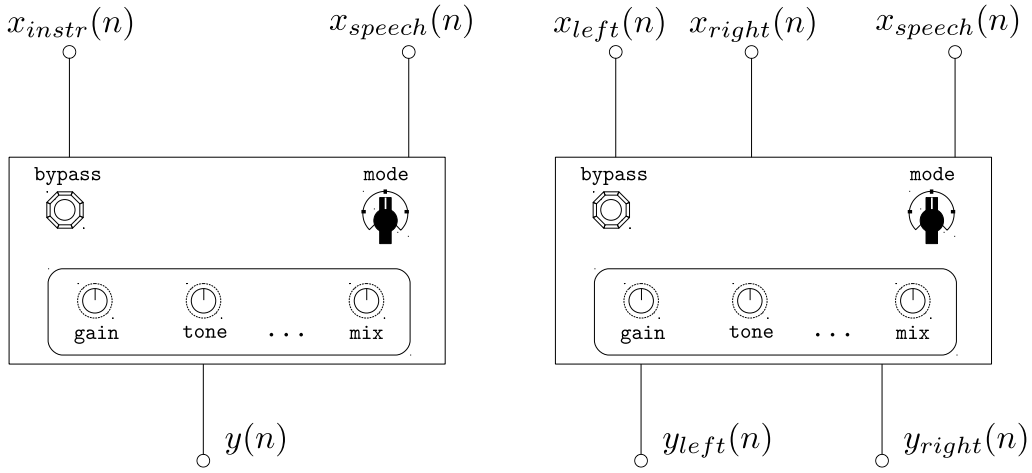


Figure 3.1: A minimal hardware setup to use the plugin.

The *formantfilter* requires two input signals: one for the instrument and one for the microphone. In order to set up the DAW to correctly run the plugin, a two channel *bus* can be added to the session with the inputs routed accordingly. Using the effect consists of playing the instrument and making vocal sounds (noises, whispers, etc.) simultaneously. The vocal does not have to be in tune with the underlying harmony, the pitch of the vocals should not have any effect on the note being played whatsoever. The instrument dictates which note will be heard, instead of the vocals.

The effect should have a few control parameters, though not too many to keep it user-friendly. The main function is to synthesize speech with the instrument sound, but other modes of operation can be imagined as well, like mixing the original instrument or vocal signal with the synthesized result. A variation or *mode* where the vocal signal is pitch shifted to the tone of the instrument and added to the resulting sound was also considered with various cross synthesis methods (using just the excitation pitch shifted vs. the complete signal...).



(a) Mono version

(b) Stereo version

Figure 3.2: Illustration of plugin instances.

A stereo and a mono configuration are taken into account, these are depicted on Figure 3.2. Ideally, the plugin logic should use independent processing chains on each channel. More channels than two are not likely to be utilized during usual operation.

A straight-forward attitude is to implement the complete algorithm offline with a rich signal processing toolkit like the one offered by MATLAB. The real plugin version with online processing can be assembled afterwards. Finally, a unique GUI can be crafted for the plugin to be more competitive against similar products. With an intention of using multiple plugin technologies (VST, LADSPA, etc.) for various platforms, an independent and simple C/C++ framework is also desired, which can be:

1. An external project attempting such issues.
2. A completely new framework written from scratch.

The first attempt would introduce an essential dependency at the very basic level, and the only considerable framework – the JUCE library – seemed to be a promising but incomplete solution, with a lot of unnecessary modules which produce a potentially huge binary.

For the above mentioned reasons, the decision is to use a tiny but functional framework written from scratch in C++. This allows a clear formulation of the final algorithm itself which will not depend on any particular API, and comes with maximum portability. Figure 3.3 shows how the algorithm will be defined over a unified plugin interface. Chapter 4 reveals further details about the implemented layers.

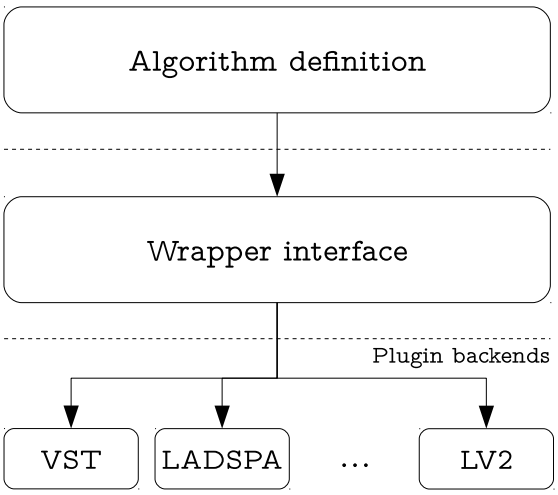


Figure 3.3: Scheme of a custom plugin framework.

3.2 Requirements and constraints

As the section title suggests, a brief summary will follow concerning the technical requirements. Being a plugin which has to support real-time processing, it is appropriate to investigate the maximal acceptable latency values, and other ways of reducing the perceived delay from the input to the output even if the physical buffering is limited to a certain amount of samples.

As stated in [14], the maximum time of human-perceptible audio latency is assumed between 20 – 30 ms. This can be assigned to a duration of a single analysis segment. The frame length is not the only factor that contributes to a latency-free operation. A smartly structured analysis window has to be used to avoid artifacts and distortion with minimal delay. Most LPC coders use 20 ms frames which are further divided to sub-frames, or the frames overlap [13].

A 20 ms long duration for each frame implies that all the calculations have to take place in the aforementioned timespan, but the situation gets more complicated by recalling the basic mechanism of audio signal processing within a plugin hosting environment. Section 1.4 gives a rough explanation of how the plugin logic is initiated by calling the `process` method (might have different names across APIs). Scheduling this callback function is completely up to the host. More precisely, it is dictated by the low-level audio subsystem of the given platform (ASIO, JACK, Coreaudio) with adjustable buffer size that can be certainly expected under 20 ms. With a well suited platform and hardware setup, this duration can be easily under 10 ms. With respect to the actual sampling frequency (usually between 44.1 kHz up to 192 kHz inclusively), the corresponding sample count is delivered

as a parameter to the `process()` function with each callback. On the other hand, successive speech analysis cannot be performed on frames with much shorter length than 20 ms. Therefore, the samples have to be collected across consecutive calls, and analyzed when the desired amount of samples become available.

As mentioned before, the processing algorithm has to adapt to different sampling frequencies, potentially variable input and output channel count, unpredictable amount of samples for each call and to fit the longest processing path into the time-span of one callback. Not satisfying these constraints can result in tiny clicks and pops also known as *Xruns*[1], or even a completely unrecognizable output.

3.3 Algorithm design and decomposition

With respect to the previously described issues and using the information gathered in Chapter 2, the next step is to outline a general solution for the desired *formantfilter* effect. This section will go through the planning stage having no feedback from the described solutions yet. Chapter 4 should provide the necessary information of how the discussed solutions perform and decide which one of them is going to be used finally.

