Redefining the Audio Editor

Toine Heuvelmans

Master of Philosophy
Bournemouth University, United Kingdom
University of the Arts Utrecht, the Netherlands

July 2014

THIS PAGE INTENTIONALLY LEFT BLANK

Abstract

This thesis describes new design principles for audio editing software. This kind of software, also called *audio editor*, is the digital cutting table for sound and music production in which audio can be loaded or recorded, then selected and edited.

First an understanding of the audio editor is established. Then a new approach to audio editing software design is developed, based on research into current software. This new approach consists of a set of design principles that aim at improving coherency, flexibility and creativity in the audio editing process. These principles are formed by carefully rethinking core elements in audio editing such as audio representation, selection and manipulation, editing flexibility, automation and personalisation.

As artefact of this research, a concept audio editor called OFFline is presented in a second section. This audio editor demonstrates a possible implementation of the new design principles.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Illustrations and Tables

Illustrations

Tables

1. Introduction

I developed my previous techniques in a classical tape studio, using only editing, mixing and speed changing.

(Trevor Wishart, 1988)

1.1 Motivation

This research is part of a personal pursuit for a specific tool. It began years ago when I was introduced to the work of composer Trevor Wishart and one of his compositions in particular, *Tongues of Fire* (1994b). For this composition, Wishart used as his only source material an audio sample of just 2 seconds long to create an immensely varied composition of almost 25 minutes. Tongues of Fire demonstrated to me the virtually endless possibilities in "sonic composition"[1], and I wanted to be able to use these techniques myself.

At that time, I had a relatively limited knowledge of available software, and my fascination was followed by disappointment, since I couldn't find any software that included these transformations in any familiar form (an *effect*, for instance in the form of a VST plug-in[2]) that I could comprehend. All of my teacher's demonstrations were programmed by himself (mainly in SuperCollider), and I had not yet heard of the Composers Desktop Project[3].

As a developer, my typical response to this disappointment was to invent something myself, that would provide this simple effect-approach to Wishart's techniques, allowing one for instance to simply select octave stacking[4] from a list of audio transformations, specify its parameters and apply it to a piece of audio. I decided the best way to approach this, was to build an audio

[1] Composing a sound. Sonic composition will be discussed in more detail in chapter 3.

[2] *Virtual Studio Technology* (VST) is a software technology that allows audio software to be extended with synthesiser and effect plug-ins. Plug-in architectures are described in more detail in chapter 3.7 (*Extendability*).

[3] The CDP is a company and cooperative network (which includes Trevor Wishart, whose work has been of great influence on the design principles discussed in this thesis) based in the UK that has been developing software for working with sound materials since 1986. The CDP software includes many techniques which, as with those used in *Tongues of Fire*, are described in Trevor Wishart's book *Audible Design* (Wishart 1994a, p.46).

[4] Octave stacking is a simple yet impressive technique described by Trevor Wishart in *Audible Design* (Wishart 1994a). It takes a piece of audio, stretches it up and/or down a number of times, each time by a factor of 2 (the ratio for an octave higher or lower). Stretching up, this causes the audio to be played twice as fast and sound exactly one octave higher. All these stretched versions are combined (*stacked*) with the original audio. If the audio has a distinct onset, all these onsets will occur somewhat simultaneously, creating a massively rich onset. Each layer of this stack will end at a different time, first the highest and shortest version, ending with the lowest and longest version.

editor, that already provided such an approach for more mainstream audio processes like normalisation, dynamic compression and reverberation, as well as including features such as cutting and pasting. The audio editor was hence suitable to be extended to include these audible design techniques.

My choice for the software type *audio editor* was based on what I thought it to be: the digital equivalent of an analogue tape studio, in which you can cut and paste audio, as well as transform it. Because I wanted my software to be recognisable in use, I felt firmly about labelling it correctly, since each label brings certain expectancies of what a software should and should not do. However, the term audio editor as well as the software it should describe appeared to have a very ambiguous definition. I required a definition because I wanted to see how far I could stretch it, extending the audio editor for sonic composition purposes, without diluting its identity. If this extending the audio editor was to be the subject of my research, it first needed to be clear exactly what that audio editor was.

Therefore, this thesis begins with a search for defining the audio editor. Through a number of surveys the area covered by it was determined, which also gave me an insight into what audio editing software is currently available.

It appeared that not much has been happening in audio editor design. Throughout the years such software has followed a small set of principles that dates back to the mid 80's, only deviating from it slightly as new features were introduced. Such new features were commonly added rather than integrated in these basic principles, accumulating into a somewhat incoherent Swiss Army knife.

I was convinced there was room for improvements in audio software design. Over the years, other kinds of software did undergo rethinking and refinements. From my experience with other kinds of software – particularly image software such as Adobe Photoshop – I saw design principles that could be adapted to audio editing software. This made my focus on extending the audio editor shift to defining a more fundamental redesign of the audio editor.

1.2 Methodology

The audio editor is a subject on which there is very little research literature[5]. The aim to redefine this type of software demanded an understanding of what an audio editor is, who uses it and for what purposes it is used. Three surveys were conducted[6] in order to delineate this area, a full treatise of the surveys

[5] Besides a handful of academic publications and books on music technology, audio editors are mainly discussed in popular literature and on the internet, mostly in the form of software reviews and incidental surveys or comparisons. A more detailed list of such resources is mentioned in chapter 2.

[6] The surveys conducted in this thesis for Bournemouth University have been conducted between 2007 and 2010, while I was registered at Portsmouth University. The MPhil was suspended and no awards were given based on these surveys so far.

is included in chapter 2. The analysis of these surveys is used as the basis on which new design principles are mapped.

Redefining the audio editor involves answering the question "how can it be made better?". Often software is made "better" by adding extra features, but it should be done with careful thought. Audio editing software throughout the years has seen quite some new features being *added* rather than being *integrated*. The tendency to add discrete features instead of rethinking the fundamental principles has led to some inconsistent cluttered audio editing software design (see *figure 1*).

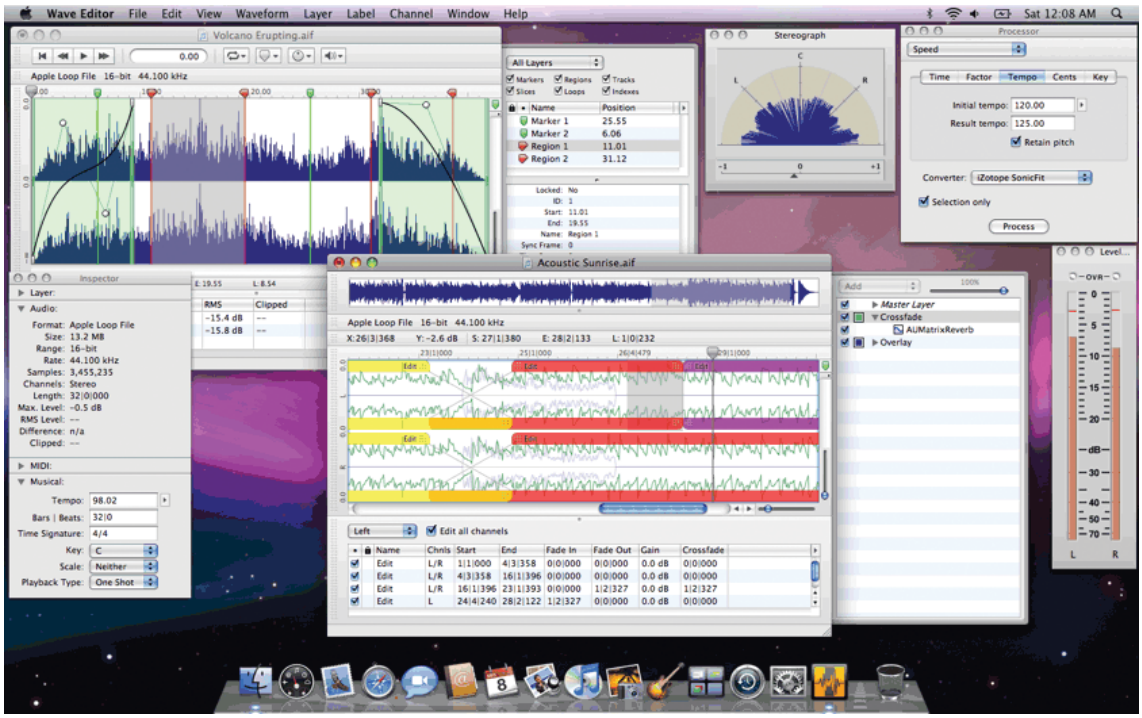


Figure 1. Audiofile Engineering's "Wave Editor"; a collection of individual windows, each with a separate function.

In this thesis an attempt is made to redefine the audio editor by focussing on software design. Seven subjects – aspects of audio editing or common software design – are explored: the process of editing, obtaining audio material, representation and manipulation, audio selection, audio layering, sonic composition, and software extendability. First, fundamental design principles and currently available features will be reflected upon, outlining the status quo and examining the usefulness of the current design. Then, new design principles are proposed, the foundations for which either originate from other audio software design, non-audio software design (of which Adobe's Photoshop is the most exploited) or recent technological possibilities. These new principles are intended to extend and enhance existing design, or replace it with an approach that unifies existing but distinct functionality. Evaluations of common scenarios in audio editing will illustrate the differences between the old and new approach. Some proposed design principles introduce concepts that have not yet been implemented in audio editing software.

1.3 Relevance

The relevance of this research has been given by developments in audio software development during the writing of this thesis (chapter 5 describes recent developments in audio editing software). Some of the described design principles are by now available – to some extent – in a number of audio editors. This proves their value as individual features, but this also emphasises the most important aspect of this research. As described in the methodology, the design principles should not be *added* to the software but be *integrated* in it. In doing so, one principle can build on another, and become more powerful than when it is implemented as an individual feature.

Therefore, read this thesis not as a list of features, but as a single approach to audio editor design.

1.4 OFFline

This thesis includes a section that presents a concept audio editor called OFFline. This audio editor is not (yet) a real and functional audio editor, it is a non-working prototype. OFFline is the author's vision of the redefined audio editor. For the reader, OFFline serves as a demonstration of how each design principle discussed in this thesis can be expressed. Throughout this thesis, mock-up screenshots illustrate a possible implementation of these design principles.

Personally, OFFline served as a structure in which all the design principles that are described in this thesis could be developed and synergistically incorporated. As described in chapter 1.1, the motivation for this thesis was the longing for an audio editor that included an effect-like approach to more artistic techniques, such as those described by Trevor Wishart (1994a). OFFline would be an audio editor, but also a sonic composition tool, yet still be easy to use. These are the basic principles on which the design was developed.

2. Literature & Repertoire

2.1 Narrative – Part 1

The design principles discussed in this thesis are a product of a long period of research into audio editors. As discussed in the motivation (chapter 1.1), it started with the desire to develop an audio editor that incorporated new features, mainly modular non-destructive editing as well as unusual audio effects. The modular editing approach would fundamentally change the editing flow that is common in many existing audio editors, which might give rise to the question if this new audio editor could still be labelled an audio editor. Labelling it as such would help to characterise the software and thus aid in discussing the audio editor's features.

That question was however immediately followed by the question of what an audio editor actually is. Available definitions are scarce and quite ambiguous. Without a more substantiated definition of the audio editor it was difficult to discuss a new audio editor's features.

It was therefore important to discover:

- What users expect from an audio editor;
- How such audio editors are used, and for what purposes;
- What kind of audio editors and features currently exist

A definition was formed in a three-step process:

1. **An initial iteration was constructed through a literature review.**

   This delineation is formed by a terminology and scope (see chapter 2.2.1). The terminology reviews the different labels used in literature to identify audio editing software, concluding on the term "audio editor" to be the most general and appropriate descriptor to identify software which main purpose is to edit (cut and paste, adjust, transform) audio.

   The definition of a scope (the sphere of action of an audio editor) was meant to set fundamental ground rules for identifying audio software as *audio editors*. From both a historical and logical point of view, key aspects of an audio editor were formulated to direct further research.

2. **Then, by asking users about what they considered to be the important characteristics of an audio editor in their work.**

   A questionnaire (see appendices 1 and 2) was constructed and distributed to audio practitioners in order to obtain a generally conceived concept of the audio editor, identifying what it is used for. It was divided as follows:

- general questioning about the participant's background.
- the participants' audio software usage, mainly to identify what audio software was used for what purpose.
- how an audio editor was used and which aspects of audio editing software the participant preferred.

This questionnaire was intended to answer the first two questions:
- what do users expect from an audio editor? and
- how and for what purposes do they use an audio editor?

Answering these questions would help identify and name specific audio editor usages in the exploration towards developing a new audio editor. But as not every user is the same, it would be useful to see if specific usages could be associated with a specific user group. The results gathered from the many participants were very varied, everyone had their own specific background, methods and preferences. Grouping these participants could be done based on their background information and particularly the descriptor they provided with that background (such as *composer* or *sound designer*). This identified a comprehensible set of user groups, with a characterisation of audio editor usage and preferences for each of these groups.

3. **Finally, by reviewing existing audio editing software.**

The questionnaire was also used as a means to gather names of existing software that participants used as audio editors. The initial delineation formed in the first step, together with the generally conceived concept formed in the second step, now functioned as a filter to identify which software was considered to be an audio editor and which software was something else. (While reviewing the terminology, other kinds of audio software and their descriptors were found – such as *Sequencer* and *Digital Audio Workstation* – which helped in this process.) Of the remaining audio editors, those that could be reviewed first-hand or through documentation or literature were listed. The reviewing of these audio editors (see appendix 3) was aimed at identifying available features, as well as identifying different kinds of audio editors (such as wave editors and spectrum editors).

For each audio editor, all included features were listed (appendix 4). Then, using a number of algorithms for comparison, averaging and deviation, the more common of those features were deduced, and for each editor a focus on either common or uncommon features, or an all-round feature set could be identified. A Self Organising Map was then used to identify which audio editors were similar, and by using the earlier reviewed terminology these groups of audio editors could be labelled. Some audio editors did not fit in such a cluster because one or more prominent features that characterised it differentiated the editor from the group. An example of this is DSP Quattro, which has basic editing features

but focuses heavily on audio CD mastering.

The features listed in the comparison were all expressed quantitatively, only identifying whether an audio editor included them or not. To a fair extent these features matched the options that were chosen by users as "preferred features of audio editors" in the questionnaire. No insight however was yet gained about qualitative aspects, such as speed and simplicity. For this a follow-up questionnaire was conducted (appendix 5). Participants had to identify themselves with one of the user groups that were deduced from the first questionnaire, and score various qualitative aspects for their relevance. The outcomes confirmed a number of assumptions, particularly about a general preference for speed and quality, and some group-specific preferences such as connectivity and compatibility often preferred by performers, and an interest for extendibility generally preferred by software developers.

The audio editor comparison mapped the breadth and width of audio editors to date. It revealed a number of different types of audio editors, ranging from basic to comprehensive, and from the most common waveform editors to less common spectrum editors. On a deeper level common design principles could be discriminated, such as how audio is displayed visually and how it can be selected for editing. In most audio editors, many of these design principles were implemented in practically the same way as in the very first audio editor (Digidesign's Sound Designer), which dates from the mid 1980's. This is not a bad practice per se; it might prove that these design principles initially were well conceived. However, many features that were included in more recent audio editors were often only made accessible through menu structures, a design approach that from an intuitive standpoint is quite cumbersome and lacks proper visual feedback on the process of editing. It moves the user away from the central user interface, which in every audio editor contains at least one visual audio representation serving as main graphical user interaction element.

The modular editing approach mentioned at the beginning of this narrative was directly inspired from image editing software such as Apple's Core Image Fun House and Adobe PhotoShop, and from that same area other design principles could be adapted better to implement a number of audio editor features currently only available through menu structures (particularly the application of certain audio effects).

A more fundamental issue, however, was also in need of improvement. Audio editing requires audio to be obtained from an external source before editing can begin. This can either be through loading a file, making a recording or synthesising an audio signal. In all audio editors these three actions were provided to the user in very disconnected ways, through separate elements of the user interface, but they all provide the user with audio that can be edited. If the first user action in editing audio is presented through such fragmented design, there is room for improvement. It was this issue that tipped the focus of developing a new audio editor, to a more fundamental deliberation on audio editing design principles. Consequently, this lead to the development of new

the design principles discussed in this thesis.

There were seven areas to be considered. First, the modular editing approach would pose a solution for various challenges in the process of editing. With the initial conceptualisation of this approach it was not yet foreseen for how many different aspects of audio editing this would turn out useful; this understanding was only achieved after conducting the comparison, and through the process of developing other design principles.

The deliberation on obtaining audio material followed, posing a unified user interface element from which file loading, recording and synthesis could be conducted. With a single user interface element for audio source selection, there was now a logical setting for addressing related matters such as batch file selection and finding audio files.

Personal experience with image editing software profoundly influenced the proposed design principles, a first example of which is the modular editing approach. Correspondingly, two other fundamental aspects of audio editing could also be improved, mainly through adapting principles from image editing software, where similar concepts have become more sophisticated than their audio editing counterparts:

- A visual audio representation as a user interface element for audio manipulation.
  In most audio editors these interface elements only allow cutting and pasting selected audio regions, but by also allowing manipulation of such selected regions, a great variety of audio transformations can become available in an intuitive way;
- The way(s) in which a specific region of audio can be selected.
  Most audio editors only allow selecting a specific time range of audio. The selection of time-frequency ranges (optionally changing in frequency range over time) can enable the user to make more precise edits.

As described in the section on motivation, there was a personal desire for an audio editor that included unusual effects and allowed for more creative forms of audio editing. Consequently, three more subjects would be explored:

- audio layering: how multiple layers of audio can be combined in different ways (such as various modulation techniques and morphing);
- sonic composition: how creative and complex edits can be made through different kinds of parameter automation; and
- extendibility: how the audio editor can be personalised by enabling the user to write their own audio effects.

2.2 Defining the Audio Editor

Although there is a wide range of audio editors available today, there isn't much written about this kind of software, other than reviews. This might be because developers of commercial software don't have the tendency to share with the public the development process; they rather advertise the end result. But perhaps it is also because there is no such thing as *the audio editor*. An audio editor is merely a collection of functions, some of which are common amongst most audio editors, and are therefore regarded as essential. *Audio editor* is therefore more a generally conceived concept (Pope 1993, p.25), rather than a definition. In this chapter that concept will be traced to formulate a definition of an audio editor. A framework will be provided by explaining a terminology and determining a scope. Then the generally conceived concept of an audio editor is distilled through three studies.

2.2.1 Delineation

2.2.1.1 Terminology

Consulting resources[7] that do write about the kind of software that can be described as an audio editor, will reveal that there is a lot of ambiguity in the assembled nomenclature concerning this subject. This thesis focuses on the type of software program that is called ***digital audio editor***. The term *audio editor* will be used from this point on, since a software program is by nature digital.

An *editor* program lets one create and modify a text, image or in this case audio (Roads 1996, p.706). One might hear a multitude of different names for this same kind of software, but *audio editor* is favoured for the following reasons.

A *sound editor* is the most common variant in use. Audio is sound, but explicitly when recorded, transmitted or reproduced. In other words, audio is sound captured on tape, disk or any other analogue or digital storage medium. Opting for *audio editor* will identify this state of sound.

Furthermore, *sound editor* is also used to describe a profession or person (in this profession), as one might find for instance when querying Wikipedia. *Audio editor* might be used in the same way, though this happens far less frequently.

A more difficult reason why not to use the term *sound editor* is that audio can be sound as well as music. Determining the boundary between sound and music is an almost impossible undertaking (Wishart 1996, p.3). One might regard sound as an individual audible event, and music (in its more traditional form) as a *composition* of multiple sounds. However, sound and music both occupy the same sonic space and can sometimes appear in such a way that they

[7] (Milano 1985; Kirby and Shute 1988; Herzberg 1989; Moorer 1990; Pope 1993a; Roads 1996; Mazzoni and Dannenberg 2001)

Further reading includes magazines such as Sound On Sound, Future Music and Interface, and websites such as KVR and Wikipedia.

are indistinguishable. In this respect, an audio editor can edit both sound and music, however it can only edit music as a whole, a single audio file; it cannot edit its individual *sounds*[8].

An audio editor is always a *graphical editor*, meaning that it can provide a graphical representation (or visualisation) of the material being edited. This representation is part of the *Graphical User Interface* (GUI), in that it provides not only a view of but also access to the audio, by allowing the user to select portions of audio through clicking and dragging in the graphical representation. Non-graphical audio editing applications go by different names, such as *audio processing command line utility*[9] or *textual programming environment* (designed for audio processing).

Almost as common as *sound editor* is the term *wave editor*, which defines a graphical audio editor that displays the most common time domain visualisation of audio, the waveform (time versus amplitude, see *figure 2a*). An audio editor primarily working in the time domain can also be called a *sound sample editor*, since it directly operates on a stream of samples. Less common is the *spectrum editor*, operating on partials in the frequency domain (*figure 2b*). A spectrum editor presumes a prior analysis stage (Roads 1996, p.768). Its type of visualisation and operations depend on the kind of analysis[10]. An *audio editor* might include both time domain and frequency domain operations, so neither *wave editor* nor *spectrum editor* fully covers the focus of this research.

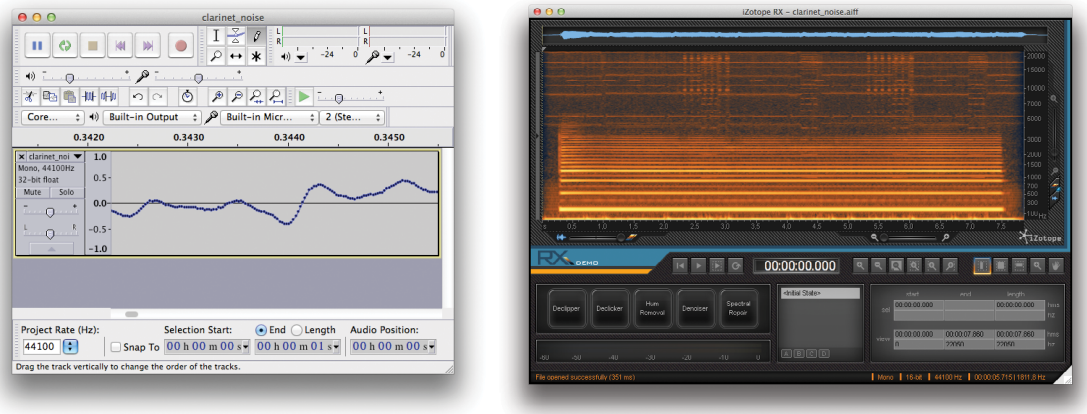


Figure 2a). A wave editor (Audacity) and b) a spectrum editor (iZotope RX)

[8] **Melodyne** implements a technique called *Direct Note Access*, which analyses a piece of music (a single file which has to be clearly tonal) and separates all notes as portions of audio, which can then be edited individually. Handling note information is outside the scope of an audio editor, and therefore Melodyne is not an audio editor. For more information about scope, please read the following section.

[9] The command line can execute a program such as **SoX**, which is provided with arguments such as input file and effect parameter settings. For example, "`sox input.wav output.wav gain -6`" will read `input.wav`, change the gain by -6dB, and save the result to `output.wav`.

[10] The most commonly used technique to transform from time domain to frequency domain is the *Fourier Transform*, which provides amplitude (and phase) information per frequency per time-frame. Another method is the *McAulay-Quatieri* technique (Mcaulay and Quatieri 1986), which attempts to represent a sound with many individual sinusoidal tracks (partials), each corresponding to a single sinusoidal wave with time varying frequency and amplitude (examples of which can be found in **SPEAR** and **O'kinshi**).

2.2.1.2 Scope

Scope defines the sphere of action of an application – the range of things that can be affected by it. An application can only access something it has an "understanding" about.

An audio editor doesn't need an understanding of music

A digital audio editor is essentially a virtual realisation of an analogue tape cutting studio. Sound and music can be recorded on tape, but only as amplitude fluctuations. Tape provides a signal representation of music, whereas a score provides a symbolic representation of music (Vinet 2003, p.194). Intrinsic musical information such as notes, tempi and instrumentation are not available from tape, hence the digital audio editor does not deal with this kind of information either: a musical understanding is not included in its scope. For example, an audio editor could not handle a request such as "Delete the C4 played by the piano in measure 36" because its knowledge of musical signals is limited to time and amplitude points (Roads 1996, p.706).

An audio editor can be used for composition

In the context of this thesis, "composition" is commonly interpreted as composition of music. Music can be regarded as emotive sound, but not all emotive sound can be regarded as music (Deutsch 2007, p.3). Any sound can be edited with an audio editor, as long as it is digitally recorded. With an audio editor a sound can be slightly adjusted, but it can also be shaped, sculptured and moulded into another sound, regardless of serving an emotive or literal purpose. This shaping of sound is sometimes referred to as *sonic composition*[11]. A sonic composer shapes the internal structure – aspects such as timbre, texture, and dynamic development – of a sound. The ways in which sound may be transformed are limited only by the imagination of the composer (Wishart 1994a, p.1). As long as the compositional process doesn't involve symbolic musical information (Vinet 2003, p.194), one can use an audio editor as a sonic composition tool. A wide range of techniques can be developed using only editing, mixing and speed changing (Wishart 1988, p.21).

Sequencers and Digital Audio Workstations

An audio editor is often confused with a *Digital Audio Workstation* (DAW) and vice versa, as became clear through the questionnaire (described in appendix 1). A DAW serves as a complete virtual production studio, facilitating multi-track recording, editing, sequencing, and mastering. The editing environment – the source of the confusion – is but one part of the application, and often not the most important.

Most DAW's present in their main window a multi-track sequencer, along with a mixer displaying virtual channel strips for each track. A track is often

[11] The term "*sonic composition*" can also be used to identify the musical organisation of sounds into compositions.

used to position sections of audio in time, but sequencers also include control-tracks, generally containing MIDI information[12] to control software or external hardware synthesizers. This control signal often contains discrete musical events that can be represented as a musical score (a *symbolic representation*, Vinet 2003, p.194). If a program has such an understanding of music, a track is often regarded as an *instrument*, along with its own mastering parameters and effects processing section. With these, audio on a single track can be shaped to become a distinguishable "object" in a composition.

The sequencer in a DAW is often an audio and MIDI sequencer, so a DAW can have an understanding of music at least to the extent of the MIDI protocol. Multi-track audio editors do incorporate a sequencer, but only for sequencing audio signal portions. A visible difference between these sequencers is that in a DAW the time-line is generally displayed in bars and beats, and playback depends on a tempo and measure parameter, whereas in an audio editor the time-line often is just divided in seconds, minutes and hours.

A survey on audio editing in Digital Audio Workstations can be found in appendix 6.

An audio editor can edit in non-real time

A real-time operation is one that processes input data within milliseconds so that it is available virtually immediately as feedback (Stevenson and Lindberg 2005). A live band instrumentalist, for example, plays an instrument, which might be processed by an effects processor in real time, making the effect immediately audible.

Real-time operations do not necessarily have to be fast, but they do have to be fast enough to keep up with the system (Loy 2007b, p.9). Processing high quality audio in real time requires a faster processor than when processing a low rate control signal.

An offline operation involves processing of a sort that cannot proceed in real time, due to causality. An operation is said to be causal if it depends only on its present and past inputs as well as its past outputs; it does not depend on future inputs. The output of a noncausal (or acausal) operation, by implication, might depend on future inputs as well (Moore 1990, p.114). Hence, acausal operations cannot exist in real time; however they are perfectly realisable in nonreal time (*offline*), such as when operating on a prerecorded signal (Loy 2007b, p.219). For example, level normalisation requires examination of an entire sound, beginning to end, before the normalisation factor can be calculated (Apple 2007).

The aforementioned Digital Audio Workstation commonly serves as a so-called host for effects plug-ins[13]. Such an effect can be designated to a track,

[12] MIDI (*Musical Instrument Digital Interface*) is a standard that allows electronic musical instruments, computers and other related devices to exchange musical information. This information can include notation, pitch and velocity, as well as control signals for parameters such as vibrato and tempo.

[13] A plug-in is a software extension, often developed by a third party, that extends the

processing all audio on that track. Effects plug-ins operate in real time, allowing an instrument connected to the DAW to be processed immediately. The host provides the effect with audio to process, which can also be done faster than real time. The effect only processes the audio it receives, it cannot inspect input audio from a different point in time, other than past inputs it might have stored. This architecture makes an effects plug-in unsuitable for acausal operations.

Audio editors operate offline, allowing for such basic operations as normalisation[14] and reversing, but also more uncommon processes as reordering and time stretching. Time stretching both covers time compression and expansion. Compressing in time requires future inputs to occur earlier in time, which constitutes an acausal process. Expanding time will cause inputs to occur later in time. All inputs will be delayed, which does not necessitate acausal processing.

2.2.2 Three studies

With the right terminology and an initial scope in place, three studies have been conducted to further define the audio editor, which are described in detail in appendix 1. From these studies, the following conclusions could be made.

The *audio editor* is an umbrella term for a number of different audio editing software types, ranging from wave editors to spectrum editors, and from basic to all-round multi-track audio editors. The majority includes a common set of basic features (file loading / recording, waveform display, selecting, cutting / pasting, amplitude adjustment, undo) that was also found in the earliest audio editors, which were little more than a virtual audio tape recorder and cutting table.

Audio editors are used in a wide variety of occupations, and for different purposes (mainly basic editing and recording, but in lesser degree also conversion and analysis). A number of user groups could be discriminated by looking at occupation, though many participants identified themselves with multiple occupations. No clear preference for an audio editor type by any user group could be identified.

Though no one definition of “the audio editor” can be formulated, the area covered by it and users associated with it are identified, forming a usable lattice for further research.

functionality of the host software.

[14] Normalisation is a way to adjust the volume range of an audio recording. The overall volume is increased or decreased so that the loudest peak is set to the digital maximum. Normalisation is often used to avoid clipping, and to ensure multiple recordings are within a common loudness range.

2.3 Design Considerations

The design of the audio editor

The audio editor (as discussed in this thesis) is a type of software that was first introduced in the mid 80's when Digidesign released Sound Designer. It was the first software that allowed one to load an audio file, see a (zoomable) waveform display, cut and paste segments of audio and change the gain[15]. It even provided freehand waveform drawing (Milano 1985; Herzberg 1989). Sound Designer became the archetype for almost all audio editors that followed to date. It presented design principles that offered the bare essentials of audio editing in an intuitive manner. Hence even in recent years new audio editors have been released that hardly diverge from what Sound Designer offered. The widespread adaptation of these design principles demonstrates their value in the field of music technology. However, at the time of this writing we are nearly 30 years further, technology has developed and with it the consumption and utilisation of it have changed.

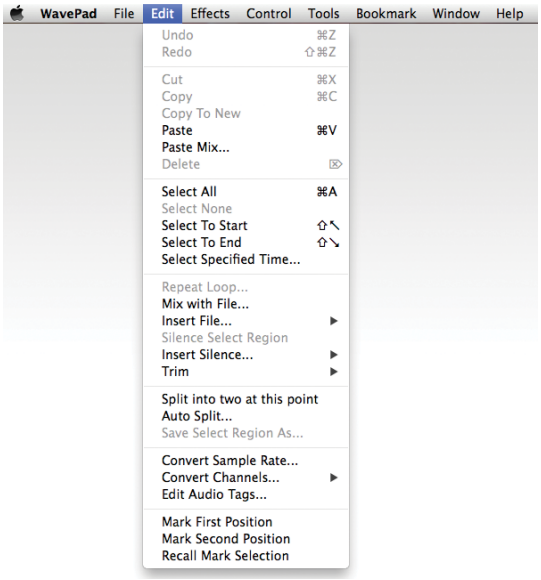


Figure 3. The Edit menu in WavePad, displaying a plethora of editing options.

It thus seems that in audio editor design little has happened. At first glance, the majority of all currently available audio editors look alike. Only when diving into the sometimes elaborate application menus (*figure 3*), real differences can be discriminated. Functionality is often not offered through the main graphical user interface, but appended to the software as a dialog that can be accessed through the menu. All these menus, submenus and sometimes multiple windows create a fragmented user experience, there is little coherency to be found in these lists of individual functions. It is even worse when an audio editor (such as Audacity and Wave Editor) hosts plug-ins such as VST by listing them in a menu, because each of these plug-ins has its own graphical user interface and style. Sometimes a functionality can only be provided through a specific user interface, and therefore presenting this in a separate dialog may be the only option. However there are many cases in which functionality is hidden in menus which could have been made accessible through the available graphical user interface, if only the design had been given a little more thought. In doing so, multiple functions can sometimes become available through a single user interface object, increasing coherency. Why up to now this happens so little, might be ascribed to a reservation towards altering

[15] The level of amplification of an audio signal.

the archetypical design principles too much; it might confuse the user. But this very same user might have seen and become familiar to the developments that have taken place in other areas of software, developments that can sometimes be adapted to audio editors.

In finding new design principles it can be beneficial to look at other *editing* software. Audio, text, image and video are all media that can be edited with a specific kind of software. Between these media there are a number of parallels – particularly dimensions such as time, position and frequency – along which one might find adaptable paradigms (Heuvelmans 2005). Think of the way(s) in which the medium is represented, selections are made along specific axes, or how a medium is composed from components such as layers. Though not directly associated with audio, some already generally accepted and understood editing design principles in other media can be translated to audio.

Modern design

In this digital era one of the most important driving forces in software innovation is the ever increasing available processing power. More can be done in less time, and on more affordable devices. Processes that were initially only feasible on a supercomputer, are now accessible through mobile phones. Real-time audio processing, high quality image processing, live video effects; with these processes becoming available to a greater user group, a fitting user interface for them is required; not only a good graphical user interface, but also one that applies to the hardware on which the software runs, an aspect that has become more relevant with the rise of smartphones. These devices are small, mobile and have a multi-touch interface. The accompanying software is typically simple and intuitive, yet very powerful.

Mobile software, called *apps*, is not window-based, as opposed to desktop software. The user is confronted with just one app at a time, that occupies the entire screen. An app has no application menu as one can encounter in most desktop software. Interaction is done through touch, and multiple touches can be interpreted, as well as gestures. This allows for very intuitive design, in which natural mappings[16] can be exploited. Where the combination of mouse and keyboard can provide an accurate user interface, the touch interface is often empowered by smart content-aware software that helps the user achieve the same precision. Through automation and interpretation, an app helps and guides the user in expressing their decisions. This can be as evident as guiding

[16] A natural mapping implies that the relationship between two things - in this case the user action and the result of that action - is based on commonly understood physical analogies or cultural standards (Norman 1990, p.23).

them to make exactly the right selection[17], but it can also produce "magic"[18], functionality such as "auto-enhance", an opaque function in Apple's iPhoto that improves your photo. Modern software can and should help the user where possible, but for each software the right balance between assistance and magic must be found.

Photoshop

The conceptualisation of the design principles described in this thesis was considerably directed by concepts from the domain of image editing software. Most notable is Adobe Photoshop, arguably the de-facto standard in image editing software, which over the years has become incredibly powerful by taking advantage of the increasing available processing power. Not only can edits be done faster, but also smarter. Photoshop allows for "quick" selections, guiding the user in defining a precise selection by finding adjacent regions similar to the selected one. It includes content-aware editing and "puppet warp", enabling the user to scale, remove or move objects in a photo, automatically filling in the areas that are left blank. With very little input the user can achieve very complex edits.

Most of these edits can be done through a single user interface element: by directly interacting with the canvas, the plane on which the image is rendered. This way there is good feedback on the user's actions. Direct feedback in audio editing can only be done when applying real-time edits[19]. To get at least some direct feedback on an edit, a good visual representation of the audio is needed. The user should also be able to interact with the representation in a logical manner; good mapping between a user action and the visual result is important (Norman 1990, p.23).

Human Computer Interface considerations

In designing software there is a trade-off between ease of learning and long-term power and flexibility (Gentner and Nielsen 1996). Audio editing is an inherently complex task, as it requires at least some knowledge about sound and how it is stored digitally; it takes time to master. Audio editing takes place in sessions that can be very short, but just as well can last for hours. Therefore the best interfaces for audio editing tend to fall towards the long-term end of the trade-off (Holland 2013, p.7). The first element with which the user is confronted is the graphical representation of sound, commonly in the form of a waveform. In order to read this waveform and work with it, the user has to know that audio is stored as time-varying amplitudes, and understand that the

[17] A selection for instance of audio, text, image or video. Making selections will be discussed in great detail further on in this thesis.

[18] A term that Apple used in describing software that, using complex automated processes without making this visible to the user, helps the user achieve wonderful things with only minimal effort.

[19] A real-time effect (see chapter 2.1.1.2 Scope) can be applied and adjusted instantly, making the effect audible on the currently playing audio.

peaks that are visualised in the waveform represent the loud parts in the audio recording. Most audio edits can be understood (up to some degree) through a trial-and-error process, but effective audio editing often does require some knowledge of digital audio processing[20]. Audio editing is therefore an implementation-centric process (Cooper 2007, p.269); the user must understand what an edit does (and what its side-effects might be, such as the introduction of distortion or phase shift artefacts) in order to apply it effectively. This doesn't mean however that the graphical interface design should also be implementation-centric and expose every parameter that concerns the edit. The design should help accomplish an audio edit. Provided that that goal can be reached, the design can be very different than a list of parameter controls. The user might have to learn how to use this interface design in order to accomplish his/her goals, but with a good design that will be easier than having to learn exactly how the edit works and how its parameters interact before being able to apply it. A good design makes the user more effective (Cooper 2007, p.149).

Dials and sliders are commonly used interface elements in audio software. As these elements resemble elements on audio hardware, users of audio software can easily understand them. A dial or slider also provides visual and easy-to-comprehend feedback on the parameter setting it represents (on both its value and range), making it an effective interface element for a single effect parameter.

More elaborate visual interface components are less feasible to technically realise in hardware, but this is exactly where audio software can differentiate itself. *Figure 4* shows a graphical equalizer[21] plug-in with two distinct interface sections. The lower section contains a series of dials to adjust centre frequency, gain and bandwidth for each of the 8 bands that constitute this multi-band equalizer. This is reminiscent to how a hardware equalizer would look. The upper section is not merely a visual graph of the equalizer curve; it is an interface element itself, allowing the user simultaneously to adjust a band's centre frequency and gain by grabbing a band's anchor (one of the coloured dots)[22]. The user is given visual feedback of his adjustments both through the visual manipulation of the curve and the real-time line spectrum of the audio that will be shaped accordingly by the equalisation.

Direct manipulation (Shneiderman 1983) of audio requires a good visual representation of the audio, as well as visible and gestural mechanisms for acting upon the audio representation. The visual representation gives direct

[20] Consumer-level music playback software such as Apple's iTunes often includes a multi-band equaliser. Many users see this as an opportunity to further increase the volume, and move every band to its maximum amplification setting. The software then punishes the user when it distorts the audio, but this isn't always followed by an understanding of how equalisation should be used.

[21] Equalisation is the process of selective filtering of audio in order to boost or cut levels within a specific frequency range.

[22] Graphical equalisers on multi-touch devices commonly also allow the adjustment of bandwidth through a pinch gesture.

Figure 4. TranQuilizr is an equaliser plug-in by A.O.M., that (like many other equalizer plug-ins) offers both a graphical equalisation curve that can be edited by dragging anchor points, and a series of (very skeuomorphic) dials for more precise control over the curve components..

visual feedback of the user's actions. The gestural mechanisms should be visible and communicate the possibility of a manipulation action[23]. For such mechanism to be intuitive in use, its mapping – the relationship between the control and its movement and the result – should be logical. For example, stretching a waveform up along the amplitude axis making it taller should not result in the audio being attenuated. It is a universal standard that a rising level represents more, and a diminishing level less (Norman 1990, p.23), hence amplification would be the logical effect.

Getting direct manipulation right is difficult in audio software design, because the user isn't directly manipulating the audio, but the visual representation of audio. The model, mapping and effect of manipulation gestures should be logical not only to the visual representation, but should also be translatable to the represented audio.

Menus

An important criticism that has driven the development of the design principles discussed in this thesis is that new features in audio editing software are frequently added rather than integrated. In worst cases these features are only accessible

[23] The gestural mechanism should communicate both *pliancy* (hinting that an object may be manipulated) and *affordance* (exhibiting how it can be used) (Cooper 2007, p.386; Norman 1990, p.9).

through the application menu – a user interface element to which an extra item can be added. Many audio edits (particularly audio effects) that are made accessible through the menu, are presented to the user via a dialog box or separate window. In his book *About Face 3*, Cooper (2007) describes this as poor design, as the subject to which the edit applies is located in the main window. Putting a function in a dialog box emphasises its separateness from the main task – in this case editing the audio that is presented in the main window. Contrary to user-interface paradigms of 20 years ago, menus and dialog boxes shouldn't be the main method by which users should apply edits. Primary interactions with the audio should be made in the primary window.

Cooper goes on to say that this doesn't mean that the menu should be empty. To provide a good pedagogic vector, menus must be complete and offer a full selection of the actions and facilities available in the application. A scan of the menu should make clear the scope of the application and the depth and breadth of its various features. Hence, the menu is the primary control for beginners, allowing them to learn the app. The menu is less direct (and actually one of the more difficult controls to physically operate), but it does provide more verbal descriptions. In short, the menu should contain all features, but it shouldn't be the primary location for these features.

3. Redefining the Audio Editor

Audio editors come in various configurations. However, the fundamental design principles are very similar, and have barely changed since the very first audio editor. Sticking to these principles impedes smooth integration of new features.

In this chapter, a set of design principles are presented, that in many cases take a different approach to some key features in audio editing. In doing so, they fundamentally change the design of an audio editor, but also incorporate existing features and present them in a more consistent way. The design principles are conceptualised with existing audio editor functionality in mind, but open up possibilities for sonic composition as well.

3.1 Modular Editing

What is an edit?

Audio editors enable the user to apply edits to audio material. This can be as elementary as cutting and pasting portions of audio, just as one would do with real tape and scissors. Edits can also involve adjustments, often in overall or frequency-specific amplitude, as one would accomplish for instance by applying gain change, normalisation or equalisation. Audibly more obvious are transformation edits, which alter the character of the original material, for instance by pitch shifting or time stretching.

The common editing workflow

The most common workflow in audio editors follows a simple sequence of decisions. If the user wants to cut or paste[24], this can be done respectively by dragging or clicking in a graphical representation of the audio. Other edits, such as gain change, normalisation or reverb, can be selected from a list (often in a menu structure), prompting a dialog in which editing parameters can be set. Either way, the software then applies the selected edit by altering the audio data accordingly. For every subsequent edit this process is repeated, and after each edit the user can play back or save the result.

The undo mechanism

But what if the user is unhappy with the result? A so-called undo mechanism allows the user to undo sequentially all actions done so far. When undoing an edit, the mechanism retrieves a temporarily stored backup of the audio data as it was before the edit was applied. As convenient as this might sound, there are however several downsides to the undo mechanism as it is implemented in the

[24] Cutting and pasting will be discussed in more detail further on.

common editing workflow. Firstly, when undoing, the entire edit is discarded, and reapplying it slightly adjusted means every parameter of the edit must be set again. Moreover, this mechanism is sequential; if one wishes to undo an edit which was followed by other edits or actions, all of these have to be undone first and cannot be reapplied without having to reconfigure each of them.

The edit chain

A different approach can circumvent this problem. A chain of edits can be constructed, which sequentially holds all edits that must be applied. Every edit, whether it is a simple cut or a more elaborate transformation, is represented as a module in this chain (see *figure 5*). The various parameters for an edit are listed in its corresponding module, presented through appropriate controls[25]. The module keeps track of the current parameter settings, which the user can change at any moment.

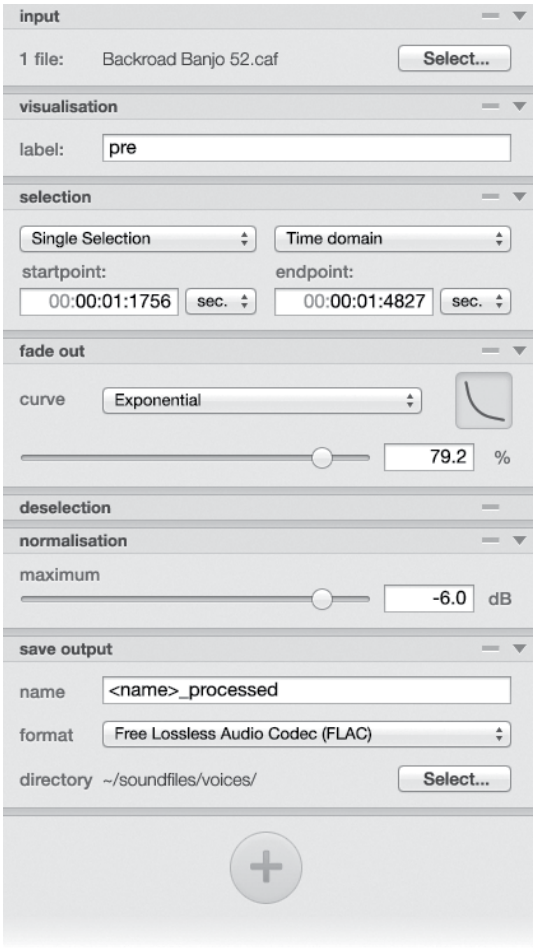


Figure 5. The editing chain, in which each edit is represented as a module.

This approach is actually very similar to the existing plug-in architecture found mainly in Digital Audio Workstations. However, that architecture is designed to operate in real time[26], which limits what can be represented as a module or plug-in. Cutting at a specific point in the audio, selecting a specific segment of audio, normalisation; these edits require access to any part of the audio being processed. Allowing these edits to also be represented as a module is what differentiates the edit chain from the existing audio plug-in architecture.

Examples of such an edit chain can be found in e.g. Apple's Soundtrack Pro and Audiofile Engineering's Sample Manager. This approach brings with it a number of benefits, that will be described in the following sections.

[25] For many parameters a slider suffices to specify a value within a certain range. Other parameters may require a switch (on/off) or a control that presents a set of options. For a filter, an editable graphical curve might be the best interface. Controls come in various shapes, whatever is appropriate.

[26] Real-time plug-ins cannot look at a specific point in time other than the present, they cannot affect the past or future. When editing offline, the entire timespan of an audio file is known, there is no "present". This for instance makes it possible to cut a portion of audio starting at a specific point in audio file and then paste it at the endany other point.

3.1.2 Non-destructive Editing

The term *non-destructive editing* is used for a variety of concepts, and not always appropriately. Ranging from the fact that one doesn't physically cut tape with audio editing software, to the fact that an earlier applied operation can be undone or even changed, *non-destructive editing* is a quite ambiguous term.

Considering the early history of audio editing, *non-destructive* illustrates the benefit of the digital audio editor, that indeed nothing is physically cut (Roads 1996, p.759). A "non-destructive digital audio editor" however already implicates this advantage because it is digital, so *non-destructive* must mean something else here.

In a digital audio editor, a file is loaded from disk memory (or other source memory) into a working memory or temporary file, and can then be edited[27]. The original file is only changed when the user chooses to save his changes to this same file. Only then is the initial state of the data in this file destroyed and replaced by new data. Since this practically is standard practice for almost all editing software, the term seems not completely appropriate here.

The ability to undo a previously applied operation is not so much a feature, it is merely a means to an end, it allows one to rehearse an operation to perfection. A downside to the general undo mechanism is that if one applies a first operation and then a second, and then wishes to change the first, the second must be undone as well and cannot be redone just by selecting "redo". In other words, it is destroyed, and this means the undo mechanism is not *non-destructive*.

An editing program is non-destructive when an edit (every edit) can be rehearsed and adjusted ad infinitum. Each edit causes the system to write a description of the edit in an "electronic logbook" (Roads 1996, p.780). This means that not the change of data is stored, but merely the operating parameters.

By storing all edit information, the user can change this information at any moment. This is called non-destructive editing. With the common editing approach, only the audio is stored, and recalculated each time an edit is applied. The non-destructive approach takes the source audio data, "renders" (calculates) the entire chain of edits at once, resulting in output audio data. Both source data and edit information remain preserved, and only the edit information is subject to change.

As said, this information includes edit sequence, parameter settings and selections. *Figure 5* shows a graphical user interface, in which each edit – just as earlier described – is represented by a module in a vertically oriented chain. By dragging a module up or down this chain, the sequence in which the edits are

[27] Some programs operate directly onto the data on disk. The altered data cannot be changed to its initial state, unless it is stored elsewhere (in working memory or temporary files). The initial state of the data is otherwise lost, so this kind of operation is therefore destructive.

applied can be altered[28]. A module can be removed from the chain, discarding the edit all together. A new edit can be inserted at any point between edits, or just at the end of the chain. Allowing the user to change the parameter settings enables him to precisely fine-tune his edits. The start- and end position of a selection can be altered just as easily, for instance when one wishes to cut-and-paste a slightly shorter portion of audio.

It must be noted that when ready, the audio processing must be rendered again in order to apply all changes. If the edit chain includes complex effects such as convolution[29], having to re-render them might slow down the editing process. For simple edits such as normalisation this most likely will not be too noticeable. An audio editor does render offline, which can mean that with a fast processor (and almost every new processor gets faster) rendering might as well be faster than real time[30].

3.1.3 Batch Processing

With the modular edit chain every edit is represented as a module. This principle can be extended by regarding every step in the editing process as a module, including loading and saving a file. A user selects an audio source (e.g. an audio file), specifies a number of edits, and also states how the output must be saved to disk. These are all modules, and when the user selects to render the whole chain, the source is loaded, the edits are applied to this source, and the result is saved to disk, all in the manner the user prescribed. Since this approach is non-destructive, every aspect of the editing process is subject to change – now also including the selected audio source. The user can change this by going to the first module in the chain which loads the file, and selecting a different audio source. When rendering again, the new source is loaded, the exact same edits are applied and the output is stored to disk in a similar fashion[31]. This way of editing is similar to using an *effects stack*, which can be found in GoldWave and Acoustica Premium[32], allowing the user to re-apply his favourite effects.

When having to edit a small number of audio files in a similar fashion this is an easy solution. But in some cases one needs to edit a huge amount of files, a *batch*, for example applying normalisation and a fade-in and -out on

[28] Altering the editing sequence can have serious effect on the end result. Take for instance compression and reverb. If you would first apply a reverb to a sound and then compress it, the soft reverb tail can become much louder as when first compression and then reverb would be applied.

[29] Convolution – in its most basic form – is a process in which each discrete amplitude value of a digital audio signal is multiplied with each amplitude value of another signal. Convolution is discussed in more detail in chapter 3.7.

[30] In this case meaning it takes as less time to apply the edits as the length of the audio material.

[31] To avoid overwriting the result of the first render (which will happen if every result is simply stored as fe.ig. *outputname*.wav), one can specify as output file name a variation on the source file name. For instance, *sourcename*.wav will result in *sourcename*_processed.wav

[32] By Acon Digital Media

every single file. Batch processing is particularly used by sound designers, producers and audio engineers, often having to uniformly process and convert large quantities of files in preparation for a production that requires all audio files to be at a specific level and of a specific file type. To achieve this, they use so-called batch processors that let the user select a list of files, specify a number of (generally basic) edits, and describe how the processed files must be saved to disk (also allowing file type conversion). The modular editing approach can also deal with batches of files, simply by introducing a file loading module that allows for multiple files to be selected. This way editing multiple files is just as easy as editing one file, providing the same editing capabilities.

Batch processing is often used for file type conversion, when a large number of audio files need to be converted for instance from an uncompressed format such as Linear PCM to a compressed format such as MP3. With the modular approach the output type (in this case MP3) can be set when specifying the output module. The processed audio will then be saved to disk in the specified format. But sometimes a user needs a number of different formats of a single file, e.g. varying compression qualities for online playback. The audio passing from one module to the next does not need to stop when an output file is generated. A second output module or even more can be appended – or even inserted at any other point in the chain – and for each a different output format can be specified (*figure 6*).

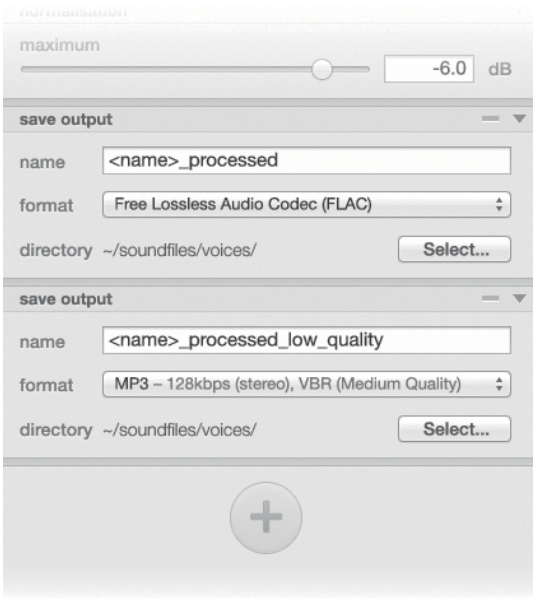


Figure 6. Multiple output modules allow simultaneous saving in different formats.

3.1.4 Summary

When regarding each step of an editing process as a module in a chain, the editing workflow is improved in several ways. The user can edit non-destructively, because all editing information is stored and can be changed at any moment. By also regarding the loading and saving of audio as modules, the editing chain can be reused and applied on multiple files, hence combining two existing concepts – the effects stack and batch processing – into a single principle.

The modular editing workflow serves as the foundation for some of the other design principles that will be discussed later on, revealing more subtle benefits of this approach.

3.2 The Source Browser

Audio editing software operates on audio files, stored on disk. An audio file can contain anything from a single sound to an entire musical composition. It might be the result of either recording natural world sounds, or generating (*synthesising*) virtual world sounds, making the possibilities of what an audio file contains endless. One might have numerous audio files, in various formats, stored on hard disks, compact disks or any other digital storage medium. Almost every user of audio software utilises a different, personal approach to organising these files, sorting them (often in folders[33]) by name, project, characteristics, or what else might be a useful parameter.

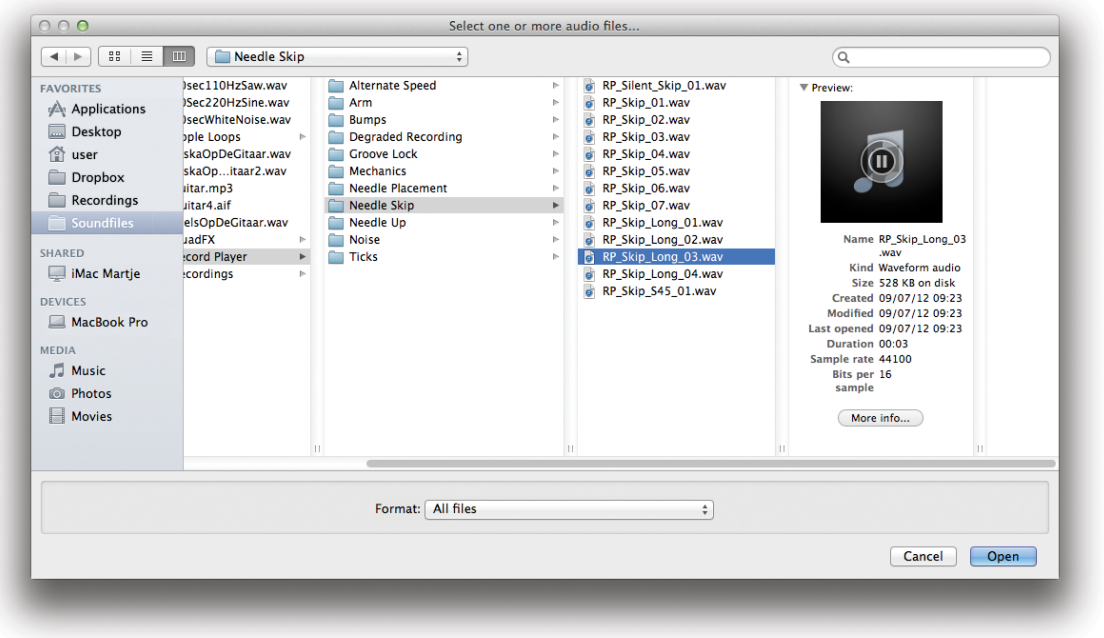


Figure 7. Mac OS X (10.9) file dialog in column view, providing duration, sample rate and bits per sample as audio-specific info. It also provides a player to pre-listen a file, though this only supports audio file formats that the system can play.

3.2.1 A Better File Dialog

Most audio editors present a default file dialog to navigate the folder hierarchy on a disk in order to locate a file. Such a file dialog has a generic design in order to fit in any kind of software running on the same operating system. Though this provides a uniform and recognisable approach to selecting a file, it lacks some aspects that would make navigating audio files easier.

The default file dialog on Mac OS X (in *column view*, see *figure 7*) provides the user with file details, such as name, type, size and creation, modification and last access date. A preview of the file is added if the file can be read by the system software; in the case of an image file, a thumbnail of this image could be displayed. For audio files – of a format that can natively be processed by the

[33] meaning disk folders

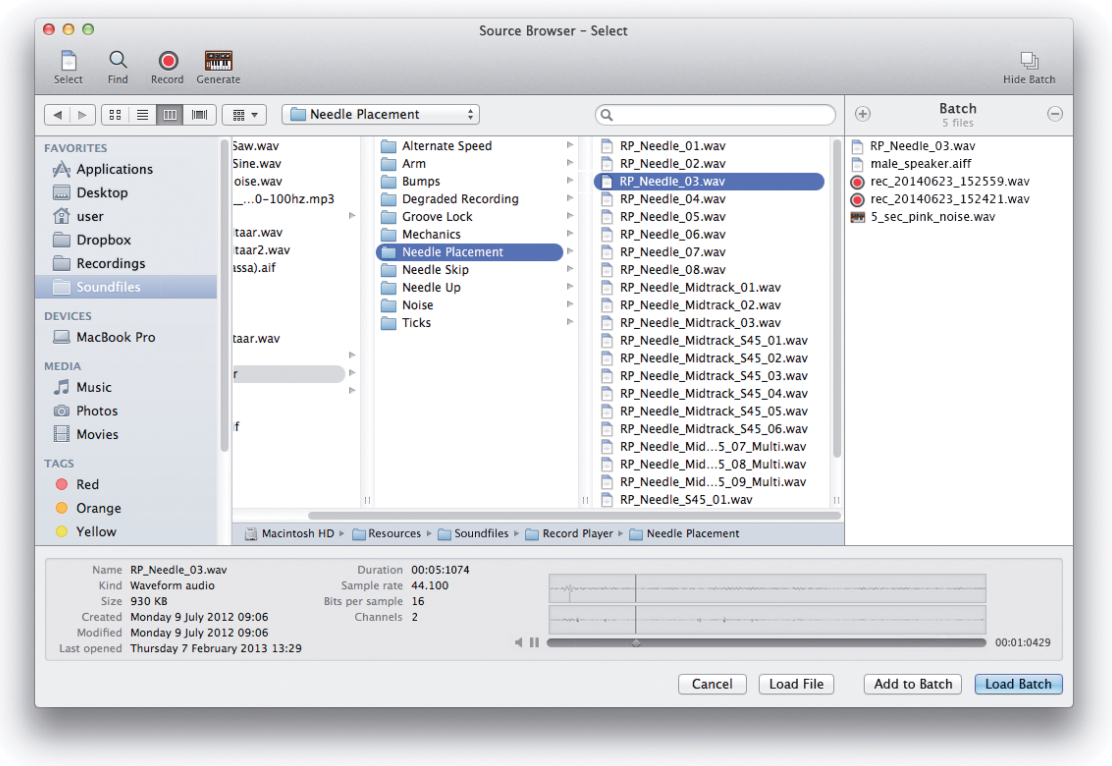


Figure 8. The Source Browser, showing a column view for browsing a file structure, and a batch list for selecting and loading multiple sources.

system software – duration info and approximate bit rate are also displayed, as well as simple playback for previewing the audio file, though this only provides play and pause functionality.

When browsing files – for instance a set of photo's – and looking for a specific file, one can quickly recognise photos by looking at the provided thumbnail. Because of the nature of audio (being time dependent), it is much harder to likewise browse a set of audio files. Being unable to play and then jump through an audio file makes it more difficult to recognise lengthy files or files with an unspecific start. A progress bar can be added to the playback functionality which indicates playback progress and can also be used to select a different position in the file to be played. This way a user can preview the portion of audio he is interested in, which might just as well be at the very end of a file.

Now the user has selected one file, s/he might want to select additional files in order to create a batch of files to be processed uniformly. A default file dialog might provide the selection of multiple files, but this comes at the expense of obtaining the above described file details for a single file (current file size, date, duration, bitrate and particularly preview); these can only be viewed when a single file is selected, because only info for a single file can be displayed. For this purpose, one part of the file dialog can be dedicated to creating a batch, enabling the user to add the currently selected file, and possibly remove previously added files. When satisfied, the entire batch can be loaded, just as easily as one would load a single file (see *figure 8*).

The selected file will be *opened*, which in most audio editors means a visual representation of the audio is provided, and the user can now edit the audio

either through the visual representation (depending on which user interface actions are implemented), or through additional windows and menus that provide editing functionality. Implementing the earlier described modular approach, a first module will be displayed representing the file or files that were selected. By doing so, the selected file(s) can at any moment be changed, regardless of any modules (representing edits) added after file selection. This effectively changes the subsequent chain of edit modules in an *effects stack*, that can be reapplied in exactly the same configuration to any other file.

3.2.2 Recording

The ability to record audio is one of the most implemented features among audio editors. The input can come from external sources such as a microphone or an instrument connected to a sound card, or from an internal source being another piece of software, for instance a software synthesizer or a radio streaming application. Multiple channels might be recorded, and recording settings such as input device (sound card), sample rate, bit depth and input gain can be specified.

The approach to recording audio varies among audio editors. While some only provide a dialog for the above described details, others provide real-time analysis of the incoming audio, visually representing what is being recorded. In either case, when the recording is done, the recorded audio is presented just as one would load an audio file. In other words, it results in having a *source* that can be edited.

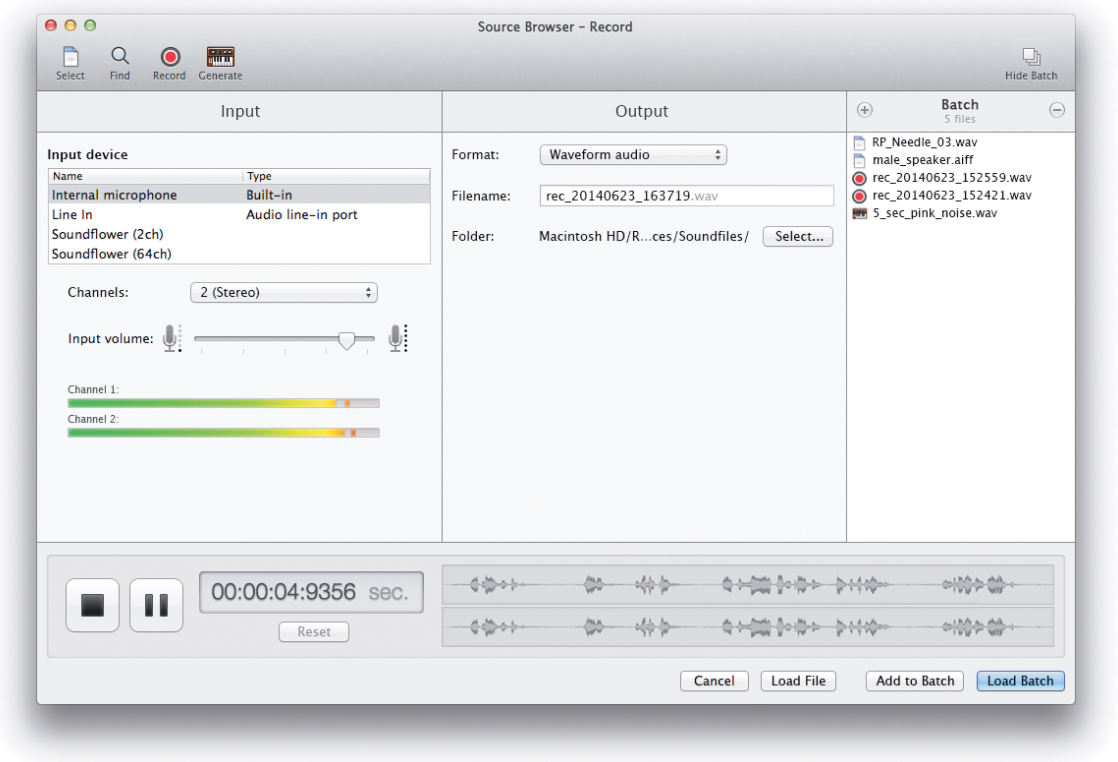


Figure 9. Recording in the Source Browser, allowing specification of input and output. A compact waveform display is provided below. New recordings can be added to the batch.

It would be logical to present recording an audio file in the same dialog as selecting an audio file (also a *source*), making the *file dialog* more a *"Source Browser"*. This also enables the user to add new recordings to a batch that can already contain files selected from disk (*figure 9*).

3.2.3 Synthesis

To extend the idea of a *Source Browser* further, another common audio editor feature[34] namely synthesis, might be added to this same dialog window. Synthesis in audio editors mostly comes in the form of signal generators. These are basic synthesizers used most often to generate arbitrary lengths of basic waveforms such as sine waves, or noise. Such basic sounds are useful for testing purposes, but might also be used as a source material for shaping a new sound, using transformations and effects to get a more complex sound. In either case, generating a signal amounts to generating a *source*, and so it fits as well in the *Source Browser* as selecting or recording a file (*figure 10*).

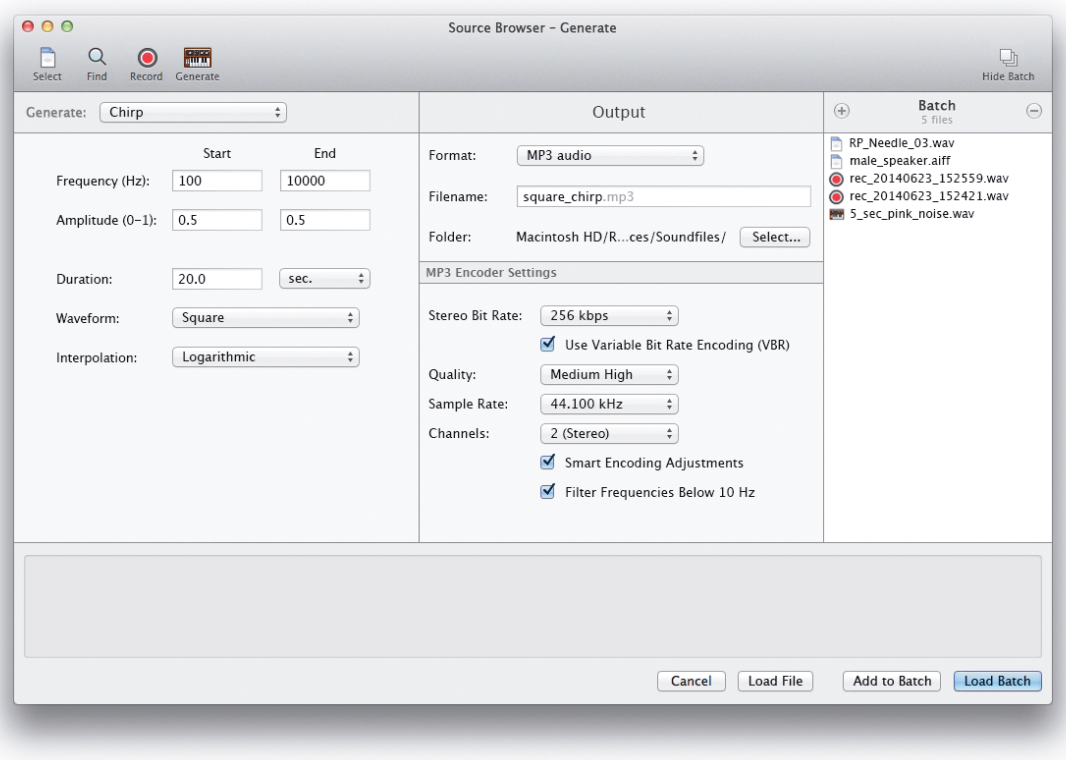


Figure 10. The Source Browser facilitates generation of audio sources using basic synthesis techniques.

3.2.4 Searching Audio Files

Searching audio files is a dreadful undertaking. File search systems are aimed at finding files by name (or textual content), so one's naming conventions for audio files must be sound, and must cover what one might be searching for. This shouldn't pose a problem when searching for a man shouting

[34] Around 45% of all surveyed audio editors; see appendix 1.

(e.g. "`shouting_man.wav`") or a damped major C chord on a piano (e.g. "`piano_major_c_chord_damped.aiff`"), but it might be problematic when you want to search for less specific details and characteristics, such as a soft high-pitched sound of specific duration, or a crescendoing sound with a wide spectrum. You might even want to find any audio file that just "is similar" to the one you have selected.

The *Source Browser* is an ideal place to include detailed audio-aimed searching functionality, as it will hopefully lead to finding files that can then be selected for editing (*figure 11*). A search filter might be constructed by adding property-value parameters, such as "channels: 2" or "duration: < 5 seconds", or using multiple values such as "spectral centroid[35]: within 2 kHz to 4 kHz". It might even be possible to specify a rough curve (moving up / down) to be able to find the crescendoing sound mentioned earlier. Other parameters can include format, sample rate, spectral density and overall magnitude.

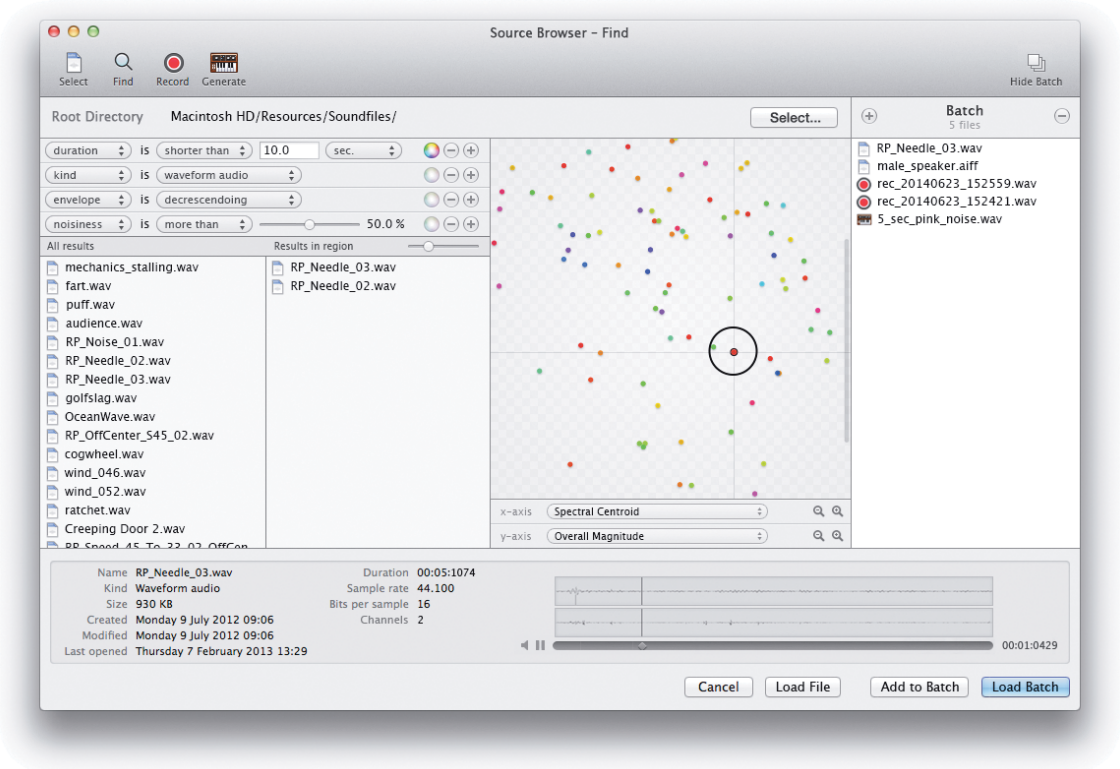


Figure 11. The Source Browser allows one to find an audio file not by name but by audible characteristics.

Searching audio files by their content requires analysis, which is a process that requires some time. For best performance, the filesystem on which the search is conducted might be scanned beforehand to index and analyse all available audio files. This makes it possible quickly to search for characteristics, but also lets you represent available files on a grid. The axes of this grid can be defined by the user, set to represent one characteristic on each axis. One might for instance select spectral centroid on one axis, and overall magnitude on the

[35] The spectral centroid is an indication of where the "centre of mass" in a given spectrum is.

other axis. Every indexed audio file will be placed somewhere along these axes, according to its spectral centroid and overall magnitude. It is not essential that the user understands exactly what these parameters indicate and how they are mapped along an axis; given these two parameters, all audio files with similar parameters will be grouped close together, and the user can discover what s/he is looking for by listening to various sounds across the grid, searching for example that soft high-pitched sound best suited for their needs.

3.2.5 Summary

An audio editor requires audio material to be edited. This audio can come from various sources, such as a file on disk, a recording or from synthesis. In current audio editors these various ways to obtain source audio material are detached, scattered throughout the graphical user interface. They do however all provide the user with audio material to be edited. The *Source Browser* provides the user with just a single window from which an audio source can be obtained. Existing audio files can be located through a file system dialog, extended to provide audio-specific details and pre-listening functionality. A search engine based on audio analysis enables the user easily to find audio files that match certain audio-specific criteria, such as temporal and spectral characteristics. Within the same window, audio can also be recorded and synthesised, both resulting in new audio files. Either way, a selected audio file can be individually loaded or added to a batch, both resulting in a source module (see the previous chapter on modular editing) representing the selected file(s). Edits will then be uniformly applied to all audio files this module represents.

3.3 Representation & Manipulation

A visual representation of audio is an integral part of audio editing software. It provides the user with visual indications of notable aspects of the loaded audio file, such as overall dynamics, peaks, balance, progression, recording or processing artefacts and noise. It can also serve as visual feedback of the user's edits, illustrating the effect of, for instance, fading audio in or out, compression, normalisation or noise removal. Used solely as a graph, visual audio representation is the primary tool for audio analysis. Used as graphic user interface element, a visual representation can also serve as an editing tool, allowing to select portions of audio on which to perform edits, to indicate time-varying curves for specific edits such as fades, or to directly perform transformations such as stretching, shifting or bending.

3.3.1 The Waveform

In the majority of audio software – not just audio *editing* software – the (default) approach to representing audio visually is by displaying a waveform. The waveform display shows amplitude changes over time, representing the pressure changes registered by a recording device. A microphone, just as the human eardrum or any physical object, registers sound (air pressure changes) as a combined pressure amplitude of all incoming pressure waves at a single instance of time. A waveform display thus represents the ***physical*** nature of sound. For a single frame of time, a single value or *sample* represents the average registered amplitude over a small span of time, the length of which depends on the sampling rate. Multiple channels might be available within the same timeframe. For example, stereo audio is generally recorded using 2 microphones, resulting in two channels of audio, represented by two *samples* for a single frame of time.

Uncompressed audio is typically stored as a list of samples. CD quality audio is stored as 2 sets (i.e. 2 channels) of samples, counting 44100 samples for every second of audio (the sampling rate), for each channel. From a developer's point of view, generating a waveform display is almost[36] as simple as plotting a line that follows each sample or amplitude value. Other representations – that will be discussed below – require complex transformations of the sample values to obtain a different data set and hence a different display.

A waveform is generally plotted on a horizontal axis representing time, against a vertical axis representing amplitude. If we were to plot every frame of samples on a single column of pixels, then on an average computer display we could only see about 40 milliseconds of CD quality audio. Zooming out will require a single pixel width to represent multiple frames of audio, meaning some kind of averaging needs to be performed. Just adding all sample values

[36] Apart from having to account for converting between integer and floating point values, endianness (byte order) and interlacing of channels.

within a single frame and dividing it by the number of samples might result in a value of 0, especially in the case of a steady oscillation such as a pure sine wave, moving equally as much above as below zero amplitude. The decision on what then to display differs among software.

Different kinds of waveforms

A common approach is to find the loudest sample, or peak value, amongst each group of samples needing to be represented by a single value. For very short timespans (zoomed in) a line can be drawn between each consecutive peak value, which can be positive or negative. For wider timespans (zoomed out) both a positive and negative peak value are determined for each group of samples, and the area between these two values is coloured in. This approach can be found for instance in Rogue Amoeba's Fission (*figure 12a*).