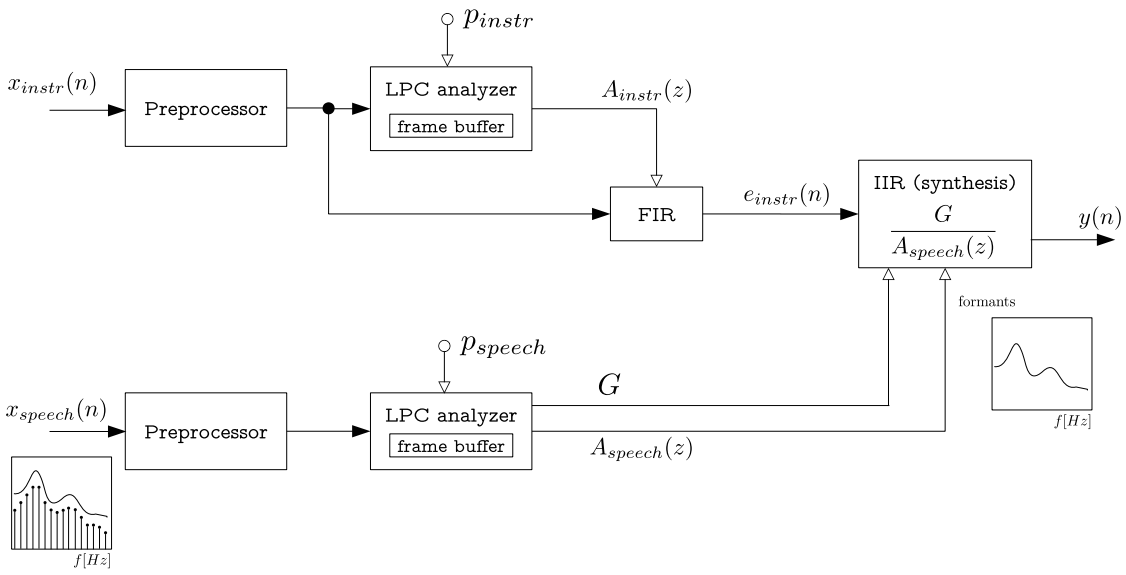


Figure 3.4: Generalized block-scheme of the *formantfilter*.

Figure 3.4 shows the basic structure under consideration. It addresses a case with a single instrument and a voice channel, which corresponds to a mono configuration depicted on Figure 3.2a. A stereo arrangement can be imagined easily by duplicating the upper part of Figure 3.4, thus yielding two distinct output signals. Each channel has a generalized preprocessing block that includes basic operations as removing the DC offset, or adjusting signal levels. The `LPC analyzer` has the responsibility to perform a source-filter separation with linear prediction and estimate the coefficients for the inverse filter $A(z)$ and the corresponding gain G for the analyzed frame. Parameters p_{instr} and p_{speech} denote the prediction

[1]Term used collectively for events like delivering the processed samples too late or not accepting the provided ones in-time.

order for both the instrument and the speech analyzer. The idea is to take the formant structure of the speech signal and apply it to the calculated excitation $e(n)$ of the instrument. Bold arrows are showing the path of audio samples while the white arrows indicate the parameters to the processing blocks. In order to remove the spectral envelope from the instrument signal, it is filtered with an FIR filter given the coefficients from $A(z)$. The final stage is an IIR synthesis filter with the input being the residual from the instrument signal and $A(z)$ placed in the feedback loop. This is an essential cross synthesis scheme and it serves as a starting point for further development. A variation (Figure 3.5) where the original signal is fed straight to the synthesis filter is also under consideration, the deciding factor is mainly the resulting sound texture. Using only the instrument excitation could destroy its musical signature over a benefit of more accurate speech reproduction. On the other hand, the lack of higher harmonics and noise may hold back the proper excitation of the spectral envelope when the original instrument signal is used. A weighted combination of the two is also possible with a dedicated controlling parameter as their ratio.

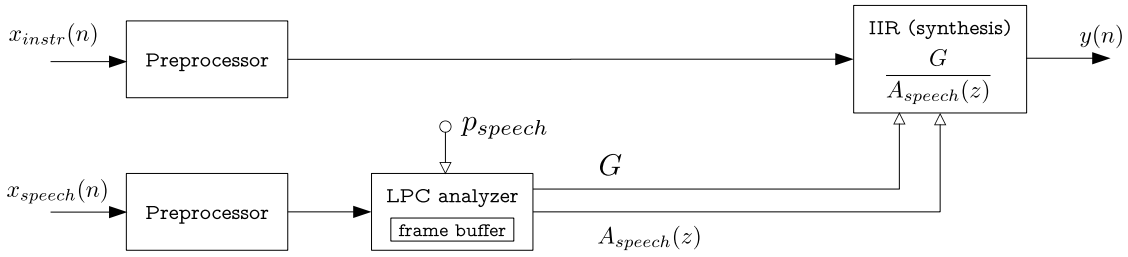


Figure 3.5: Synthesis with the original instrument signal.

Let the `LPC analyser` be examined in more detail. This should include a buffer for collecting the input samples and arranging them into a stream of frames. The correct placement and the frame size are also affecting the sound quality, processing time and perceived latency. A few possibilities of sample buffering and windowing will now be discussed.

The simplest possible structure is to use a buffer with the appropriate size and collect the incoming samples till the buffer gets full. When this happens, calculate the inverse filter $A(z)$ and repeat the whole procedure. Figure 3.6 depicts the situation considering a window function for multiplying the frame samples after being ready for analysis.

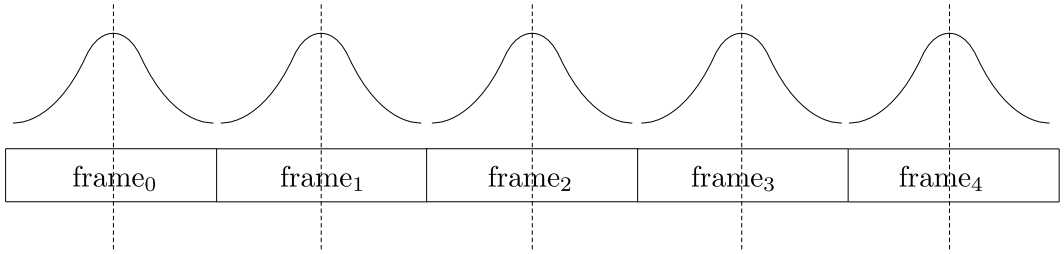


Figure 3.6: Simple buffering of samples into frames and windowing.

The solution in Figure 3.6 has an advantage of simplicity which makes it less vulnerable to errors, requires relatively low processing power and can be developed and tested faster than the other, more complicated variations. As a major drawback, the minimum latency

has to conform to the exact duration of one frame which is at least 15 ms even with the shortest usable frame size. Additionally, the accuracy of prediction may not be acceptable.

An enhanced version of the previous structure is shown on Figure 3.7. This arrangement is suggested by [9] and its focus is to maximize the positive effects of the window function and the accuracy of prediction with an implicit overlap of the windowed segments. The samples are collected into a window with twice the size of a frame which is centered under the window segment. This arrangement is complicated for implementation and is taken as a backup plan if every other fails.

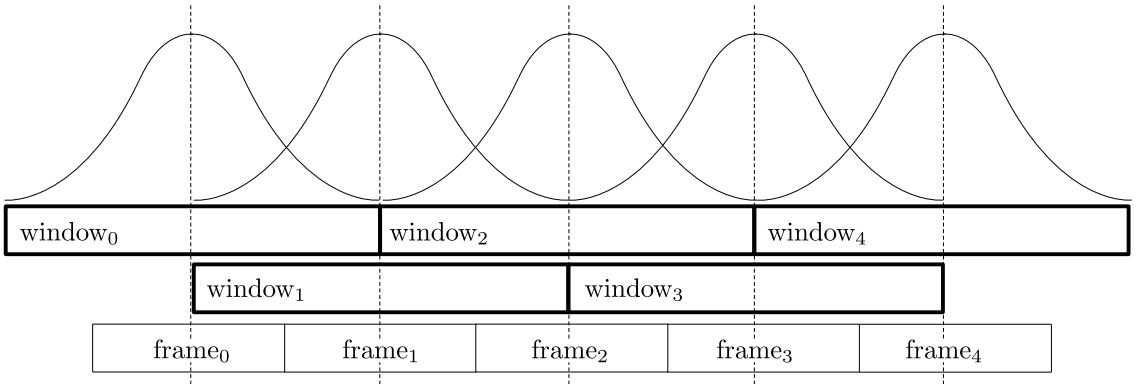


Figure 3.7: Enhanced buffering and windowing.

A compromise between the previous two arrangements is a simple but overlapped frame structure illustrated on Figure 3.8. The exact size of the frame and the overlap still remains a question. The next section will try to provide some estimations related to such values including prediction order and suitable window functions. However, the final decision relies on practical experiments as well. Although this arrangement is a bit more complicated than the simple structure on Figure 3.6, it is tempting due to the scalable overlap and thus the overall latency.