An elaboration on this approach is to draw the *root mean square* or RMS value on top of the peak value graph. The RMS is the square root of the average of the squares of each sample within a timeframe, and as it uses averaging it will never exceed the peak value, thus fitting nicely within the peak graph. Combined, it provides feedback on the average loudness and maximum amplitude of a timeframe. Audacity and Sweep are examples of audio editing software implementing this approach (*figure 12b*).

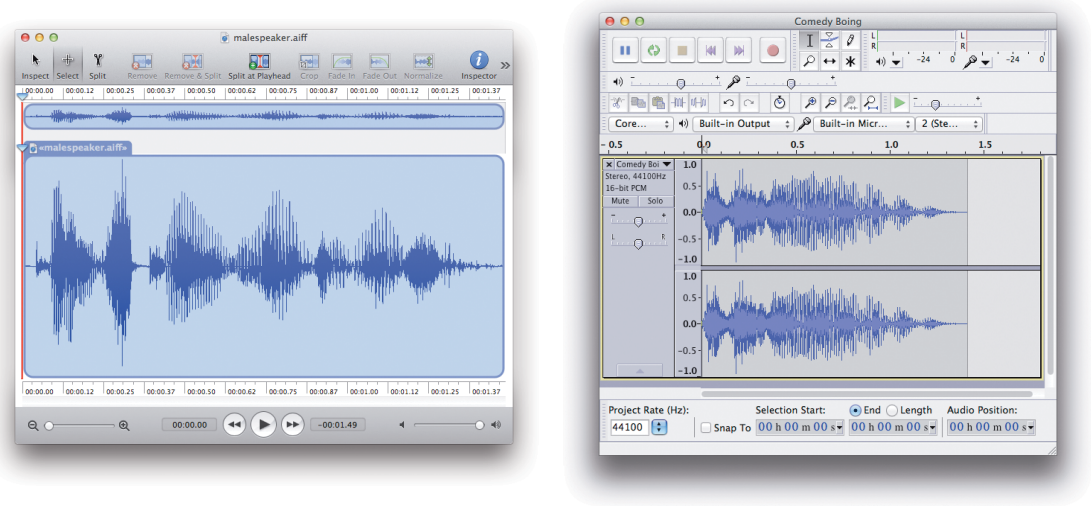


Figure 12. a) The waveform in Rogue Amoeba's Fission only shows positive and negative peak values. b) Audacity (by Roger Dannenberg) displays a waveform that shows both peak and RMS.

In most cases, amplitude is plotted on the vertical axis on a linear scale. This might however be misleading, as it doesn't correspond to the way the human ear works. If we halve the amplitude of a sound, we won't perceive it as half as loud. Utilising a logarithmic scale gives an image more approximate to our loudness perception. The logarithmic scale is used in decibel meters, as decibel is a logarithmic unit. Making a sound softer by a factor of 2 (halving the *power*) means decreasing it by 10 log(2) = 3dB. For sound *pressure* (dB SPL), a change by a factor of 2 means a change by 20 log(2) = 6dB. Our loudness perception differs in relation to frequency (Roads 1996, p.38; Collins 2010, p.8), but by

approximation an increase of roughly 10dB SPL is perceived to be twice as loud.

Waveform displays are used for a great variety of purposes, and depending on the importance of speed, accuracy and aesthetics, different approaches are taken in transforming sample values to a visual graph. Surprisingly often the bottom half of the waveform data (below zero) is discarded and replaced by a mirror of the top half, just for nicer appearance. When accuracy is of less importance than visual quality, the rough contour of the waveform might suffice, and one might even opt to leave out the bottom half all together.

Certain DJ applications need to emphasise visually beats so that the DJ can visually cue to specific points to prepare a transition. One way to accomplish this is to take the *n*'th power (e.g. the fourth) of the amplitude value, which will emphasise the difference between lower and higher amplitudes, hence accentuating peaks. The popular DJ software Traktor (by Native Instruments) takes the waveform a step further by colouring it with extra analysis data. This can be a spectral analysis, from which a spectral centroid or spread[37] can be deduced. By using a colour gradient to discriminate high values from low values, a waveform can be coloured in, displaying for instance high spectral density with a bright colour, and low density with a darker colour. Sonic Visualizer[38], a true audio analysis application, goes even further by allowing the user to overlay different analysis visualisations, making it possible to see a wide range of sonic parameters and characteristics in a single view, on a single timeline.

3.3.2 The Spectrum

Spectral analysis can provide valuable insights in audio editing. It describes the energy distribution of an audio wave over frequency bands, one band constituting a range of frequencies. As mentioned earlier, sound, being air pressure waves, is registered by our eardrum. The vibrations then pass a number of different and complex components of our inner ear to finally reach the cochlea, an organ lined with nerve cells, each of which resonates with a different frequency. These nerve cells signal the brain, enabling us to perceive different frequencies (Marieb and Hoehn 2007, p.587). If we perceive a high-pitched sound, this means there's much energy in the higher frequencies. A spectrum display can show this energy distribution, and thus closely relates to the ***perceived*** nature of sound.

A frequency can only be determined by observing the number of cycles of a sound wave over a period of time. The more cycles, the higher the frequency. As the amplitude values of the waveform are measured against time, the waveform is expressed in the *time domain*. To see the power of individual frequencies (or frequency bands), a transformation to the *frequency domain* is required. The

[37] Spectral spread is an indication of the bandwidth of a spectrum.

[38] Developed by Queen Mary University of London. www.sonicvisualiser.org

most widely used mathematical technique for this is the Fourier Transform[39]. There's an abundance of literature on this subject, a fine example of which is Loy (2007b), to which is referred for further details on this technique. The Fast Fourier Transform (FFT) is, as the name implies, a fast Fourier algorithm, and therefore so popular and thus widely used that in audiophile parlance a frequency analysis (or spectral analysis) is often simply referred to as an FFT (Rockmore 1999; Loy 2007b, p.111).

It must be noted however, that much as with plotting a waveform, there is no single approach to implementing the Fourier Transform. The Fourier Transform itself is a purely mathematical theory, its implementation for digital audio is generally the (Discrete-Time) Short-Time Fourier Transform. This implementation determines the spectral content for a small *window* in time. A windowing function[40] must be applied to each window to get a more accurate analysis. The window size determines the frequency resolution of the analysis, but also inversely influences the time resolution, always posing a trade-off. Analysis windows can be overlapped to improve time resolution. The choice of window size, overlapping method and windowing function make the number of possible implementations countless. These decisions are important for a developer as it will affect the quality, speed and usefulness of the spectral analysis; a user will rarely be directly confronted with them.

Audio is divided in a number of small, optionally overlapping, time frames (the aforementioned *windows*) and a single analysis is made for each time frame. This analysis reveals the magnitude for a series of frequency bands or *bins*. These values can be plotted either by drawing a time-varying line on a frequency versus magnitude grid, known as a *discrete* or *line spectrum* (*figure 13a*), or by using a (colour) gradient to express the magnitude of each bin plotted on a time versus frequency grid, known as a *spectrogram* (*figure 13b*). Another approach is the *waterfall plot* (*figure 13c*), a 3D landscape-like plot of magnitude on the vertical axis versus frequency on the horizontal axis and time along the third axis optionally moving in real time (Roads 1996, p.541).

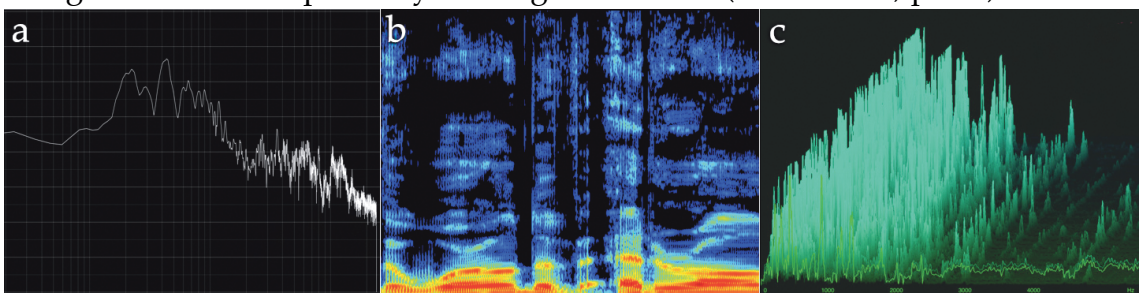


Figure 13. a): a line spectrum with frequency on the horizontal and magnitude on the vertical axis. b) a spectrogram with time on the horizontal and frequency on the vertical axis, using color to represent magnitude. c) a waterfall plot with frequency on the horizontal and magnitude on the vertical axis. Time is represented on the third axis.

[39] There are other techniques to derive a spectral analysis, among which the Wavelet Transform, Constant Q Transform and Dictionary-Based Methods (Sturm et al. 2009).

[40] A windowing function is a mathematical function that has a value of 0 outside of a specific range, and commonly a positive value within that range. For the use of windowing functions in spectral analysis, see Loy (2007b, p.140-145).

Each frequency band is equally wide (for instance 25 Hz), consequently often resulting in bins plotted along a linear frequency axis. The visual space occupied by frequencies between 1000 and 2000 Hz then is equally wide as between 8000 and 9000 Hz. The human perception of musical intervals is however (more or less) logarithmic, meaning 2000 Hz sounds an octave higher than 1000 Hz, while 9000 Hz only sounds a major second higher than 8000 Hz. Plotting bins over time against a logarithmic frequency axis, as for instance IRCAM's AudioSculpt does, results in a spectrogram that can be read as a piano roll, a much more informative approach for musical purposes (Bogaards 2008). This way musical intervals look invariant over the frequency scale (Risset 1991).

A spectrogram uses a gradient to express bin magnitudes. Commonly used are grayscale brightness gradients, where white indicates a high magnitude value and black a low magnitude value, and rainbow gradients, where – resembling temperature maps – blue indicates a low magnitude value, running through green and yellow to red indicating a high magnitude value. The human eye can discriminate colour differences more easily than brightness differences, which is why many spectrograms are coloured using a rainbow gradient. This gradient is however contrasting over the entire magnitude range. Enabling the user to modify the gradient would allow him to specify regions of interest where highly contrasting colours are used. In order visually to emphasise small differences in low magnitude frequency bins, one could use a gradient such as in *figure 14*. AudioSculpt also incorporates a contrast filter that can be applied to the spectrogram, for example to remove noise or to bring out details within a noisy spectrum.

Figure 14. A rainbow gradient

3.3.3 Reading Waveforms And Spectra

A waveform display offers sample-accurate timing information and allows the user to see overall magnitude and magnitude changes over time. A spectrum display provides frequency information, valuable in music production and analysis environments, but due to the nature of the analysis there is always a trade-off between frequency resolution and time resolution (Roads 1996, p.557).

To get an idea of what can be determined from both audio representation methods, a set of 18 test signals (following as signals *a* to *r*) are used, selected to illustrate a variety of sound properties such as waveform, noise, harmonic structure, modulation, distortion and recording artefacts.

a) A steady-frequency 440Hz sine tone at fixed gain 0.3
   - *Waveform:* Because the signal is only one steadily oscillating wave, the 0.3 amplitude can clearly be determined. The sine wave can be seen, but there is no certainty this is a pure sine.
   - *Spectrogram:* A single line, indicating a pure sine, can be seen at 440Hz. It is uniformly coloured, so there is no substantial gain change.

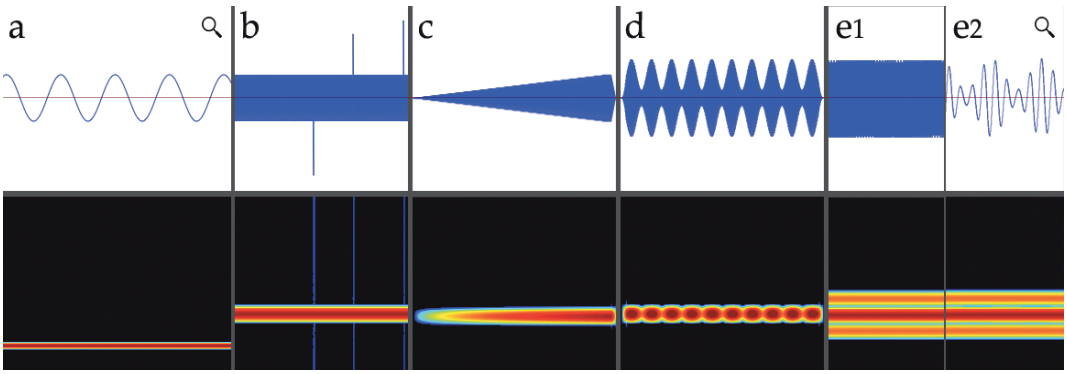


Figure 15. a) A steady-frequency 440Hz sine tone at fixed gain 0.3. b) Signal a, with spikes (1-sample ticks) added. c) Signal a, crescendoing linearly, from 0 gain to 0.3. d) Signal a, with a 5Hz tremolo (amplitude modulation) between 0.1 and 0.5 gain. e) Signal d, but with the modulation frequency at 100Hz.

b) As signal *a*, but with spikes (1-sample ticks) added
   - *Waveform:* The spikes can clearly be seen with their individual amplitudes and they each span only a single sample.
   - *Spectrogram:* The spectral impact of the spikes can be seen as a narrow (one window width) light line across the entire spectrum.

c) As signal *a* crescendoing linearly, from 0 gain to 0.3
   - *Waveform:* A triangular envelope exactly describes the crescendo.
   - *Spectrogram:* The brightness of the line gradually changes, but no exact values or shape can be determined.

d) As signal *a*, with a 5Hz tremolo (amplitude modulation) between 0.1 and 0.5 gain
   - *Waveform:* As with the crescendo, a smoothly oscillating envelope describes the kind of modulation. The speed cannot be determined exactly.
   - *Spectrogram:* The brightness of the line periodically changes from dark to light, but here too no exact modulation speed cannot be determined.

e) As signal *d*, but with the modulation frequency at 100Hz
   - *Waveform:* No details can be deduced, other than that (when zoomed in, *e2*) it appears to be a sinusoidal oscillation with changing amplitudes.
   - *Spectrogram:* A denser and slightly wider line indicates there might have been introduced some side frequencies. That this might be due to the rapid modulation cannot be determined.

f) A square wave with fundamental frequency 440Hz and fixed gain of 0.3
   - *Waveform:* The waveform shows why this signal is called a square wave.
   - *Spectrogram:* The harmonic structure of a square wave consists only of odd-integer harmonics, consecutively getting softer. On a linear scale this shows equally spaced lines, brightness gradually getting less in the higher frequencies.

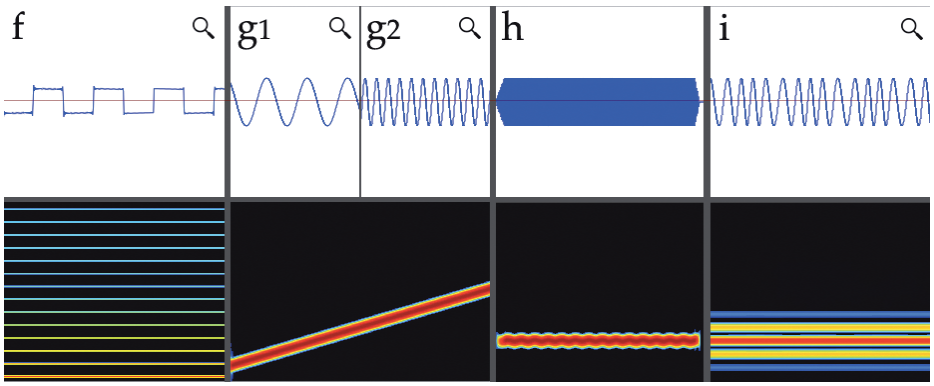


Figure 16. f) A square wave with fundamental frequency 440Hz and fixed gain of 0.3. g) A sine wave at 0.3 gain, going linearly from 220Hz to 880Hz. h) A sine wave with a 6Hz vibrato (frequency modulation) between 435 and 445Hz. i) A sine wave with a 100Hz frequency modulation between 340 and 540Hz.

g) A sine wave at 0.3 gain, going linearly from 220Hz to 880Hz
   - *Waveform:* The signal looks much as *a*, but it can be seen the wave period is getting smaller.
   - *Spectrogram:* A straight line going from 220Hz to 880Hz exactly describes this glissando.

h) A sine wave with a 6Hz vibrato (frequency modulation) between 435 and 445Hz
   - *Waveform:* This signal cannot be distinguished from signal *a*.
   - *Spectrogram:* The frequency line has a slight oscillating character. The speed of the oscillation cannot be determined. The upper and lower frequency are approximately 10Hz apart, but it cannot be determined exactly.

i) A sine wave with a 100Hz frequency modulation between 340 and 540Hz
   - *Waveform:* This somewhat looks as signal *g*, but now the wave period gets longer and smaller periodically.
   - *Spectrogram:* A dense line at around 440Hz is surrounded by a softer area between approximately 340 and 540Hz. Due to the time-frequency trade-off it can hardly be seen that there is any modulation.

j) A sine wave getting louder than maximum, being clipped
   - *Waveform:* Every amplitude above 1 is set to 1. Values below −1 are set to −1. This is hard clipping.
   - *Spectrogram:* A very bright line is accompanied by evenly spaced harmonics when the signal goes above maximum. The sharp edges of the clipping have introduced overtones.

k) A sine wave getting louder than maximum, being wrapped (aurally very painful!)
   - *Waveform:* The parts of the oscillation that should be above 1 are now popping up at the bottom (from −1) and vice versa. They are wrapped around.
   - *Spectrogram:* Very many and very bright (loud) overtones are introduced, as well as a lot of noise in-between them. Wrapping is disastrous.

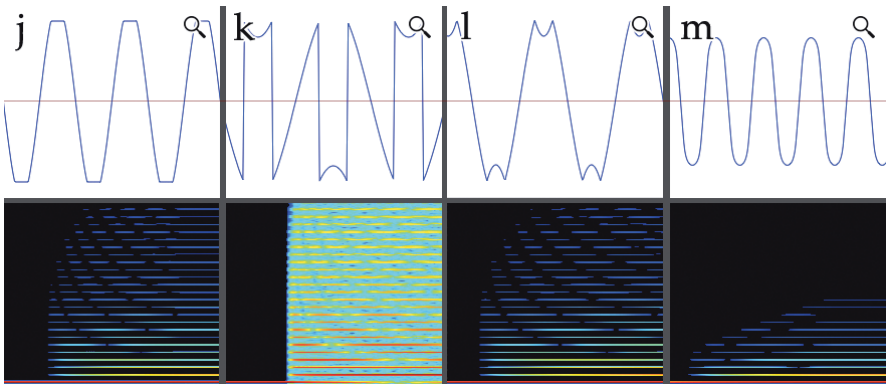


Figure 17. j) A sine wave getting louder than maximum, being clipped. k, l, m) A sine wave getting louder than maximum, respectively being wrapped, folded and soft-clipped.

l) A sine wave getting louder than maximum, being folded (every amplitude above 1 is subtracted from 1)
   - *Waveform:* The parts that were wrapped in signal *k* are now folded back, creating dips at the peaks of the oscillation.
   - *Spectrogram:* Similar to signal *j*, but slightly more extensive.

m) A sine wave getting louder than maximum, being soft-clipped
   - *Waveform:* As the signal gets louder, the sinusoidal waveform gets rounder, slightly wider at its peaks. It looks as signal *j* but with smoothened edges.
   - *Spectrogram:* Similar to signal *j*, but slightly less extensive.

n) A single decaying piano note[41]. This intentionally is a bad recording, having a DC offset[42], a noisy background, and a hard cut at the end.
   - *Waveform:* A DC-offset can clearly be seen as the amplitude center is not at 0. The signal starts out loud, but then decays rapidly. At the end of the decay the line quickly jumps from its DC-offset to 0 amplitude.
   - *Spectrogram:* The piano note has a number of harmonics, each decaying at a different rate. There is an irregular distribution across

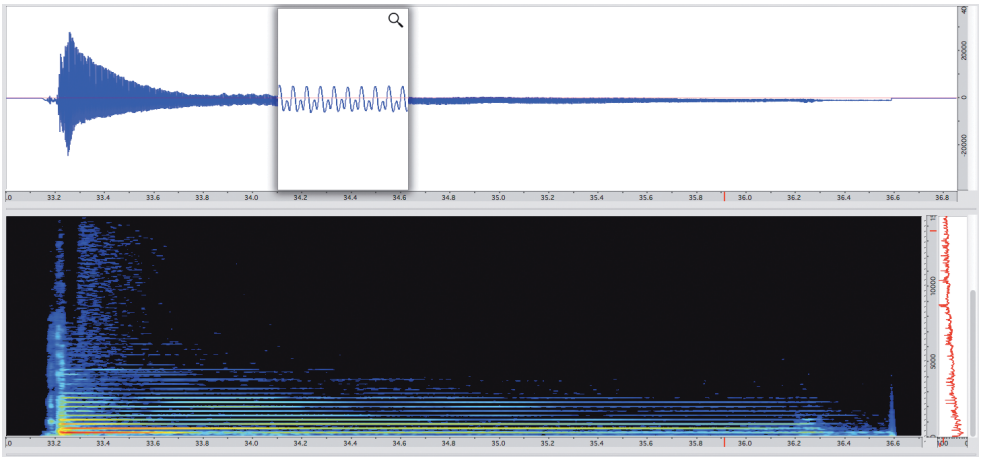


Figure 18. n) A single decaying piano note.

[41] Freesound: Grandmither's Piano (28).wav (Techsetsu - 11647)

[42] If a signal has a DC offset, it oscillates not around an amplitude value of 0, but just above or below it. Playing such a signal for a prolonged time can harm your speakers.

these harmonics, indicating a more complex harmonic structure than for instance the square wave of signal *f*. The spectrum has a lot of background noise, as well as a steady gradient at the very bottom, which might be the DC-offset being interpreted as a very low frequency. At the end of the signal there's a very sharp line across a wide frequency range, the result of the hard cut from offset to 0.

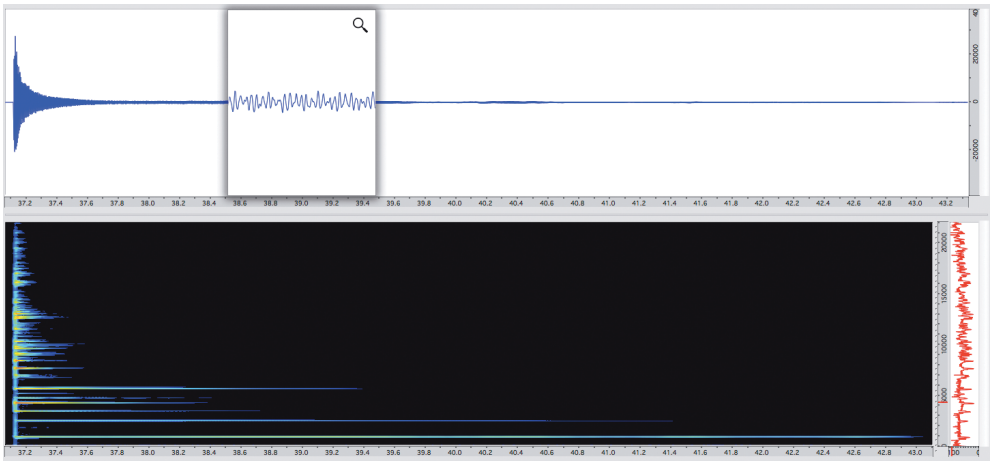


Figure 19. o) A small Tibetan bell.

o) A small Tibetan bell[43] struck once, decaying
- *Waveform:* The signal starts out loud, but then decays more rapidly than signal *n*.
- *Spectrogram:* This signal is much cleaner than signal *n* and has a far more complex harmonic structure. There are many overtones, even in the very high frequency ranges, varying in decay length, loudness and even amplitude modulation.

p) A steady clarinet note[44]
- *Waveform:* A relatively complex but quite periodical waveform can be seen. Overall it looks a bit similar to signal *f*.

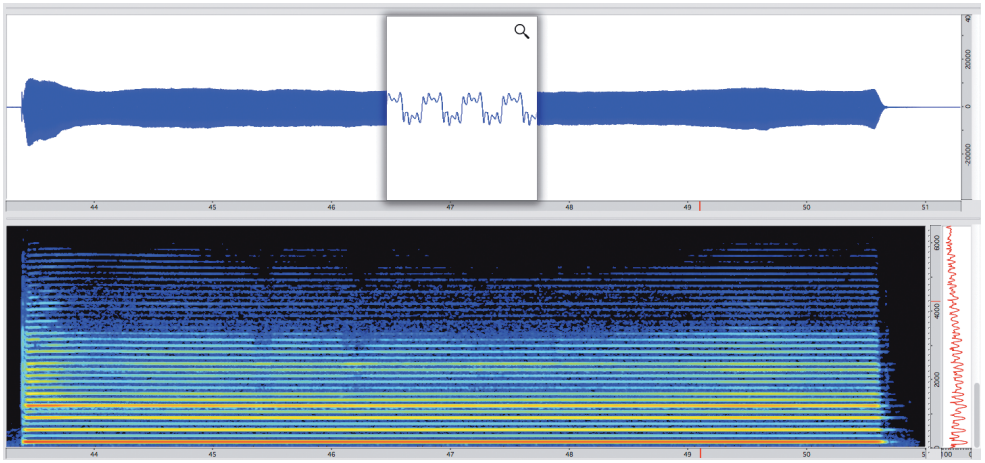


Figure 20. p) A steady clarinet note.

[43] Freesound: Tibetan bell.wav (djgriffin - 15401)

[44] Freesound: Largo.wav (insinger - 25898)

- *Spectrogram:* A harmonic structure similar in complexity to the piano is displayed, as well as some varying noise. Where there's little noise, there are more or brighter harmonics in the higher frequency range, indicating the noise might be unvoiced tones and breath.

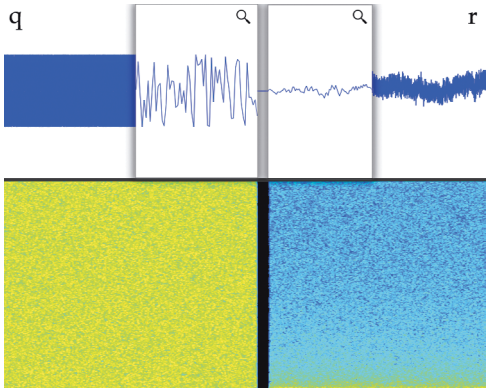


Figure 21. q) White noise. r) Pink noise.

q) White noise
- *Waveform:* Zoomed out this signal looks much as a steady signal such as a or *f*. Zoomed in it appears to be an irregularly and extremely varying signal. White noise is total randomness, so this might be it.
- *Spectrogram:* Equally distributed noise from left to right, from top to bottom, much as when your old TV isn't working; this most likely is white noise.

r) Pink noise, having a 3dB falloff per octave
- *Waveform:* Overall it looks much more random than white noise, but when zoomed in one can see the variations are more gradual, indicating less high-frequency variation.
- *Spectrogram:* This signal looks like white noise but with a gradient, getting softer towards the higher frequencies.

A waveform display can best be used when overall characteristics of a signal must be monitored. Amplitude changes and anomalies as well as low-frequency oscillations are clearly visible and sample-accurate timing information is available, so very precise selections can be made (selections will be discussed further on). Depending on their severity, recording artefacts such as clipping, ticks and DC-offset are visible on a waveform display. It must be noted that a periodic waveform with a certain amplitude spectrum can look very different when altering the phase spectrum. It will however sound the same, as the ear is phase-deaf (Risset 1982, p.55). In this respect a signal might sound such as a square wave (signal *f*) but look very different.

A spectrum display can provide insights in spectral content such as fundamental frequency, frequency modulation, harmonics and noise. It can reveal individual parts such as the breathing noise in signal *p* between its tonal parts, or the intricate harmonic structure in the Tibetan bell (signal *o*). It can also reveal recording artefacts such as undesired harmonics, the effect of DC-offset

and background noises (signal *n*). Though a spectrum display provides valuable information for music production, recording and analysis, due to its time-frequency tradeoff it lacks the time precision the waveform display can offer.

For the discussed signals it was often a combination of a waveform display and a frequency display that revealed the entire "picture" of a signal, one example of which was the square wave (signal *f*), which has a typical waveform as well as spectral characteristic. Signals *j* to *m* demonstrated a cause visualised in the waveform display with great effect that was revealed in the spectral display. Therefore it would be ideal if audio editing software presents both a waveform display and spectrum display. Fine examples of software already incorporating both alongside are Adobe Audition and IRCAM's AudioSculpt.

3.3.4 Quick Editing

3.3.4.1 Editing Using The Waveform Display

The waveform display can also serve as a user interface control. Most often the waveform display is used as an interface to define selections of audio on which to perform edits. This practice is so common that a whole group of software can be discriminated that has a waveform control as centrepiece of the user interface: the *wave editor*.

Basic Selection

The simplest user action is to click[45] on the waveform. A single click will specify a point in time within the waveform, which can be used as start point for playback, or to position a marker. Markers serve as an aid for subsequent selections, indicating points of interest. Selecting a segment of audio can be performed by clicking and then dragging over the waveform along the time axis, defining a start and end point. The selected segment can now be separately edited, without affecting the unselected audio. Audio selection will be discussed further in the following chapter, *"Smart Selections"*. Here, the manipulation of a selected segment will be discussed.

Axes

The waveform display represents audio as amplitudes on the vertical axis, along time on the horizontal axis. A selection on a waveform display is only a selection along the time axis. Selecting a range of amplitudes might serve useful only for processes for which a more fitting user interface already exists[46].

Basic editing actions on a segment of audio are cutting, copying and pasting. This way, audio can be moved, removed, and duplicated, just as one might do

[45] In this thesis, for the examples of user input the combination of computer mouse and keyboard is used as input device, which currently is still the most predominant input method.

[46] One example might be wave shaping, mapping amplitudes to different amplitude values.

with text in a text editing application. Along this resemblance, a less common procedure in audio editing is repositioning, "grabbing" a selected segment and dragging it along the time axis to the desired position.

So far, these basic editing procedures do not alter the audio within the selected segment. Considering the two axes along which our user interface actions might operate, two types of transformations can easily be applied. Stretching the selection along the time axis can equate to time-stretching the audio. This is becoming a common practice in modern music production software such as Apple's Logic (*"Flex Time"*) and Ableton Live (*"Warping"*), where it becomes a powerful tool when for instance synchronising a sampled piece of music to the beat of the final project.

As the selection is only a selection on the time axis, it cannot be stretched out vertically. Therefore a horizontal line inside the selected segment (either along zero amplitude or along the bottom of the display) might be used to present a value that can be shifted up or down. Shifting along the vertical axis can be associated with a gain change, as this will change the amplitude up or down (see *figure 22*). An example of this can be seen in Adobe Audition. A less straightforward association may be pitch shifting, often appearing paired with time stretching and therefore a suitable candidate for the vertical axis transformation. Pitch being conceived as relatively "higher" or "lower" to another pitch is also strongly directionally associated with the vertical axis.

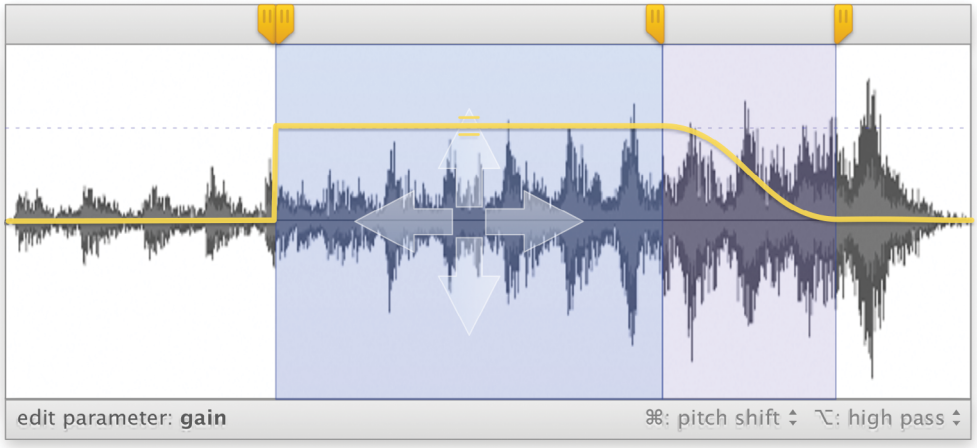


Figure 22. A waveform display that provides gain adjustment by dragging up or down. Other parameters can be adjusted using moderator keys (see bottom-right).

Modifier Keys

Time stretching and pitch shifting are but two possible processes that can be applied by manipulating the waveform display. Looping a selection by extending it along the time axis can be desirable in (combination with) music production software such as Apple's Garage Band. Considering the loudness war (Katz 2007), dynamic compression by dragging along the vertical axis (louder, louder!) is conceivably also convenient. In any way, the employed process might be replaced through a settings menu, or alternated using so-called *modifier keys*. These keyboard keys, among which most common the

Shift, Control, Alt and Command key, modify the behaviour or action of other computer keyboard keys and mouse actions. As *figure 22* illustrates, a possible implementation can have gain change as the default vertical axis process, while pressing the Apple Command key (⌘) to alternate this to pitch shifting, or pressing the Apple Option (or Alt) key (⌥) to alternate to high pass filtering. Combinations of modifier keys, such as Command + Shift, can extend the number of available processes even further[47].

Transitions

The waveform display in *figure 22* also provides a basic means for parameter automation. In the illustrated example, the gain is increased for the selected audio. The unselected audio is kept unchanged, which can be regarded as having a gain change of zero. On the left side of the selection, the direct transition from unchanged audio to the increased gain within the selection will be unpleasant audibly. On the right side of the selection, a transition area is defined by dragging a handle at the top of the display apart from the selection. Within this area a smooth transition from increased gain to zero is performed by *automating* (meaning to automatically change) the gain parameter, making the audible result more pleasant. Some audio editing software (such as Audiofile Engineering's Wave Editor) incorporate this approach to create fades (fade in / out) at the edges of a selection, but no other effect parameters can yet be automated this way. Parameter automation will be discussed in detail in chapter 3.6, *"Sonic Composition"*.

3.3.4.2 Editing Using The Spectrum Display

Spectral Selections

A spectrogram is a two-dimensional image on a time axis and a frequency axis. As with a waveform, a selection in time can be made, but now a specific frequency range can also be specified. Using a rectangular selection tool (in Photoshop called a *marquee* tool) a portion of the two-dimensional image can be selected that delineates both a timeframe and frequency range. Another commonly implemented and more precise tool is the *free selection* tool or *lasso*. With it a specific region can be drawn, allowing the user to trace a portion of the spectrum, varying over time. This is particularly useful for selecting partials that change over frequency, or small bits of noise between other components, without also selecting and affecting (see spectral editing further on) the directly surrounding spectral components. Adobe, creator of both Photoshop and Audition, implements another selection tool called *paintbrush*, allowing the user to paint the selection using a round brush tool of a specified width, which results in a similar time-varying frequency selection. AudioSculpt also includes a *Magic Wand* tool that allows the user to click anywhere on the spectrum display, automatically selecting contiguous spectral bins within a predefined

[47] In a touch user interface, alternate control can be realised by using two or more fingers instead of just one.

range in dB around the clicked bin. This makes it easy to for instance select a frequency-modulated partial, or a cloud of noise, which can then be moved or otherwise manipulated (Bogaards 2008).

Spectral Editing

The ability to select both a segment in time and in frequency allows for effects to be applied to just the specified time frame, and to just the specified frequency range. This can be as simple as amplifying or attenuating partials or noise elements, but any effect that can be applied to a standard time selection can also be applied to a time-frequency selection. Think of reverberating only the higher frequencies of a voice, or compressing specific components of a recording. Some processes do make less sense, such as high-pass-filtering only high frequency components.

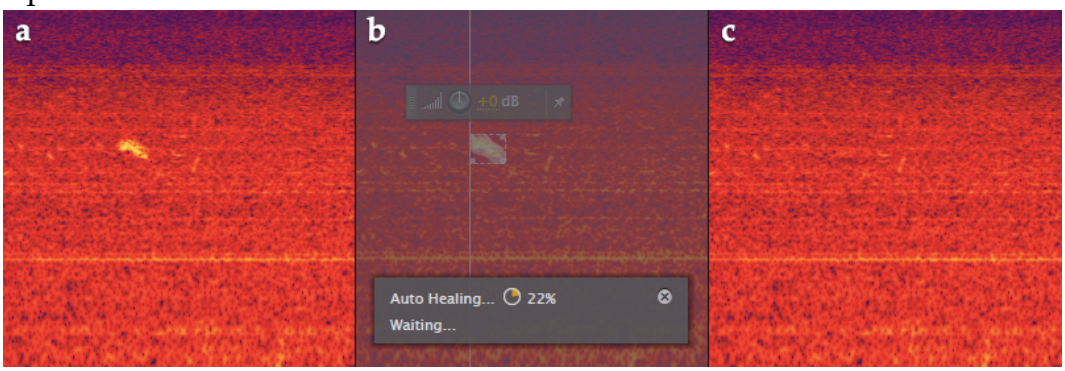
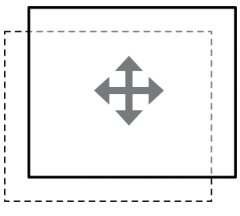


Figure 23. Spot healing in Adobe Audition. a) shows a small but audible noise element amidst background noise. Spot healing allows selection and automatic removal of this element, filling it up with similar background noise.

To continue with Audition, another tool implemented is the *spot healing* tool (*figure 23*), which exactly resembles an image editing tool from Photoshop, to draw over a time-frequency region that will be levelled with the surrounding spectral content. This is ideal for removing sharp noises on a slightly noisy background, not removing all the noise but just the sharp aspect. Audition also provides a gain adjustment control directly above the selection, which makes amplifying or attenuating selections easy.

Editing by directly manipulating the selection becomes more advanced now there are two axes along which audio can be affected. In a two-dimensional space, we have the following transformations at our disposal: move, stretch, scale, skew, distort and flip. Photoshop does offer more transformations (rotate, warp and perspective), but these are less meaningful in audio editing.

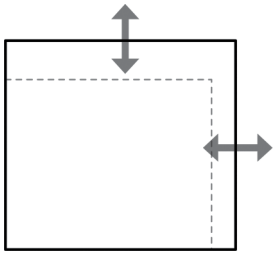
- **Move**



Moving along the time axis equals the aforementioned repositioning, though now it can be frequency-independent. Moving along the frequency axis will *shift* or *transpose* the frequencies, depending on the selected scale (respectively linear and logarithmic). While shifting moves all frequencies

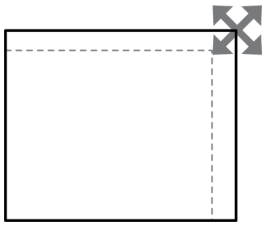
a same amount, transposing moves all frequencies a same ratio. The latter will preserve the harmonic structure of for instance an instrument and the resultant will hence be more recognisable to the original than when frequencies are shifted[48].

- **Stretch**



Stretching up along the frequency axis on a linear scale will effectively be the same as transposing, moving the frequencies up a same factor. On a logarithmic scale the harmonic structure will be stretched out in the higher frequencies. Stretching downwards will have the same effect on the lower frequencies, while preserving the harmonic structure within the higher frequencies. Stretching along the time axis accounts to frequency-independent time stretching. This time stretching can be made content-aware. This is interesting, considering the onset problem. If one were to simply stretch out a single recorded trumpet tone, the most important part of that recording for us to recognise it as a trumpet, the onset, will be stretched out as well. The short burst of noise parts and inharmonicity that forms the onset of the played tone is what guides us in recognising the instrument (Howard and Angus 2001, p.213). This goes for trumpets as well as violins and practically most instruments that can be used to produce sustained instrumental tones. Stretched far enough, the onset and thus the instrument will no longer be recognisable. The onset is followed by a more harmonic and less varying spectrum which dictates the timbre, which is more suitable for time stretching.

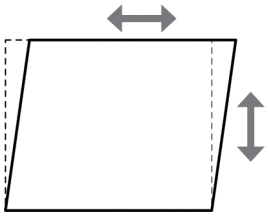
- **Scale**



Scaling is stretching along both axes. An interesting examination of *tape-speed variation* can be made here. When playing back audio twice as fast, the audio is twice as short, and all wavelengths are halved making the audio sound twice as high. This then effectively scales down along the time axis (twice as short) and up along the frequency axis (twice as high). The inverse is true when playing back audio twice as slow; the audio becomes twice as long, and sounds twice as low. When stretching along both axes, any variation and deviation on this process can be made, including time-preserving pitch shifting and pitch-preserving time stretching, or anything in-between.

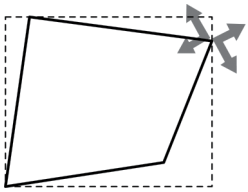
[48] It must be noted that in this case the *formant structure* is lost. A formant is a peak of energy in the spectrum, which can include both harmonic and inharmonic partials as well as noise (Roads 1996, p.296). The formant structure can be regarded as the filter that defines the character of a voice (particularly the vowels) or a musical instrument. A pitch-shifted voice will not be recognised as originating from the same person.

- **Skew**



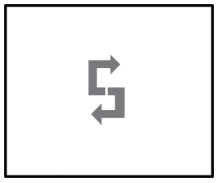
Skewing allows one side of a selection to remain fixed while the opposite side is shifted. Along the frequency axis this allows for applying a gradual pitch shift over time. Along the time axis a very interesting effect called *spectral delay* can be introduced, meaning components of the selected audio are individually delayed, which can for instance result in having the lowest frequency component played first, followed progressively by the higher components.

- **Distort**



Distorting the selection means corners can freely be moved. It offers simultaneous time stretching, spectral delaying and time-varying pitch shifting, making it an extremely powerful tool.

- **Flip**



Flipping spectral content along the time axis will effectively reverse the selected audio. Along the frequency axis the results will be obscure, as recognising audio depends heavily on relationship between the lower and higher frequency components.

These more artistic transformations are rarely found in current audio editing software. The ability to amplify, attenuate and remove spectral content is found in various applications, and is particularly useful for audio restoration and cleaning. One might also find use of such software outside the music industry, for instance in forensics, where spectral editing is used to isolate and enhance certain parts of audio recordings.

3.3.5 Other Representations

The waveform is by far the most predominant visual representation method in audio software. Spectrum displays are more often found in the more comprehensive audio editing software packages, where it is generally presented as a secondary display. Some audio editing software, such as iZotope RX, is aimed at editing spectral aspects and therefore presents the spectral display as the key display. Such software is typically referred to as a *spectrum editor*, as opposed to the *wave editor*.

It must be noted that, though rare, there are spectrum editors that employ a different or alternative approach to spectrum analysis and the associated editing. SPEAR as well as AudioSculpt[49] use a sinusoidal modelling technique

[49] Using the sinusoidal modelling technique instead of a Fourier Transform is optional in AudioSculpt

(Mcaulay and Quatieri 1986) to attempt to represent a sound with many individual sinusoidal tracks called *partials*. This looks similar to a spectrogram, but instead of displaying amplitude changes within a frequency range (a bin), each partial corresponds to a single sinusoidal wave varying over time in both magnitude and frequency. This approach effectively separates the tonal parts of a sound from its noise parts, as noise cannot be captured as time-varying sinusoids. Editing using this model allows for example to shift individual harmonics of a pitch-varying note of a recorded instrument. The technique is however heavily arbitrary, because the decisions needed to track partials (determining partial birth, death, cross-frame movement, etc.) are not set in stone and are open to interpretation. Therefore there is no guarantee that what you see in SPEAR resembles what you see in AudioSculpt.

A unique approach to audio analysis and editing is Celemony's *Direct Note Access*, as found in their landmark software Melodyne. An analysis on polyphonic audio material identifies the individual notes, which are presented on a piano-roll grid as small waveforms, thus forming a kind of mix between waveform and spectral display. Each waveform is one note that can be transposed, stretched, moved or removed, allowing for intuitive retuning, recomposing and decomposing of polyphonic audio material. As this software is aimed at accessing and editing the *musical* content of audio, it is not regarded as an *audio editor*. Audio editors, as the name implies, aim at providing an *audio signal representation*. A musical representation as provided by Melodyne can be regarded as a *symbolic representation*. This differentiation is important. A symbolic representation can only represent what is within the limits of its symbolic vocabulary (here music theory), and can only represent this discretely. Yet an audio signal representation can continuously[50] transmit any, non-musical kind of sound, even non-audible signals (Vinet 2003, p.194).

3.3.6 Summary

The most common method visually to represent audio is to plot a waveform, which represents the physical nature of audio. Depending on the importance of speed, accuracy and aesthetics, the appearance of such a waveform can vary. A waveform display can give precise timing information and a clear image of amplitude characteristics.

An approximation of the perceived nature of audio can be visually represented by plotting a spectral analysis. The appearance of this is also dependent on various choices, such as algorithm, representation type and colouring method. A spectrum display can reveal individual parts within the spectral content of audio.

For accurate representation of audio characteristics a combination of these two displays is recommended, allowing precise time and amplitude inspection using the waveform display, whilst observing the related spectral content using the spectrum display. Being able to see the relation between these two can give

[50] apart from being digitised through sampling

insights in cause and effect of certain audio artefacts. For instance, sharp edges in a waveform caused by a sudden gain change or clipping will effect in added harmonics, which will be visible in the spectrum display.

Both a waveform display and a spectrum display can serve as audio editing interface. By manipulating and dragging a selected time or time-frequency segment of audio, various transformations can be applied, ranging from gain change to time stretching and pitch shifting, even spectral delaying. Finding logical analogies between the audio content that is visually represented and the editing gesture of that representation can result in an intuitive and powerful editing interface. For sound designers and composers – people who use an audio editor to shape a sound towards a desired result and who value a good visual representation – such an immediate and clear approach can prove very useful, allowing them directly to see what they are doing.

3.4 Smart Selections

As described in the previous chapter, a visual representation of audio can be used to select a segment of audio on which to perform edits. This chapter explores how this most basic and essential action in audio editing can be extended.

3.4.1 The Standard Selection

The basic selection, as found in most audio editing software, involves clicking and then dragging over a waveform display, demarcating the desired audio segment. Once a selection is made, edits can be applied to only this segment of audio – the audio that is not selected is generally[51] not affected by the applied edit. The selected segment might also be cut, copied and pasted somewhere else along the time axis. The selection can be discarded at any time, but when doing so, it cannot be retrieved. In some editing software it can be reconstructed if markers can be set along the time axis.

When observing the modular editing approach described in chapter 3.1, a selection can be interpreted as an individual module (*figure 24*). This module keeps track of the start and end time of the selected segment. By doing so, a selection is ever retrievable as long as it exists as module, but moreover it is adjustable. The segment boundaries can be shifted, affecting the edits following the selection module. Every time the modules are *rendered*, the specified selection is taken and as long as no other selection or deselection is specified, consecutive edits will be applied to *that* selection. The selection module can also be copied, making the selection reusable.

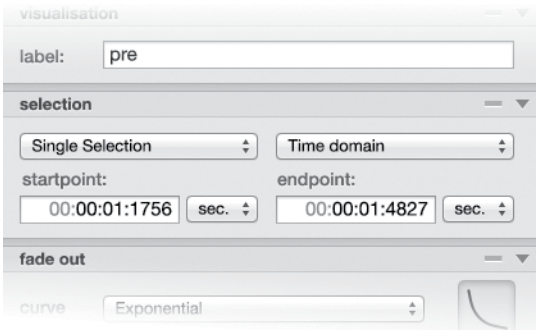


Figure 24. A selection displayed as a module in the editing chain.

When clicking and dragging over a waveform to specify the boundaries of a selection, commonly the resulting start and end points in time are determined by the point where the mouse was pressed down and where it was released. In situations where in the visual representation a single pixel width spans a considerable amount of audio samples, setting a start or end point is far from precise. When the boundaries are not on or near a so-called zero-crossing[52], an edit can cause a sharp change in amplitude, resulting in a hard tick (see *figure 25*).

[51] Some edits might involve tails. For instance, a reverb might be applied to the selected segment, of which the delayed reverberations might be added to the subsequent audio.

[52] A zero-crossing is a point where the signal values go from positive to negative or vice versa, crossing a value of zero.

In another scenario, one might require to apply an edit to a specific audible event such as a beat, but selecting the exact onset of the beat can prove to be difficult when having to set the selection manually.

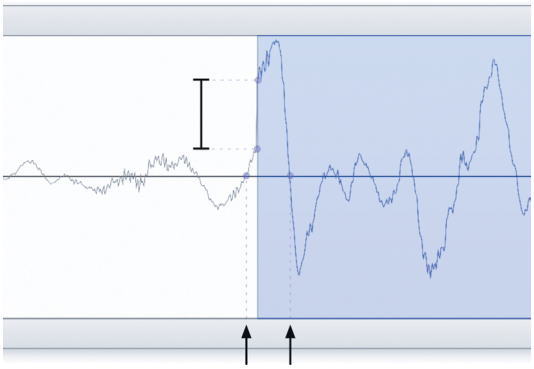


Figure 25. A gain change on a section that doesn't start at a zero-crossing (arrows) can result in a hard tick.

To aid manual selection, *snapping* can be introduced. Snapping means that when dragging a selection near a snapping point, that point is given priority over the cursor point to determine the selection border. For audio, an analysis of the audio is made to identify zero-crossings, transients, onsets and beats (Collins 2010, p.99). Now, when dragging over a waveform display, the selection boundary is *snapped* exactly to any identified point of interest, aiding the creation of precise selections. Snapping to zero-crossings is quite common, examples can for instance be found in TwistedWave, which also snaps to transients, and Wave Editor by Audiofile Engineering, which also supports snapping to time units, markers and the playback head. Beat detection in conjunction with selection can be very useful when slicing samples for looping purposes, aiding in precisely selecting a desired number of beats or bars.

As described in the previous section, selections can be extended with transitional edge areas. At the edges of the selection small handles are provided in order to define an area adjacent to the selection, in which a *dry-wet*[53] transition is made between the processed selected segment and the unprocessed adjacent audio, smoothening in and out the effect of the applied processes.

3.4.2 Multiple Selections

In the previous section we discussed various audio transformation methods that find their roots in image editing software. These transformations were based on manipulating a selection. This section discusses the process of making a selection, where again concepts from image editing software can be adapted for audio editing software.

In image editing software it is very common to make multiple selections. When a normal selection is made, a selection mode for succeeding selections can be defined. Photoshop provides the following:

- **New**
  The next selection will replace the current selection.
- **Add**
  The next selection is added alongside or, wherever selections intersect, to the current selection. This allows for multiple basic selections.

[53] Dry indicates the original unprocessed signal, wet indicates the processed signal. A dry-wet mix combines a bit of dry and a bit of wet signal.

- **Subtract**
  Wherever the next selection intersects with the current one, the selected segment is deselected.
- **Intersect**
  Wherever the next selection intersects with the current one, this becomes the new selection. Everything else is deselected.
- **Exclude**
  This is the inverse of *intersect*, deselecting only where selections intersect.

Being able to define multiple selections allows the user to process all selected parts at once, instead of having to select and process each part individually. The selection modes help to refine and shape selections. They are however quite uncommon in audio editing software; most editors do implement additive selection to extend the current selection, but that is it. Only rarely[54] is it possible to define multiple selections, whilst this can be very useful. Imagine a recording with a number of coughs in it, and you want to filter out these coughs. The easiest way would be to select each cough and then process them all together equally. Similarly, a set of beats or onsets can be uniformly processed for more oomph by selecting each and processing them together.

With the above described additive selection mode, overlapping selections are joined into one selection. It can however be desirable to define multiple overlapping selections. Techniques such as random cutting, shredding and brassage (as described in Wishart 1994a, p.59) are based on reordering segments of audio. Segments are cut from the source audio and then *respliced* together to produce a new sound (see *figure 26*). The selection here functions to define a segment to be repositioned instead of a segment to be processed. Overlapping segments cause certain parts of the selected source to be captured multiple times in the result.

3.4.3 Automated Selections

A couple of coughs or a handful of beats can quite easily be selected manually. But if, for instance, you want to select ticks and pops from a digitised vinyl recording, it becomes much harder. Not only are these probably much more numerous, they are sometimes much harder to identify. Therefore a selection can be automated. Given a definition of what needs to be selected, everything matching that definition is selected (*figure 27*). This can be ticks and pops that can then be attenuated or filtered out, or beats that can be sliced into individual samples. Everything under or above a given volume threshold can be selected, for instance to completely silence out almost-silent parts (again, see TwistedWave), or to soften the loudest bits of a recording. In spectral editing mode, background hums or other components with a distinguishable character might also be automatically selected. Automation can be very precise as well as

[54] see again TwistedWave

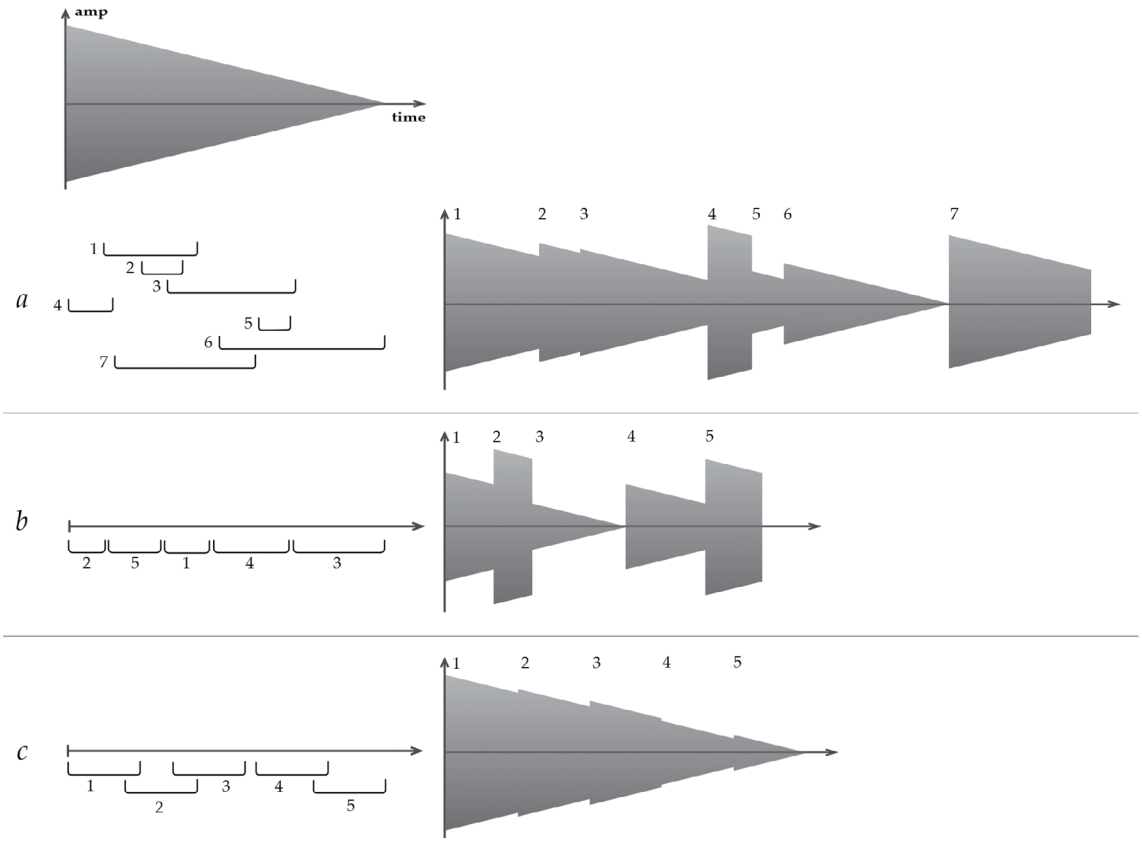


Figure 26. Random cutting (a), shredding (b) and one form of brassage (c). Random cutting: the segments have a random position and length, and can overlap. Shredding: similar, but no overlap. Brassage: overlap is allowed, but segments are sequential, which is characteristic for this technique. (Wishart 1994a, p.60)

timesaving when needing to select many segments.

This way, automated selections can also be applied in re-synthesis and reordering techniques such as described earlier. A number of segments can be selected, possibly of random length and at random points, which are then cut and recomposed in a specific way. As the process of selection is now automated, the size of these segments can be very small and the number of segments can be very large. It can even be applied to select wavesets or grains[55], allowing for more techniques described by Wishart (1994a) to be applied, such as waveset reversal, shuffle, and shaking, granular time-stretching and reordering.

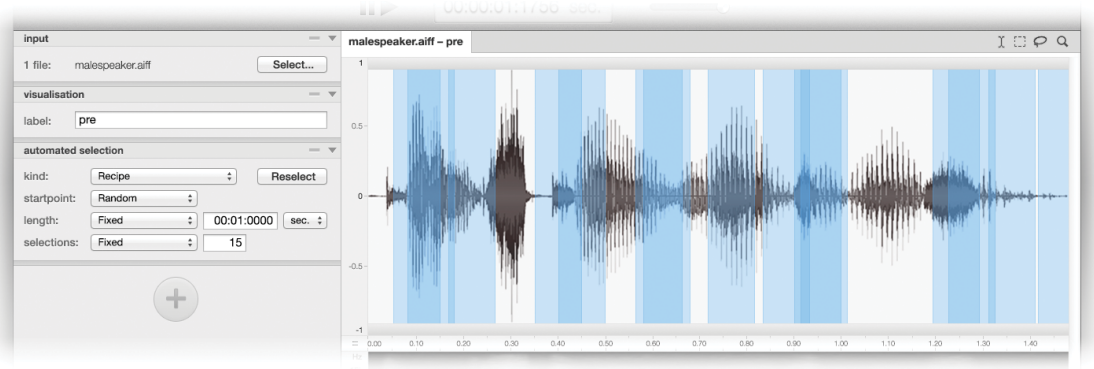


Figure 27. Multiple selections are automatically made by specifying selection criteria.

[55] A waveset is the signal between any pair of zero-crossings (Wishart 1994a, p.17). A grain is a small segment of audio, normally in the region of 10-100 milliseconds long (Roads 1996, p.168).

3.4.4 Finding Audio

With automated selections, most of the criteria of what needs to be selected are predefined by the software. By letting the user specify all the criteria, a *find* function becomes available. The user can now say "give me everything that meets these criteria", specifying the minimum and/or maximum length, the average amplitude or volume thresholds and perhaps a spectral centroid or density. Every segment of audio that matches these criteria is then selected. This process is similar to finding entire audio files, described in the chapter on the "Source Browser".

In case of the coughs-example, it can even be made easier to select all coughs. A *find-similar* approach requires selecting one segment (one cough in this case) that serves as benchmark. Given the characteristics that are determinant for this selection, such as length, loudness or spectral characteristics, every segment that matches the benchmark on the specified characteristics is selected as well (*figure 28*).

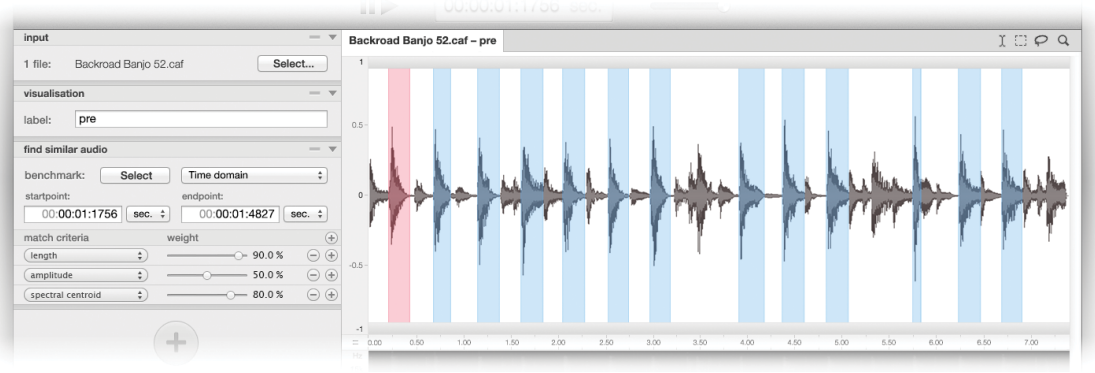


Figure 28. Finding similar audio based on a benchmark selection.

3.4.5 Summary

The ability to select segments of audio to edit is a fundamental aspect of audio editors. It allows for applying edits not only to an audio file in its entirety but to specific time or time-frequency segments within that file. Selecting exactly what you want can be difficult. Aiding the selection process by providing content-aware snapping speeds up and eases this process.

A selection is made in order to apply an edit to the selected audio segment. If such an edit should be applied to multiple segments, the specification of multiple selections can offer a swift approach to applying a uniform edit to these segments.

To make it easier to select all required segments, a multi-selection can be automated to pick just the segments that match certain criteria, or that are similar to a specific segment. Automating the selection process also allows for selecting very many, very short segments.

Multiple selections, whether specified manually or automated, can be used to quickly apply an edit uniformly, but also for reordering techniques. This extends the possibilities for using an audio editor for sonic composition.

3.5 Layered Audio

Tracks, channels and layers

One common aspect of many audio editors is the support for multiple *tracks*. A track is a placeholder along the time axis onto which audio can be recorded or loaded. Multiple tracks are represented under each other (with time progressing along the horizontal axis) to illustrate the simultaneous playback of these tracks. A single track needs not be filled entirely with audio, a section of audio can be represented as a container on a track that can be moved and positioned along the time axis (see *figure 29*).

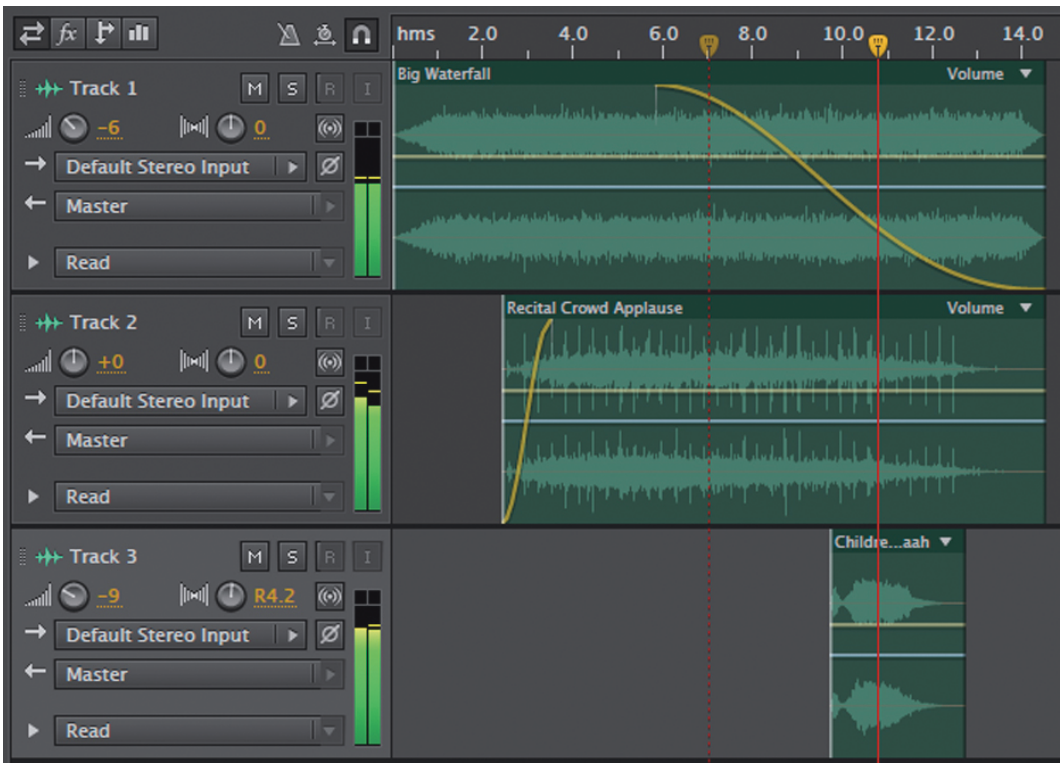


Figure 29. Audio tracks in Adobe Audition. Audio is represented as a container so that it can be repositioned along the time axis.

An audio editor that supports multiple tracks is often called a *multi-track editor*. Multi-track editing allows for a sound to be constructed out of multiple sounds. Each individual sound can be regarded as a layer of the final composition that can be edited separately and can be moved in time relatively to the other layers. During playback a mix is made of all tracks, at a single time instance producing the sum of all audio present in any track at that time. The resultant can also be bounced, registering the track mix to a single track or to disk.

Each track can consist of multiple channels, each representing one part of a multiphonic (multichannel) section of audio. In a mix all audio is mixed per channel. If one track contains a mono recording and a second track contains a stereo recording, the resultant will be a stereo mix where for each channel of the stereo track the mono track is added.

In chapter 2.1.1.2 a distinction was made between audio editors and Digital Audio Workstations (DAW's). In DAW's, a track is often regarded as an instrument, along with its own mastering parameters and effects processing bank (an *instrument track*). Such a track can contain an audio signal, just as a track in a multi-track audio editor can. But in a DAW, a track can also contain a symbolic music representation (Vinet 2003, p.194), such as a MIDI control signal. This control signal can be visually represented as a musical notation score or a *piano roll*, and is interpreted by a synthesiser associated with the instrument track. This way an instrument track can for instance represent a piano or a guitar (synthesiser) that plays the score that is placed on this track.

Whereas music production software such as a DAW uses tracks to construct music out of multiple instruments, an audio editor uses tracks to construct a sound out of multiple layers of sound. By using the term *layer* instead of track a firmer distinction between instrument tracks and nonmusical tracks is made. It also establishes a connection to imaging software, where layers are an essential composition tool and layering possibilities have become plentiful over the years. Inspired by these possibilities, in this section I will describe a number of audio layering techniques.

3.5.1 Blending Audio

In a multi-track editor the mix that is played back or bounced is created by adding all tracks together. In most of these editors a gain and a panning can be defined for each track, allowing some control over how each channel on each track is added to those of other tracks. When observing tracks as layers, one might regard a layer to be a part of a composite. This composite can be made up simply by the sum of its parts, but it can just as well be a more dynamic body of sound, where layers influence each other (*figure 30*).

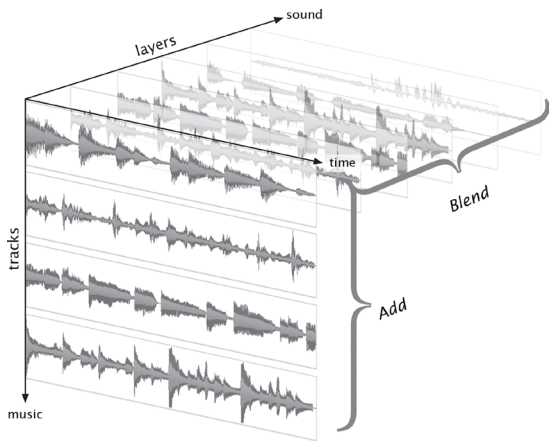


Figure 30. Adding tracks versus blending layers.

In digital image editing the compositing of layers is called *blending*. Layer blending uses a background-to-foreground (or bottom-to-top) approach; what is visible on the top most layer (the actual *mix*) is influenced by the content of the layers beneath it. There are several blending modes, aimed at either accentuating or attenuating parts of the colour spectrum as well as affecting colour saturation, brightness and contrast. This is done by taking the content of two layers and using colour info from one layer as control for adjusting the other, or by compositing individual components. The content of the bottom layer that is taken for this blend is actually the resultant of a blend between its original content and the content

of the layer beneath it. This in turn can also be a resultant of a blend, so on repeating down to the bottommost layer.

Layer blending in image editing is often used to accentuate components of an image by adding contrast or by increasing colour saturation. For such purposes the layer dictating this effect is carefully constructed to have the desired effect on specific areas of the original image, while leaving the rest as much unchanged. A good knowledge of the mathematics behind blend modes is required to effectively apply them. A blend between two layers can be very intricate, therefore a layer is rarely the result of a blend with a blend with a blend and so on, because anticipating the result gets harder when adding more layers and blends to the process.

The simple mix of audio channels can be expressed as adding all layers together, meaning for each frame of time all samples are summed up. No particular relationship between any two layers is defined, other than that their blend is their sum; you will hear one layer regardless of the content of the other. With other basic arithmetic operators a first step towards more interesting blend modes for audio can be made.

Negation can be used to cancel out audio. Though rarely truly effective, imagine one can use a recording of one instrument with unwanted spill[56] of another on one layer, and the recording of the second instrument on the other layer. In theory, when the spill and second recording are exactly the same and perfectly aligned, they should cancel out. In practice this is nigh on impossible to achieve, and audio subtraction is not very useful for anything else.

Multiplication of two audio signals can be seen as a process called *ring modulation*[57] or *amplitude modulation*, depending on both speed and shape of the modulating waveform. By multiplying two signals, one signal (the *modulator*) effectively modulates the amplitude of the other signal (the *carrier*). For ring modulation the modulator waveform is commonly of a simple shape (such as a sine) and a low frequency. A very well known application of ring modulation is for producing the voice of the Daleks from the BBC television series Doctor Who[58].

Side-chain Effects

Blending two audio signals per each individual sample value can be too convoluted. Using an abstraction is a way to obtain more foreseeable effects. One such an abstraction can be an average loudness curve or *envelope* of the modulator audio signal. Attenuating the carrier signal whenever the

[56] *Spill* is sound picked up by a microphone from a source other than that for which the microphone is intended.

[57] *Ring* refers to the shape of the analogue circuit of diodes originally used to implement this technique.

[58] Attributed to Brian Hodgson of the BBC Radiophonic Workshop. See YouTube for various *Dalek Voice Tutorials*.

modulating signal is loud creates an effect called *ducking*. It allows for the modulating signal whenever present to be more perceivable as the level of the carrier signal is lowered. Commonly a compressor is used to attenuate the carrier. Ducking can be used for voice-overs, where the music is lowered in volume when the voice starts, smoothly returning to its original volume whenever the voice stops. A pumping effect can be achieved by more tightly following the loudness curve[59] of the modulator audio signal, changing the volume of the carrier more quickly and varyingly. A simple example of this effect as applied in popular music can be found in Benny Benassi's *"Satisfaction"* (2003), where the kick drum is used to reduce the volume of the bass-line, resulting in a pumping sound.

In plug-in architectures found in Digital Audio Workstations, using one track of audio in the processing or synthesis of another is commonly referred to as using a *side-chain*. The side-chained track can be used to trigger or modulate one part of an effect or synthesis process. Using a compressor that is controlled by a side-chain track, as is just described for ducking, is often called *side-chain compression*. Side-chain compression can also be employed to reduce sibilant or plosive sounds. These are respectively the hissing and popping sounds in vocals. For sibilants, this process is known as *de-essing*, for plosives one might call it *de-popping*, though this term is used less commonly. A copy of the original vocal track is made, in which these sounds are accentuated using equalisation. Using this track as a modulator for the compression will emphasise on these accentuated sounds, reducing them in the original track that is compressed. In contrast with ducking, the modulating signal is not taken into the resulting mix, it only serves as modulator.

As illustrated by these approaches to side-chain compression, a blend between two layers of audio can be far more than applying simple arithmetic to them. It can just as well be an effects processor that takes one layer as the carrier and the other layer as the modulator. With ducking, de-essing and de-popping, the modulation is fairly simple, it is a single envelope curve that controls the gain of the compressor. A popular technique called *vocoding* uses a more complex form of modulation. The modulator audio is passed through a filter bank, and for each filter band the envelope is followed. The carrier is also passed through a filter bank set at identical frequencies as the modulator filter bank, and each filter band is amplified by the envelope of the corresponding modulator filter band (Roads 1996, p.197). Originally developed for encoding and decoding voice for effective and secure transmissions, it is a technique that would later be used particularly for voice synthesis effects such as robot voices. Such synthesised voices can be found in the works of many music artists, such as Kraftwerk (among others in *"The Robots"*, 1978) and Isao Tomita (*"The Visionary Flight to the 1448 Nebular Group of The Bootes"*, 1978), to name just a very few.

[59] This process is called *envelope following*.

Blend masking

A blend between two layers describes the effect one layer has on the other. In the chain of layers the result of a blend between two layers replaces the top most layer of these two. The content of this result layer is determined by the relationship between the two blending layers.

Blending as implemented in image editing software allows for a *mask* to be assigned to the modifying layer in a blend. A mask can be regarded as an additional layer that reveals parts of a layer and hides others. It is a monochrome layer (using just one colour dimension), ranging from minimum value black to maximum value white. This black-and-white mask serves as a multiplier of the layer to which it is applied. Black stands for 0, white for 1, with a grey area in-between. When an area of the mask is black, the area of the layer will be hidden (any colour value multiplied by 0 will result in 0); it is fully masked. A white area in the mask will reveal the layer content (multiplied by 1 it will stay the same); it is unmasked. Grey areas will partially hide or reveal layer content. Applied to a modifying layer in a blend (see *figure 31*), an unmasked area can fully modify the other layer, a partially masked area will only partially have effect, and a fully masked area will have no result at all, revealing the other layer unblended.

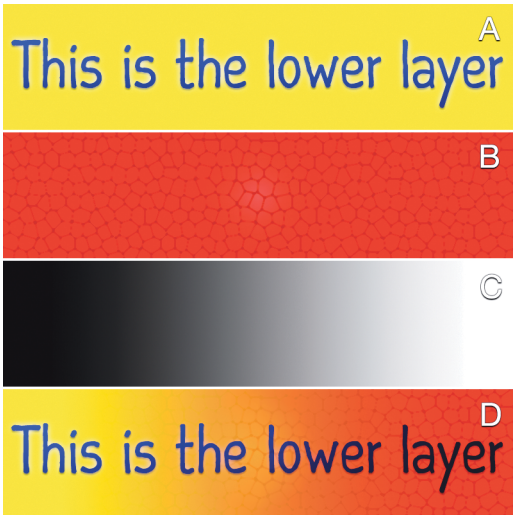


Figure 31. Layer B will be blended with layer A using multiplication. Mask C is applied to layer B before blending. The result of the blend is shown in D.

In audio processing, a similar paradigm can be found in the *dry/wet* mix parameter that is commonly available when applying an audio effect. This is a mix between the original audio and processed audio. Dry indicates only the original audio is heard, wet indicates only the processed audio is heard. When blending audio, a fully blended layer (unmasked) can be considered a completely wet mix, whereas dry would mean no blending takes place and just the content of the lower layer is heard.

Spectral blending

The filter bank used in vocoding constitutes a very rough spectral analysis. We use the spectral contour – that is the shape of the spectrum as dictated by the magnitudes of all spectrum bands – of one layer of audio and impose this on the other layer. Though a vocoder generally uses time domain envelope followers and amplitude modulators, in effect the spectral domain is the domain in which the blend is determined. A frequency analysis offers an envelope for every single frequency band, providing a richer basis for blending.

One interesting technique that can be achieved using spectral blending is spectral interpolation. If you want to gradually change one sound into the other, you need to interpolate between the audio data of both signals. In the time

domain, gradual interpolation based on the amplitude values constitutes what is known as a cross-fade; one sound is faded out, while the other is "faded" in. Far more interesting is interpolation between the time-changing spectra of the two sounds (Wishart 1994a, p.96). The spectral contour of one sound, while changing over time, is gradually moved towards the contour of the other sound, resulting in a morph[60] between these sounds.

Earlier we discussed ducking, a technique that can be used to make one sound give way for the other to be heard more easily. Ducking is based on loudness envelope following; for each timeframe it analyses the loudness of the sound in its totality. Spectral analysis can provide a more fine-grained approach to ducking, which might be called *spectral masking*[61]. Now each individual spectrum band of one sound can be attenuated (*ducked*) dependent on the loudness of the corresponding spectrum band of the other sound. If the sound that will be ducked has a broad spectrum and the other sound only has a narrow spectrum, only the parts that overlap will be attenuated, the rest of the spectrum will remain unchanged – time domain ducking will attenuate the sound in its entirety.

3.5.2 Special Pasting

Pasting is one of the fundamental editing actions, next to cutting and copying, to be found in any editing program, being it text, image, video or audio – even files on a file system can be cut, copied and pasted. These editing actions don't involve scissors or glue, they are often provided in multiple ways; through menus, key combinations and/or buttons. In order to cut or copy something – a word, a sentence, a specific region in an image or audio recording – the user first selects it, as described in the previous section. If it is then cut, it is effectively removed from the original and depending on the software this either leaves blank space or silence, or it pulls together the ends between which is cut. Copying keeps the original intact, it only copies the selected content, as one might expect.