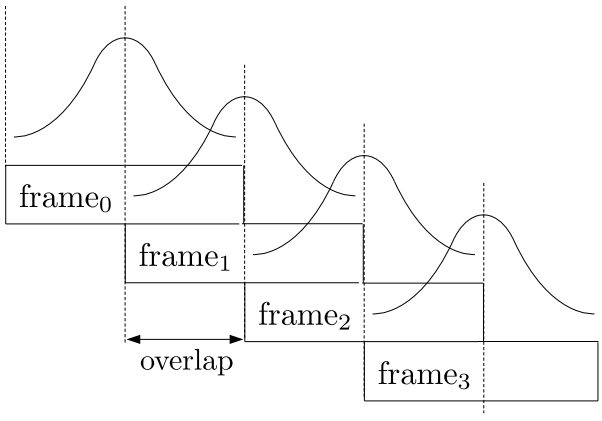


Figure 3.8: Simple overlapped buffering and windowing.

3.4 Choosing the optimal parameters of prediction

The development time needed for examining all possible parameters, measuring prediction gains, and generally playing around with different parameters blindly is unacceptable. This section will gather the best recommendations found in the literature concentrating on audio quality and performance. The parameters in question are:

- Sampling frequency
- Prediction order
- Frame length
- Type of window function

Parameters are qualified by the ability to give a reasonable accuracy (though not necessarily perfect) with no unpleasant frequencies, buzz or anything non-musical. Secondly, the processing time has to be minimized thinking of the fact, that in a real situation there are possibly multiple plugins and effects running simultaneously, sharing the same resources. The primary measure of accuracy for linear predictive coding is the prediction gain defined by equation 2.12. It depends mainly on the prediction order and the size and shape of the analysis window.

Sampling frequency The sample rate affects nearly all parts of the processing chain. "Fortunately", audio plugins are not responsible for choosing the exact value, they just need to be prepared for the possible variations (or inform the host about the inability to use a particular sampling frequency). The majority of audio interfaces operate with a sampling rate of $48\,kHz$, but more extreme products can easily reach $192\,kHz$. A higher frequency also requires more processing power, not only for the increased number of samples that has to be processed. The following discussions will highlight that – besides other factors – the prediction order has to be increased as well, resulting in higher CPU usage.

Prediction order As stated in [15], the memory of the inverse filter $A(z)$ has to cover at least twice the duration required for sound waves to propagate from the glottis to the lips which is $\frac{2L}{c}$ where L is the length of the vocal tract (cca. $17\,cm$) and c is the speed of sound ($340\,m/s$). This means that a memory of at least $1\,ms$ is required and by speaking of the filter $A(z)$, the memory is equal to the number of coefficients, thus the prediction order. The exact order has to be chosen prior to the usual sampling rates that audio plugins are dealing with. As stated in section 3.2, the expected sampling frequencies should be around 44,1 and 192 kHz depending on the audio interface. Generally,

$$\frac{10^{-3}\,[s]}{F_s\,[Hz]} = F_s \times 10^{-3}$$

coefficients are required which leads to roughly an order between 41 and 192 for a completely intelligible speech reconstruction (which is not *primarily* the case). Moreover, this deliberation does not take into account the glottal and lip radiation characteristics and other factors that would lead to even more coefficients. At the same time, these values are to be taken as a minimum, meaning that for the sake of audio quality, they may have to be increased further. At the same time, if the sound quality is not degraded, there is no need to make a super-accurate speech reproduction, as the final product is intended to be a musical effect in the fashion of the talk-box, and not a speech encoder.

Frame length The previous section has outlined a few solutions for buffering and windowing. The proper size of the analysis window is evidently dependent on the sampling frequency. Several actions happening in the vocal tract are shorter than the many times mentioned $20\,ms$ time-span which is considered as a usual frame length in speech coding. As explained before, accuracy is not the primary concern, therefore, this value seems to be acceptable without further discussion. However, given the fact that $20\,ms$ is an almost noticeable latency for a musician – especially for a singer with a microphone – and knowing that the latency accumulates with other effects potentially working with small but existing latencies, this value can be troublesome. Using overlapped frames can solve the physical delay from the input to the output, but the perceptual latency would still remain around the length of the frame itself, although slightly decreased with the overlapping prediction coefficients. Using a $15\,ms$ overlap should not be noticed at all by the ears of listeners and performers. With a bit of luck, the slightly lagging formant structure will be unnoticeable as well.

Type of window function Perhaps the most difficult recommendation is the type of window function. Section 2.5 has introduced common functions used in signal processing. Presumably, many of these would produce very similar results — which makes the decision mostly an experimental concern. The primary literature on the subject of audio effect creation [6] suggests the *Hanning* window, although it does not provide an exhaustive reasoning regarding this choice other than the good periodicity with the edges of the function close to zero.

Chapter 4

Implementation of the Audio Plugin

Now that all the crucial parts have been introduced and several recommendations are available for the most significant issues, this chapter will describe the development process with all the experiments and final decisions.

4.1 Development environment and workflow

The development platform was chosen to be a GNU/Linux system for being free and open-source and having excellent low-latency audio capabilities. As a drawback, it may be lacking the myriad of DAW applications found to be available for other major platforms. Nevertheless, *Ardour3* can be an acceptable reference DAW, despite of being a beta version (see Figure 1.5 for screenshot). The straight-forward workflow is the development of the algorithm in Matlab, followed by the adoption to a plugin API of choice. This approach was slightly extended during the programming work. Offline operation could generally mean any non-realtime usecase, which is amongst the basic capabilities of every plugin framework. With a bit of effort, the off-line plugin can be bridged into Matlab that is an overall workbench for the development. While some parts were implemented first in Matlab, others started their existence in C++. At the end, all the parts were transferred into C++ and incorporated into the final program. The bridge between the two environments was a simple `MEX` function capable of loading the plugin and sending/receiving the audio and control data via function arguments and return value.

In order to shorten the development time, the *Itpp*[1] library was used for several signal processing operations in the C++ code. In some ways, it resembles the functionality provided by Matlab, which is reflected in the function signatures as well.

The easiest way to run real-time audio applications under GNU/Linux is to use the *Jack Audio Connection Kit*. With a correct configuration and a dedicated real-time kernel, the reference system's (see appendix B) latency has been successively set to $5\,ms$, which corresponds to 256 samples under a sampling frequency about 48 kHz. Fortunately, several Linux distributions are dedicated to audio or multimedia related tasks and come pre-installed with applications like Ardour3 and Jack. Additionally, they are pre-configured for real-time use. An example of such distribution is the one called *AVLinux*. The easiest way to try the plugin is probably to use a LiveUSB setup with a similar distribution.

[1] http://itpp.sourceforge.net/

4.2 Software framework

A custom C++ framework was written to serve as the playground for audio related classes, hiding and unifying most of the plugin specific overhead. This framework was initially built on top of the LADSPA API due to its simplicity. Figure 3.3 shows this concept being a wrapper interface layer. Special care was taken for using *virtual* functions only when necessary and avoid them in places with frequent usage (e.g., accessing the audio samples) as this could degrade the performance right from the beginning. Figure 4.1 shows a simplified class diagram of how the framework can be imagined. The class `AudioFilter` is an abstraction of a plugin that uses ports for audio data and control parameters. The subclass `FormantFilter` derived from `AudioFilter` is the plugin under development. Ports and parameters are captured in class `Port` and their accompanying subclasses. The real framework defines several more subclasses for `ControlPort` to capture different types of controls like a linear or logarithmic pot-meter, a switch control or a selector for various modes of operation.

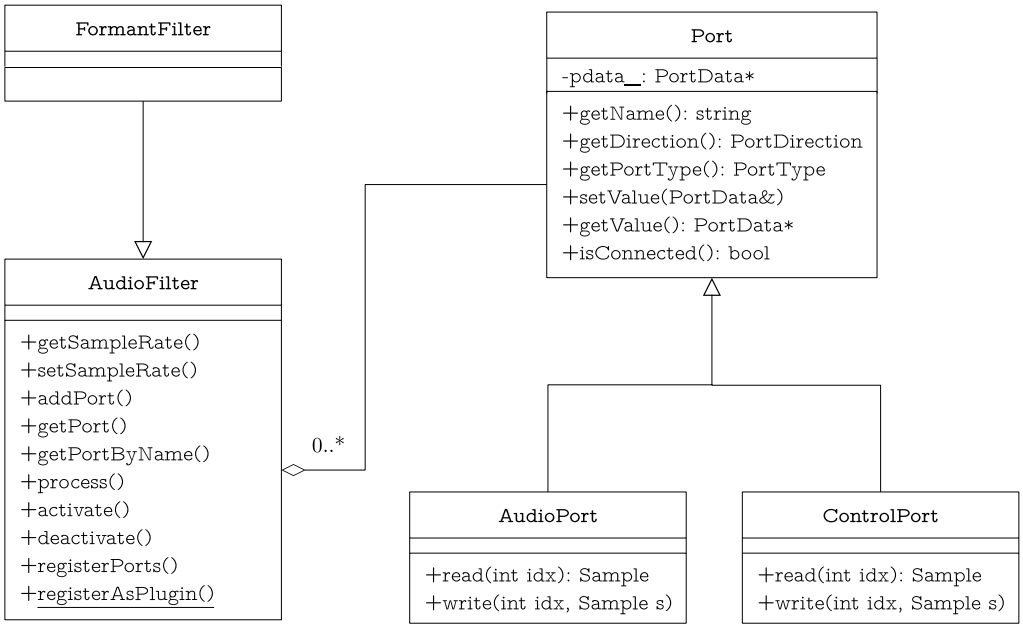
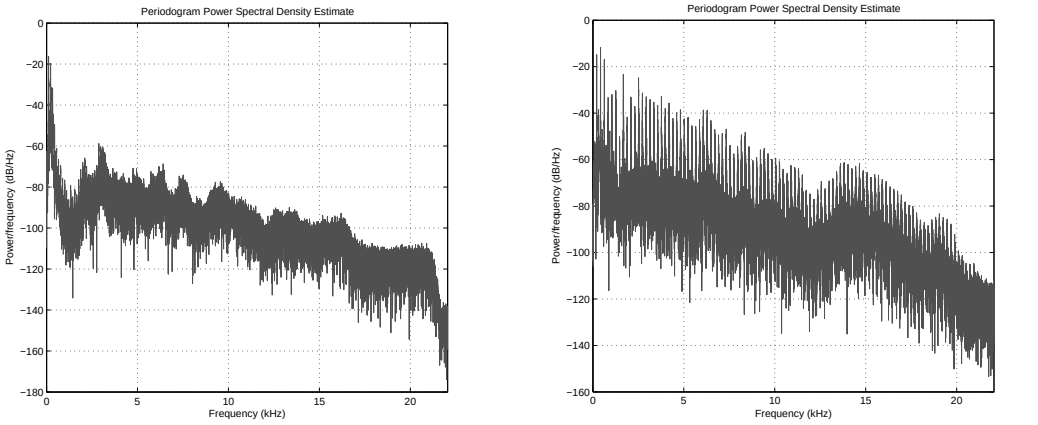


Figure 4.1: Simplified class diagram of the developed software framework.

The implementation of specific structures and methods are left to the module representing a particular plugin technology. For example, if a plugin is going to be built for LADSPA, then the appropriate module has to be specified for compilation. This is the module where the declared structures are actually defined, like the private `PortData` which varies across plugin technologies. Several plugins can be compiled at the same time by automating the compilation. The low-level module implementing the LADSPA interface can be found in the `ladspaplugin.h` and `ladspaplugin.cpp` source files. Others plugin technologies can be implemented similarly. Due to the limited time frame, only the LADSPA and the JUCE back-end was implemented exclusively.

4.3 Development stages and milestones

Although the concept of cross-synthesis may seem to be a straight-forward procedure, a lot of issues occurred during the development. It took a long time to establish a clear sound without any artifacts, noise, clicks, pops, buzz, or any unpleasant features. The biggest challenge was to get rid of the high energy peaks occurring when a new inverse filter $A(z)$ was ready to replace the old one. Many solutions had been proposed addressing this issue. A short pair of audio samples (about 2.5 seconds) containing an *"i"* vowel and a musical *"A"* note will be used for demonstrating different approaches of cross-synthesis to show how the audio output evolved to be acceptable. The spectra of the two input sounds are shown in Figure 4.2. A lot of experiments and repeated evaluation were performed with only tiny incremental changes in the designed architecture, therefore the following part of this section will present only the significant changes made towards the final solution.



(a) Spectrum of the *"i"* vowel

(b) Spectrum of an *"A"* note (dist. guitar)

Figure 4.2: Illustration of plugin instances.

The first attempt was to use the plain coefficients of the $A(z)$ filter and a usual Hamming window for multiplying each frame. The output generated this way was full of distorted high frequencies both with Matlab and in C++ using the filter classes provided by the Itpp library (Figure 4.3). The filter's state was saved before changing the coefficients and restored right afterwards.

Initially, two possible solutions were proposed:

1. Interpolate the coefficients using a suitable representation like the line spectral frequencies

2. Use the *zero input response* of the synthesis filter for smoothing out coefficient changes instead of saving and restoring the filter memory (see section 2.7).

Additionally, a constant assumption was that a correct window structure discussed in section 3.3 can always reduce unwanted sound features. Interpolation of the coefficients was not a welcomed solution as it requires high amount of processing overhead, considering both the coefficient conversion from the LP domain and the linear interpolation itself. For this reason, the second alternative was implemented being simple enough to be acceptable as a fast solution.

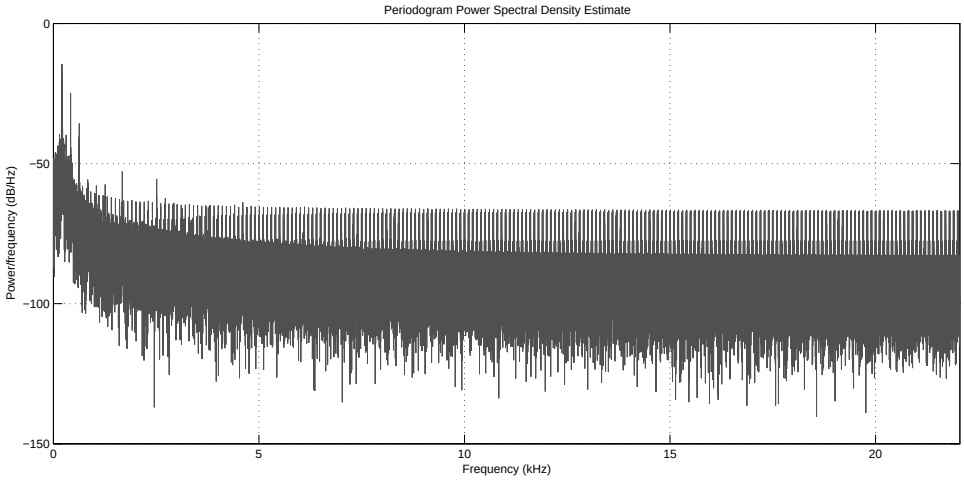


Figure 4.3: Output PSD of using plain LP coefficients with state-restored filters.