Editing involves an imaginary *clipboard*. Whenever something is cut or copied, it or its copy is put on the clipboard, a temporary storage. Whatever ends up on the clipboard (generally wiping off anything that was on there before) can then be pasted. Here it becomes interesting, because there are various ways in which something can be pasted.

Figure 32 shows the *Special Pasting* dialog found in TwistedWave. It offers three ways of pasting: insert, mix or replace. Additionally both the document[62]

[60] The quality of this morph is strongly dependent on the perceptive similarity of the sounds and the time that is given to recognise these sounds, see (Wishart 1996, p. 101/2).

[61] In (Wishart 1994a, p.100), a technique called spectral masking is proposed, which for each frequency band takes the loudest amplitude of two sources. This is all-or-nothing approach can be considered as one variety of the approach discussed in this thesis.

[62] The *document* refers to the audio „"document"" which the user is editing. Any cuts, pastes and other edits such as effects are applied to this document.

and the clipboard content can be faded in and out, optionally using a crossfade. These pasting methods will be described in more detail shortly, but first an explanation for this digression into pasting audio. Earlier on we discussed non-destructive editing, as well as layered audio. Editing actions such as cutting, copying and pasting generally are destructive. The start and end points of a region that is cut or copied, are definitive, they cannot be tweaked after the edit is performed. A pasted segment is merged into the whole of the subject audio and cannot be discerned afterwards. That is, unless a modular editing approach as described earlier is used. Cutting and copying segments as well as pasting points can be altered at any time when they are regarded as module parameters. For audio pasting, audio layers might offer a non-destructive approach that allows for special pasting methods to be explored.

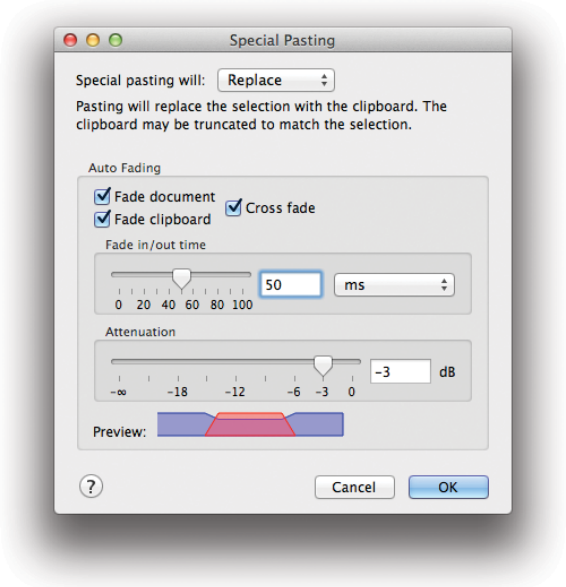


Figure 32. The Special Pasting dialog in TwistedWave.

Insert

The default approach, that is for text and audio, is to insert the clipboard content wherever an insertion point is specified, done in a similar fashion as to making a selection. Insertion is logical for text and audio as these can gradually be observed along a single dimension, from the beginning to the end. Inserting text makes the sentence longer, not taller. Inserting audio makes the duration longer, nothing else. Insertion on an image, something which has two dimensions and which content is observed as a whole, would make little sense. We can't make room for extra image content without seriously interfering with the original content.

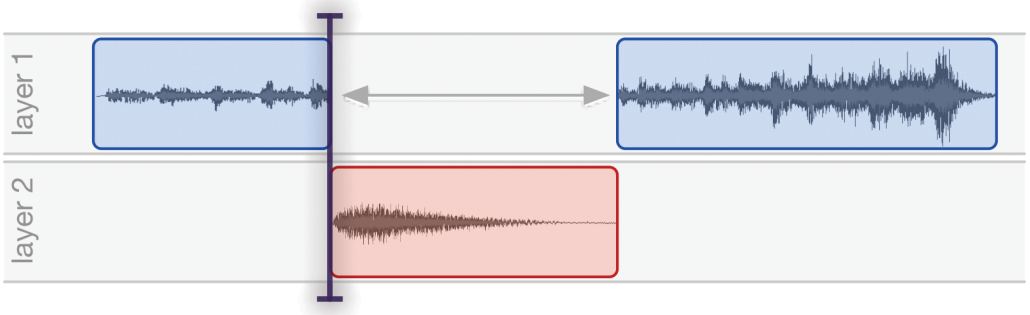


Figure 33. Inserting audio using layers.

Inserting audio by using layers will imply that the clipboard content is put on a secondary layer, below the primary audio layer on which the insertion takes place (see *figure 33*). This primary layer is split on the insertion point, the

first part is aligned tail to head with the secondary layer content, which in its turn is tail to head with the last part of the primary layer. This way a visual representation of the insertion action is made.

This visual representation can be instructive when specifying crossfades (*figure 34*). With audio, insertion can also interfere with the content, for example if a vocal recording is interrupted by an inserted bell sound. Here a smoother insertion can be realised by crossfading on the start and end point of the insertion, gradually leaving the vocal track for the bell sound and returning back.

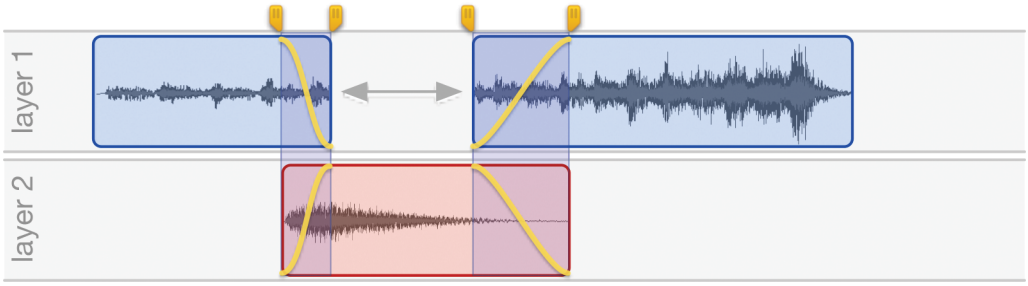


Figure 34. Inserting audio with a crossfade. The yellow line indicates gain adjustment.

Insert-pasting is particularly useful for reconstructing and reordering purposes. When for instance working with a voice or instrument recording, words or phrases can be inserted or moved by cutting away and inserting them at different points. This can be done more smoothly when using short crossfades.

Mix

Earlier I described the use of ducking for voice-overs, where music is attenuated when a voice-over starts. Special pasting using mixing is somewhat similar. Mixing in the clipboard content will position this content at the insertion point on a secondary layer, and will optionally attenuate the primary layer. Opposite to ducking, this method is more static, it does not perform any envelope following. This means that when the clipboard content is silent, it can still attenuate the primary layer content. Though less dynamic, the user now has more control over the attenuation, along with fading in and out (*figure 35*).

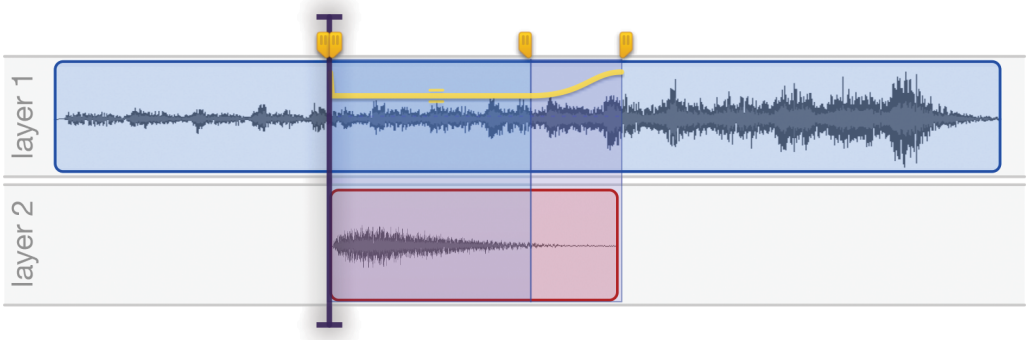


Figure 35. Mixing audio using layers. Layer 1 is slightly attenuated.

Replace

Pasting audio as replacement involves selecting a segment of audio that will be sacrificed in favour of the clipboard content. The clipboard content is trimmed to the length of the selection. This way of pasting is useful for instance if one has multiple takes of a guitar recording and wants to select a short bit from one take and paste it in another (final) take. By trimming the clipboard the overall timing of the primary audio is preserved. It might however be desirable to replace a short bit of audio with a considerably longer bit of audio. This in effect will be a combination of *replace* and *insert* pasting; the selection is replaced with the clipboard content, everything after that is shifted to make room for the remaining clipboard content to be inserted[63].

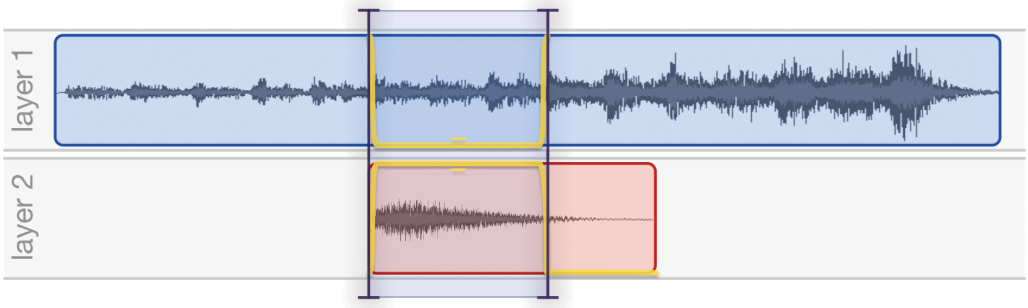


Figure 36. Replacing audio using layers. The gain of layer 1 is set to 0, while the gain of layer 2 is set to 1.

All these pasting methods can be approached using layers, as illustrated by figures 33 to 36. Using layers the pasting parameters become non-destructive; boundaries and insertion points can be shifted, attenuation levels and crossfades can be altered at any moment. Perhaps equally important is that the layer approach gives the user a clear image of what is actually happening when pasting in a specific way – what remains, what is replaced, and when does it start and end.

3.5.3 Summary

A sound can be constructed out of multiple layers. An audio layer can represent a single component of a sound. Layers can simply be added together, but can also influence each other. The act of combining two layers of audio in a specific way is called a blend. Blending can be as straightforward as adding two layers, but can be as unusual as masking one layer based on the spectral content of the other layer.

By analogy with image layers, audio layers are useful for visually representing audio compositing. Moreover, audio layers offer a non-destructive and highly flexible approach to a variety of audio processes such as ducking and de-essing, as well as special pasting edits.

[63] When the clipboard content is shorter than the selected region, the layer approach cannot be applied, as the primary layer has to be contracted. The only way to perform such a replace-paste is to first cut away the selected region – which can be done non-destructively using the modular approach – and then inserting the clipboard content at that point.

3.6 Sonic Composition

> *The digital domain permits not merely composing with sounds, but composing the sounds themselves.*
>
> *(Risset 1998)*

In delineating the scope of audio editing, we discussed *sonic composition*, which focusses not just on composing *with* sounds, but also on the composing *of* the sounds themselves. The elements with which the sonic composer works are sound aspects such as timbre, texture, and dynamic development. Outside the digital domain, only limited influence can be expressed on these parameters. Take for instance a note played on a wind instrument; its dynamics can be altered and, to a certain extent, its pitch. On some instruments a mute[64] can be used to apply some time-varying filtering. However, the instrument's nature will always stay the same, and through its fixed physical form and its possible ways of playing it imposes constraints on the sonic possibilities. Perhaps the most sonically versatile instrument is the human voice[65]. It can produce a wide variety of timbres with time-varying dynamics, as well as noise-like sounds. More interesting is that it can gradually change between tonal and noise sounds, or produce both simultaneously. Still there are limits imposed by physical form, lung capacity and muscle flexibility.

It has only been since the advent of the digital domain, that total control over sonic parameters can be achieved. Using a computer, sounds can be shaped into anything within reach of the composer's imagination. In the time domain, the amplitude curve can be manipulated with sample-accurate precision, and filter or effect parameters can be altered over time. Using techniques such as spectral analysis and re-synthesis, sound can be decomposed into frequency components, which can individually be altered.

This section focusses on two aspects of sonic composition. When shaping a sound, the sonic composer can specify time-varying parameter settings for a filter or effect, a technique called *parameter automation*. An approach to parameter automation will be discussed, in which the earlier described multiple and automated selections are integrated. Then, taking this approach a step further, the creation of a *sound morphology* is presented, with which a tree of sonically related sounds can be produced.

3.6.1 Parameter Automation

Parameter automation is a technique that is mainly found in multi-track audio editors and DAW's. These kinds of software allow the user to apply

[64] A mute is a device fitted to a musical instrument to alter the sound produced, by affecting the timbre, reducing the volume, or most commonly both.

[65] Trevor Wishart describes in detail the "human repertoir", the possibilities of the human voice, in his book *On Sonic Art* (1996, p.263).

effects to a single track. Then, using parameter automation, the settings of these effects can be changed over time. This generally works by defining an envelope curve. This curve runs parallel to the audio waveform, giving the user a visual guide to determine where to set breakpoints in the curve. Curves for different parameters can be overlapped, making it possible to easily synchronise changes in various parameters.

In an audio editing environment which does not use the notion of tracks and track-wide effects, automating parameters is far less common. In such an environment an effect is selected, specified and then applied destructively. The dialog in which the effect settings were specified is then closed, losing any reference to the applied effect with its specific settings. Parameter automation requires a non-destructive approach (and hence a persistent reference to the effect) to allow the automation curve to be adjusted. The modular, non-destructive approach to applying effects that was described earlier, provides a means to keeping reference to effects and parameter settings and can therefore provide a structure for parameter automation.

Whenever an effect is selected, it will be displayed as a module in the editing chain. Its parameters can be set to a specific value, but can also be selected for automation. A curve representing the parameter value over time is displayed below or on top of the waveform. An important point is that the waveform here displayed reflects the audio state on input of the effect. If the waveform reflects an earlier state, the timing or length of the audio might have been altered by a preceding effect.

Due to the offline nature of the modular approach, it requires all effects in the chain, or at least up to the effect the user is currently setting, to be rendered first before the parameter automation is audible. However, it also allows for parameter automation on offline effects. A simple example of what could be achieved this way is gradually increasing the playback speed[66] of an audio file.

Defining a curve

An envelope curve used for parameter automation is often a segmented line defined by a number of breakpoints. By default, values between points are derived by linear interpolation. Some audio editors, such as Adobe Audition, also offer a spline curve interpolation[67] for smoother transitions along breakpoints, which also is more appealing audibly.

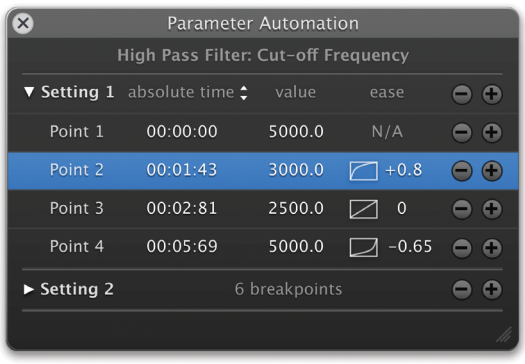


Figure 37. A head-up display for defining a parameter automation curve.

Defining a curve can be done more

[66] Known by an old-fashioned term, *Tape Speed Variation* is playing an audio file at a different rate, resulting in time-stretching and pitch-shifting.

[67] A spline curve is a curve constructed to pass smoothly through a given set of points. The values between the points are derived from this curve.

detailed when adding a non-graphical overview of the breakpoints, for instance listed in a head-up display (or HUD[68], see *figure 37*). Per point the exact value and time position can be inputted. By default, the parameter value will change linearly from one point to the following. Specifying an interpolation or smoothing function between a point and its preceding point will make the parameter value change smoothly.

For now this HUD will only add a level of precision in defining a parameter automation curve, but further on in this chapter the added value of the HUD will be made more clear.

Selections

When applying an effect to only a selected segment of audio, one of two approaches to parameter automation can be taken. Either the parameter value is deduced from only the selected time segment of a curve that spans the full length of the audio file (*figure 38-a*), or a curve is defined that will be applied relatively to the length of the selected segment (*figure 38-b*).

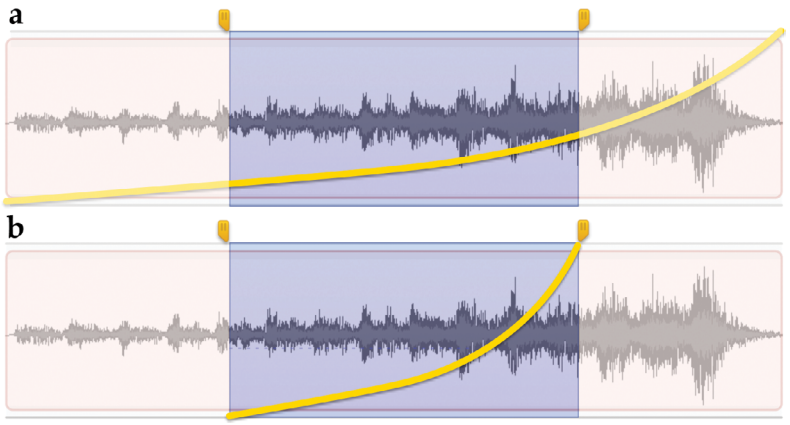


Figure 38. a) A curve that spans the full length of the audio file. b) The same curve but applied relatively to the selected segment.

Both approaches appear to have a downside. In the first case, defining a curve of which only a part will be used seems unnecessary. The parts of the curve outside of the selection are not used, so there is no reason for them to be defined. In the second case, positioning breakpoints graphically makes less sense because the points at which breakpoints will be effective will be different when applying the curve to a different (smaller) audio segment. However, the latter case is preferred, when we look at applying curves to multiple selections.

Applying an effect to multiple selections is comparable to applying an effect to multiple files: a batch edit. A simple scenario might be to fade in at the beginning and fade out at the end. How much time is taken for the fade might be fixed, say 0.5 seconds, but it might also be relative, such as 10% of the total duration. Using the HUD one can define for each setting if its position is fixed

[68] A head-up display is a transparent, commonly small display component in the graphical user interface. It can be moved around and miniaturized so that it interferes very little with the users gaze. It is used mainly for displaying additional information, but can also contain adjustable parameters.

or relative. A fade out at 10% before the end effectively starts at 90% of the total duration of the selected segment. Now, to all selected segments a fade in and fade out can be applied uniformly, without having to define a curve for each segment.

Segment Tables

Having multiple segments selected, another way of automating parameters can be introduced. Where the earlier described envelope curve would define a parameter value at a specific point in time, it can also be interesting being able to define a different parameter value for each individual selected segment. Enumerating segments on one axis and parameter value on the other, a curve can be defined passing through these segments, effectively resulting in a table with parameter values per segment index. This for instance makes it possible to apply a gain of 0 to the first selected segment, gradually increasing the gain over consecutive selections until a gain of 1 is applied to the last selection (*figure 39a*). The curve can be set to a fixed number of selections, but a relative curve as just described is more flexible in use, particularly when using automated selections, which can result in an unexpected amount of selections.

It becomes truly flexible when using a segment table to define an automated selection.

The table can now be used to specify an aspect of each selection that will be created, such as length or relative starting position, and for time-frequency selections the lower and upper frequencies (*figure 39b*).

These two uses of segment tables, for defining per-segment parameter values as well as automated varying segment selection, is particularly interesting regarding the aforementioned brassage technique (Wishart 1994a, p.60), which is based on cutting up, altering and repositioning audio segments. For example, an automated selection is made separating an audio file in a number of segments. Then – similar to increasing the playback speed gradually along the file – for each segment a different playback speed can defined, resulting in a stepped speeding up of the audio. Segment tables are also suitable for other sonic composition techniques described by Wishart (1994a, p.55), such as granular or waveset based transformation and repetition techniques. Grains are normally in the region of 10-100 milliseconds long (Roads 1996, p.168) and

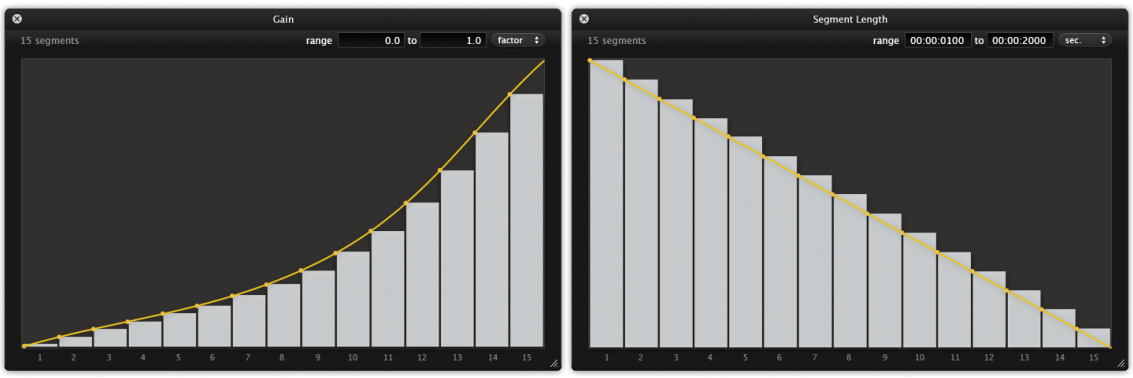


Figure 39. a) A segment table to gradually increase gain over consecutive selections. b) A segment table specifying automated selection segment length.

wavesets are only the distance from a zero-crossing to a third zero-crossing (Wishart 1994a, p.17). Using an automated selection, these short segments can be selected, on which then techniques such as incremental repetition or varying pitch shifting can be applied.

3.6.2 Branch Editing: Creating a Sound Morphology

The parameter automation HUD provides a detailed overview of the parameter value curve. For each automated parameter it lists all line segments of a single curve. From this user interface it is only a small step to introducing multiple settings per parameter. What this means is that for a single parameter, there can be a number of variations that will be applied. These multiple settings can be automation curves, but can just as well be fixed settings. Starting with a single input source, an effect can be applied a number of times, each time with different settings, resulting in multiple outputs, all having the same origin. The number of results can grow exponentially when defining multiple settings for multiple parameters. Take for instance a simple band pass filter, define 3 settings for centre frequency and 3 settings for band width. Now there will be 9 outcomes, because for each centre frequency each band width will be applied.

This technique might be called *"branch editing"*, as a variation on batch editing. With batch editing, you start with multiple input files and commonly end with that same amount of files, all processed equally. With branch editing, as with a tree, you start with a single source, and each parameter can be a point where the tree branches, resulting in multiple, possibly thousands of leaves or outcomes.

Branch editing can be useful in different situations. It is ideal for "gold digging", a top-down approach to finding the right sound. If a user has a vague idea of how a sound should be processed, they can apply some effects, specify variations on parameter settings of which they are uncertain, and render all possible variations. Doing this manually can be very exhausting and one can easily miss one combination of settings if it gets numerous.

It is well possible these variations will sound very different, but it can also be that the variations are ever so small. Obtaining all these variations can increase the chance of finding the perfect combination of settings, hence the term gold digging.

Another use for branch editing is found in composition. Branch editing facilitates the creation of a *sound morphology*, a tree of interconnected sounds that can be (but are not necessarily) perceptually related. This must be regarded as merely a generative method, not a compositional rationale (Wishart 2000, p.22). However it can ease the process of finding sounds with the right audible connection. Two sounds can have significant audible similarities without being generatively related, and the opposite is also true. When composing a morph from one sound into another, the best route needn't follow a single branch to create the most interesting result. A sound morphology can stimulate the composer in exploring unexpected directions in transforming a sound.

3.6.3 Summary

Parameter automation allows the sonic composer to apply time-varying transformations to audio. The common approach to parameter automation is by defining a curve along the time axis, so that a parameter value can be deduced from it for each point along this time axis.

Combining parameter automation with multiple selections allows for a curve to be applied relatively to each selected segment. Using a segment table, the parameter value can differ per selected segment. Automatically selecting very many very short segments on which such parameter automation can be applied facilitates interesting transformation techniques.

Parameter automation can also be extended to provide more than one value setting or curve for a parameter. When multiple settings are provided, for each setting the audio will be processed, producing a different output. Consecutive parameters with multiple settings will create a tree-like branching of editing variations, hence this technique can be called *branch editing*. The earlier described modular editing approach can facilitate this technique.

Branch editing is ideal for "gold digging", when looking for the perfect combination of effects settings. It is also useful when creating a convincing morph, requiring audibly related sounds.

The combination of parameter automation with more advanced selection methods and the modular editing approach can make the audio editor a powerful tool for sonic composition.

3.7 Extendability

> *The ways in which audio may be transformed are limited only by the imagination of the composer.*
>
> *(Wishart 1994a)*

This quote is true in theory. In practice, the ways in which a composer *can* transform audio are limited by the tools s/he uses. These tools are limited by the imagination and skills of the developer[69].

Most audio editing software is designed in such a way that it is not extendable. This means that the functionality provided to the user cannot be extended, other than by updating to a new version, if there is one. Users can make requests to the developer for new features, but there is no guarantee these will be implemented in a future release of the software. If a user is not entirely happy with an audio editor's functions, this leaves them either waiting for an update or looking for an alternative editor.

There are some audio editors that function as a so-called plug-in *host* for audio effect plug-ins. A plug-in is a means to dynamically add functionality to a piece of software (the host), either by the same developer or by a third party if the plug-in architecture is open. Audio plug-in architectures such as VST, Audio Unit and LADSPA have been widely acknowledged and there is a great variety of effects plug-ins available in these formats. Audacity also supports effects plug-ins written in the *Niquist* programming language, which can be developed in an ordinary text editor. For audio analyses there exists a plug-in system called VAMP, developed by Queen Mary University of London. Their Sonic Visualiser is built on this plug-in system, and Audacity also supports these kind of plug-ins.

Adding effect plug-ins is a means to extend the effects arsenal in an audio editor. VST, Audio Unit and LADSPA are however designed to process audio in real time and by this nature such effect plug-ins are inherently causal. An effect plug-in is applied offline by passing an audio file through it from start to end, much as a meat grinder works. Such an effect cannot inspect audio at a specific point in an audio file, it just receives something and outputs it processed. Processes and effects that require this inspection such as normalisation, reverse reverb[70] or speed variation are not possible through this architecture.

Offline effect plug-ins do exist. Apple's Audio Unit specification (Apple 2006) describes an offline type, though implementations of both hosts and effects are hard to find. Pro Tools[71] includes an offline variant of their Real Time AudioSuite (RTAS) plug-in format called the Offline AudioSuite. These plug-ins are mainly used for pre-processing to avoid heavy and repetitive real-time processing.

---

[69] Which can be the same person as the composer.

[70] Reversing a sound, applying a reverb and reversing the sound back again.

[71] Pro Tools is an industry standard Digital Audio Workstation by Avid.

Noise removal, compression and equalisation can be such processes that don't need to change over time and can be expensive to calculate in real time. The Offline AudioSuite also includes some processes that can only be calculated in non-real time, such as normalisation, reversing and time compression.

Where a realtime effect plug-in works relatively straightforward (such as a meat grinder), an offline audio effect plug-in is more complex. It must be able to inspect any point in the audio file, it must respect selections, and it requires a strategy for compression and expansion of selections (see *figure 40*). It depends on the possibilities of its host.

The design principles discussed in this thesis stretch the possibilities of the audio editor. In an audio editor that implements these principles, each edit module in the edit chain can exist as a plug-in. By allowing the user to develop new modules, such an audio editor is not limited to the developer's skill or imagination. The user can become the toolmaker for his own imagination as well as the imagination of other users.

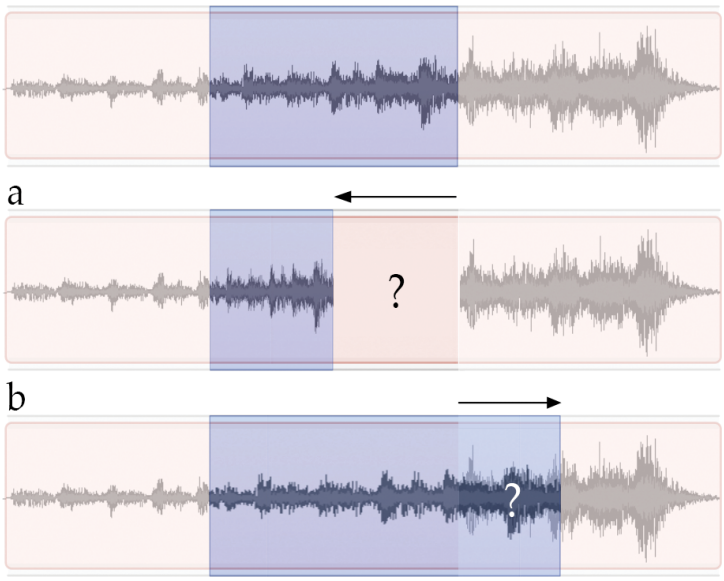


Figure 40. When time-compressing a selection (a) it can either leave a gap, or all subsequent audio can be pulled towards it to fill the gap. When time-expanding a selection (b) the extra audio that is generated can either be mixed with or replace the audio it will overlap in time, or it can push all subsequent audio further in time. The same problem occurs when applying a reverb that has a tail longer than the selection.

3.7.1 The Module Development Environment

In essence an edit module needn't be complex. Audio data, optionally with a specification of selections, is provided to be processed and then passed to the next module. Hence, the development of a module can also be relatively[72] simple.

Earlier mentioned plug-in architectures require a separate development environment and downloading of software development kits. Both generally have a steep learning curve. Once a plug-in is developed, testing happens in a different application, a plug-in host such as a Digital Audio Workstation. A developer can choose his own development environment and host application, but developing in this way can be elaborate.

By integrating a simple development environment within the audio editor, there is no need for complex all-purpose development software, and testing can be done instantly. *Figure 41* shows such a module development environment

[72] Software development and digital audio processing are inherently complex.

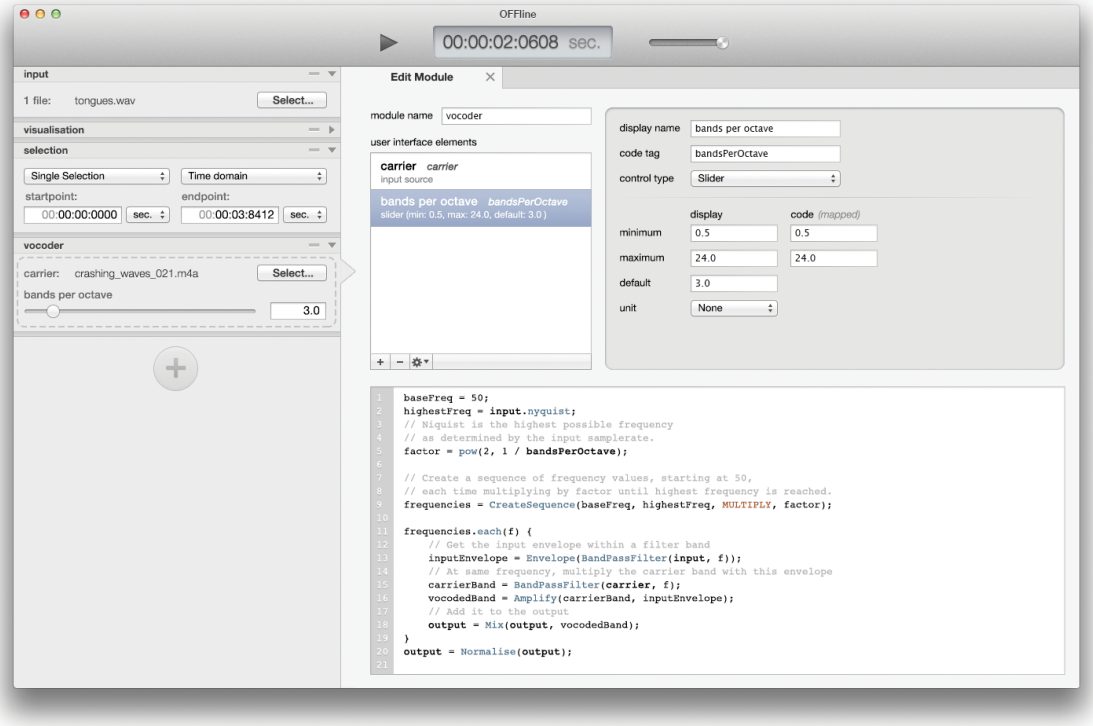


Figure 41. The module development environment in OFFline. A module can be created or edited through this interface. Parameters can be specified for the graphical user interface, and a code editor allows for programming the processing part of the module.

in OFFline (see chapter 1.4). Parameters can be specified for the graphical user interface, and a code editor allows for programming the processing part of the module.

An integrated audio module development environment serves more than extending the audio editor. It can be a didactic tool for educating audio processing. A technique can be implemented without worrying about file import and export, knowledge of development software or kits. This lowers the bar for students to try things out, and clears the view when a technique needs explaining[73]. For developers of audio software, it can also serve as a rapid prototyping environment. They can quickly try out a new audio processing concept before creating a full-fledged product.

Language

The audio processing part of a module needs to be described in some computer language. The language should be capable of the following:

- signal processing (generation and manipulation)
- analysis (for instance spectral analysis)
- visualisation (a module can produce a visual representation)
- understanding selections (time or time-frequency)
- receiving input from the graphical user interface

This does not mean the language should specifically include all of this

[73] See the section on *motivation* (chapter 1.1).

functionality, but at least should be versatile enough to facilitate it[74].

An investigation into existing audio related programming languages was conducted for this thesis. The full list of investigated languages can be found in appendix 7. Many of these languages were aimed at composition, control or sound synthesis, sometimes involving complex syntax for creating intricate compositional structures. Some were too basic, others such as SuperCollider were too sophisticated and extensive.

A language for composition might benefit from a special syntax. Through the language a specific compositional structure and order must be defined, the syntax – the structure of statements – can aid or even guide this. Signal processing does not require any special syntax. A declaration can be very similar to the mathematical formula describing the signal processing. A very simple low pass filter:

math: $y(n) = (x(n) + x(n\text{-}1)) \ / \ 2$

code:
```
output[n] = (input[n] + input[n-1]) / 2
```

This example displays signal processing at the level of the individual audio samples; input[n] stands for a single sample at time n. For some developers, signal processing at this low a level might be desirable, but then in every module code one has to explicitly consider aspects such as multichannel audio and selections. This unnecessarily complicates the language, particularly with respect to the aforementioned possible didactic purposes of such a language.

Trevor Wishart developed a number of techniques "using only editing, mixing and speed changing"[75]. Indeed, a great variety of audio processing techniques can be derived from a small set of procedures. A scripting language with some well chosen fundamental building blocks (*procedures*) such as amplification, mixing and speed changing, can be versatile enough to create numerous different edit modules. The intricacies of handling multiple channels and respecting selections can be implicitly dealt with by these procedures.

Procedures

An edit such as amplification is very straightforward and unambiguous. Therefore it can be notated as a simple procedure such as in the following statement:

```
output = Amplify(input, selection, factor);
```

Three arguments are taken into account: the audio that is being amplified (`input`); the `selection` of this audio on which the amplification should be applied; and the amplification `factor`. This one line can constitute the

[74] If a function is not included directly in a language, but the language does allow for the creation of such a function, than the language can facilitate this function.

[75] Also quoted at the beginning of chapter 1, (Wishart 1988, p.22).

amplification module since it does not need to do anything else. The factor parameter can be associated with a graphical user interface element such as a slider. If this parameter is automated (see chapter 3.6.1), the `Amplify` procedure will internally determine the time-varying factor to be applied to each audio sample.

Other procedures that are equally unambiguous are cut, insert (paste), add (mix) and delay. These can be regarded as equally fundamental as mathematical operations such as multiplication and addition. More procedures can be derived from these basic procedures. A fade is in essence a time-varying amplification. Normalisation is finding a maximum in a section of audio and then multiplying (amplifying) the entire section by the reciprocal of this maximum. However, because both fading and normalisation are common elements in more complex operations, these should also be available as a procedure.

A different reason to provide an operation as a procedure is because of its complexity. The aforementioned speed changing is easy to accomplish with analogue tape. In digital signal processing it is more complex to achieve[76]. Likewise, the Fourier Transform requires a series of complex mathematical operations. The user need not be bothered with implementing this, but should be able to readily use it as a procedure. This way, speed changing can easily be used to construct the *octave stacking* technique mentioned in chapter 1.1:

```
octaves.each(i) {
    speedFactor = pow(2, i);
    output = Mix(output, Speed(input, speedFactor));
}
output = Normalise(output);
```

'`Octaves`' is the parameter that defines which octaves to stack. We iterate over each octave, where `i` represents the current octave. The speed factor is calculated by taking 2 to the power[77] of `i`; if we want the input to sound three octaves higher (`i` = 3) the speed factor needs to be 8. Then the output is cumulatively formed of itself and the speed-changed input at the given speed factor[78]. The final mix can now be very loud, so we normalise it before we are done.

A spectral operation, which might necessitate a Fourier Transform or other complex transformation, can be done without exposing this complexity. In the following example, spectral frequency shaking is implemented:

```
inputSpectrum = Spectrum(input);
output = Shake(inputSpectrum.frequencies, shakeFactor);
```

`Spectrum()` performs a spectrum analysis on the input signal. The specifics

[76] Digital audio is sampled at a certain rate. Playing it back at a lower speed requires approximation of unsampled audio using interpolation methods.

[77] `pow` is an exponentiation function as available in languages such as C, where the first argument is the base and the second is the exponent.

[78] For sake of clarity, a selection parameter is omitted from this example.

of this analysis – such as window size, step size and windowing function – might be accessible through a general settings dialog. `Shake()` is a procedure that adds random offsets to any numerical data; in this case the frequencies of the input spectrum, but it can just as well be the amplitudes of this spectrum, a collection of formants or a set of delay times.

As the language is an integrated part of the audio editor, it should not distract the user from their main goal, editing audio. The language should be expressive enough to let the user extend and personalise the audio editor, allowing them to become a toolmaker. On the other hand it should be simple enough not to obfuscate how an edit operation is constructed. Hence a bit of "magic"[79] – taking care of some distracting or complex details – can be part of this language. In the above *shake* example, it is not visible that the spectrum is time-varying and that for each separate spectral analysis frame the frequencies should be shaken. The fact that Shake here performs an operation on spectral information but outputs a time domain signal is also not noticeable. These details are implicitly dealt with, giving clarity to the language.