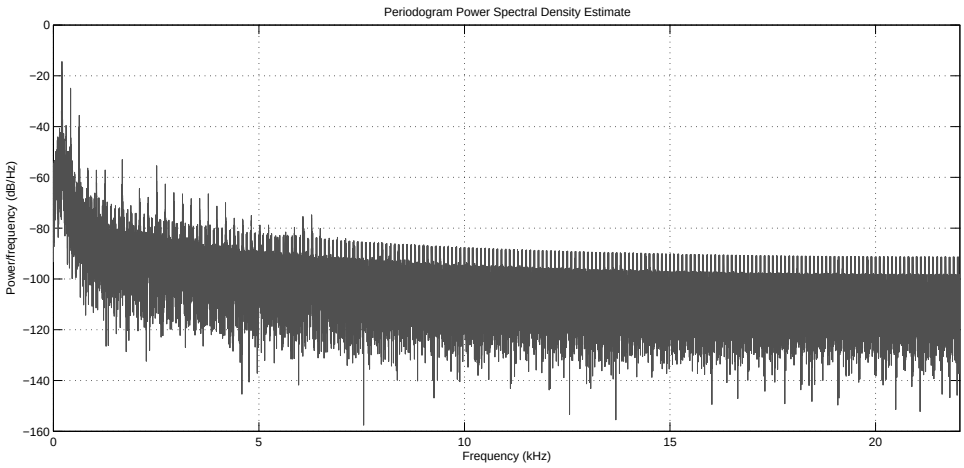


Figure 4.4: Output PSD of using ZIR and ZSR for cross-synthesis.

Using the zero input response and zero state response for cross-synthesis has made a few enhancements. With some instrument input signals, the unwanted frequencies were almost completely gone from the output. However, the perceived presence of the formant structure seemed to appear weaker than previously. Nevertheless, using a real unfiltered distortion guitar sound with the sample *"i"* vowel produced an output with still unusable buzzing high frequencies (the spectra is shown in Figure 4.4). The weakness of the applied formant structure has been considered to be the consequence of poor harmonic content of the input instrument signal. If there are no frequencies at the position of the formants, there is nothing that excites the $A(z)$ filter on those positions and the formant structure will not be strong enough on the output signal. For this reason, an embedded nonlinear distortion (or overdrive) effect was built into the plugin, with a gain control parameter on the UI. The nonlinear processing can produce richer harmonic content in case the problem above really exists. If it does not, a built-in overdrive can still come handy in many situations. The "A" note used with these outputs is a distorted guitar sound, therefore, this problem

is eliminated.

Eventually, the issue with filter changes was found out to be caused by the implementation of filters used both by Matlab and the Itpp library. Namely, the canonical or *Direct Form 2* (see section 2.6). This form has a very bad behavior with instant coefficient changes even with saving and restoring the filter memory. DF1, however, has shown to have much better properties regarding this issue. To completely smooth out the filter changes, the final structure of the frame buffer was chosen to be similar to what is illustrated on Figure 3.8 with a massive overlap. After every $5\,ms$, a new frame is produced from the actual window content by multiplying it with a periodic Hanning window function. This way, a high amount of correlation is introduced between the coefficients of consecutive inverse filters and the perceptual delay is also minimized. Figure 4.5 shows the mentioned *"i"* vowel and *"A"* note synthesized with the final approach. It shows that the formant structure of the input speech is perfectly recognizable. Higher frequencies are still present, but being mostly periodic, it can be addressed to the distortion effect of the guitar tone.

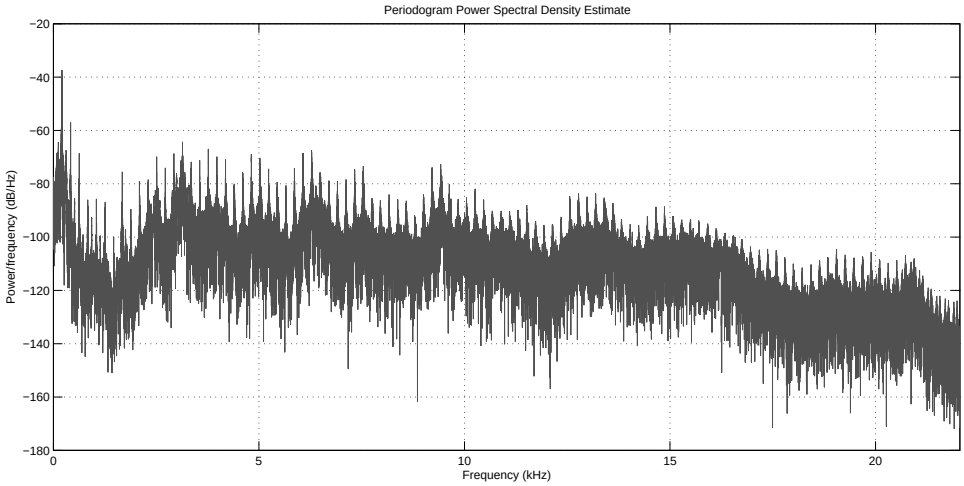


Figure 4.5: Final output spectrum with a DF1 synthesis filter.

4.4 Assembly of the plugin and final notes

The signal processing aspects of the *formantfilter* plugin are depicted on Figure 4.6. A few controlling parameters have been added, these are shown at their respective positions. Perhaps the most interesting feature is the one named *presence* that is a balance specifier between two synthesis paths described in section 3.3. The two approaches are implemented at the same time and their respective amount can be selected with a simple coefficient from the interval $\langle 0, 1 \rangle$. In the first path, the instrument excitation is extracted, while in the second path, the signal is left unprocessed and goes straight into the synthesis. Other parameters are the *gain* that specifies the amount of distortion on the input and the *level* which is the overall signal level of the plugin. The latter is needed to compensate the wide range of input signal levels that are affecting the output as well. In order to prevent potential overflow of the signal with wrong settings, a *limiter*[2] is placed at the end of the processing

[2] A nonlinear signal processing block that detects possible clipping and holds back the gain to decrease the amount of distortion.

chain. It was shown in many practical use-cases, that this is a welcomed feature. The *mix* parameter is a "joker" control which specifies the amount of the original instrument signal that is mixed to the processed one. This can be achieved with the DAW as well, therefore, it should be less important, but cannot be ignored due the offered creative potential.

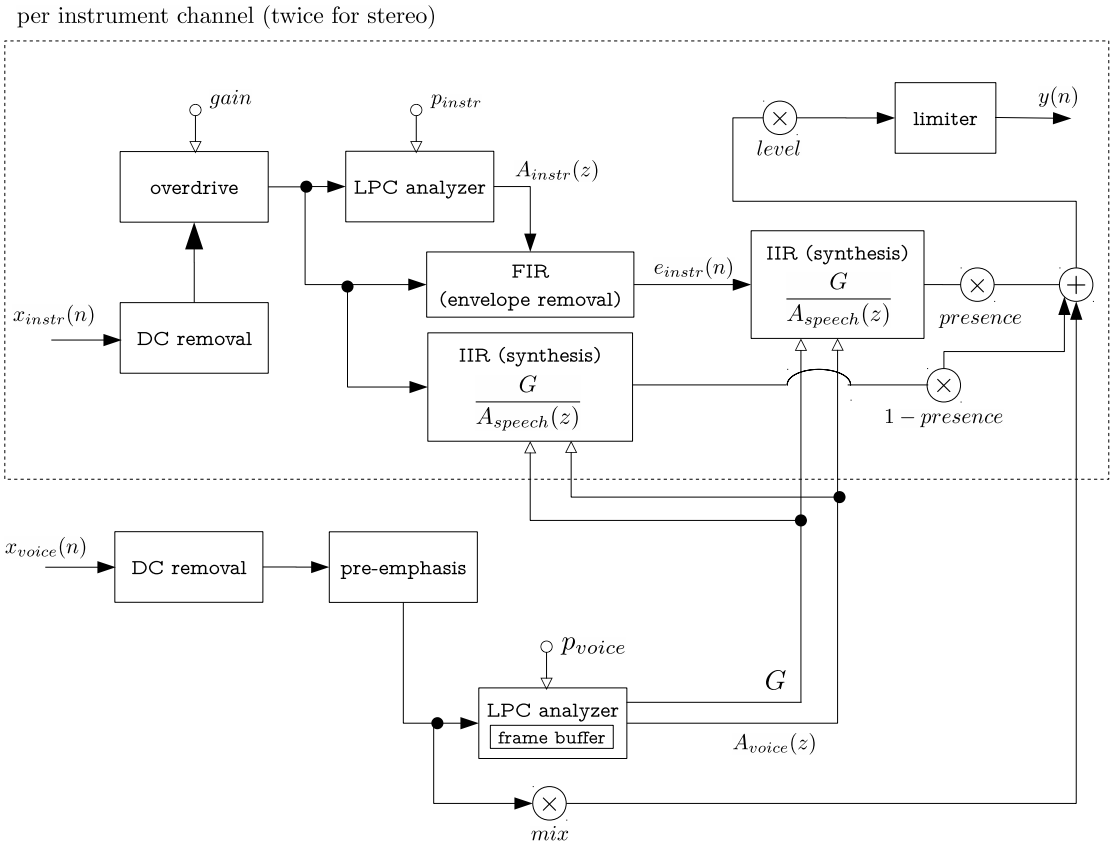


Figure 4.6: Detailed scheme of the *formantfilter*

The function of the `overdrive` block can be expressed with the following formulas[3]:

$$k = \frac{2a}{1-a} \tag{4.1}$$

$$x(n) = \frac{1+k}{1+|x(n)|} \tag{4.2}$$

with the coefficient a determining the amount of distortion — this value is controlled by the *gain* parameter (from the interval $\langle 0, 1)$) shown in Figure 4.6.

An important factor is the interoperability of the plugin, and the overall dependency on other signal processing blocks. Ideally, the effect should be usable without any further processing. On the other hand, it should not limit other effects to be placed before or after the *formantfilter* in the signal chain. For example, the microphone input left uncompressed leads to huge dynamic variations that can be hard to handle, or may just be the thing

[3] Matlab code available at www.musicdsp.org

that a musician is going for. A solution could be to have a built-in compressor at the input of the speech signal. Knowing that a compressor is amongst the most common plugin effects, writing a custom version, or even using an external (open-source) project was not a priority. However, if a boxed, hardware distribution of the effect is considered, these would be incorporated, together with better equalization and a true bypass switch.

An additional note addressing the effect usage is the optimal position in a typical signal chain. Focusing on the electric guitar, the *formantfilter* can be considered to be a *post-distortion* effect, which means that at least the overdrive effects should precede it. Spatial effects, on the other hand, should be placed after the *formantfilter*. Despite this recommendation, experimenting with different arrangements is not forbidden and may lead to usable results.

A unique user interface for the plugin was initially planned as well, but did not make it to the final release, due to the limited time frame and the fact that it can be generated perfectly. Figure 4.7 shows the automated GUI created by *Ardour3* during an editing session. The indicators on the right side are displaying the exhausted CPU time compared to the available time interval between consecutive audio callbacks. These are implemented using just a few lines of code and are intended for testing and measuring purposes.

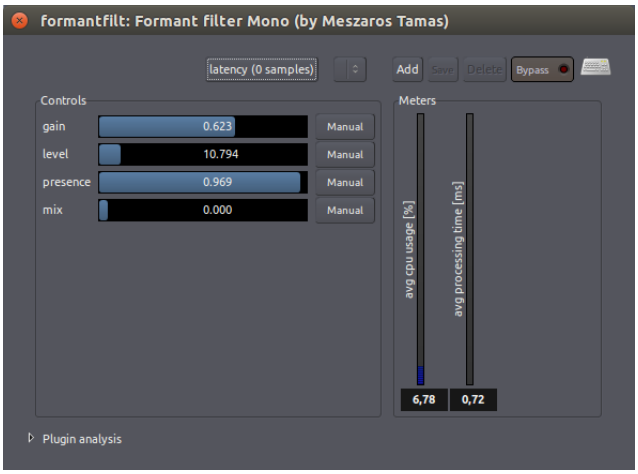


Figure 4.7: Plugin UI generated by *Ardour3*

4.4.1 Compilation and build issues

The filtering problems mentioned in the previous section has led to a decision to write the basic signal processing operations in C++ from scratch including digital filters, basic algorithms and calculations together with linear prediction. These are grouped under the `sigproc` namespace. This way, the Itpp library is not a crucial dependency anymore and used only for building unit tests. The CMake configuration will ignore these tests if the Itpp library is absent and the compilation of the plugin should not stop.

The JUCE library has a custom build system, this was integrated into the existing CMake project by generating the JUCE configuration files straight with CMake. This solution makes it possible to use the *formantfilter* in applications supporting VST plugins and the advantage over writing a dedicated VST backend is that the JUCE library can automatically compile the plugin into many formats if the appropriate SDK is provided. Using the JUCE library as an optional and temporary backend still makes the project

compilable without this heavyweight dependency. By configuring the project without the JUCE SDK, only the LADSPA backend will be compiled, but other dedicated backends are waiting to be implemented. More details about the framework can be found in the doxygen generated documentation and `README` files.

The supported compilers are the `GCC 4.8` and above with finished C++11 standard and the *Microsoft Visual Studio 2013 C++* compiler on Windows systems. However, the Windows version will compile only the JUCE backend for which both the JUCE SDK and the VST SDK is required. *The JUCE plugin version is included rather as a tech-preview and has not been tested extensively.* Compiling the complete solution does not differ from a typical sequence of commands used by CMake projects. More details can be found in the project `README` files regarding the compilation process.

Chapter 5

Testing and Evaluation

5.1 Technical testing

Several unit tests were developed to ensure flawless operation of the basic low level functions like filtering, ring buffers used for windowing, and overall audio data pass-through tests. These can be initiated with the `make test` command after building the project tree.

In order to measure the consumed processing power, the plugin was equipped with two additional output ports, emitting the measured processing overhead. Let T_{cb} be the time available between consecutive calls to the `process()` method. This interval can be calculated from the product of the sampling frequency and the provided sample count — both values are available at any time during the processing. Furthermore, let T_s be the time spent in the `process()` method. The ratio of $\frac{T_s}{T_{cb}}$ gives an indication of the current processing overhead which can be observed in Figure 4.7. The occupation of the CPU was below 10 % almost all the time on the development hardware (see Appendix B).

5.2 User testing

To examine the plugin as a product, it is very hard to define objective measures that can be put under evaluation. The generated sound quality is free of unwanted artifacts, every other feature is dependent on the actual signal chain in which the *formantfilter* effect is located. Therefore, a survey is used to collect user opinions in a structured manner with the following questions:

1. How do you rate the overall output quality?
   (a) Poor with unpleasant features, clicks, pops, noise or non-harmonic frequencies.
   (b) Usable but not perfect.
   (c) Completely acceptable.
   (d) Excellent.
2. Can you imagine a musical situation, where you could use this effect?
   (a) No, not at all!
   (b) Yes, in rare situations.
   (c) Yes, in many occasions throughout a single tune.

(d) Yes, complete songs can be based on this effect.

3. Do you find the effect capable of co-operating with other existing effects?
   (a) No, it always sounds the same with any input signal.
   (b) Yes, but only limited sound textures are usable (e.g., only clean or distorted sounds).
   (c) Yes, the output sound responds meaningfully to every input variation.
4. To what degree is this effect distinguishable from existing vocoder effects?
   (a) Sounds completely like a channel or phase vocoder.
   (b) Similar but has some unique features.
   (c) Completely different.
   (d) I have no experience with vocoder effects.

From ten musicians (mostly guitarists) only four has provided a dedicated review (see appendix A for the full reviews). Others have completed the test survey only. The results are collected in Table 5.1 with a value indicating the number of testers giving an answer respective to the cell position. The **Overall Success** column should indicate an intuitive summary from the received answers.

An interesting conclusion can be made based on the answers for question 4. The majority of guitarists does not use vocoder or talk-box effects making it hard to get a reasonable end-user comparison to similar products. It was a new experience for nearly all the test subjects, mostly with a positive impression.

Question number	a	b	c	d	Overall Success
1. Output quality	0	0	8	2	★★★★★
2. Usability	1	3	5	1	★★★☆☆
3. Cooperativeness	0	4	6	-	★★★★☆
4. Novelty	1	2	1	6	★★★★☆

Table 5.1: Results from user reviews

5.3 Demonstration of the effect in-use

To demonstrate the usage of the final plugin, several example sessions are provided both in exported `flac` format and in a complete project representation for *Ardour3*. The session files contain all the effect configurations and signal routing for the plugin itself with other accompanying effects and the recorded audio regions, to illustrate a practical recording session.