One other such magical procedure can be `Impose`, which takes an analysis as argument and imposes it onto an audio signal. This analysis can be an amplitude envelope, effectively resulting in amplitude modulation, but can just as well be a spectral contour, resulting in vocoding (also mentioned in chapter 3.5.1). The procedure internally finds out what exactly to do, the user can just state what to impose on what.

When not just using `Impose`, vocoding requires a secondary input source and a user-defined number of filter bands per octave. Within each filter band, the envelope of the input source (the *modulator*) is applied to the secondary source (the *carrier*).

```
baseFreq = 50;
highestFreq = input.nyquist;
// Niquist is the highest possible frequency
// as determined by the input samplerate.
factor = pow(2, 1 / bandsPerOctave);

// Create a sequence of frequency values, starting at 50,
// each time multiplying by factor until highest frequency is reached.
frequencies = CreateSequence(baseFreq, highestFreq, MULTIPLY, factor);

frequencies.each(f) {
    // Get the input envelope within a filter band
    inputEnvelope = Envelope(BandPassFilter(input, f));
    // At same frequency, multiply the carrier band with this envelope
    carrierBand = BandPassFilter(carrier, f);
    vocodedBand = Amplify(carrierBand, inputEnvelope);
    // Add it to the output
    output = Mix(output, vocodedBand);
}
output = Normalise(output);
```

[79] "Magic" as described in chapter 2.2.

This example demonstrates the creation of numerical sequences, in the form of `CreateSequence()`. It takes an initial value and a limit, as well as a calculation mode and a constant to use in this calculation.

The two input parameters are `bandsPerOctave`, controlling the number of filter bands, and `carrier`, the secondary input source. Note also that in this example, `Amplify()` does not take a single amplification factor as argument, but a time-varying envelope curve. Again for clarity, details such as selection and filter parameters such as Q and gain were omitted.

As an example of the combination of this language with the graphical user interface, brassage, a technique discussed in chapter 3.4.2, assumes multiple (selected) segments of audio to be cut and reordered. The selection process is done through the graphical user interface of the audio editor. There, parameters such as length, spacing and order influence the brassage character. With the selections defined, the brassage edit module can be described as:

```
selection.each(s) {
    output = Append(input, s, output);
}
```

For each selection as `s`, that segment of input audio is cumulatively appended to the output.

Brassage is an acausal operation; a single output value can depend on any input value, from the start to the end of the input source. Offline editing makes such acausal operations possible (see also chapter 2.1.1.2). Reversing a source is the most basic of acausal operations, where the first output value equals the last input value and vice versa. Acausal operations can also be exotic, like *preverberation*, a variation on reverb where a sound "reverberates" before it is heard. It can be implemented as follows:

```
output = Convolve(input, impulseResponse, -1.0, ADD);
```

Reverberation can be implemented using convolution[80], by using an *impulse response* that in essence is a recording of the reverberation of an *impulse*, a sharp tick. Convolving this with the input will give the input the same reverberation. Note in this example the –1.0, which represents a negative delay; the impulse response is offset backward in time relative to the input (see also *figure 43*), effectively resulting in preverberation.

A more practical example of an acausal operation is a look-ahead compressor[81]. Real-time compression faces the compromise between reacting smoothly but also sufficiently to sudden peaks in volume. A look-ahead compressor can identify these peaks on forehand and gradually adjust the compression rate in time.

[80] A good treatise on convolution is Smith (1997, p.107). Convolution reverb is discussed in Zölzer (2011, p.184).

[81] Compression is discussed in chapter 3.5.1.

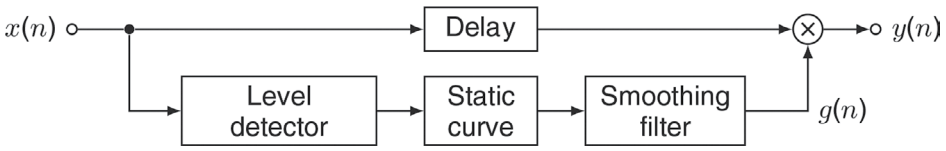


Figure 42. Compression is based on level detection (an envelope follower), a static curve to derive a gain factor from the result of the envelope follower, a smoothing filter to prevent too abrupt gain changes and a multiplier to weight the input signal (Zölzer 2008, p.213; 2011, p.100).

Following the basic compressor schematics illustrated in *figure 42*, a look-ahead compressor can be implemented as:

```
// 1. Create a smoothing function: (length 50 values)
//    f(x) = (1-|x|)^s    for -1 <= x <= 1  where s is smoothingFactor
functionSequence = CreateSequence(-1, 1, ADD, 2/50);
smoothingFunction = pow(1 - abs(functionSequence), smoothingFactor);

// 2. Determine peaks in envelope
envelope = Envelope(input);
peaks = Threshold(envelope, threshold);

// 3. Create smoothed curve using convolution (fig. 45c)
smoothedPeaks = Convolve(peaks, smoothingFunction,
   -(smoothingFunction.length / 2), MAX);

// 4. Calculate gain factor
gainFactor = (threshold + (ratio * (1 - threshold))) / smoothedPeaks;
output = Amplify(input, gainFactor);
```

Three parameter settings are provided by the user: a `smoothing factor` which determines how fast the compressor responds to a peak as well as how fast it turns back to normal level, a `threshold` above which the compressor should compress and a `ratio` that determines how strong it should adjust the gain (a ratio close to 0 will clip everything above the threshold to that value).

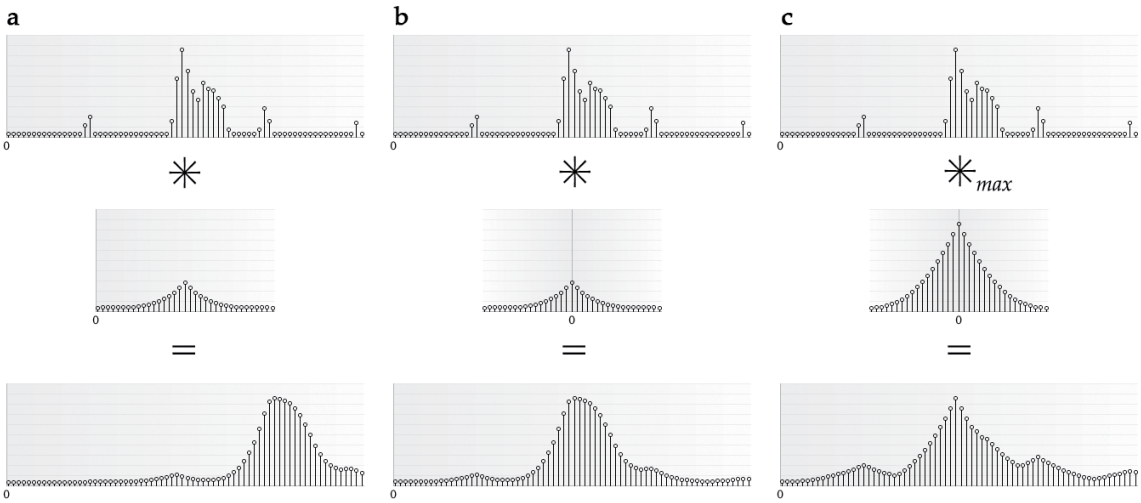


Figure 43. Three modes of convolution. a) shows regular convolution in which every value of the input (top) is multiplied with every value of the impulse response (middle). b) shifts the impulse response relative to the input sample, allowing the convoluted samples to occur before the input sample. c) doesn't add all convoluted samples but returns only the maximum convoluted sample values.

A smoothing function serves as impulse response. It is convolved with the peaks, in which the offset is negative half the impulse response's length. It uses a `MAX` mode, in which for each audio sample the convoluted samples aren't added together as usual, but only the highest sample value is returned (see Smith (1997, p.107) for details on convolution). This results in a curve which is less smooth but still has enough attack and decay (*figure 43c*).

It is also important to note the implicit calculation power resulting in the time-varying gain factor. It does not matter that `smoothedPeaks` is a time-varying signal and `threshold` and `ratio` can be either a single value or varying over time (using parameter automation). Under the hood the right values for each instance in time are determined to use in the calculation.

One final example illustrates the use of analysis methods (other than envelope following) in the language. It is a rudimentary *auto-tune*, correcting off-key pitches in music using transposition.

```
// calculate equal tempered scale note frequencies (Loy 2007):
noteOffsets = CreateSequence(-50, 50, ADD, 1);
frequencies = 440 * pow(pow(2, 1/12), noteOffsets);
// obtain all "voiced" segments, i.e. not containing noise.
voicedSegments = Voiced(input);
voicedSegments.each(s) {
    fundamental = FundamentalFrequency(s.input);
    tuned = Quantize(fundamental, frequencies);
    ratio = tuned / fundamental;
    s.output = Transpose(s, ratio);
}
```

> Analysing the input for voiced segments (any part not containing noise that can be described as a sum of sinusoids), for each of those segments the (possibly time-varying) fundamental frequency is determined. This frequency is then quantised to the pre-calculated set of note frequencies in the equal tempered scale. The ratio between the original and quantised frequency is used to transpose the input so that its fundamental frequency now is on this scale.

These few examples illustrate a language suitable for swift development of audio editing operations. Already a great variety of techniques can be developed with building blocks such as described here.

Expressiveness

The procedures form the vocabulary of the language, and hence its expressiveness. A number of existing languages, among which SuperCollider and Pure Data, allow developers to extend this vocabulary by adding and modifying procedures. The available procedures are stored in and loaded from a single location. The implementation of such a procedure is often written in a lower level language that allows the specification of the aforementioned sample-accurate multichannel audio processing and processing of selections. This approach to language implementation effectively creates three levels of use:

- At the highest level, a user only sees an edit module in the audio editor, that performs a single specific audio editing task. This user is the ordinary audio editor user.
- One level lower, still in the audio editor, a user can see and modify the workings of an edit module, using a language of procedures that allows them to focus on the characteristic elements of this editing task. New edit modules can be developed at this level.
- At the lowest level, a developer can extend the vocabulary of procedures by developing their own. This development is not done inside the audio editor itself.

There is a clear separation between these levels, each satisfying a different kind of user. If you don't want to know what happens under the hood of the audio editor, you are not bothered with it. If on the other hand you want to extend the audio editor's range of editing modules, it can be done relatively easily. If the language with which you extend the audio editor isn't expressive enough, it can also be extended, though this is far more complex. This approach allows for the audio editor's editing capabilities to be extended indefinitely without having to modify the actual code of the audio editor itself.

3.7.2 Summary

The limits to what an audio editor can do restrict the creative process of its user. An audio editor need not be limited to the set of editing operations with which it is distributed. Extending the audio editor with an environment in which its editing capabilities can be extended, its editing possibilities can become virtually limitless. The practicality of such a development environment is determined by the language on which it is based. A simple grammar and a well chosen vocabulary, combined with some implicit processing, can yield a clear yet powerful language suitable for quick development of new editing operations. Such a development environment can also serve didactic purposes as well as provide a means for rapid prototyping.

4. Narrative – Part 2

The modular editing approach was the first design principle that was developed. Its initial incarnation originates from 2004, at that point merely reflecting real-time audio effect plug-in architectures in the way effects can be chained non-destructively. As such, similar implementations can be found in AudiofileEngineering's Sample Manager and Apple's Soundtrack Pro.

Extending the design by including the input audio file as a module, enabled it to be used as an effect stack and for batch editing, a feature desired by multiple user groups (as identified from the survey). Other novel aspects presented here are module representations of the output audio file and of selections.

The Source Browser is a unified approach to obtaining an audio source for editing. It combines audio file selection, audio recording and audio synthesis in one user interface element, instead of accessing them in disparate ways. As such it is unique, as it is not found in any audio editor to date.

The audio "find" feature originates from a personal desire to be able to find audio files in a more intuitive way than through filename search. It can also be regarded as an approach to "audio file management", an audio editor feature preferred by producers and audio engineers (see appendix 1.2.1).

The deliberation on visual audio representations is one that any developer of audio software faces at some point. The waveform is relatively easy to implement, but is it the most useful? Its most common counterpart, the spectrogram, is more difficult to implement, but for this as well can be asked how useful it is. A varied series of test signals was used to identify strengths and weaknesses of both waveform and spectrogram. It appeared that a combination of the two gives the most complete insight in the audio; implementing only a waveform or spectrum view would leave out a lot of info.

It was also known from professional experience that there is no single approach to drawing a waveform or a spectrogram. Many audio editors currently available have slightly different drawing implementations. This aspect was also considered in this thesis, because a developer should not only carefully consider which visual representation(s) to include, but also how best to present them.

The comparison of audio editors revealed several that offered audio editing through manipulating the waveform display. In most cases this encompassed only gain change, but Apple's Logic also included a feature they called *Flex Time*, which can be regarded as a form of time stretching. This encouraged the deliberation on finding other edits that could be made accessible through intuitive waveform display manipulations. Similarly, audio editors such as Adobe Audition, iZotope RX and AudioSculpt offered spectral editing tools

that allowed direct manipulation of selected audio content, but it wasn't possible to transform the audio by transforming the selection. Applying such transformation concepts from image editing to the spectrogram revealed how a variety of audio effects would become more accessible.

Selecting an area for editing is an essential aspect of the audio editing process. Compared to the selection possibilities in image editing software, selection in audio editing software is quite unsophisticated; it is a single, one-dimensional (time axis only) selection, only in some cases guided by intelligent snapping (to zero-crossings, beats, or other points of interest). Again inspired by image editing software, the first logical step toward improving the audio selection seemed to be the introduction of multiple selections. This was already implemented by TwistedWave as a means automatically to select and then edit all silences in an audio file. The concept was then extended further to allow selection based on characteristics other than silence, such as noisiness or recording artifacts, and to select automatically audio regions similar to a preselected one. The automated selection of very many very short regions facilitates some more creative audio reordering techniques described by Trevor Wishart (in *Audible Design*, 1994a).

The concept of layered audio was formed partially by insights from image editing software, and partially by the deliberation on what an audio track represents in an audio editor as compered to a track in a Digital Audio Workstation. As it appeared from the audio editor comparison, Audiofile Engineering already included a very similar concept called Layers in their WaveEditor. It allows layers of audio to be blended together using a specific blend mode.

Layered audio can also be regarded as a means to provide better insight in certain audio edits involving multiple sources of audio. With this in mind, the blend modes could be extended to include various effects that use a so-called side-chain. Taking this further, spectral blending between two sources can also be investigated using audio layers.

A constant throughout this research has been the desire to design an audio editor that allows for more creative and unusual effects – particularly audio transformation techniques described in *Audible Design* by Trevor Wishart (1994a). One recurring aspect in many of these techniques was that parameters could change over time, gradually changing for instance the amount of spectral morphing between to audio sources. A concept called parameter automation (defining such a time-varying curve for a parameter) was already present in most Digital Audio Workstations. As discussed in chapter 3.6.1, translating parameter automation to audio editors posed a challenge to make it work intuitively with selections. After introducing multiple selections and trying to combine this concept with parameter automation, the idea for segment tables was conceived, a concept that allows the user to define different parameter

settings for each selected region.

Looking at combining parameter automation with other design principles, the combination with the modular editing approach then gave rise to the concept of branch editing. This is a novel approach to audio editing with which different combinations of parameter settings can easily be explored.

As became apparent from the survey, people do want to be able to personalise and extend an audio editor. An investigation towards a programming language or environment with which audio editors could be extended was conducted early in this MPhil trajectory as a possible research subject. It was discontinued in favour of the design principles, but could still be incorporated in an extended project which would focus on finding a solution for audio editor extendability. With the modular editing approach, each single edit is represented by a module through which its settings can be accessed and altered. Integrating a development environment for such a module would allow the user to focus on developing a single effect or other editing task, without having to worry about audio input or output technicalities (the details of which are described in chapter 3.7).

The design principles were not developed in a strict sequential order, they rather evolved simultaneously. Parameter automation and branch editing are not possible without the use of the nondestructive modular editing approach. Batch editing is also facilitated by that approach, and the batch of files can be defined through the Source Browser. The possibility to make multiple selections (optionally automated using segment tables) allows for very interesting sonic composition techniques, which can easily be programmed using the module development environment described in chapter 3.7.1. This synergy demonstrates the value of combining design principles, rather than implementing just one of them. It echoes the initial theme in the discussion that instead of merely *including* a feature it is better to *integrate* it; good integration might require more fundamental changes.

5. Recent Developments in Audio Editing Software

Software development is an ongoing process. It takes place in many different areas, and little of which finds its way into commercially available software. This study has focused on commercially available software because a wider focus is beyond the scope of this thesis.

As mentioned in chapter 1.3 (*"Relevance"*), during the writing of this thesis, a number of design principles have become available in new audio editors. This section focuses on these developments, as well as on other developments in audio editing software that have a significant influence on how people edit audio.

"A Simpler Interface"

One interesting development is the discontinuation of Audiofile Engineering's Wave Editor. A screenshot of this audio editor was used in chapter 1 to illustrate the cluttered user interface resultant of adding features in separate windows. Audiofile Engineering replaced Wave Editor with Triumph, and audio editor and post-production tool with, to cite them literally, "a simpler interface, which focuses on content".

Mobile Audio Apps

The mobile app market has grown immensely over the past few years, not just in size but also in quality. Mobile devices are becoming increasingly powerful and the user by now is well accustomed to those concepts that differentiate mobile software from desktop software, particularly in regards to user interface.

The specific architecture of apps and the platform on which they run – an app is an isolated entity on the device, it has no awareness of other apps and hence cannot communicate with them – for years meant that one couldn't really use professional audio apps the way one would use desktop audio software. Apps could not send or receive audio to/from other apps, making a DAW app practically useless, unless it included its own synthesisers and effects. On Apple's iOS this problem was lifted in 2012 by a platform called Audiobus, soon to be followed by a similar native solution called Inter-App Audio. With these technologies – which allow audio apps to be chained so that they can send and receive audio to and from other apps – an audio app could focus on just one thing, such as synthesis, processing or recording. This development

has initiated a quick growth in new professional audio apps. Though this market is still relatively young and users have only just started to integrate mobile devices in their audio setup, such development seems to be continuing rapidly. This doesn't stop developers from creating new and innovative touch interfaces for audio apps. These interfaces often make use of multi-touch gestures, and when the mapping between such a gesture and what it controls is well designed, such an interface can be controlled more intuitively than through mouse and keyboard operations. Mapping and guidance (see also chapter 2.3) are of great importance in the design of a touch interface, because a finger is less precise than a mouse. For this same reason, app interfaces are often much cleaner and uncluttered than desktop apps.

There is much debate about the position of mobile devices (smartphones and tablets) in a professional music environment. Though it is now possible to exchange audio between apps, a mobile device does have considerably less hardware connectivity options than most desktop computers. Hardware audio interfaces generally still require wires, but increasingly more hardware controllers can be used wirelessly. As for processing power, many recent mobile devices can match desktop computers from only a few years earlier. For both mobile and desktop software, ease of use depends on the users and the context in which they use audio software. An abundance of (often tiny) controls in desktop software doesn't necessarily make it easy to use. Mobile audio software is of course ideal in mobile audio setups.

Audio editing apps are relatively rare. As there is generally no accessible file-system on mobile platforms, one cannot simply select a file to edit. From early on there has been a focus on voice recording/editing and ringtone editing. Thanks to cloud/sharing services such as Dropbox it has become easier to load and save audio files across devices (both mobile and desktop), and with technologies such as AudioCopy & AudioPaste by Sonoma Wire Works it has become possible to exchange audio between compatible apps.

The desktop audio editor TwistedWave is also available on iOS, and together with Hokusai by Wooji Juice Ltd and Pocket Wavepad HD by NCH Software, these can be regarded as the more feature-rich audio editors. These apps offer the essential audio editing features such as audio recording, audio import and export, a zoomable waveform display, cut/paste actions, an undo/redo mechanism, as well as a wide variety of audio effects. Apps such as Monle (Ochen K.), Audioforge and Reforge (Audioforge Labs Inc.) are more basic editors with a smaller feature set.

Layers

In chapter 3.5 audio layers are introduced as a different way to look at and combine audio tracks. At the time of writing that chapter, only one audio editor, the earlier mentioned Wave Editor by Audiofile Engineering, included such a concept. Both Wave Editor and its successor Triumph offer *add*, *subtract* (add out-of-phase) and *ring* as "layer combination types" (blend modes), and

each layer can be assigned its own effects and fades. Masking the blend over time is however not possible. Audiofile Engineering calls this their "Layers technology", and since recently holds a patent on this technology.

Another recent audio editor called Fluctus also uses the term "layer", but here it is merely a separate layer in which the amplitude of the audio file layer can be edited non-destructively (by applying functions for fades, mute, pass).

Complex Analyses

As long as Moore's law remains in force, commercially available processing power will continue increasing. In recent years there have been introduced a number of software applications that, through very complex and particularly computationally heavy algorithms, enable to user to perform operations that seem magical. As mentioned above in chapter 2.1.1, Melodyne introduced Direct Note Access in late 2009. This technique, also known as polyphonic pitch correction, was long regarded as a 'holy grail' in digital audio processing. In more recent years, pitch correction algorithms (mainly monophonic) have been adopted by an increasing number of audio applications, particularly DAW's. Polyphonic pitch correction has not been much implemented by other developers, apart from a company called Zynaptiq, creator of a range of audio plug-ins that seem to include multiple audio processing techniques that until recently were regarded nigh on impossible. Zynaptic's plug-ins include real-time polyphonic pitch correction, de-filtering and de-reverberation. The latter technique can by now also be found in the *Advanced* version of iZotope RX 4, the audio repair and enhancement software.

Thanks to the increase in available processing power, it has become feasible to include complex analysis algorithms in commercial audio editing software that enable the user to take control over more naturally understandable aspects of audio such as pitch, filtering and reverberation, instead of just samples and frequency bands.

6. Conclusions

This project aims at redefining the audio editor by introducing new design principles. Through these design principles coherency, flexibility and creativity in the audio editing process can be improved. Furthermore, this thesis proposes the construction of an audio editor according to these principles, in which OFFline, the concept audio editor described in chapter 1, can serve as a conceptual example.

Briefly summarising the discussed design principles, it becomes apparent how these design principles can interact.

Modular Editing is an approach to represent consecutive applied edits. It is non-destructive, allowing any previously applied edit to be altered, removed or repositioned in the chain of edits. Representing each editing action as a module also allows adjustment of selections, input source(s) and output configuration(s). It also facilitates batch editing (editing multiple files uniformly), parameter automation (see chapter 3.6.1) and branch editing (see chapter 3.6.2).

The Source Browser unifies the various ways in which the audio source to be edited is obtained. Whether this is a file on disk, a recording to be made or a signal to be synthesised, ultimately it is an audio file that will be edited. The Source Browser provides detailed information on selected files on disk, as well as a means to find specific audio files based on audible characteristics rather than by name. It also allows to create a batch of sources (files, recordings, syntheses) that can then be uniformly edited using the modular editing approach.

Representation and Manipulation are united in a single key element within the graphical user interface of an audio editor. The choice of representation type influences the visual cues offered to the user that tell them where to apply an edit and what the effect is. It determines the ways in which a specific segment of audio can be demarcated through selection, as making such a selection is often carried out on this visual representation. It also determines the ways in which audio can be manipulated through this visual representation.

The waveform and spectrum display – the most common visual representations of audio – need to be displayed together to provide sufficient visual cues needed to precisely inspect and edit a variety of audio signals. For both types of representation, a number of intuitive manipulations are proposed that in some cases (spectral delay, time-varying pitch shifting) considerably simplify the application of such an edit.

Smart Selections extend the standard single time or time/frequency selection by introducing multiple selections, as well as ways in which these multiple selections can be made. Defining multiple selections allows audible elements such as beats, onsets or noises to be edited quickly and uniformly. It also facilitates sonic composition techniques that involve cutting, reordering and repeating multiple audio segments. Selections can be made manually, but can also be made automatically, either by specifying selection criteria or by selecting a single segment based on which similar segments are sought.

Layered Audio is a visual approach to combining multiple layers of audio. It allows for defining a relationship between two layers (a *blend*), which can be anything from mixing to modulating. It provides a transparent and uniform graphical interface for a great variety of effects in which one audio source influences another. Through this same interface, fundamental edits such as cutting and particularly pasting become obvious and can easily be adjusted.

Sonic Composition, the composing of sounds, is what I aspired to do using an audio editor. Within the context of the audio editor, sonic composition is stimulated through parameter automation and *branch editing*.

Parameter automation is made possible in the audio editor using the non-destructive modular approach. It allows for parameters to be varied in time by defining a curve along the time axis. In the offline environment of an audio editor, acausal operations such as tape speed variation now can also be precisely controlled. Parameter automation can be applied to multiple selections by defining a relative curve that is applied to each selected segment. Using segment tables, parameter automation can also be used to define a different parameter value for each individual selected segment.

Branch editing builds further on the combination of the modular approach and parameter automation, allowing one edit module to apply multiple variations of its edit by defining multiple settings for a parameter. Each consecutive edit module can multiply the number of outputs by also defining multiple parameter settings. A single input source can thus result in a multitude of related output sources. This is an ideal approach for finding just the right combination of parameter settings, and for finding sounds with the right audible connection.

Extendability of the audio editor is provided through an integrated module development environment, in which the user can inspect and modify existing edit modules or create their own. Modules can be developed using a relatively simple yet powerful scripting language. This language implicitly handles details such as data types, multichannel audio, parameter automation and dealing with (multiple) selections, hence clearing the view for the user to focus on the edit process itself. By integrating this development environment directly in the audio editor, it also becomes suitable for didactic purposes and rapid prototyping of audio effects.

Each design principle on its own unites a number of existing features that in existing audio editors are generally found disjointed. Attention to intuitive user interfacing makes these design principles highly flexible in use by allowing a variety of edits through a single principle, unveiling more creative and novel methods of audio processing that are as yet uncommon to audio editors.

Taken together, these design principles provide a synergetic approach to audio editing in which one principle builds on the other. This demonstrates how pervasive the contribution of some of these design principles can be. As noted before, by now some elements of these design principles have been implemented in existing audio editors, but it is only through combining these design principles that the audio editor is profoundly redesigned.

A redesign however doesn't necessarily redefine the audio editor. Redefining the audio editor means to define it again or differently. In this thesis this is carried out by observing how various elements are implemented, and then extending on existing approaches as well as introducing different approaches to these elements. Through the presented design principles, the audio editor is still a software with which one can edit audio. Characteristic features to such software – file loading and recording, graphical display of audio, selecting, cutting and pasting audio and audio processing – are all still available, yet the principles through which these features are offered differ. The audio editor is now more coherent, as fundamental aspects in the workflow have been rethought and redesigned. Through the same design principles, the audio editor is also extended to allow for more creative editing processes without requiring a design tailored to these processes. In other words, the audio editor is still what one may expect, but it is no longer the same; it has been redefined.

An actual implementation of the design principles has not yet been developed. Hence, though theoretically improving coherency, flexibility and creativity, it cannot yet be said if the proposed design principles indeed make the audio editor *better* as long as no one has actually experienced them in practice. However, through the various real-life scenarios discussed in this thesis it becomes apparent that many edits can be made easier, and in turn easier can be considered as better. Moreover, given that some elements of what has been discussed are by now implemented in existing software, this may also prove their practicability. Observing the preferences for the various user groups in appendix 1.2.1, the design principles do provide in the market's needs.

This thesis addresses just an instant of a continuously developing subject. Though for decades indeed little has changed in the design and development of audio editing software, the recent developments in mobile software cannot be unaddressed. Mobile devices more and more often replace laptops and desktop computers in audio setups. Though this is especially true in live performance sets, in which the audio editor is seldom used, it is also true for mobile audio recording solutions. In the production studio, in which the audio editor is

most often used, it is not as necessary to go mobile, so it will be unlikely that the audio editor ceases to exist in a desktop format any time soon. That said, the innovations in mobile and touch software give rise to increasingly smarter and easier (more accessible) software applications. This makes it worthwhile to re-evaluate the status quo and perhaps again redefine the audio editor at some point in the next decade. For that, there now lies a foundation in the form of this thesis.

OFFline

The Redefined Audio Editor

– a concept –

OFFline

As artefact of this research, a concept audio editor called OFFline is presented that illustrates a possible implementation of the discussed design principles. Throughout this thesis, mock-up screenshots of OFFline have been presented to exemplify aspects of these design principles. In this section, a more comprehensive overview of OFFline is provided.

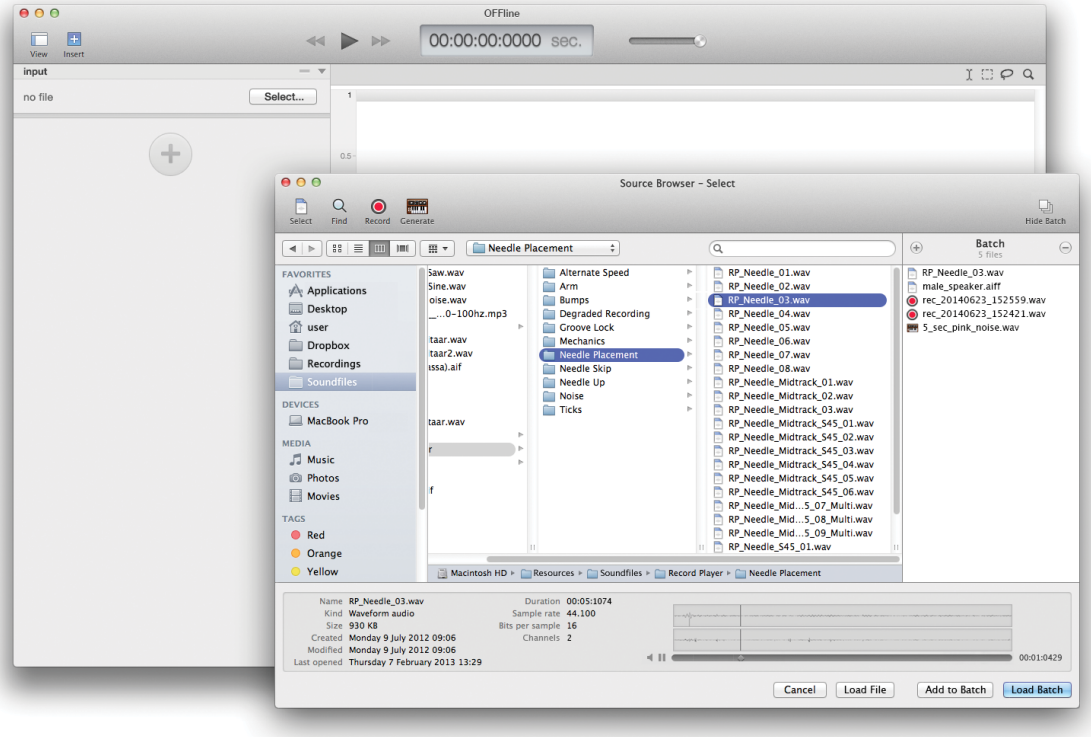


Figure 44. OFFline at launch. The module chain is empty except for an input module. The Source Browser is displayed to allow selecting one or more sources to load through this input module.

A source

Any audio edited with OFFline is regarded as a source, whether it is an audio file on disk, a generated signal or a recording. When opening OFFline, it immediately displays the *Source Browser* (chapter 3.2), with which one or multiple sources can be selected (*figure 44*). When selecting multiple sources, this creates a *batch* of which all files will be edited uniformly.

The Source Browser provides default file selection as well as a sophisticated finder, allowing the user to find files on disk based not on filename but on audible properties, such as duration, envelope, spectral characteristics and noisiness (*figure 45a*, also chapter 3.2.4).

Recording audio also takes place in the Source Browser, as this also leads to a source that will be edited. The specification of input and output is provided as

well as input VU meters and an output waveform (*figure 45b*, also chapter 3.2.2).

The Source Browser also includes signal generators for creating test tones, noise, chirps (or sweeps) and other basic signals (*figure 45c*, also chapter 3.2.3).

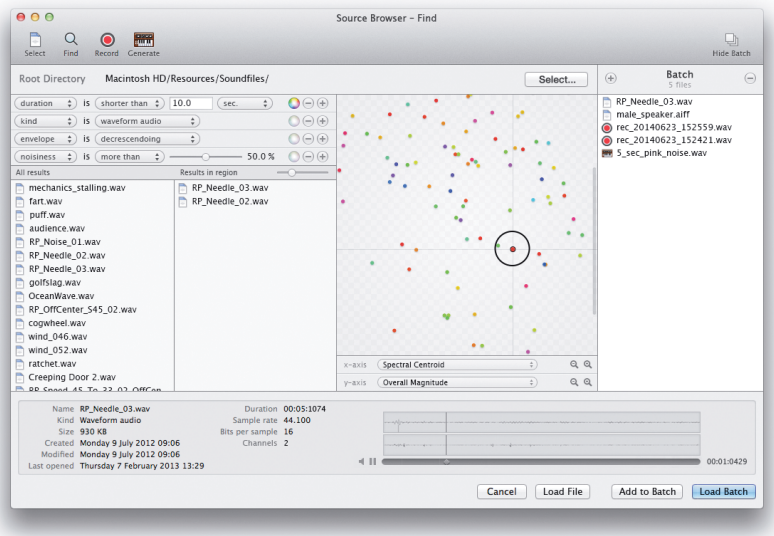
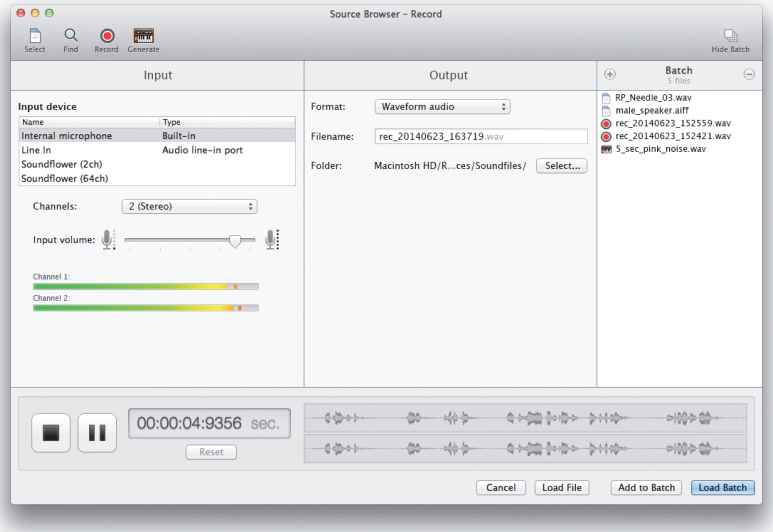
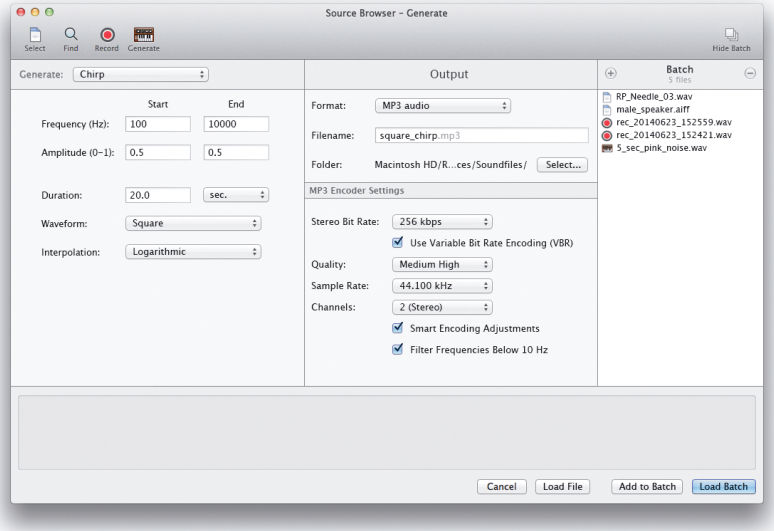


Figure 45. Different modes in the Source Browser: a) Finding an audible properties.

b) Recording an audio file.





c) Generating an audio file using signal generators.

The module chain

As is visible in *figure 46*, a single input module is present by default. This module can represent a single input source as well as a batch of multiple sources. When a source is specified, a second module, *visualisation*, is added that renders both a waveform and spectrum in the righthand side of the window (*figure 48*).

There are now a number of ways to add new edit modules to the chain. Selecting audio and then copying, cutting or pasting it are fundamental actions that almost always are performed in the same way: selecting audio is done by clicking and dragging over the visual representation, copying, cutting and pasting is done either through the Edit menu or through keyboard shortcuts. Whenever one of these actions is performed, this automatically adds an associated module for that edit action in the module chain (*figure 46*). This way, the user does not even need to be aware of the edit chain, allowing him to focus on basic editing.

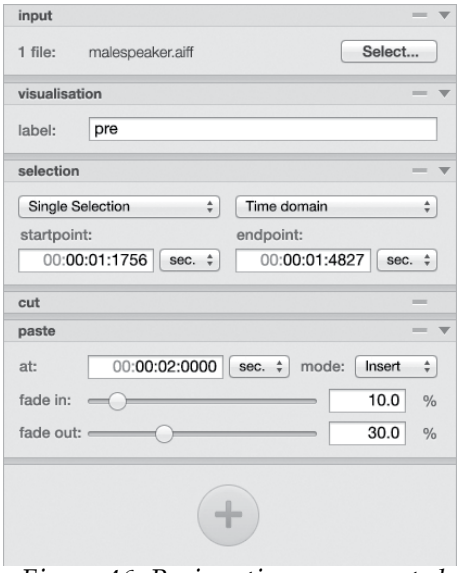


Figure 46. Basic actions represented as modules: input, visualisation, selection, cut and paste.

Similarly, quick edit actions that are applied through interaction with a visual representation are also automatically listed as edit module in the chain. These can include gain change, pitch shifting, time stretching and many more, as described in chapter 3.3.4, *Quick Editing*.

An edit module can also be added by clicking the ⊕ button at the end of the chain. A popup menu will appear that groups and lists all available edits that can be applied (*figure 47*).