Additionally, a cover version of the famous song *Livin' on a Prayer* containing well known marks of a talk-box effect was re-assembled using the created plugin. Here, a backing-track free of guitar parts was used as a basis, and the traces of the talkbox are replaced using the *formantfilter* effect. Furthermore, the solo part is also enhanced with it.

A second track is a simple improvisation with various articulations in the solo guitar part.

Chapter 6

Conclusion

6.1 Summary of the performed work

The official goal of the thesis was to create an audio effect making use of speech analysis techniques. The method for formant extraction was chosen to be a type of linear prediction. After getting familiar with various speech detection and synthesis algorithms, an effect was designed inspired by the legacy of the *talk-box*. Alternative textures can also be dialed in with the provided controlling parameters. The designed unit was packaged into multiple software plugin formats to be usable in today's audio workstations. Special effort was made towards a cross platform implementation, with a simple framework supporting various plugin technologies. Finally, the plugin was tested by musicians and a summary was made about the practical usability and quality of the effect.

Despite the fact, that the idea of cross-synthesis is by no means a novelty and quite a few theoretical sources are available regarding source-filter processing, implementing a nice sounding audio effect requires a lot of experiments. Small parameter changes can make a difference between an unusable output signal and a completely appealing sound. A good example is the problem with the canonical filter structure and its bad resistance to coefficient changes that caused a lot of trouble and delay during the development stage. The effect was tested for real-time performance and latency free operation. As far as the perceptual limits go, these requirements were satisfied with no inconveniences and effortless playability.

Fortunately, the audio quality and texture was not subject to any complaints. *Dávid Zoltán* has made a comment, that the effect needs several external processing blocks to get a nicer result (equalizer, compressor), making it less comfortable: *"...As a drawback, I could mention the need of external effects for a really usable sound, like the compressor for the microphone input or an overall equalizer – perhaps a simple tone and bass parameter would be enough..."*

Gábor Szabó has made a similar comment, stating: *"...As a backdraft of this effect I would mention – it's complicated. I would like to try it either as a part of a vocal performer multieffect (like the ones that are made by TC Helicon) or as an independent effect pedal. But using it through a computer – it might be useful, but personally I would refuse this kind of option...."* As a reaction, software plugins are not far from being utilized in hardware multi-effect units. A few solutions – like the *MOD duo* project[1] – are being on the way, loaded with an embedded Linux system for hosting plugins. Therefore, a suitable hardware

[1] http://portalmod.com/home

platform can make this effect usable in live situations as well.

A few testers have mentioned the possibility of microphonic feedbacks, to which this effect can be sensitive indeed. An anti-feedback correction can be a solution. As a drawback, it may restrict the performer from achieving desirable instrument feedback sounds. The correct acoustical arrangement and the general feedback-resistance of the particular microphone can also reduce the number of unwanted feedbacks.

6.2 Prospects for the future

The primary plan is the integration of the effect into an embedded multi-effect pedalboard. Either by designing a dedicated hardware, or – more ideally – using an existing platform addressing such concerns. A potential candidate is the mentioned *MOD Duo* project that gathers existing open source software plugins into a hardware device targeted to live performances. Secondly, the implementation of several more plugin back-ends to support many other DAW applications. This is rather a time consuming work and not a real engineering problem.

The user reactions have shown that this effect can raise interest in musicians and provide an inspiration for artistic creation. This can be considered as a big success from a practical perspective.

Bibliography

[1] S. Cai, P. Gandhi, and C. Guida, "The music really speaks to me", Carnegie Mellon University College – Department of Electrical and Computer Engineering, Nov. 2009. [Online]. Available: https://www.ece.cmu.edu/~ee551/projects/F09/Group6FinalReport.pdf.

[2] R. Mannel. (Jan. 4, 2010). A brief historical introduction to speech synthesis, A macquarie perspective, Macquaire University – Department of Linguistics, [Online]. Available: http://clas.mq.edu.au/speech/synthesis/history_synthesis/ (visited on 05/05/2015).

[3] "Now a machine that talks with the voice of man", *Science News Letter*, p. 19, Jan. 14, 1939. [Online]. Available: http://scienceservice.si.edu/newsletters/39019p.htm (visited on 05/04/2015).

[4] P. R. Cook, "Singing voice synthesis: history, current work, and future directions", *Computer Music Journal*, vol. 20, no. 3, pp. 38–48, Autumn 1996. [Online]. Available: http://www.jstor.org/stable/3680822 (visited on 01/14/2015).

[5] S. Lemmetty, "Review of speech synthesis technology", Master's Thesis, Helsinki University of Technology – Department of Electrical and Communications Engineering, Mar. 30, 1999. [Online]. Available: http://research.spa.aalto.fi/publications/theses/lemmetty_mst/.

[6] U. Zölzer, *DAFX: Digital Audio Effects*, 2nd Edition. Wiley, 2011, ISBN: 9780470979679.

[7] W. Pirkle, *Designing Audio Effect Plug-Ins in C++: With Digital Audio Signal Processing Theory*. Taylor & Francis, 2012, ISBN: 9780240825151. [Online]. Available: http://books.google.cz/books?id=v0ulUYdhgXYC.

[8] V. Goudard and R. Müller. (Jun. 2, 2003). Real-time audio plugin architectures, IRCAM, [Online]. Available: http://mdsp2.free.fr/ircam/pluginarch.pdf (visited on 11/27/2014).

[9] T. Islam, "Interpolation of linear prediction coefficients for speech coding", Master's Thesis, Department of Electrical Engineering McGill University Montreal, Canada, Sep. 2000. [Online]. Available: http://digitool.library.mcgill.ca/R/?func=dbin-jump-full&object_id=30253&local_base=GEN01-MCG02.

[10] D. O'Shaughnessy, *Speech communications: human and machine*. Institute of Electrical and Electronics Engineers, 2000, ISBN: 9780780334496. [Online]. Available: http://books.google.cz/books?id=yHJQAAAAMAAJ.

[11] C. Drentea, *Modern Communications Receiver Design and Technology*, ser. Artech House Intelligence and Information Operations. Artech House, 2010, ISBN: 9781596933101. [Online]. Available: http://books.google.cz/books?id=9juUwbKP-58C.

[12] G. McNally, "Digital audio: recursive digital filtering for high quality audio signals", BBC Research Department, Report, Oct. 1981. [Online]. Available: http://downloads.bbc.co.uk/rd/pubs/reports/1981-10.pdf (visited on 05/10/2015).

[13] M. Hasegawa-Johnson and A. Alwan, "Speech coding: fundamentals and applications", University of Illinois at Urbana–Champain, Urbana, Illinois, report. [Online]. Available: http://www.seas.ucla.edu/spapl/paper/mark_eot156.pdf (visited on 05/10/2015).

[14] J. Schimmel, "Studiová a hudební elektronika", Brno University of Technology – FEEC, Purkynova 118, 612 00 Brno, Tech. Rep., 2012.

[15] J. E. Markel and A. H. Gray, *Linear Prediction of Speech*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982, ISBN: 0387075631.

Appendix A

Full User Reviews

"I play bass guitar, yet I am very opened to use any – really, any – effect that I come across. Honestly, trying to use this kind of effect was very new to me. Playing a tone plus singing any tone I want – the experience was really shocking but it opened my mind. I was thinking about versions "OK, just admit it, you are not a singer. But, what about leaving the instrument behind – let it be played by others – and try to say something into the mic, and see what happens." That was a completely new experience. I never tried to "sing" a song but the feeling that no pitch will be false because it's actually corrected by the guitar gave me such a self-confidence that I felt more than powerful. It was really interesting to feel something that is in some way different to the vocoder-sounds that we have been used to at bands like Pendulum, but in some ways quite similar.