Editing

The user can thus choose to edit audio by only interacting with the visual representation (*quick editing*), or by only specifying modules in the chain, or a combination of these two approaches. In this combined approach, an edit can be roughly specified through quick editing and then fine-tuned through its associated edit module.

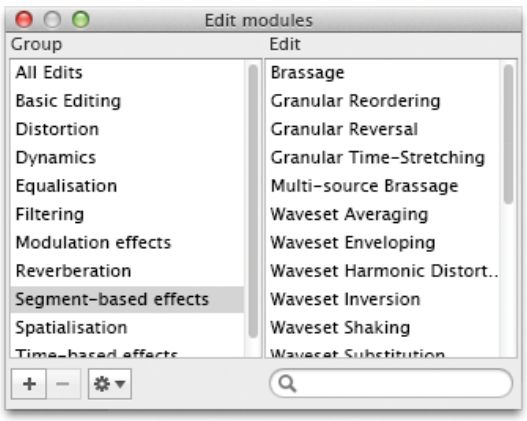


Figure 47. The edit module selector.

Quick editing can be done either through the waveform display or the spectrum display. Selections can be made by dragging over a display, and the kind of selection can be changed by selecting the mode in the top-right of the visualisations section (see *figure 48*

– options are time selection, rectangular selection and lasso, the latter two are only applicable in the spectrum display). Automated selections, as discussed in chapter 3.4, can be made by adding the respective module to the chain. When a selection is made, its contents can be manipulated, as is illustrated in *figure 48*. In this example, a time-frequency selection made in the spectrum display is stretched and skewed, respectively resulting in transposition and spectral delay of the selected audio.

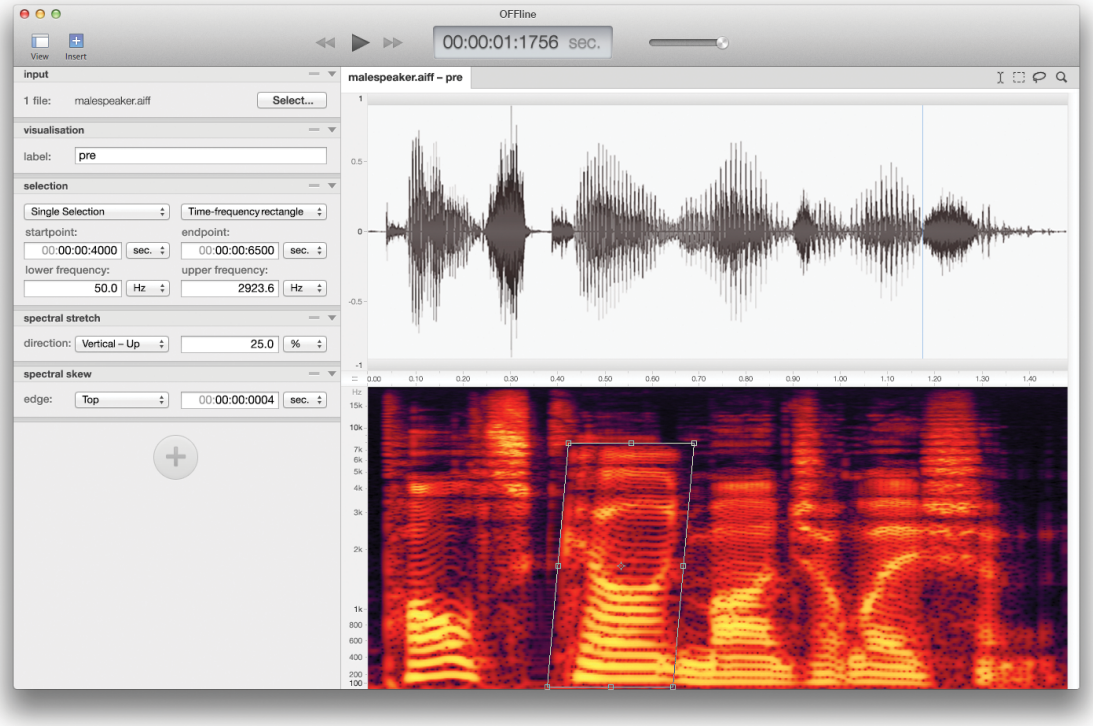


Figure 48. OFFline with a single audio file loaded. Both a waveform and a spectrogram are displayed, visualising the audio at a specific point in the editing chain (here right after the input module). A spectral edit is made through manipulating a spectral (time-frequency) selection.

OFFline supports defining multiple selections, which allows the user to edit multiple segments uniformly and simultaneously. The selections can be defined by hand or, as just mentioned, through automation, either by providing a specification that includes details such as number of segments, segment length and position, or by providing a benchmark segment that is used to select similar segments of audio. Selections may overlap, allowing for segments of audio to be processed multiple times (*figure 49*). This is particularly interesting for techniques in which the selected segments are reordered and concatenated, causing a segment that is selected multiple times to also occur multiple times in the result of the reordering.

Parameter Automation

The visualisations can be used to define a great variety of edits, but they can also be used to define time-varying parameters, by drawing a curve on top of a visualisation along the time axis. The curve now specifies the time-varying

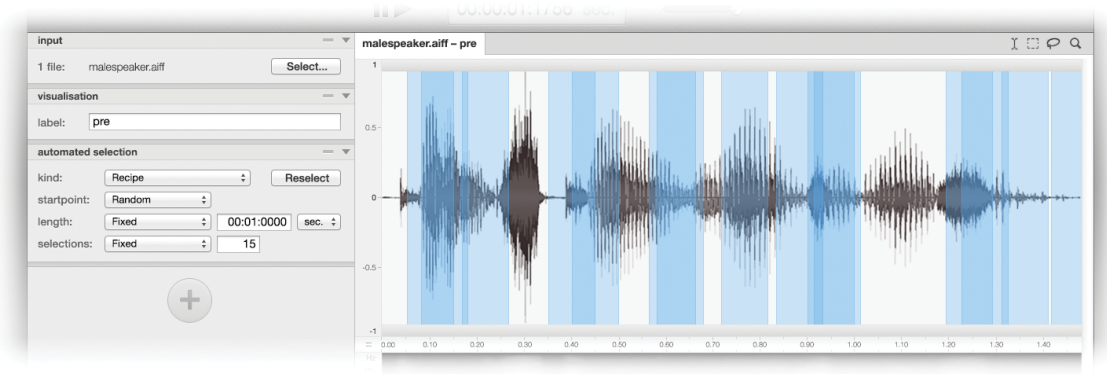


Figure 49. Automated selections based on a recipe that describes the number of selections and the length to be a fixed value, but the position to be random. This causes certain segments to be selected multiple times.

value of an edit parameter. This process is called parameter automation (chapter 3.6.1). Every one-dimensional numerical parameter is suitable for parameter automation, as its value can change over time. *Figure 50* illustrates how a parameter can be automated, in this case the playback speed, following the example described in chapter 3.6.1.

The curve is drawn on top of the waveform by setting breakpoints that can be moved to a different time or value position. Using a HUD the breakpoints can be precisely set to a specific time or value and an easing function can be defined for a smooth transition to the next breakpoint. Through this same HUD the user can specify multiple settings for the same parameter, resulting in multiple variations of the edit. This technique of creating multiple outputs from one input is called *branch editing*, which is described in chapter 3.6.2.

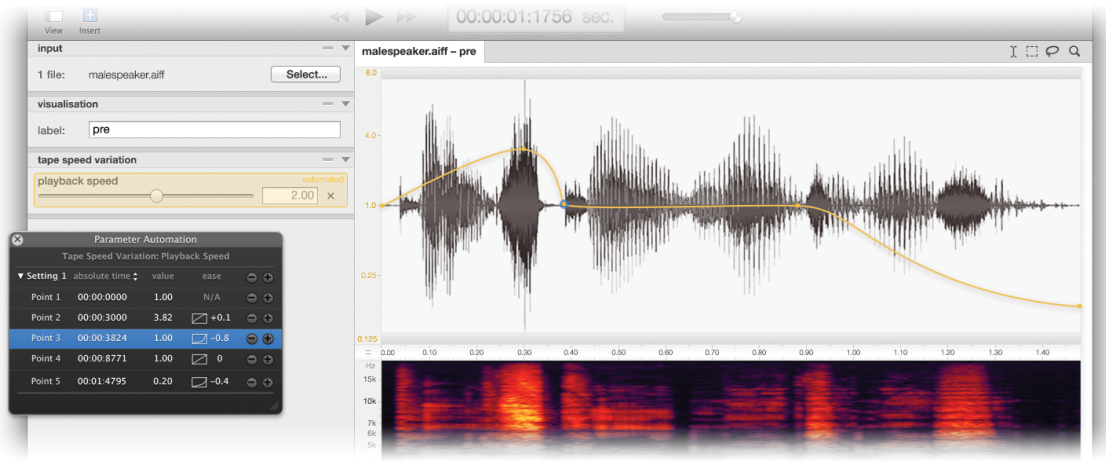


Figure 50. Parameter automation in OFFline. Here the playback speed is varied over time, as specified by the curve that is drawn over the waveform.

Layers

OFFline implements a principle called Layered Audio. In essence this is multi-track editing, but it goes beyond simply mixing together tracks. Layered Audio is foremost a means to visually apply edits in which one or more layers of audio influence another or in which multiple layers are combined in a specific way. Such edits can include ducking, modulation-based techniques

such as ring modulation and vocoding, side-chain compression, morphing and many more.

When selecting such an edit, it requires specifying a secondary input source (or more when applicable). The visualisation section will now show each input source as waveform under each other. Here each waveform can be positioned horizontally along the time axis. *Figure 51* shows how this is done for vocoding, which requires a modulator and carrier input source.

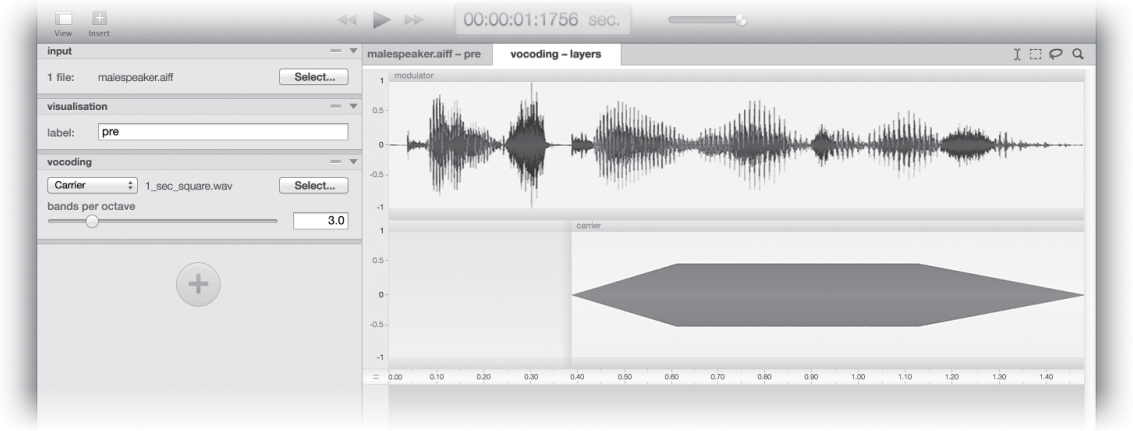


Figure 51. Audio layers for vocoding. The modulator is the file we loaded, displayed as the topmost waveform. The carrier is a secondary input source loaded through the vocoding module and is displayed beneath the modulator waveform.

By visually layering the involved audio sources in an edit, this edit becomes easier to understand and adjust. Layers can visually be repositioned relative to each other, enabling the user to fine-tune their edit.

Module development

OFFline can offer a great variety of possible edits, but can also be extended by the user to include even more edits. Through an integrated development environment, edit modules can be constructed using a dedicated and powerful, yet simple programming language. *Figure 52* (also *41*) shows how any existing module – in this case the vocoding module from *figure 51* – can be inspected and adjusted. This way OFFline can be personalised to exactly meet the user's needs. In a similar fashion new edit modules can be constructed. In the top section of the module edit screen, parameters can be defined by specifying a parameter type (such as input source, slider, drop-down), a name to display in the user interface as well as one for use in the module code (a *code tag*), and aspects applicable for the selected type. In the lower section of the screen, the code editor allows the user to write the audio processing code for the module, providing access to the module parameters through their code tags. Intricacies such as correctly processing multi-channel audio and handling selections are dealt with implicitly by OFFline, making module development relatively easy.

Summary

OFFline demonstrates a possible implementation of the design principles discussed in this thesis. It is recognisable as an audio editor in that it offers the

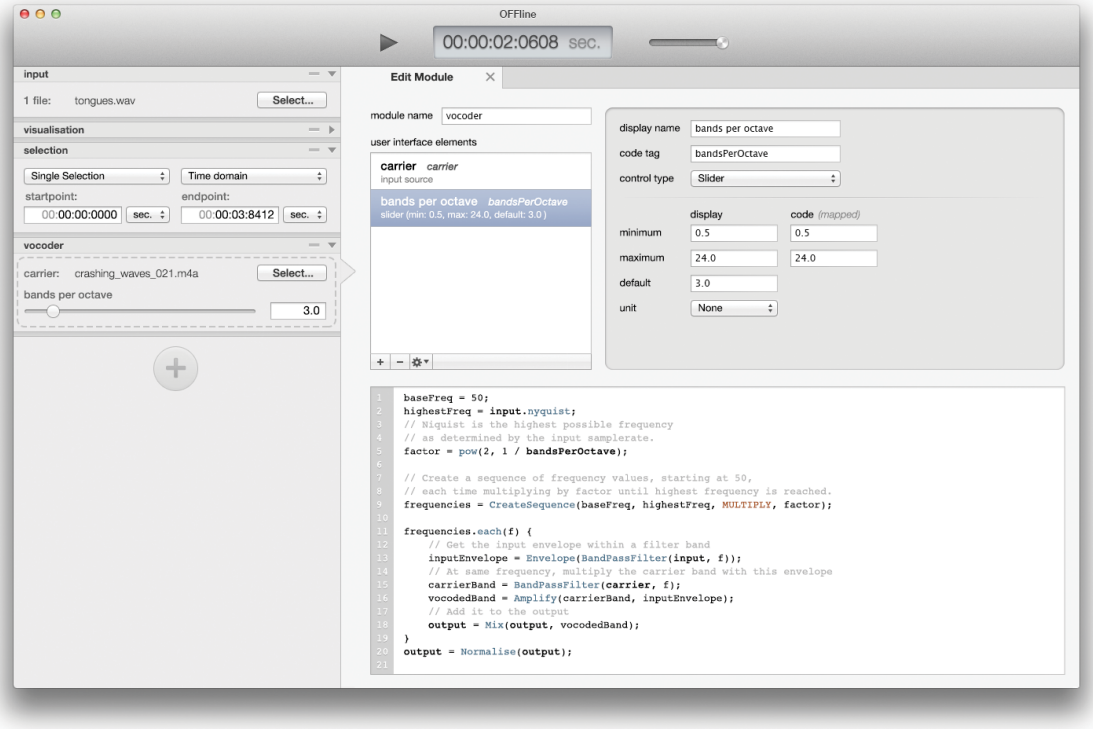


Figure 52. The module development environment in OFFline. A module can be created or edited through this interface. Parameters can be specified for the graphical user interface, and a code editor allows for programming the processing part of the module.

same functionality that is common to most other audio editors: file loading and recording, defining selections, cutting and pasting audio, a visual representation of the audio as the most prominent user interface element and of course the possibility to apply a variety of edits and transformations to the audio.

OFFline is different because it offers this functionality in a more coherent and flexible way. Through combinations of modular editing, quick editing, multiple selections, parameter automation and layered audio, a great variety of audio edits become available in an intuitive, informative and non-destructive manner.

Appendices

1. Research Methods

1.1 Three Studies

Three studies have been conducted to define the audio editor:

1. A **questionnaire** on user background and usage determines user types and audio editor usage for each user type.
2. A **comparison** between audio editors determines the current state of audio editors, available audio editor types, and common features among these audio editor types.
3. A **follow-up** questionnaire joins these first two studies by determining which editors are used by which user types.

This section will describe how each of these studies was set up. The following section will describe the results.

1.1.1 A Questionnaire

A survey was conducted in the form of an online questionnaire, for which more than 700 people active in the field of sound & music technology were invited to participate. The survey was divided into 5 pages:

1. **Occupation & Method**
   A number of 7-point semantic differential scales (poles such as *music* ↔ *sound* / *hardware* ↔ *software* / *graphical* ↔ *textual*) was used to determine the participants' professional background and method of working.
2. **Work Division**
   The participants specified how much time they spend on composition, performance, recording, editing, synthesis and analysis.
3. **Software Usage**
   Different types of audio software were listed, for which participants specified how much they use each kind, through how many different software applications. Additionally they could specify how much free and commercial software they use, and if they use default settings or presets, or personal presets.
4. **Audio Editor Usage**
   The participants could specify for what purpose they use an audio editor (editing, recording, conversion, analysis, synthesis, arranging), and which features they regard as unimportant or essential.
5. **Field Research**
   A list of audio editing software was provided, which could be supplemented by the participant. Any available research literature concerning audio editing software the participant might know of, was also requested.

A definition of the software type *audio editor* was intentionally not provided beforehand, because through their answers, the participants might help form this definition. The survey was designed to answer two main questions:

Users: What kind of people use an audio editor?
- Which user groups can be distinguished?
- What are their characteristics?

Usage: How do they use an audio editor?
- What role does the audio editor have amongst other audio software?
- For which purposes is it used?
- Which features are essential?

Additionally, the survey would provide a list of audio editors that would function as the basis for the second survey described further on, the comparison.

Target Group and Turnout

The survey was aimed at people who are in any way involved in sound and/or music technology. An invitation for participation was sent by email to the entire list of members of the International Computer Music Association, students and teachers from the Utrecht School of Music & Technology, people from the Sonic Arts Research Centre at Queen's University in Belfast, and a large number of colleagues from the author's personal address book. Additionally, some participants proposed to forward the invitation to fellow colleagues and/or students, which brought the total number of invited people somewhere between 700 and 800. The survey website tracked around 350 visits, and registered some 230 completed surveys, of which after investigation 211 provided valid data. With this selection of people it is not the intention to cover fully every possible user group, but just to gain a clearer understanding of which user groups exist in the field. For that purpose this selection should be appropriate for an acceptable representation. The survey was conducted between July and November 2008.

1.1.2 A Comparison

The software type *audio editor* covers a collection of widely varying applications. Each one displays a variety of features, principally including the most general features, but also including less prevalent features. Only when one has an overview of all editors, it is possible to say whether a feature is common or not. A comparison was conducted between close to 50 audio editors, scoring the available features, which resulted in such an overview. This comparison was conducted in the first quarter of 2010.

Editors

The list of audio editors used for this comparison was initially assembled from personal knowledge, then extended by reading literature[1] and searching on internet. Additionally, through the questionnaire the participants were asked to list what they thought were audio editors. The full list of editors included in this comparison can be found in the appendices. A number of software applications from the original list are excluded from the comparison, for various reasons; some were discontinued, others didn't have any editing capabilities or could better be categorised as a different kind of audio software. These editors are listed in *table 1*.

Reason not included	***Applications***
Discontinued *(no available literature)*	Spark XL – TC Electronic Sound Sculptor – Jeff Smith
Analysis / Visualisation *(no editing capabilities)*	Sonic Visualizer – Queen Mary University London Raven – Cornell Lab INA Acousmographe – GRM
Synthesizer	Coagula – Rasmus Ekman
MIDI editor	Rosegarden – Chris Cannam & Andy Green
Digital Audio Workstation	Pro Tools – Digidesign Cubase / Nuendo – Steinberg Logic – Apple XO Wave – XO Audio Digital Performer – MOTU Ardour – The Ardour Company Reaper – Cockos Inc.
No editing capabilities	Jokosher – Jokosher

Table 1. Editors not included in the comparison.

Features

The list of features was initially quite small, but grew whenever an editor revealed an unlisted feature. The features were grouped in the following categories:

- **Operating Systems**
  On which Operating Systems does the software run?
- **Source Selection / Import / Export**
  How is the source audio material loaded, recorded, generated, organised?

[1] Mostly magazines such as *Sound on Sound* and *Interface* (a Dutch magazine for musicians and producers – not to be confused with the similarly named predecessor of the Journal of New Music Research)

- **Work-flow / Basic Actions**
  What kind of editing tools are available?
- **Effects**
  What kind of audio transforming processes ("effects") are available?
- **Plug-in Support / Extendibility**
  Through what kind of architectures can the software's functionality be extended?
- **Miscellaneous**
  Various extra features, often presented in individual windows or dialogues.

The full list of features can also be found in appendix 4. Qualitative features, such as high resolution or phase-correct signal processing, are not taken in account, since it is hard to express these quantitatively. A good example of the consequence of this decision can be found in the two highest scoring audio editors, being WaveLab and Audacity. WaveLab costs over $500, while Audacity is free. The difference can mainly be found in aspects that influence how well it can be used in a professional setting, such as format support, usability and integrability.

Comparison Tool & Statistics

For each editor that was inspected, a checklist of its features had to be stored. An online application was built to insert the audio editor data and automatically (re)calculate comparison statistics. It can be found on www.audio-editor.info/comparison. Audio editors can be sorted by name, as well as by the following values:

- **Feature count**
  How many features the audio editor includes.
- **Score**
- Similar to feature count, but features are weighed by their occurrence. If the score is much higher than the feature count, the included features are the more common ones.
- **Average match**
  An indication for how well an audio editor matches to each other audio editor in the comparison. The number of features that an audio editor has in common with an other audio editor is expressed as a *match percentage*. The average of the match percentages against all other audio editors in the comparison is expressed as the *average match* of an audio editor.

Features can be listed by category, as well as by *weight* (in how many audio editors it is included) and *score* (similar, but weighed by the feature count of the audio editors in which it is included).

These statistics provide insight in which audio editors are the most versatile, the most common or most eccentric, as well as which features are most common. The results follow in section 1.2 of this appendix.

1.1.3 A Follow-up

In order to join the results of the questionnaire and the comparison, a shorter questionnaire was written, as a follow-up. This consisted of only 5 questions:

1. With which user group(s) do you identify yourself?
2. Are you a student or teacher?
3. When choosing audio editing software, which quality / qualities do you prefer?
4. Which of the following applications do you use?
5. What, do you believe, is missing in current audio editors?

All questions except question 5 are multiple-choice. The options for question 1 are the user groups deduced from the questionnaire, which are listed in the following section. The qualities that can be selected for question 3 are listed below, and the applications listed as answers for question 4 are all audio editors that are included in the comparison.

- **Speed**
  Fast rendering waveforms, swift processing
- **Accuracy**
  Sample-accurate control, precise processing parameters
- **Quality**
  Pristine processing, without artefacts that might occur from resampling, resynthesis or (de)compression
- **Simplicity**
  Easy interface, no unnecessary adornments, basic actions
- **Diversity**
  A wide range of functions, all-in-one, can be used for different purposes
- **Compatibility**
  Can be used in different set-ups, on different Operating Systems, with different file types
- **Connectivity**
  Fits in a digital studio with all sorts of signals and files moving between applications and devices
- **Creativity**
  Flexible, including non-standard functionality, allowing creative editing
- **Personality**
  Can be personalised through settings and presets
- **Extendibility**
  Can be extended through programming or scripting

This follow-up should provide insight in which user group uses which audio editor type(s). The comparison only included quantitative data, and therefore question 3 is included to gain some understanding of quality as well. For this article, data was gathered from 93 participants, 43 of which had

also participated in the first questionnaire. The follow-up was conducted in November 2010.

1.2 Results

1.2.1 User Groups

Participants were asked for a concise description of their occupation. Most of them used conventional descriptors such as *"producer"*, *"sound designer"* or *"composer"*, though in practice people gave them a highly personal

Classification	*Activities*	*Preferences*
Sound Designer	***recording*** ***editing*** synthesis	• sample-accurate, nondestructive edits • batch editing • various kinds of (fast rendering) analysis methods • multiple channels • advanced synthesis options
Producer	***recording*** ***editing*** sequencing	• efficient recording • batch editing • audio file management
Audio Engineer	***recording*** ***editing*** analysis	• efficient recording • batch editing • personal presets • audio file management
Composer / Sound Artist	composing ***editing***	• plug-in support • fast waveform and spectrogram • signal generators • nondestructive editing
Composer / Researcher	composing ***editing*** ***conversion*** analysis programming	• format conversion • analysis methods
Performer	***performance*** ***composition***	• memory usage • personal presets
Software Developer	***programming*** ***synthesis*** ***analysis***	• technical terminology

Table 2. Classification of audio editor users. This table lists their everyday activities, of which those that are performed using audio editors are ***emphasised****. It also lists which features each group generally regards as essential in audio editing software. (September 2010)*

interpretation, resulting in widely varying characteristics among people with a similar occupation. These descriptors however appeared to be the most useful starting point for categorising participants. Between all participants a match was calculated, based on the answers they gave. An algorithm used these matches to group users. The algorithm was fine-tuned using the descriptors participants provided. With participants grouped together, a piece of software was written to analyse the questionnaire data.

For each group, answer averages and differences were identified, as well as a common conventional descriptor. A descriptor such as *"computer music composer"* was derived from similar descriptors each participant in this group gave themselves. Among all groups, the averages and differences were compared to see which questions were answered distinctively by individual groups, which were answered commonly, and on which questions participants within groups didn't agree. To further compare groups, a *self-organising map* was used. A self-organising map (SOM) reduces the dimensionality of vectors (feature sets), and can be used to map and organise multidimensional data – in this case for each participant the answers to all questions – into two dimensions[2], making it possible to see which participants answered similarly. Each participant was colour-coded according to the group he was identified with. This way, an image was created of which groups were well distinguishable and which groups mingled together.

After carefully merging overlapping and indistinctive groups, a selection of occupations remained. The conventional descriptors did provide a fairly consistent image of each occupation's background, roughly corresponding to generally understood characterisations associated with it. For this research the following classifications are selected, in descending order according to how much a group uses audio editors (*table 2*).

These groups were also listed in the follow-up, where participants were asked to select with which user group they identified themselves. They were explicitly asked to only select an option if it applied considerably, but more than 75% of the participants listed 2 or even more user groups with which they identified.

Observing the activities listed in *table 2* reveals that no single user group has a set of unique activities; each user group overlaps in activities with one or more other groups. This might explain why so many participants are active in multiple occupations.

1.2.2 Students and Teachers

Not every participant described their occupation as a profession such as one of the above. Some stated their occupation was *"student"*, *"teacher"* or *"professor"*, occasionally also providing some area of expertise. Students and teachers might be active in any of the above professions, though most teachers had a great similarity with the *"researcher"* (which was confirmed through the

[2] The self-organising map only uses the two spatial dimensions to distribute participants; no specific parameters are mapped along the axes.

follow-up), who often happens to be connected to an educational institute. Students, often still developing a focus, on average provided the same answers as teachers, though students were more moderate in their answers. Teachers, as it seems, have a clear idea of how they spend their time on their occupation, and have developed a firmer opinion about what is important and what not.

1.2.3 Average Usage

An audio editor is a tool that is used very often, compared to other kinds of audio software. It is primarily used for basic editing actions (cut/pasting, amplitude adjustments) and recording, but in a lesser degree also for conversion and analysis of audio files. Some also use an audio editor for synthesis (using signal generators), or for arranging audio files. Features essential to almost everyone include a waveform display, the ability to record audio, multichannel (2+) support, and a good undo mechanism (*undo* is explained in detail in chapter 3.1).

1.2.4 Types of Audio Editors

Various types of audio editors could be identified from the statistics derived from the comparison. *Figure 53* shows a self-organising map[3] that was used to identify groups of audio editors with common features. The resulting audio editor types are listed in *table 3*.

The highest scoring editors, such as the above mentioned WaveLab and

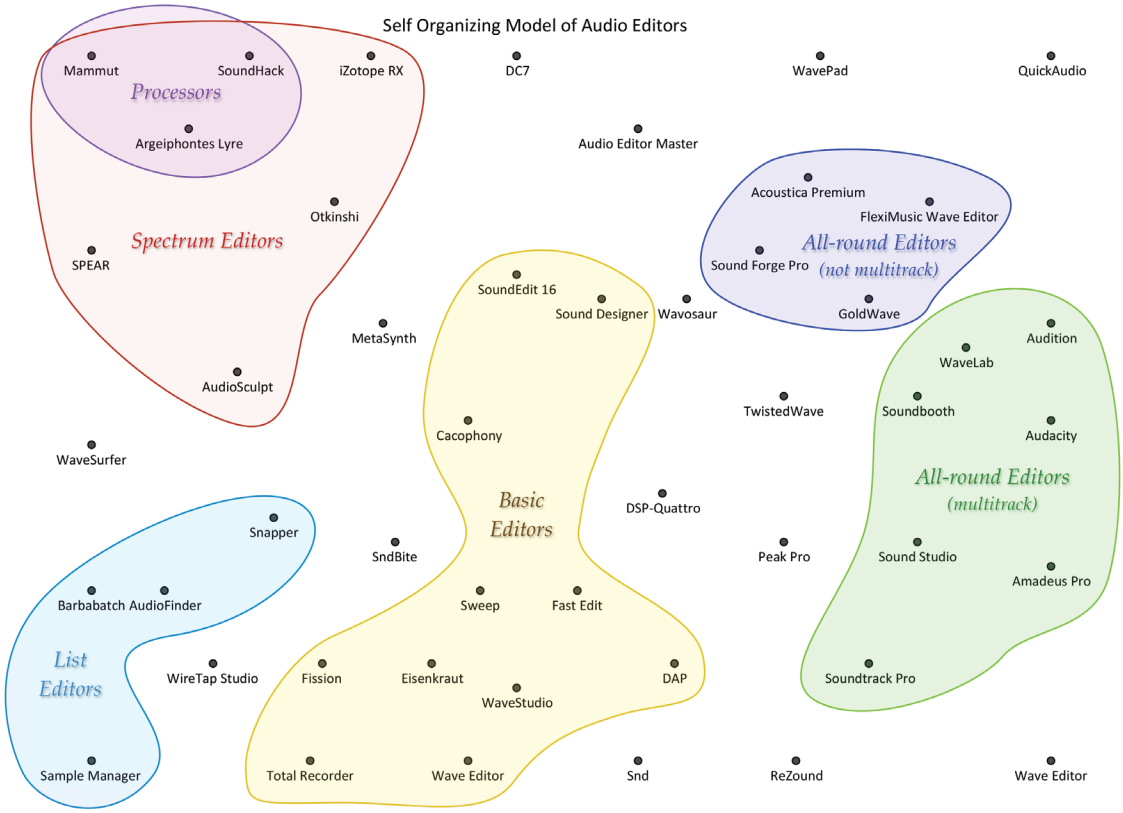


Figure 53. A self-organising model of audio editors. In this model, audio editor features are compared to organise the editors in a two-dimensional plane.

[3] See *User Groups* for a description of self-organising maps.

Type	*Description*	*Examples*
Wave Editor	See *Terminology* for a description	
Spectrum Editor	See *Terminology* for a description	AudioSculpt, SPEAR, iZotope RX
All-round Editor	Multipurpose audio editor, with wide supply of processing, synthesis and analysis features. Multitrack editors resemble DAW's (without musical understanding. See Scope).	Audacity, Amadeus Pro, Soundtrack Pro
Basic Editor	Opposite of all-round, only including basic actions. Basic editors often excel in innovative user interfaces.	Fission, Cacophony, Eisenkraut, WaveStudio
List Editor	Operating either on multiple files, presented as file lists or folders, or on individual files directly from a directory structure. Batch editors and Asset Management applications are included in this category.	Barbabatch, Snapper, Sample Manager, AudioFinder
Processors	Not so much an editor, having no basic editing capabilities like selection, cut/paste. Processing is applied to the entire file.	SoundHack, Mammut, Argeïphontes Lyre
Partial Editor	Sinusoidal-model based partial editing as basis	SPEAR, Oᵗkinshi
Nondestructive Editor	Operations can be altered at any time	Sample Manager, Soundtrack Pro
Abstract Editor	Extreme processing with fuzzy parameters	Argeïphontes Lyre, Mammut

Table 3. List of possible audio editor types, some of which can be discriminated from the self-organising map.

Audacity, are *all-round multi-track audio editors*, which can be utilised for multiple purposes, but the choice depends on price, quality and operating system. WaveLab is only available for Windows, while Audacity is cross-platform, and free, making it the application of choice for various audio hardware developers to ship it with their products.

The opposite of all-round audio editors, *basic audio editors* can be identified by looking at how much they match with other editors (this is called the *average match* in the statistics). Editors such as Fast Edit, Cacophony and Fission, having a high average match, are focused solely on providing the most basic editing features. These audio editors are the quick-cut-and-paste applications.

The Apple Mac Operating System is the most supported platform, quickly followed by Microsoft Windows. Less editors are available for the GNU/Linux Operating Systems, though Audacity is among them, able to provide in most

needs.

One remarkable outcome was that one of the first audio editors in history, the archetypical Sound Designer, ended up almost in the centre of the self-organising map, confirming most editors have something in common with it.

1.2.5 Basic Features

The comparison is dominated by *wave editors*, applications built around an amplitude waveform display as the centrepiece of the Graphical User Interface. Such editors can quickly load an audio file through a standard dialog or record audio from an external input, thereby displaying the waveform of the audio. Dragging over this waveform lets the user select a portion of the audio file, which can than be cut, copied and/or pasted again. Volume can be adjusted (gain, normalise) over the entire file or the selection. In most editors, all of these actions can also be undone. More than two-thirds of all editors includes this basic functionality. Every feature beyond that is supported in less than two-thirds of all editors, and thus might be considered as nonstandard.

Interesting is that for both time and frequency domain, the ability to visualise it seems as common as to perform transformations in it. The frequency domain is a transformed version of the time domain. In the time domain audio can be observed as time-varying amplitudes. Using mathematical techniques such as the Fourier Transform, a frequency analysis can be made on a period of time, revealing the harmonic and phase spectrum within that period of time. These spectra together form what is called a frequency domain representation (Risset 1991, p.9). A frequency domain representation is both more difficult to read and to implement; it requires a specific knowledge. Perhaps due to this relatively complex required transformation and/or possible audible artefacts, frequency visualisations and transformations are less implemented in audio editors than their time domain counterparts. Time and frequency domain will be explained in more detail in chapter 3.3, *Representation & Manipulation*.

Less common features, such as additional analysis methods, batch editing (explained in chapter 3.1), signal generators (chapter 3.2.3), expression evaluator or parameter automation (chapter 3.6) are in many cases added in a separate dialog window. Perhaps this is because redesigning the entire workflow so that its features are more completely integrated, hence offering a workflow that is unlike the more traditional workflow such as can be found in the archetypical Sound Designer, might confuse the user.

1.2.6 Audio Editor Preference

Audacity, the highest scoring audio editor when looking at included features, is also the most used editor among all user groups, with 62 out of 94 participants using it. Surprisingly it is followed by SPEAR (44 participating users) and SoundHack, both spectrum editors. Though all three are freely available which might explain their popularity, the latter two are not among the most predominant editor type, being Wave Editors (which includes multi-track

#	*Audio Editor*	*Users*	*Editor Type*
1	Audacity	62	All-round Multi-track
2	SPEAR	44	Spectrum (sinusoidal model)
3	SoundHack	33	Spectrum (processor)
4	Peak Pro	23	Wave Editor
5	iZotope RX	15	Spectrum Editor
6	Audition	15	All-round Multi-track
7	AudioSculpt	12	Spectrum Editor
8	Snapper	11	List Editor
9	Soundtrack Pro	10	All-round Multi-track
10	WaveLab	10	All-round Multi-track

Table 4. Top 10 Audio Editors, among 94 participants

editors). That this isn't per se the most used kind of audio editor is also confirmed when looking at the top 10 editors (*table 4*), which includes iZotope RX at 5th and AudioSculpt at 7th position, both also spectrum editors.

No real preference for a specific type of audio editor by a specific user group can clearly be identified. Notable however is that some audio editors that were developed in the academic domain, particularly Snd and AudioSculpt, are also mostly used by people active in this area.

1.2.7 Qualitative Aspects

As one might expect, high quality processing at high speed is valued by most participants, quality being especially essential to sound designers and audio engineers. Speed is considerably more essential to producers and engineers than to researchers, which is perhaps attributable to different demands in commercial and academic sectors. Producers and engineers responded much alike in the first questionnaire as well as the follow-up. For instance, both also have a relatively higher need for diversity in software functionality compared to other participants. One difference though is that audio engineers seem to demand less room for creativity compared to other participating user groups. This might confirm a common work division in a studio production, being that the engineer takes care of the technical part of the production, executing the demands of the producer or artist who are more concerned with the creative part.

No further remarkable qualitative highlights were revealed by the follow-up, except from more or less predictable outcomes, such as the greater need for compatibility and connectivity among performers (as performance setups might change with each venue) and the relatively higher interest in extendibility (through programming, scripting and other software development) among software developers.

1.3 Summary

The *audio editor* is an umbrella term for a number of different audio editing software types, ranging from wave editors to spectrum editors, and from basic to all-round multi-track audio editors. The majority includes a common set of basic features (file loading / recording, waveform display, selecting, cutting / pasting, amplitude adjustment, undo) that was also found in the earliest audio editors, which were little more than a virtual audio tape recorder and cutting table.

Audio editors are used in a wide variety of occupations, and for different purposes (mainly basic editing and recording, but in lesser degree also conversion and analysis). A number of user groups could be discriminated by looking at occupation, though many participants identified themselves with multiple occupations. No clear preference for an audio editor type by any user group could be identified.

Though no one definition of "the audio editor" can be formulated, the area covered by it and users associated with it are identified, forming a usable lattice for further research.

2. A Questionnaire

The first of three studies used to define the audio editor was an online questionnaire. It focussed on user background and usage. It was used to determine user types and audio editor usage for each user type. The questionnaire is listed in this appendix.

Occupation & Method

What is more important in your work?

music	□ □ □ □ □ □ □	sound	□ *N/A*

What is the nature of your work?

acoustic	□ □ □ □ □ □ □	electronic	□ *N/A*
analogue	□ □ □ □ □ □ □	digital	□ *N/A*
live	□ □ □ □ □ □ □	studio	□ *N/A*

How do you approach your work?

emotional	□ □ □ □ □ □ □	formal	□ *N/A*
hardware	□ □ □ □ □ □ □	software	□ *N/A*
graphical	□ □ □ □ □ □ □	textual/notation	□ *N/A*
abstract	□ □ □ □ □ □ □	sample-accurate	□ *N/A*
top-down	□ □ □ □ □ □ □	bottom-up	□ *N/A*

What is your background?

music	□ □ □ □ □ □ □	technology	□ *N/A*

How would you describe your occupation?

[…]

How many years have you been involved in this?

years	□ <5 □ <10 □ <20 □ 20+	□ *N/A*

Work Division

How often do you busy yourself with the following activities?

composition	never	□ □ □ □ □	very often	□ *N/A*
performance	never	□ □ □ □ □	very often	□ *N/A*
recording	never	□ □ □ □ □	very often	□ *N/A*
editing	never	□ □ □ □ □	very often	□ *N/A*
synthesis	never	□ □ □ □ □	very often	□ *N/A*
analysis	never	□ □ □ □ □	very often	□ *N/A*

How much time of the total time you spend on your occupation, is for (personal) experiments?

0% □ □ □ □ □ □ □ □ □ □ □ 100%

Software Usage

How often do you use the following types of software?

audio editor
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

sequencer
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

digital audio workstation
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

graphical programming environment
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

textual programming environment
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

notation software
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

synthesizer (including plug-ins)
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

effect plug-ins
never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

How many different applications/packages of the following types of software do you use?

audio editor
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

sequencer
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

digital audio workstation
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

graphical programming environment
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

textual programming environment
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

notation software
☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ more ☐ *N/A*

synthesizer (including plug-ins)
☐ 0 ☐ <5 ☐ <10 ☐ <20 ☐ 20+ ☐ *N/A*

effect plug-ins
☐ 0 ☐ <5 ☐ <10 ☐ <20 ☐ 20+ ☐ *N/A*

How often do you use the following kinds of software?

free/open source never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*
Commercial never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

How often do you rely on…?

default parameter settings never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*
factory presets never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*
personal presets never ☐ ☐ ☐ ☐ ☐ very often ☐ *N/A*

What kind of terminology do you find easier to use?

abstract □ □ □ □ □ □ □ technical □ *N/A*

Audio Editor Usage

How would you describe the role of audio editing software in your work?

supportive □ □ □ □ □ □ □ central □ *N/A*

For what purpose do you use audio editing software?

basic editing	never	□	□	□	□	□	very often	□ *N/A*
recording	never	□	□	□	□	□	very often	□ *N/A*
conversion	never	□	□	□	□	□	very often	□ *N/A*
analysis	never	□	□	□	□	□	very often	□ *N/A*
synthesis	never	□	□	□	□	□	very often	□ *N/A*
arranging	never	□	□	□	□	□	very often	□ *N/A*

How much do you value the presence of the following features in audio editing software?

recording
unimportant □ □ □ □ essential □ *N/A*

arranging
unimportant □ □ □ □ essential □ *N/A*

batch editing
unimportant □ □ □ □ essential □ *N/A*

non-destructive editing
unimportant □ □ □ □ essential □ *N/A*

unlimited undo/redo's
unimportant □ □ □ □ essential □ *N/A*

effect presets
unimportant □ □ □ □ essential □ *N/A*

sample-accurate control
unimportant □ □ □ □ essential □ *N/A*

control over memory usage
unimportant □ □ □ □ essential □ *N/A*

single-window User Interface
unimportant □ □ □ □ essential □ *N/A*

extendibility (programming interface)
unimportant □ □ □ □ essential □ *N/A*

plug-in support (VST, AU, LADSPA)
unimportant □ □ □ □ essential □ *N/A*

>2 channel audio support
unimportant □ □ □ □ essential □ *N/A*

midi-to-audio
unimportant □ □ □ □ essential □ *N/A*

module file import (*.mod, *.xm …)
unimportant □ □ □ □ essential □ *N/A*

waveform display
unimportant □ □ □ □ essential □ *N/A*

fast waveform rendering
unimportant □ □ □ □ essential □ *N/A*
frequency analysis
unimportant □ □ □ □ essential □ *N/A*
signal generators
unimportant □ □ □ □ essential □ *N/A*
more advanced synthesis options
unimportant □ □ □ □ essential □ *N/A*
audio file management
unimportant □ □ □ □ essential □ *N/A*

Research

Do you know of any audio editors other than the following?
(A list of audio editors was provided)
[…]

Do you know of research literature on audio editing software?
[…]

3. A List of Audio Editors

Following are all audio editors included in the comparison described in chapter 2. The version number refers to the version that was used in the comparison, not the current version.

Editor name	*Developer*	*Version*	*Price*
Acoustica Premium	Acon Digital Media	4.1.0	$ 119,90
Amadeus Pro	HairerSoft	1.4.5	$ 40-
Argeïphontes Lyre	Akira Rabelais	5.06	Free
Audacity	Dominic Mazzoni, Roger Dannenberg	1.26a	Free
Audio Editor Master	Metrix Audio Solution Inc.	5 Plus	$ 29,95
AudioFinder	Iced Audio	4.9.8	$ 69,95
AudioSculpt	IRCAM	2.5	€ 250,-
Audition	Adobe	3	$ 349-
Barbabatch	Audio Ease	4.0.33	€ 403,41
Cacophony	Richard Bannister	1.3.1	$ 25-
DAP	Richard Kent	2.1.5	Free
DC7	Diamond Cut Productions	7.15	$ 149,99
DSP-Quattro	i3 s.r.l.	3.1.2	$ 199-
Eisenkraut	Hanns Holger Rutz	0.72	Free
Fast Edit	Minnetonka Audio Software Inc.	4.0	$ 199-
Fission	Rogue Amoeba	1.6.8	$ 32-
FlexiMusic Wave Editor	FlexiMusic	6.3.0.1	$ 20-
GoldWave	GoldWave Inc.	5.55	$ 49-
iZotope RX	iZotope	1.2.1	$ 349-
Mammut	Oyvind Hammer, Kjetil Matheussen	V0.58	Free
MetaSynth	U&I Software	5.0	$ 599-
Otkinshi	Naotoshi Osaka, Takafumi Hikichi		
Peak Pro	BIAS	6.1.1	$ 399,95
QuickAudio	Sion Software	2.0	Free
ReZound	David W. Durham		Free
Sample Manager	Audiofile Engineering	3.1	$ 79-

Editor name	*Developer*	*Version*	*Price*
Snapper	Audio Ease	1.0.11	$ 59-
Snd	Bill Schottstaedt	11	Free
SndBite	Bill Poser	3.3	Free
Sound Designer	Digidesign		$ 995-
Sound Forge Pro	Sony	10.0b	€ 351,95
Sound Studio	Freeverse	3.6	$ 79,99
Soundbooth	Adobe	CS4	$ 199-
SoundEdit 16	Macromedia		$ 299-
SoundHack	Tom Erbe	896	Free
Soundtrack Pro	Apple	2.0.1	$ 500-
SPEAR	Michael Klingbeil	0.7.4	Free
Sweep	Conrad Parker	0.9.3	Free
Total Recorder	High Criteria Inc.	8.1	$ 35,95
TwistedWave	TwistedWave	1.8	$ 79,90
Wave Editor	Audiofile Engineering	1.4	$ 79-
Wave Editor	Abyss Media Company	3.0.3.1	Free
WaveLab	Steinberg	6.0	$ 549,99
WavePad	NCH Software	4.25	Free
WaveStudio	Creative Technology Ltd.	7.10.17	Free
WaveSurfer	Kåre Sjölander and Jonas Beskow	1.8.5	Free
Wavosaur	Wavosaur	1.0.5.0	Free
WireTap Studio	Ambrosia Software Inc.	1.1.0	$ 69-

4. Audio Editor Features

Listed here are all features on which the audio editors in the preceding appendix (48 in total) were compared. With each feature the number of audio editors in which this feature is included is listed.

Operating Systems

- **Mac OS** *(28)*
  The software runs on a Macintosh Operating System, such as Mac OS X.
- **Windows** *(24)*
  The software runs on a Microsoft Windows Operating System.
- **GNU/Linux** *(9)*
  The software runs on a GNU/Linux Operating System.

Source Selection / Import / Export

- **Default file dialog** *(46)*
  A standard file dialog is presented to select a file from the disk. Playback of audio files may be integrated.
- **Recording** *(34)*
  The software provides means to record audio data, either from a hardware input (possibly a microphone), or from a virtual input (being another software application running on the same machine).
- **Synthesis / Signal generators** *(19)*
  The software includes rudimentary synthesis models to generate waveforms, noise or more complex signals to be used as source material.
- **Project sound library / Sound hierarchy** *(10)*
  The software facilitates organisation of sound files currently opened or relating to a project currently opened.
- **Non-audio format import** *(6)*
  The software can open and process non-audio file formats (such as MIDI, SDIF, imagery, analysis data etc.) and convert these into audio.
- **Non-audio format export** *(8)*
  The software can analyse audio files and write non-audio file formats derived from this analysis to disk (such as MIDI, SDIF, imagery, analysis data etc.)

Workflow / Basic Actions

- **Waveform-centred workflow** *(39)*
  A waveform display is the centre part of the Graphical User Interface of the software, through which most audio operations are performed. The software is thus a “Wave Editor”.
- **Manual selection** *(42)*
  A selection on the time axis can be made in a visualisation of the audio, by dragging over it.

- **Nonstandard Selection** *(7)*
  Selections other than a single portion of time (with amplitude or frequency information) can be made, such as multiple selections (manually or automated), frequency bands, or time/frequency chunks (a frequency band at a time segment), either automatically or manually, using a rectangular or lasso-like selection tool.
- **Cut / Paste / Copy** *(41)*
  Basic editing actions (cutting, pasting and copying a portion of audio) are included in the software.
- **Special Pasting** *(19)*
  Audio data from the clipboard can be pasted at a certain position, being inserted before, mixed with or replacing the adjacent audio data. A fade-in/out of the clipboard data might be specified.
- **Track mixing** *(10)*
  Multiple tracks are provided to arrange (at the time axis) and mix audio material.
- **Envelope drawing** *(8)*
  Envelopes for amplitude or other parameters can be drawn directly over a visualisation of the audio.
- **Markers / Labels** *(30)*
  The software allows placing markers or labels at certain positions on the timeline.
- **Unlimited Undo / Redo** *(36)*
  Subsequently performed operations can all be undone one by one, and redone if no new operation was performed after undoing it. There is no limit to the amount of operations that can be undone or redone this way.
- **Non-destructive** *(8)*
  Subsequently applied operations can each be altered at any point in time, without having to undo any other operation.
- **Individual sample alteration** *(12)*
  By zooming in far enough so that individual samples (amplitude points) can be distinguished, dragging or drawing on such a point changes its amplitude value (which can be useful to manually erase spikes).
- **Varispeed / Scrubbing** *(21)*
  The audio can be played back at different speeds, or can be scrubbed (moving the playhead manually back and/or forth over the audio).
- **Batch Editing / Processing** *(21)*
  The software provides a dialog for selecting a batch of files, and equally processing these automatically.
- **Effect Chain** *(10)*
  A bit similar to batch editing, a chain of effects can be preset, to be applied again and again over currently loaded audio data.
- **Expression Evaluator** *(1)*
  A dialog is provided to program and execute/generate simple audio processes or signals using a basic syntax.

- **Layered Audio** *(1)*
  The software provides alternative audio track layering. Default track mixing simply sums the amplitudes of all tracks. Audio Layering enables blending (masking, ring modulation, ...) consecutive tracks, much as Photoshop can blend graphic layers in different ways.

Effects

- **Amplitude Adjustment** *(43)*
  The software includes audio operations and effects affecting amplitude, such as amplification, normalisation and compression.
- **Frequency Adjustment** *(26)*
  The software includes audio operations and effects affecting frequency (or frequency band amplitude), such as filtering, equalisation, Bass Boost / Loudness and spectral modifications.
- **Noise Removal** *(16)*
  The software includes audio operations and effects for removing noise, artefacts and other undesired aspects of sound.
- **Tempo / Pitch Change** *(31)*
  The software includes time-preserving pitch alteration and/or pitch-preserving time alteration.
- **Delay and Reverb** *(21)*
  The software includes audio operations and effects based on delay, such as chorus, echo and reverb.
- **Unusual / Radical Effects** *(11)*
  The software includes audio operations and effects that may alter the character of the sound considerably, more applicable for artistic purposes than for enhancing audio quality. Effects might include vocoding, convolution, freezing, and the Phase Vocoder.

Analysis / Visualisations

- **Time domain visualisations** *(44)*
  The software includes visualisations based on time domain information (not requiring any complex transformation such as FFT). The standard waveform, a decibel plot, etc.
- **Frequency domain visualisations** *(28)*
  The software includes visualisations based on frequency domain information (requiring a Fourier- or similar analysis). Spectrogram.
- **Pitch / Partial Tracking** *(6)*
  The software is capable of tracking the time-varying fundamental pitch or frequencies of individual components of a sound.

Plug-in Support / Extendability

- **VST / AU / LADSPA** *(22)*
  The software supports one or more plug-in architectures, which allow the

user to apply external effects plug-ins.

- **Scriptable** *(9)*
  The software allows scripts, written e.g. in AppleScript, to be executed to automatically perform certain tasks within the software.
- **Programming** *(5)*
  The software can be extended through an integrated programming environment, an API (Application Programming Interface), or by providing a custom plug-in architecture.

Miscellaneous

- **Edit / Insert Meta Data** *(21)*
  Meta Data, such as ID3, can be inserted or edited.
- **CD Mastering / Burning** *(11)*
  The software provides means to master and burn a CD.
- **Corrupt File Repair** *(1)*
  The software includes means to repair files that cannot initially be loaded, due to errors in the raw data structure of the file.

5. A Follow-up

An online follow-up questionnaire was designed to join the first two studies (the questionnaire in appendix 1 and the audio editor comparison) by determining which editors are used by which user types. The user type definitions are not universal, they are resultant of the questionnaire (appendix 2) and only apply in the context of this thesis.

1. With which user group(s) do you identify yourself?

Multiple answers are possible, but do so only if an answer applies considerably to you.

- **Sound Designer**
  A Sound Designer acquires, specifies and manipulates audio, to create a desired effect or mood. This involves manipulation of previously composed or recorded audio, or composition of new audio material. Sound Design is employed in a variety of areas including film, TV commercials and games.
  emphasis: Sound
- **Producer**
  A Producer oversees and manages the recording of an artist's music, supervising the entire process through mixing and mastering, shaping the music according to an overall creative vision for the production. The Producer might also be responsible for arranging and managing recording sessions, musicians, budgets and other entrepreneurial aspects.
  emphasis: Music
- **Audio Engineer**
  An Audio Engineer records, edits, manipulates, mixes and/or masters sound by technical means in order to realise an artist's or producer's creative vision. This can be employed amongst others in music production, post-production for video or film, broadcasting, live sound reinforcement and games. In smaller productions the engineer is often also the producer.
  emphasis: Technology
- **Composer / Sound Artist**
  A Composer / Sound Artist makes extensive use of synthesisers and effects in his compositional process. In many aspects a Sound Artist is the same as a Sound Designer, though there is a much greater emphasis on composition.
  emphasis: Composition and Sound Synthesis
- **Researcher**
  A researcher (often connected to an institute as professor or educator) explores (new) methods for music composition and / or sound processing, working on the foundations of music production rather than the end result.
  emphasis: Music- and Sound Technology

- **Computer Music Composer**
  A composer of computer (or electronic) music focusses on music production, sound design (and sonic composition) and the development of music systems, exploring possibilities of composing with computers and electronic devices. Composers of computer music have much in common with a "researcher", though they often focus more on practice rather than theory.
  emphasis: Music- and Sound Technology
- **Performer**
  The Performer composes and / or produces music from the viewpoint of a musician (instrumentalist), while considering an audience. Activities may include performance, production and post-production, but also design of new instruments or playing methods.
  emphasis: Music and Play
- **Software Developer**
  A Software Developer (for sound or music) can be designer and computer programmer of software that is used in music production, sound synthesis, manipulation and analysis, or (algorithmic) composition. The latter requires a background in musical composition, whilst the others require expertise in digital signal processing.
  emphasis: Digital Technology
- **Other**
  None of the above descriptions apply to you.

2. Are you a student or teacher?

- Student
- Teacher
- Both
- Neither

3. When choosing audio editing software, which quality / qualities do you prefer?

- **Speed**
  Fast rendering waveforms, swift processing
- **Accuracy**
  Sample-accurate control, precise processing parameters
- **Quality**
  Pristine processing, without artefacts
- **Simplicity**
  Easy interface, no fancy stuff, basic actions
- **Diversity**
  A wide range of functions, all-in-one, can be used for different purposes
- **Compatibility**
  Can be used in different set-ups, on different Operating Systems, with different file types

- **Connectivity**
  Fits in a digital studio with all sorts of signals and files going between applications and devices
- **Creativity**
  Flexible, including non-standard functionality, allowing creative editing
- **Personality**
  Can be personalised through settings and presets
- **Extendability**
  Can be extended through programming or scripting

4. Which of the following applications do you use?

(Here a list of all audio editors used in the comparison was provided)

5. What, do you believe, is missing in current audio editors?

[…]

6. Digital Audio Workstations

In defining the audio editor, the software type called Digital Audio Workstation[1] has been mentioned in the exegesis (see *2.2.1.2: Scope* and appendix *1.1.2: Comparison*) but deliberately not explored in detail. So far the discussion has focussed on delineating the software type **audio editor**, and although most Digital Audio Workstations contain an audio editing section, DAW software itself is distinctively different. Digital Audio Workstations typically include multitrack audio recording and sequencing capabilities, as well as editing and mixing sections. Modern DAW's include MIDI recording, editing and playback (sometimes in the form of an extensive music notation editor), as well as sophisticated audio routing capabilities and plug-in support for software effects and instruments.

A Digital Audio Workstation can be used for editing audio, but that is just one of its functions. It was kept out of the comparison of audio editor types, because its functions go beyond audio editing. However, though DAW's were excluded from the discussion what an audio editor is, they (or at least their editing sections) can be included in surveying what an audio editor can do. The complete list of the DAW's that have been surveyed can be found in the appendices.

Common Ground

Audio editing in a DAW is done at the level of a *clip* (other names are *region*, *item*, *event* or *object*), a single piece of audio positioned on a track. In the user interface, a clip is represented by a rectangular block, commonly with a waveform drawn into it, that can be dragged and dropped horizontally to a different point along the time axis, or vertically to a different track. Many audio editors (particularly multi-track editors) adopt a similar concept for the visual representation of a segment of audio. Aside from repositioning, a clip can also be trimmed, removing audio from the start or the end. It can be copied and pasted, cut, split, spliced and reversed. A very common feature that is accessible through the graphical user interface of a clip are fades (in and out). These can often be applied simply by dragging the edges of a clip inwards, from a specific corner or handle. Equally common is overall gain adjustment of a clip, which can be applied by either dragging vertically inside the clip, adjusting a gain line that runs through the clip or by adjusting a pop-up gain knob or slider.

Different Scope

The main difference in editing capabilities between audio editors and DAW's can be attributed to the difference in scope. As described in chapter 2.2.1.2, an

[1] The term Digital Audio Workstation historically describes a specialised hardware setup (possibly computer-based) that facilitates recording, editing and producing audio. With the advent of software that was capable of facilitating this in a single application, the term Digital Audio Workstation became a more general term, identifying either hardware or software that facilitates audio recording, editing and producing.

audio editor doesn't normally use musical markers or reference points, whereas a DAW includes various musical reference points, such as pitch, tempo, metre - its representations are essentially musical. A DAW has a timeline that can be divided in clock time (seconds and minutes) as well as in musical time (beats and bars). Musical tempo and timing are core concepts in DAW's. Most DAW's also offer MIDI note sequencing for playing virtual instruments and synthesisers, making tuning also a core concept.

Tempo & Timing

In most DAWs tempo and timing can be controlled very precisely. A bar can be defined, as can tempo, and these can be specified to changed over time. The bars and beats are displayed along the time axis. Beats have a fixed width; a tempo change will make the play-head progress at a different speed over the beats. Audio clips can be positioned freely at any point along the time axis, or snap to bars, beats and subdivisions, easing the synchronisation of clips on different tracks. Snapping makes looping a clip each specific number of beats or bars very easy too. Clip looping is often provided in a similar way to trimming; a clip edge is dragged outwards to loop the audio in the clip to fit the new clip length.

Not only can a clip be snapped to beats, but its individual contents can be independently snapped to beats as well. If a clip contains audio with distinctive transients, these can be analysed and serve as points at which the clip is effectively split. After this analysis stage, the individual segments can then automatically be moved and quantised to the timeline. It is then also possible to introduce variations to the timing, for instance to add some swing to an otherwise straight drum track.

This process of analysing for transients, splitting up a clip and moving around its segments can cause gaps to occur between segments. If the recording is really dense or has a distinctive background noise, this might be undesirable. A relatively recent development alleviates this problem. *Flex-Time* (also known as *Time Warp*, *Elastic Time* and other variations) also analyses a clip for transients, but does not split it up. Instead it allows for individual segments to be time-stretched, allowing some segments to occur earlier or later on in time, while maintaining a continuous clip of audio.

Tuning

Very similar to Flex-Time is *Flex-Pitch* (also called *Elastic Pitch*, *pitch shift* or *pitch correction*), a feature present in a growing number of DAW's. Flex-Pitch can be applied to monophonic audio recordings. It performs a pitch analysis and then commonly splits the audio clip on distinctive pitch changes, repositioning each segment vertically over a piano-roll to indicate the pitch of each segment. Any deviation from the specified tuning scale will be visible as an offset from the piano-roll grid. Pitch can be adjusted by dragging segments up or down the piano-roll to snap to an exact note, or to a whole different note, allowing for correction of misplayed notes and for recomposition. The pitch of multiple

segments can also be quantised, often in a gradual way (using a slider to vary the approximation towards exact pitches in the tuning scale), allowing for some natural-sounding deviation to remain. Vibrato, often recognized as pitch variation within a segment, can also be removed by straightening a segment to its particular pitch. This pitch straightening sounds particularly unnatural when applied to voice recordings, which is what characterises severe auto-tune as sometimes heard in popular music (when it is used more as an effect instead of a corrective tool). One artist who is well known for making use of this technique (and for naming a popular auto-tune app for iPhone) is T-Pain.

In Search of Perfection

Many DAW's include a feature which is called *Comping* (short for "compilation"). It allows the user to compile an ideal final take of a performance by selecting the best parts of multiple previous takes. Multiple takes can often be recorded in a single recording session by defining a loop region on the timeline. When the play head reaches the end and jumps back to the beginning of the loop region, a new take is recorded and positioned underneath the previous take (or above, it differs per DAW). The common approach to comping is then to simply select start and end points in a take, and cross-fades between different takes are added automatically. Though comping is regularly used in search of the best available possible take of a musical performance, it is not a musical feature per se, but can just as well be regarded as a convenient cut-ting-and-pasting mechanism.

Another feature that can be found increasingly often in DAWs is more musical. Groove extraction and, related to that, beat replacement are features that are related to the transient detection mentioned in the paragraph on tempo and timing. Given a recording of a drumbeat, the groove can be extracted by analysing the transients and converting this to pure note information (not so much a pitched note, but particularly the timing of each note). This note information can then be used to trigger synthesised or sampled beats of which volume and timing can be quantised or manipulated more precisely.

Corrective Editing

Overall, the audio editing feature set offered by most DAW's serves a clear purpose: it focuses on corrective edits. The user begins with trimming, slicing and fading, removing any unwanted material. Then using comping, the user selects the best parts of multiple recording takes. The various tempo and timing tools allow the user to arrange everything in sync and in the right rhythm. Finally, everything is harmonized using the tuning tools.

Whereas an audio editor is a fully packed tool shed for multi-purpose audio editing, in which the focus commonly lies on a single audio file, the audio editing in a DAW is very focused on making the many audio clips in a single project work well with each other.

These corrective edits are all on the level of an audio clip. More transformative editing functionality can be found in the effects section of each

track. Though there are many plug-ins that serve a corrective purpose (for tuning, noise removal, gain adjustment), edits in this section are generally referred to as *effects* and hence as a different division.

For this survey, the following DAW's have been inspected (in alphabetical order):

Company	***Software***
Ableton	Ableton Live
Sony	Acid
The Ardour Company	Ardour
Bitwig	Bitwig Studio
Steinberg	Cubase
MOTU	Digital Performer
Image Line	FL Studio
Apple	Garageband
Apple	Logic
Acoustica	Mixcraft
MuTools	MuLab
Steinberg	Nuendo
AVID	Pro Tools
Cocos Inc.	Reaper
Propellerhead	Reason
Magix	Samplitude Pro
Cakewalk	Sonar
PreSonus	Studio One
Tracktion	Tracktion
XO Audio	XO Wave

7. Audio Programming Languages

In search of a suitable existing computer programming language for audio, the languages listed in this appendix were examined. The list is constructed from multiple resources, most important of which are the PLUM list[2] and a list of music composition languages constructed by William Alves[3]. See also Kornfeld (1980), Krasner (1980), Pope (1993a), Loy and Abbot (1985), Scaletti (2002).

Language name	*Author*	*Year*
4CED	Curtis Abbott	1979
ABC	Chris Walshaw	1991
AC Toolbox	Paul Berg	
Adagio	Dannenberg, R. B.	1984
Arctic	Dannenberg, R. B., P. McAvinney, and D. Rubine	1986
ARES/MARS	IRIS s.r.l.	1991 (prototype version), 1997 (commercial version)
CAL	Twelve Tone software (Cakewalk)	1992?
ChucK	Ge Wang and Perry R. Cook	2003
CLM (Common Lisp Music)	William Schottstaedt	1990
CMIX	Paul Lansky	
Cmusic	F. Richard Moore (UCSD)	198?
Common Music	Heinrich Taube	
Csound	Barry Vercoe (MIT)	1986
CYBIL	Alexandre Burton and Jean Piche	1995
Foo	Gerhard Eckel	1994
FORMULA	Dave Anderson (UC Berkeley) & Ron Kuivila (Wesleyan U)	1991
Fugue	Roger B. Dannenberg, Chris Fraley	1988
GROOVE	Mathews, Moore (Bell Labs)	1968
JSyn	Phil Burk	1997

[2] The PLUM (Programming Languages Used for Music) list by Tim Thompson can be found at http://www.nosuch.com/tjt/plum.html

[3] William Alves' list of music composition languages can be found at http://ftp.cs.uu.nl/pub/MIDI/DOC/music.languages

Language name	*Author*	*Year*
Kyma	Carla Scaletti	1986
Loki	Juniper Moon	1996
Max	Miller Puckette (IRCAM) and David Zicarelli (Opcode Systems, Cycling 74)	1986
McLeyvier Command Language	David McLey (et al)	1981
MODE	Stephen Travis Pope	1987
MSP	David Zicarelli	1998
MUS10	David Poole	1974
MUS8	R. Boudinot	1976
MusBox or MBox	Gareth Loy	1979
Music 11	Barry Vercoe	1973?
MUSIC 360	Barry Vercoe (MIT)	1969
MUSIC 4B	Godfrey Winham & Hubert Howe (Princeton)	1964
Music Composition Language	Fairlight Instruments	1980
MUSIC N: MUSIC I/II/III/IV	Max Mathews (Bell Labs)	1957, ‘58, ‘60, ‘63
MUSIC V	Max Mathews (Bell Labs)	1969
Music-1000	Dean Wallraff	1978
MUSIC7	Lejaren Hiller (SUNY Buffalo)	
MusicKit	David Jaffe, Julius Smith	1988
MusicScript	David Piott	2000
MusicXML	Michael Good	2000
MUSIGOL	Donald McInnis, Paul S. Davis, William A Wulf (U. of Virginia)	1966
NetSound	Michael Casey and Paris Smaragdis	1996
Nyquist	Roger Dannenberg	1992
OpenMusic (OM)	Gérard Assayag and Carlos Agon with Olivier Delerue	1997
ORGANUM 1	Tisato	1974
OSW - Open Sound World	Adriien Fried and then open source	2000
OUTPERFORM	D. Jaeger, D. Lester (University of Toronto)	1972

Language name	*Author*	*Year*
PATCH	Lynx Crowe	
Patchwork	Mikael Laurson, Jaques Duthen and Camilo Rueda	± 1990
Pcmusic	F. Richard Moore	1995
Petal	Chris Cannam, Andy Green, Richard Bown and Guillaume Laurent	1995
PILE	Paul Berg (Inst. of Sonology, Utrecht)	1977
Pla	Bill Schottstaedt (Stanford)	1983
PLACOMP	D. Murray, J. Beauchamp, G. Loitz (Univ. of Illinois)	1978
PLAY (PLAY1, PLAY2)	Joel Chadabe & Roger Myers (NYSU Albany)	1977
PMML	Satoshi Nishimura	
POD6, POD7	Barry Truax (Simon Fraser University)	1973, 1975
PROD	M. Green	
Pure Data (PD)	Miller Puckette	1997
Pyrite	James McCartney	
pysco	Paul Winkler	1999
Q	Albert Graef	1991, 2002 (midi interface), 2003 (audio interface)
Quasimodo	Paul Barton-Davis	1998
Ravel	Jim Binkley	1988
Realtime Composition Library for MAX	Karlheinz Essl	1992 ff.
Sambox	Bill Schottstaedt	1979
SAOL (MPEG-4 Structured Audio Orchestra Language)	ISO MPEG, project led by Eric Scheirer	1997-1998
Sapphire	Jim Finnis	1995
Scala	Manuel Op de Coul	1996
SCRIPT	New England Digital Corp.	1984
SDIF	Matthew Wright, Adrian Freed, David Wessel, et al	1997
Silence	Michael Gogins	1996
SKINI	Perry Cook	1996

Language name	*Author*	*Year*
SMDL	Stephen R. Mounce	
SMOKE		
Snd	Bill Schottstaedt	
SoundModel (CCPL)	Peter Lunden	1993
SSP	G. M. Koenig (Inst. of Sonology, Utrecht)	1975
SSSP	Buxton et al. (University of Toronto)	1978
ST	Xenakis (CEMAMu, Paris)	
STK (Synthesis Toolkit)	Perry Cook	1996
STORM	Lynx Crowe	
SuperCollider Server	James McCartney	2003
SuperCollider	James McCartney	1996
Symbolic Composer	Pekka Tolonen	1991
SYMPFONICS	B. Vassaur (U. of Tulsa)	1972
SYN4B	Neil Rolnick & Phillipe Prevot (IR-CAM)	1978
SynSeq.pm	kanak	2000
SYNTA L-II	Wayne Slawson (University of Pitts-burgh)	1977
Tclmidi	Mike Durian	
TREE/COTREE	Curtis Roads (MIT)	1978
UPIC	Xenakis (CEMAMu)	
Zel		1998

References

Apple, 2006. Audio unit programming guide. Available from: https://developer.apple.com/library/mac/documentation/musicaudio/Conceptual/AudioUnitProgrammingGuide/Introduction/Introduction.html .

Apple 2007. Audio unit component services reference. Available from: http://developer.apple.com/iphone/library/documentation/AudioUnit/Reference/AUComponentServicesReference/Reference/reference.html .

Benassi, B. 2003. (music recording) Satisfaction: Energy Production.

Bogaards, N., 2008. Sound editing on the sonogram. *In: Proceedings of the 14th International Conference on Auditory Display,* Paris, France.

Collins, N., 2010. *Introduction to computer music.* Chichester: John Wiley and Sons.

Cooper, A. R., Robert; Cronin, Dave. (2007). *About face 3: the essentials of interaction design*: John Wiley & Sons, Inc.

Deutsch, S., 2007. The soundtrack (putting music in its place). *The Soundtrack,* 1 (1).

Evangelista, G., 1991. Wavelet transforms that we can play. *In:* Giovanni De, P., Aldo, P., and Curtis, R. eds. *Representations of musical signals*: MIT Press, 119-136.

Gentner, D. N., Jakob. (1996). The Anti-Mac interface. *Communications of the ACM,* 39(8), 70-82.

Giovanni De, P., Aldo, P., and Curtis, R. (Eds.). 1991. *Representations of musical signals*: MIT Press.

Haus, G., 1993. *Music processing.* Madison, Wis.: A-R Editions.

Herzberg, L., 1989. Review: Sound designer. *ST-Log*.Heuvelmans, T. 2005. Overeenkomsten tussen dip & dap, *not published*.

Heuvelmans, T. 2006. Approaching digital audio – a technical audio software classification, *not published*.

Holland, S. (2013). *Music and Human-Computer Interaction*: Springer Publishing Company, Incorporated.

Howard, D. M., and Angus, J., 2001. *Acoustics and psychoacoustics.* Butterworth-Heinemann.

Katz, R. A., 2007. *Mastering audio : The art and the science.* 2nd ed. Amsterdam; Boston: Elsevier/Focal Press.

Kirby, D. G., and Shute, S. A., 1988. *The exploitation and realization of a random access digital audio editor.* Paper presented at the IEEE Broadcasting Convention.

Kornfeld, W. A., 1980. Machine tongues vii: Lisp. *Computer Music Journal,* 4 (2), 6-12.

Kraftwerk. 1978. (music recording) Die roboter: Kling Klang (EMI).

Krasner, G., 1980. Machine tongues vii: The design of a smalltalk music system. *Computer Music Journal,* 4 (4), 4-14.

Liscio, C., 2009. Drawing waveforms. Available from: http://supermegaultra-groovy.com/2009/10/06/drawing-waveforms/.

Loy, D. G., 1985. Designing an operating environment for a realtime performance processing system. *In:* Truax, B. ed. *Proceedings of the international computer music conference* Vol. 1985. San Francisco: Computer Music Association.

Loy, D. G., 2007a. *Musimathics: The mathematical foundations of music.* Vol. 1. Cambridge: MIT Press.

Loy, D. G., 2007b. *Musimathics: The mathematical foundations of music.* Vol. 2. Cambridge: MIT Press.

Loy, D. G., and Abbot, C., 1985. Programming languages for computer music synthesis, performance, and composition. *ACM Comput. Surv.,* 17 (2), 235-265.

Marieb, E. N., and Hoehn, K., 2007. *Human anatomy & physiology.* 7th ed. San Francisco: Pearson Benjamin Cummings.

Mazzoni, D., and Dannenberg, R., 2001. A fast data structure for disk-based audio editing. *Computer Music Journal,* 26 (2), 62-76.

Mcaulay, R. J., and Quatieri, T. F., 1986. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Trans. on Acoust., Speech, and Signal Processing,* ASSP-34 (4).

Milano, D., 1985. Digidesign sound designer for emulator ii. *Keyboard Magazine*.

Miranda, E. R., 1998. *Computer sound design: Synthesis techniques and programming*. Second Edition ed.: Focal Press.

Moore, F. R., 1990. *Elements of computer music.* Prentice Hall.

Moorer, J. A., 1990, September. *Hard-disk recording and editing of digital audio.* Paper presented at the 89th AES convention.

Norman, D. A., 1990. *The design of everyday things.* 1st Doubleday/Currency ed. New York: Doubleday.

Osaka, N. H., Takafumi. 1999. Visual manipulation environment for sound synthesis, modification, and performance. *ICMC Proceedings*, 1999, 429-432.

Osaka, N. S., Ken-Ichi; Hikichi, Takafumi. 2002. The sound synthesis system "otkinshi": Its data structure and graphical user interface. *ICMC Proceedings*, 2002, 188-191.

Pope, S. T., 1993. Music composition and editing by computer. *In:* Haus, G. ed. *Music processing* Vol. 9: A-R Editions, Inc., 25-72.

Pope, S. T., 1993. Machine tongues xv: Three packages for software sound synthesis. *Computer Music Journal*, 17 (2), 23-54.

Risset, J.-C., & Wessel, D. (1982). Exploration of Timbre by Analysis and Synthesis. In D. Deutsch (Ed.), *The Psychology of music* (pp. 26-58). New York: Academic Press.

Risset, J.-C., 1991. Timbre analysis by synthesis: Representations, imitations, and variants for musical composition. *In:* Giovanni De, P., Aldo, P., and Curtis, R. eds. *Representations of musical signals*: MIT Press, 7-43.

Risset, J.-C., 1998. Forerword. *In:* Miranda, E. R. ed. *Computer sound design: Synthesis techniques and programming*. Oxford: Focal Press.

Roads, C., 1996. *The computer music tutorial.* MIT Press.

Rockmore, D., 1999. The fft: An algorithm the whole family can use. *Computing In Science and Engineering*, 2 (1), 60-64.

Scaletti, C., 2002. Computer music languages, kyma, and the future. *Comput. Music Journal*, 26 (4), 69-82.

Shneiderman, B. (1983). Direct Manipulation. A Step Beyond Programming Languages. *IEEE Transactions on Computers*, 16(8), 57-69.

Smith, S. W., 1997. *The scientist and engineer's guide to digital signal processing.* California Technical Publishing.

Stevenson, A., and Lindberg, C. A. 2005. New oxford american dictionary. In Mckean, E. (Ed.), *The New Oxford American Dictionary* (Second Edition ed.): Oxford University Press.

Sturm, B. L., Roads, C., Mcleran, A., and Shynk, J. J., 2009. Analysis, visualization, and transformation of audio signals using dictionary-based methods. *Journal of New Music Research*, 38 (4), 325-341.

Tomita, I. 1978. (music recording) The visionary flight to the 1448 nebular group of the bootes, *The Bermuda Triangle*: RCA.

Vinet, H., 2003. The representation levels of music information. *CMMR 2003*, 193-209.

Wishart, T., 1988. The composition of vox-5. *Computer Music Journal,* 12 (4), 21-27.

Wishart, T., 1994a. *Audible design: A plain and easy introduction to practical sound composition.* York, UK: Orpheus the Pantomime.

Wishart, T. 1994b. (music recording) Tongues of fire: Orpheus The Pantomime.

Wishart, T., 1996. *On sonic art.* New and rev. ed. Amsterdam: Harwood Academic Publishers.Wishart, T., 2000. Sonic composition in tongues of fire. *Computer Music Journal,* 24 (2), 22-30.

Wishart, T. (2000). Sonic Composition in Tongues of Fire. *Computer Music Journal*, 24(2), 22-30.

Zölzer, U., 2008. Digital audio signal processing. John Wiley & Sons Software.

Zölzer, U., 2011. *DAFX: Digital audio effects.* Wiley Publishing.