An absolute – and, unfortunately untested – unique thing would be to use this effect with a jaw-harp or any wind instrument. The backdraft of a jaw harp is that its soundscape is limited by the instrument itself and the technique of the player. This effect would eliminate at least one – if not both – of them. What about a blues harp? Most of the blues guitarists play and sing simultaneously, it would be quite interesting to use this effect to spice up their performance.

As a backdraft of this effect I would mention – it's complicated. I would like to try it either as a part of a vocal performer multieffect (like the ones that are made by TC Helicon) or as an independent effect pedal. But using it through a computer – it might be useful, but personally I would refuse this kind of option. I can imagine that some day this one receives its own firmware and box-on-floor layout and the computer can be avoided because this one is quite complicated to use on stage.

However, it can be quite useful in studios, or in the "garages", rehearsal rooms: for bands who like to experiment, this gadget can be a way finding their own and unique sound."

Gábor Szabó
Bass guitarist (H.A.L.)
(full unedited review — English)

Tamás Mészáros gave me his own programmed voice effect to test. I run my Tumblr site since august last year. There are my humorous, sarcastic poems and hopefully I will release some minimal synth-pop songs later on. I have only two instruments so far. One is a semi-acoustic guitar "Dowina" and a Korg MicroKorg *synth.*

I tried this effect only with the guitar. It sounds very strange. It was very hard to get used to it in this short time frame but it is definitely something I haven't met so far. In something, what I can imagine as a song, I don't think that such an effected voice would go through the song all the way, but it can be interesting for "bridge" or "interlude" parts of it. Or simply something used in the background, probably by adding some hall/delay effect on top of it. I found this effect very interesting. I don't know how it would sound using it together with a Clouds texture synths and adding some Wiard synths. But I suppose it would be very a special, unworldly, transcendent trip. Maybe not the most pleasant one ever but absolutely one to remember."

Ma.F.F.
maganyosfarkasokfalkaja. tumblr. com
(full unedited review – English)

"Mészáros Tamás told me about his thesis subject and asked me to test it. Myself being a guitarist as well, I have immediately accepted his request. My preferred musical styles are oriented towards heavier rock genres where the usage of this effect appears to be rare. For me, it was just one more reason for excitement.

Before testing the effect, I have made an overview amongst the bands I'm familiar with, searching for appearances of similar effects and found some interesting songs from Bon Jovi, Black Label Society, Joe Satriani *or from* Daft Punk *as well, although from a different genre. I have never used a talkbox before and felt myself as a newbie. It was hard to get used to talking while playing at the same time, but after some practice, I got synchronized with the effect.*

In my opinion, the software is working very well and I think that Tamás has made the most of it. The test was accomplished with a middle-class microphone and a cheaper kind of electric guitar. Even so, I haven't experienced any significant feedback with a high signal output, which is impressive even compared to commercial effects. Despite having just a few controlling parameters, a lot of different textures could be set with the presence and gain "knob" which I find to be a big advantage. After getting familiar with the effect, I tried to play different parts of songs from the above mentioned artists. The best match I could get is the style of Daft Punk. *Additionally, a good approximation can be made to the talkbox guitar intro heard in the song Livin' on a Prayer from Bon Jovi. As a drawback, I could mention the need of external effects for a really usable sound, like the compressor for the microphone input or an overall equalizer – perhaps a simple tone and bass parameter would be enough.*

After the test, my overall impression was very positive."

Dávid Zoltán
***guitarist from band* Ankh**
(full review — translation from hungarian by Mészáros Tamás*)*

"Although the talkbox was a popular modulation effect in music production during the late 20th century, various reissues are still frequently used today. Human voice is a versatile tool for sound creation and by fusing it with instrumental playing, we can get a very exciting and colorful musical accessory.

As an active musician and music devotee, I find it very important for new tones to emerge in the contemporary musical genres. Mészáros Tamás has developed a modern music creation tool, that is able to make some melodies richer, more interesting and more "alive". The created tool connects the musician's own vocal organs with the instrument in his hands, opening up new artistic possibilities. I would also list the extendability of digital processing amongst the benefits of this effect. The result is a usable modulation effect that may become popular amongst today's modern musicians.

The behavior of the effect in live situations is still uncertain with the apparently high input sensitivity. Microphonic feedback is perhaps the most potential threat. However, it would certainly perform well in a studio environment."

Ing. Pál Tamás
drummer and sound engineer from the band H.A.L
Ph.D. student at BUT FEEC Department of Telecommunications
(full review — translation from Hungarian by Mészáros Tamás*)*

"I have been testing the demo application on my usual gig-setup: Through a set of few basic pedal effects, I normally run the guitar signal directly to the top-boost channel of a vintage VOX AC30 amplifier. For the demo, I have set the input gain of the amplifier to low in order to allow good headroom without too much distortion (the amplifier is known to easily run into saturation due to lack of the internal negative feedback). I have been testing both with the distortion pedals on as well as fully by-passed.

The testing microphone was a dynamic Shure SM57 — perceived as an established standard in the community. The room setup didn't let me move the microphone too far from the AC30 (around 1.5 m), so I had to tweak the gain knobs on all devices in the chain to avoid the feed-back. The room was a 2-by-4 m "garage-band" style rehearsal room with carpets on the walls and ceiling to eliminate.

Immediately after connecting the application, I was surprised by the quality of the sound, especially on the louder side of the master volume knob. The sound was very pronounced and clean with no obtrusive artifacts. There was no lack of bass nor I had to adjust the treble setting. It took me a while to synchronize playing guitar and singing at the same time to achieve the desired talk-box-like effect. The application offers limitless number of creative sounds and ways to play and sing. The application can be used both for solo-style of guitar playing, as well as for power-chords — again resulting in different sounds. Not only the player is not limited by the style of guitar playing, but also the vocal part of the effect is very versatile and yields interesting effects with different style of vocal input. E.g. the most obvious was to use long voiced singing modulated by the vocal tract to achieve "wah" style of effect. However, I have also tried rhythmic modulation using e.g. fricatives and various styles of "funny cha-cha" sounds. Personally, I liked the rhythmic sounds with simple guitar solos.

From the user-point of view, the biggest advantage is the modularity of the application, i.e. it's built as a plugin and apparently, the effect is available also as a VST plugin. Apart

from my stomp-box effects, I used a software compressor on the guitar to smooth out the playing. This makes the usage and development of the effect very versatile and creative.

A suggestion for further development would be to add a dynamic envelope shaping of the parameters of the effect, i.e. attack, sustain, release to smooth out the triggering of the microphone input by the guitar.

In general, this effect is very creative and offers interesting sounds. Once I found my style of playing, it was difficult to stop playing. This effect has exactly what it is supposed to have: very high "fun" and "inspiration" factors."

Ing. Ondrej Glembek, PhD.
lead guitarist of the band RockMood
(full unedited review – English)

Appendix B

Recording Environment

Operating system Ubuntu 14.04.2 x86_64

Kernel version 3.16.0-38-lowlatency

Host CPU Intel® Core™ i3 CPU M 370 @ 2.40GHz

Host memory 4 GB

Reference DAW Ardour3; version *Ardour3.5.403-dfsg-3-ubuntu14*

USB audio interface Mackie *Onyx Blackjack*

Figure B.1: Onyx Blackjack recording interface.

Microphone Behringer *XM 1800 S*

Instrument used on recordings Epiphone LP Special

Appendix C

DVD Content

The following files and folders can be found on the DVD:

`/AudioSamples` Contains the demo audio tracks with the plug-in sounds.

`/Distribution` Compiled binaries of the plug-in.

`/ProjectTree` The project tree managed by CMake. It contains the LaTeX source code of the thesis as well.

`/RecordingSessions` Contains the recording session project files of the demo tracks for opening in *Ardour3*. The Bon Jovi track can be opened using the session file `livinonprayer.ardour`

`runardour.sh` A bash script for running Ardour3 with an environment set up to recognize the plugins correctly. This can be used with a LiveUSB setup of a Linux distribution intended for multimedia editing purposes (*AVLinux*, *Ubuntu Studio*, etc.).